Un análisis criptográfico de la
Protocolo de protocolo de enlace TLS 1.3

Benjamín Dowling1 , marc fischlin2 , Félix Gunther1 , y Douglas Stebila3

1Departamento de Ciencias de la Computación, ETH Zürich
2TU Darmstadt
3Universidad de Waterloo

22 de febrero de 2021

Resumen

Analizamos el protocolo de negociación del protocolo Transport Layer Security (TLS), versión 1.3. Abordamos tanto el protocolo de enlace TLS 1.3 completo (el modo de tiempo de ida y vuelta, con firmas para autenticación y (curva elíptica) Diffie-Hellman efímero (intercambio de clave (EC)DHE)), y el modo de reanudación abreviado/“PSK” que utiliza una clave precompartida para autenticación (con clave (EC)DHE opcional intercambio y establecimiento de clave de tiempo de ida y vuelta cero).

Nuestro análisis en el marco de seguridad reduccionista utiliza una seguridad de intercambio de claves de varias etapas. modelo, donde cada una de las muchas claves de sesión derivadas en un solo protocolo de enlace TLS 1.3 está etiquetada con diversas propiedades (como no autenticado versus autenticado unilateralmente versus autenticado mutuamente, si está destinado a proporcionar seguridad hacia adelante, cómo se usa en el protocolo y si la clave está protegida contra ataques de reproducción). Mostramos que estos TLS 1.3 Los modos de protocolo de protocolo de enlace establecen claves de sesión con sus propiedades de seguridad deseadas bajo supuestos criptográficos estándar.

Contenido

1. Introducción

El *protocolo Transport Layer Security (TLS)* es uno de los protocolos criptográficos más implementados en la práctica y protege numerosos accesos a la web y al correo electrónico todos los días. El *protocolo* de negociación TLS permite que un cliente y un servidor se autentiquen entre sí y establezcan una clave, y el *protocolo de capa de registro* posterior proporciona confidencialidad e integridad para la comunicación de los datos de la aplicación. Originalmente desarrollado como el protocolo Secure Sockets Layer (SSL) versión 3 en 1996, TLS versión 1.0 fue estandarizado por Internet Engineering Task Force (IETF) en 1998 [DA99], con revisiones posteriores a la versión 1.1 (2006) [DR06] y la versión 1.2 (2008) [DR08]. A pesar de su despliegue a gran escala, o quizás gracias a él, hemos sido testigos de frecuentes ataques exitosos contra TLS. A partir de 2009, hubo muchos ataques prácticos a la versión 1.2 de TLS entonces vigente que recibieron mucha atención, explotando las debilidades en las primitivas criptográficas subyacentes (como las debilidades en RC4 [ABP+13]), errores en el diseño del protocolo TLS (p. ej., BEAST [Duo11], el ataque Lucky 13 [AP13], el ataque de triple apretón de manos [BDF+14], el ataque POODLE [MDK14], el ataque Logjam [ABD+15]), o fallas en las implementaciones (p. ej., el ataque Heartbleed [Cod14], ataques de máquinas de estado (SMACK [BBDL+15])).

1.1 Desarrollo y estandarización de TLS 1.3

Ante la creciente preocupación por la seguridad de la versión 1.2 de TLS debido a los numerosos ataques, pero también motivada por el deseo de desaprobar los algoritmos antiguos, mejorar la privacidad y reducir la latencia del establecimiento de conexiones, en 2014 el grupo de trabajo de TLS del IETF inició un proceso de varios años para desarrollar y estandarizar una nueva versión de TLS, eventualmente llamada versión 1.3. Desde 2014 hasta 2018, se publicaron un total de 29 borradores de TLS 1.3, con comentarios activos de la industria y el mundo académico, incluidos extensos análisis de seguridad realizados por varios equipos académicos (consulte [PvdM16] para obtener una crónica del desarrollo y análisis de TLS 1.3). El documento que estandariza TLS 1.3, RFC 8446 [Res18], se publicó en agosto de 2018 y ya ha tenido una adopción generalizada.

Desde una perspectiva criptográfica, los principales cambios de diseño en TLS 1.3 en comparación con la versión 1.2 incluyen: (1) cifrar algunos mensajes de reconocimiento con una clave de sesión intermedia, para proporcionar confidencialidad de los datos del reconocimiento, como el certificado del cliente; (2) firmar la transcripción completa del apretón de manos para la autenticación; (3) incluir hashes de mensajes de apretón de manos en una variedad de cálculos clave; (4) usar diferentes claves para cifrar mensajes de protocolo de enlace y datos de aplicaciones; (5) desaprobar una variedad de algoritmos criptográficos (incluido el transporte de claves RSA, el intercambio de claves Diffie-Hellman de campo finito, SHA-1, RC4, modo CBC, MAC-entonces-codificar-luego-encriptar); (6) usar esquemas modernos de encriptación autenticada con datos asociados (AEAD) para proteger los datos de las aplicaciones; y (7) proporcionar apretones de manos con menos flujos de mensajes para reducir la latencia.

Hay dos modos principales del protocolo de protocolo de enlace TLS 1.3. Uno es el protocolo de enlace completo de un tiempo de ida y vuelta (1-RTT), que utiliza certificados de clave pública para el servidor y (opcionalmente) la autenticación del cliente, y (curva elíptica) el intercambio de claves efímeras Diffie-Hellman ((EC)DHE), inspirado por el diseño 'SIGn-and-MAc' (SIGMA) de Krawczyk [Kra03]. Se establecen varias claves de sesión para una variedad de propósitos en este modo: para cifrar parte del protocolo de enlace, para permitir la exportación de material de claves a otras aplicaciones, para la reanudación de la sesión y, por supuesto, para cifrar los datos de la aplicación.
Este modo recibe su nombre del hecho de que los datos de la aplicación se pueden enviar desde el cliente al servidor con la finalización del protocolo de enlace después de un viaje completo de ida y vuelta, lo que significa que hay un tiempo de ida y vuelta (1-RTT) hasta que se puede enviar el primer mensaje de la aplicación. enviado (sin contar las operaciones de red que no son TLS, como las búsquedas de DNS o el protocolo de enlace de 3 vías TCP).

El otro modo principal del protocolo de protocolo de enlace TLS 1.3 es el modo de reanudación o clave precompartida (PSK), en el que la autenticación se basa en una clave precompartida simétrica, con opción

Intercambio de claves (EC)DHE para confidencialidad directa; esto generaliza el protocolo de enlace de reanudación de sesión abreviado de versiones anteriores de TLS. El modo PSK se puede aumentar opcionalmente con un establecimiento de clave de tiempo de ida y vuelta cero (0-RTT), lo que permite al cliente enviar, junto con su primer flujo TLS, datos de la aplicación encriptados con una clave derivada del PSK.

1.2 Análisis de seguridad de TLS

TLS 1.2 y versiones anteriores. Una larga línea de trabajo ha analizado varias versiones del protocolo SSL/TLS utilizando tanto métodos formales como pruebas de seguridad reduccionistas. En el paradigma de seguridad reduccionista, los primeros trabajos [JK02, MSW08, Gaj08] sobre el protocolo de protocolo de enlace trataron con versiones modificadas o truncadas del protocolo, necesarias porque TLS 1.2 y versiones anteriores no tenían una separación estricta de claves: la clave de sesión también se usaba para cifrar los mensajes dentro del protocolo de protocolo de enlace, salvo pruebas de seguridad en modelos de intercambio de claves autenticadas basadas en la indistinguibilidad fuerte en el estilo Bellare-Rogaway [BR94] . También hubo formalizaciones de la seguridad del cifrado autenticado en la capa de registro [Kra01, PRS11]. Un hito importante en los análisis reduccionistas de TLS fue el desarrollo del modelo de seguridad de establecimiento de canal autenticado y confidencial (ACCE) que permitió el análisis combinado de un protocolo de enlace TLS 1.2 completo y un canal seguro en un solo modelo [JKSS12], eludiendo la clave antes mencionada . cuestión de separación; este trabajo fue seguido por una variedad de otros trabajos que analizan la seguridad de varios aspectos de TLS 1.2 [KPW13, KSS13, LSY+14, GKS13, DS15]. Otros enfoques para probar la seguridad de TLS 1.2 dentro del paradigma de seguridad reduccionista incluyen una gama de enfoques modulares y de composición [BFS+13] , así como enfoques que combinan el análisis formal y la seguridad reduccionista [BFK+13, BFK+14].

Borradores de TLS 1.3. El protocolo de protocolo de enlace en los borradores iniciales de TLS 1.3 se basó en parte en el protocolo OPTLS [KW16]. Hubo una variedad de investigaciones sobre la seguridad de varios borradores a lo largo del proceso de estandarización de TLS 1.3. Utilizando el paradigma de seguridad reduccionista, se han realizado análisis del protocolo de protocolo de enlace [DFGS15, KMO+15, DFGS16, KW16, LXZ+16, FGSW16, Kra16b, FG17, BFG19a] y la capa de registro [BMM+15, BT16, LP17, GM17, PS18].
Ha habido una variedad de trabajos que involucran métodos y herramientas formales, como verificadores de modelos y análisis simbólico [CHSv16, CHH+17], y enfoques que combinan implementaciones verificadas con análisis formal y seguridad reduccionista [BFK16, BBD+15, BBF+16, DFK+17].

Estándar TLS 1.3. Desde que se publicó TLS 1.3 como RFC en agosto de 2018, algunos trabajos han abordado el estándar TLS 1.3 final. El ataque Selfie [DG21b] condujo a análisis actualizados de los apretones de manos de PSK [DG21b, AASS19]. Arfaoui et al. [ABF+19] investigó las funciones de privacidad del protocolo de enlace TLS 1.3. Pruebas de seguridad computacional revisadas del protocolo de enlace 1-RTT completo por Diemert y Jager [DJ21] y Davis y Günther [DG21a] técnicas traducidas de Cohn Gordon et al. [CCG+19] para establecer reducciones más estrictas. También ha habido propuestas académicas para mejoras o modificaciones de TLS 1.3, considerando la seguridad directa para el protocolo de enlace 0-RTT [AGJ21], ejecutando TLS 1.3 sobre un protocolo de red diferente [CJJ+19], o definiendo una alternativa basada en KEM apretón de manos que permite el despliegue de esquemas post-cuánticos [SSW20].

1.3 Nuestras contribuciones

Damos un análisis de seguridad reduccionista de tres modos del protocolo de enlace TLS 1.3: el protocolo de enlace 1-RTT completo, el protocolo de enlace PSK (con modo 0-RTT opcional) y el protocolo de enlace PSK-(EC)DHE (con modo 0-RTT opcional)); basado en una abstracción criptográfica de los protocolos que proporcionamos

en la Sección 3. Para llevar a cabo nuestro análisis, formalizamos un modelo de seguridad de intercambio de claves de múltiples etapas que puede capturar una variedad de características asociadas a cada clave de etapa. Nuestro análisis muestra que el diseño del protocolo de enlace TLS 1.3 sigue principios criptográficos sólidos.

modelo de seguridad. Nuestro modelo de seguridad, presentado en la Sección 4, sigue el modelo Bellare-Rogaway (BR) [BR94] para la seguridad del intercambio de claves autenticadas basado en la indistinguibilidad de la clave de sesión, tal como lo formalizaron Brzuska et al. [BFWW11, Brz13], y nuestro modelo se basa específicamente en el modelo de etapas múltiples de Fischlin y Günther [FG14, Gün18]. Este último se ocupa de los protocolos de intercambio de claves que derivan una serie de claves de sesión en el transcurso de múltiples etapas de protocolo. Nuestra extensión de su modelo de intercambio de claves de múltiples etapas nos permite capturar las siguientes características asociadas a la clave de sesión establecida en cada etapa, a la que llamamos clave de etapa:

- *Autenticación:* si una clave de etapa no está autenticada, unilateralmente autenticada o mutuamente autenticada. Ampliamos aún más el modelo de múltiples etapas para capturar *la autenticación actualizable:* la clave de una etapa se puede considerar, por ejemplo, no autenticada en el momento en que se acepta, pero el nivel de autenticación de esta clave se puede "elevar" a autenticado unilateralmente o, potencialmente, en un segundo paso, mutuamente autenticados después de algunas operaciones posteriores, como la verificación de una firma en un mensaje posterior.

- Secreto directo : si una clave de etapa está destinada a proporcionar secreto directo, es decir, que permanece segura después del compromiso de un secreto a largo plazo involucrado en su derivación.

- *Uso de la clave:* si una clave de etapa debe usarse internamente dentro del protocolo (por ejemplo, para cifrar mensajes de protocolo de enlace posteriores) o externamente (por ejemplo, compuesta con un esquema de cifrado simétrico para proteger los mensajes de la aplicación o utilizada en algún otro protocolo de clave simétrica).

- *Rejugabilidad:* si se garantiza que no se establezca una clave de etapa como resultado de un ataque de repetición; Las primeras etapas de los modos 0-RTT no tienen esta garantía.

Nuestro modelo de seguridad viene en dos versiones que capturan la seguridad establecida a través de dos tipos de credenciales: claves públicas o claves simétricas previamente compartidas. Siguiendo el modelo BR, nuestro modelo de compromiso incluye compromiso de clave a largo plazo (corrupción) y compromiso de clave de etapa (revelación). Mientras que otros modelos [CK01, LLM07] capturan aún más el compromiso del estado de la sesión o la aleatoriedad efímera, TLS no está diseñado para ser seguro contra dicha exposición de valores efímeros y, por lo tanto, no incluimos estas capacidades de compromiso en nuestro modelo.

Además de capturar la indistinguibilidad de las teclas del escenario, el modelo también garantiza la solidez de identificadores de sesión utilizando la noción de seguridad de coincidencia de [BFWW11, Brz13].

Análisis de protocolo. Aplicamos nuestro modelo de seguridad de intercambio de claves de varias etapas en las Secciones 5 y 6 para analizar los tres modos del protocolo de enlace TLS 1.3: 1-RTT completo, PSK y PSK-(EC)DHE, y los dos últimos tienen 0-RTT opcional. teclas. Hay cuatro clases principales de claves de etapa cubiertas en el análisis: cifrado de datos temprano y claves de exportación (ETS, EEMS, solo presentes en PSK con modos 0-RTT); secretos de tráfico de apretón de manos (tkchs, tkshs); secretos de tráfico de aplicaciones (CATS, SATS); y claves exportadas (RMS para reanudación de sesión, EMS para otras claves exportadas). Esto da como resultado seis teclas de escenario en el modo 1-RTT completo y ocho teclas de escenario en los modos PSK.

Como se señaló anteriormente, nuestro modelo de seguridad nos permite capturar con precisión varias características de diferentes claves de etapa. Por ejemplo, considere el secreto de tráfico de protocolo de enlace del cliente tkchs, que se utiliza para cifrar los mensajes de protocolo de enlace del cliente al servidor. En el protocolo de enlace 1-RTT completo, esta clave es inicialmente

no autenticados, luego autenticados unilateralmente a través de una firma de servidor después de alcanzar la etapa 3 y, en última instancia, pueden autenticarse mutuamente después de alcanzar la etapa 6 si el cliente se autentica; es secreto directo; está destinado para uso interno dentro del protocolo; y se garantiza que no se reproducirá. Por el contrario, en el protocolo de enlace de PSK, esta clave se autentica mutuamente tan pronto como se establece, pero no tiene confidencialidad hacia adelante. Finalmente, en el protocolo de enlace PSK-EC (DHE), esta clave no se autentica inicialmente, luego se actualiza a autenticación unilateral y eventualmente mutua después de las etapas 5 y 8, cuando se verifican los MAC dentro de los mensajes Finalizados; y es adelante
secreto.

Las reducciones que muestran la seguridad de los modos de protocolo en el modelo siguen una técnica de salto de juego y se basan principalmente en la firma estándar resp. Imposibilidad de falsificación del esquema MAC (para autenticación en el protocolo de enlace 1-RTT resp. PSK completo), resistencia a la colisión de función hash, seguridad PRF (y en algunos casos seguridad PRF dual) y una suposición interactiva Diffie-Hellman (una variante de PRF- Suposición Oracle-Diffie-Hellman [JKSS12, BFGJ17] llamada dual-snPRF-ODH).

Observaciones sobre el diseño y la seguridad de TLS 1.3. En la Sección 7, incluimos una discusión sobre varias características de TLS 1.3 en función de los resultados de nuestro análisis de seguridad, incluido cómo una variedad de decisiones de diseño de TLS 1.3 impactan positivamente en el análisis de seguridad (separación de claves e independencia de claves, incluido el hash de sesión en firmas y derivación de clave), algunas sutilezas sobre el papel del cifrado de protocolo de enlace y la confirmación de clave a través de mensajes Finalizados, así como la susceptibilidad de las claves 0-RTT a las repeticiones.

Relación con nuestro trabajo anterior. Este documento es un trabajo sucesor de [DFGS15, DFGS16] y [FG17], así como de [Dow17, Gün18]. En [DFGS15], primero ampliamos el modelo de intercambio de claves de varias etapas de [FG14] según fuera necesario, luego lo aplicamos para analizar dos borradores iniciales de TLS 1.3: borrador-05, que tiene la misma estructura básica firmada Diffie-Hellman pero un programa de claves simplificado en comparación con la versión final, y una propuesta alternativa llamada draft-dh que incorpora ideas del diseño de OPTLS [KW16], en el que los servidores podrían tener una clave compartida DH semiestática. En [DFGS16], actualizamos nuestro análisis al borrador 10 y agregamos un análisis del modo de protocolo de enlace de clave precompartida, entonces revisado. En [FG17], un subconjunto de nosotros analizó la clave precompartida 0-RTT y el modo PSK-(EC)DHE en el borrador 14, así como el modo 0-RTT basado en Diffie-Hellman más tarde en desuso usando semiestático La clave DH se comparte en el borrador 12, que introdujo la noción de etapas reproducibles en el modelo de seguridad de intercambio de claves de múltiples etapas. En una tesis doctoral [Dow17], uno de nosotros actualizó el trabajo de [DFGS16] para abordar los apretones de manos completos, PSK y PSK-(EC)DHE en el borrador 16; en otra tesis doctoral [Gün18], otro de nosotros unificó el modelo MSKE y los resultados antes mencionados sobre los protocolos de enlace completos y PSK del borrador-10 y los protocolos de enlace 0-RTT del borrador-12 y borrador-14.

Este documento actualiza este trabajo previo a la versión final de TLS 1.3 tal como se publicó en RFC 8446 [Res18] (recuerde que hubo 29 borradores que condujeron al estándar final). Aborda, en un modelo de seguridad unificado, los protocolos de enlace completos, PSK y PSK-(EC)DHE, los dos últimos con claves 0-RTT opcionales.
El modelo de seguridad de este documento incluye mejoras que no están presentes en trabajos anteriores, en particular para capturar la autenticación actualizable. El modelo y el análisis para el modo PSK se han actualizado para reflejar las observaciones del ataque "Selfie" de Drucker y Gueron [DG21b] mediante la asociación de funciones previstas con una clave precompartida.

La Sección 7.1 proporciona más detalles sobre las diferencias técnicas entre este documento y nuestro trabajo anterior.

Limitaciones. El protocolo TLS 1.3 permite a los usuarios admitir y negociar diferentes algoritmos criptográficos, incluidos los esquemas de firma utilizados, los grupos Diffie-Hellman y los en autenticados.

esquemas de cifrado. Muchas implementaciones admitirán simultáneamente TLS 1.3, TLS 1.2 e incluso versiones anteriores. No pretendemos capturar la seguridad de este proceso de negociación ni la seguridad cuando se reutiliza una clave criptográfica (por ejemplo, una clave de firma) en diferentes combinaciones de algoritmos o con versiones anteriores de TLS [JSS15]. Para los modos PSK de TLS 1.3, no tratamos cómo las partes negocian qué clave precompartida usar. Nuestro análisis asume que todas las partes usan solo TLS 1.3 con una sola combinación de algoritmos criptográficos y no reutilizan material de claves fuera de ese contexto (más allá de consumir claves de sesión establecidas por el protocolo de enlace TLS 1.3).

En nuestras pruebas de indistinguibilidad de claves para los tres modos de protocolo de enlace TLS 1.3, algunos de nuestros pasos de prueba implican adivinar partes y/o sesiones y, por lo tanto, no son estrictos, similares a la mayoría de las pruebas de protocolos de intercambio de claves autenticados. Recientemente, Diemert y Jager [DJ21] , así como Davis y Günther [DG21a] han establecido nuevas pruebas de seguridad para el protocolo de enlace TLS 1.3 full 1-RTT con reducciones estrictas a la suposición fuerte de Diffie-Hellman, traduciendo técnicas de Cohn-Gordon et al. . [CCG+19].

Nuestro enfoque está completamente en el protocolo de protocolo de enlace TLS 1.3 y, por lo tanto, no aborda la seguridad del cifrado autenticado de la capa de registro. TLS 1.3 también incluye una variedad de funcionalidades adicionales fuera del protocolo de enlace central que tratamos como fuera del alcance. Los ejemplos incluyen tickets de sesión, autenticación posterior al protocolo de enlace [Kra16b], el protocolo de alerta y cambios para Datagram TLS (DTLS) 1.3 [RTM19], así como otras extensiones de TLS 1.3 actualmente en estado Borrador de Internet.

La seguridad en la práctica obviamente depende de muchos más factores, como buenas implementaciones y buena seguridad operativa, que son importantes pero están fuera del alcance de este análisis.

2 preliminares

Comenzamos introduciendo la notación básica que usamos en este documento y recapitulando algunos elementos fundamentales y suposiciones criptográficas empleadas en nuestro análisis de seguridad.

2.1 Notación

Con N denotamos los números naturales. Escribimos un bit como *b* ÿ {0, 1} y una cadena (bit) como *s* ÿ {0, 1} ÿ con | , *s*| indicando su longitud (binaria); {0, 1} *n* es el conjunto de cadenas de bits de longitud *n*. Escribimos $ para asignaciones de valor y para el muestreo uniforme de *x* de un conjunto (finito) X.

Para un algoritmo A escribimos *x* ÿ *A(y),* resp. *x* ÿ$ *A(y),* para el algoritmo de forma determinista, resp. probabilísticamente, dando salida a *x* en la entrada y. Indicamos por AO un algoritmo A que se ejecuta con acceso de Oracle a algún otro algoritmo O.

2.2 Funciones hash resistentes a colisiones

Como suele ser el caso en la práctica, las funciones hash criptográficas utilizadas en TLS 1.3 no tienen clave. Al considerar la resistencia a la colisión de una función hash, exigimos que una reducción de seguridad proporcione medios efectivos para construir un algoritmo concreto que genere una colisión (cf. Rogaway [Rog06]).

Definición 2.1 (Función hash y resistencia a colisiones). *Una* función hash H: {0, 1} ÿ ÿ {0, 1} *asigna mensajes de longitud arbitraria m* ÿ {0, 1} ÿ *a un valor hash H(m)* ÿ {0, 1} *de longitud fija ÿ* ÿ *N .*

Ahora podemos medir la resistencia a la colisión (COLL) *con respecto a un adversario* A *a través de la ventaja*

$$\mathrm{Adv}^{\mathrm{COLL}}_{\mathrm{H,A}} := \Pr\,(m, m0) \ddot{y}\$\ A : m \neq m0 \text{ y } H(m) = H(m0)\ .$$

En la noción asintótica común exigiríamos que no se pueda construir un adversario A eficiente *donde esta ventaja no sea despreciable con respecto al parámetro de seguridad ÿ.*

2.3 HMAC y HKDF

TLS 1.3 emplea HKDF [Kra10, KE10] como su función de derivación clave, con HMAC [BCK96, KBC97] en su núcleo. Recapitulamos brevemente su definición y uso.

HMAC [BCK96, KBC97] se basa en una función hash criptográfica H: {0, 1} ÿ y se codifica con alguna clave *K* ÿ {0, 1} ÿ {0, 1} (el material de clave más grande se procesa a través de H para obtener una clave de ÿ-bit).
Calcular el valor HMAC en algún mensaje *m* se define como HMAC(K, m) := H((K ÿ opad) k H((K ÿipad) k m)), donde opad e ipad son dos rellenos de ÿ-bit valores que consisten en bytes repetidos 0x5c y 0x36, respectivamente.

HKDF sigue el paradigma de *extracción y luego expansión* para la derivación de claves [Kra10, KE10], instanciado con HMAC. Adoptamos la notación estándar para las dos funciones HKDF: HKDF.Extract(XTS, *SKM)* en la entrada, un extractor salt (no secreto y potencialmente fijo) *XTS* y algún material de clave fuente (no necesariamente uniforme) *SKM* genera una clave pseudoaleatoria *PRK* . HKDF.Expand(PRK, *CTXinfo,* L) al ingresar una clave pseudoaleatoria *PRK* (desde el paso Extraer) y alguna información de contexto (potencialmente vacía) *CTXinfo* genera material de clave pseudoaleatoria *KM* de longitud *L* bits.
(Para simplificar, omitimos el tercer parámetro *L* en Expandir cuando *L* = ÿ, que es el caso en TLS 1.3 excepto cuando se derivan claves de tráfico (cf. Tabla 2).) Ambas funciones se instancian con HMAC, donde directamente HKDF.Extract (XTS, *SKM) :*= HMAC(XTS, *SKM)* y HKDF.Expand invoca iterativamente HMAC para generar una salida pseudoaleatoria de la longitud requerida (ver [Kra10]).

2.4 Seguridad PRF Dual y el Supuesto PRF-ODH

La mayoría de los pasos de derivación clave en TLS 1.3 se basan en la seguridad de la función pseudoaleatoria (PRF) regular para las funciones HKDF y HMAC. En nuestro análisis de los protocolos de enlace de PSK, también tratamos HMAC como una función hash sin clave resistente a colisiones sobre el par de entradas, como en la Definición 2.1. Sin embargo, para algunas de sus aplicaciones, necesitamos implementar suposiciones más sólidas que recapitulamos aquí.

La primera suposición se refiere al uso de HMAC como PRF dual (cf. [Bel06]).

Definición 2.2 (Seguridad Dual PRF). *Sea f :* K × L ÿ O *una función pseudoaleatoria con espacio de claves* K *y espacio de etiquetas* L *tal que* K = *L. Definimos la* seguridad PRF dual *de f como la seguridad PRF de f* swap(k, l) := *f(l,* k) *y la función de ventaja correspondiente como*

$$\mathrm{Adv}^{\text{dual-PRF-sec}} := \mathrm{Adv}^{\text{PRF-sec}}_{f,A\, f\,\text{swap},A} .$$

La segunda suposición, la llamada suposición de función pseudoaleatoria oracle-Diffie-Hellman (PRF-ODH), ha sido introducida por Jager et al. [JKSS12] en su análisis del intercambio de claves TLS 1.2. Es una variante del supuesto oráculo-Diffie-Hellman introducido por Abdalla et al. [ABR01] en el contexto del esquema de cifrado DHIES. Básicamente, la suposición PRF-ODH establece que el valor PRF(g *uv, x?)* para una clave tipo Diffie-Hellman *g* es indistinguible de una cadena aleatoria, incluso cuando se dan *g u* y *g v* y cuando se pueden [illegible] valores PRF(S, *T* , x) y la suposición PRF-ODH viene en varias variantes, que han sido generalizadas y estudiadas por Brendel et al. [BFGJ17].

Para nuestro análisis de TLS 1.3, implementaremos solo la suposición snPRF-ODH que brinda acceso de Oracle limitado a solo un único valor relacionado PRF (S *u* , x), así como su variante dual, dual-snPRF-ODH.
Ambos han sido establecidos por Brendel et al. [BFGJ17] para HMAC en el modelo de oráculo aleatorio bajo la suposición fuerte de Diffie-Hellman.

Definición 2.3 (supuestos de snPRF-ODH y dual-snPRF-ODH). *Sea ÿ ÿ N, G un grupo cíclico de primer orden q con generador g, y* PRF: G × {0, 1} ÿ ÿ {0, 1} *Sea una función pseudoaleatoria.*

Definimos el juego de seguridad snPRF-ODH *de la siguiente manera.*

1. *El retador muestrea b* ÿ$ {0, 1}, *u, v* ÿ$ *Zq, y proporciona G, g, g* ^{tu}, *y G* ^{v} *a A, que responde con una etiqueta de desafío* $x^{?}$ ·

2. *El retador calcula y* $_{0}^{?}$ = PRF(g *uv, x?) y muestras y* $_{1}^{?}$ ÿ$ {0, 1} ^{ÿ} *uniformemente al azar, proporcionando* $y_{B}^{?}$ *a a*

3. A *puede consultar un par (S,* x), *en el que el retador primero asegura que S /*ÿ G *o (S,* x) = *(g v , ¿x?) y, si es así, devuelve y* ÿ PRF(S *u ,* x).

4. *Eventualmente,* A *se detiene y genera una suposición* b^{0} ÿ {0, 1}.

Definimos la función de ventaja snPRF-ODH *como*

$$\mathrm{Adv}^{\mathrm{snPRF\text{-}ODH}}_{PRF,G,A} := 2 \cdot \Pr[b^{0} = b] ÿ 1.$$

Definimos la variante dual del supuesto, dual-snPRF-ODH, *como el* supuesto snPRF-ODH *para una función* PRF: {0, 1} ÿ × G ÿ {0, 1} *con entradas intercambiadas, es decir, toma el elemento de grupo en la segunda entrada y tomando la etiqueta como la primera entrada.*

3 Protocolo de protocolo de enlace TLS 1.3

En esta sección, describimos los modos del protocolo de protocolo de enlace TLS 1.3, específicamente el protocolo de enlace de tiempo de ida y vuelta completo (1-RTT), que se muestra en el lado izquierdo de la Figura 1, y el tiempo de ida y vuelta cero combinado (0-RTT) y el protocolo de enlace de clave precompartida, representado en el lado derecho de la Figura 1. Nuestro enfoque en la Figura 1 y en todo el documento se centra en los aspectos criptográficos del protocolo de enlace TLS 1.3. Como tal, omitimos muchos otros componentes del protocolo, incluida la mayoría de las extensiones de saludo, aspectos de la negociación de versiones y algoritmos, mensajes posteriores al protocolo de enlace, el protocolo de capa de registro y el protocolo de alerta.

En TLS 1.3, los protocolos de enlace 1-RTT y PSK se dividen en dos fases distintas: una *fase de intercambio de claves,* en la que el cliente y el servidor intercambian mensajes de saludo para indicar la compatibilidad con diferentes opciones criptográficas y utilizan los parámetros seleccionados para generar material de intercambio de claves; y una *fase de autenticación,* en la que el cliente y el servidor intercambian mensajes CertificateVerify y Finished, autenticándose mutuamente mediante valores asimétricos (o simétricos) a largo plazo. La Figura 2 ilustra el programa de claves de TLS 1.3, la Tabla 1 enumera las abreviaturas de los mensajes y las claves que se utilizan en todo el documento y la Tabla 2 detalla algunos de los cálculos y las entradas.

3.1 Fase de intercambio de claves

La *fase de intercambio de claves* consiste en el intercambio de mensajes ClientHello (CH) y ServerHello (SH), durante los cuales se negocian los parámetros y se realiza el intercambio de claves principales, utilizando el intercambio de claves Diffie-Hellman o basándose en una clave simétrica precompartida.

ClienteHola. El cliente comienza enviando el mensaje ClientHello, que contiene *rc* (un valor nonce de 256 bits muestreado aleatoriamente), así como información de negociación de versión y algoritmo.

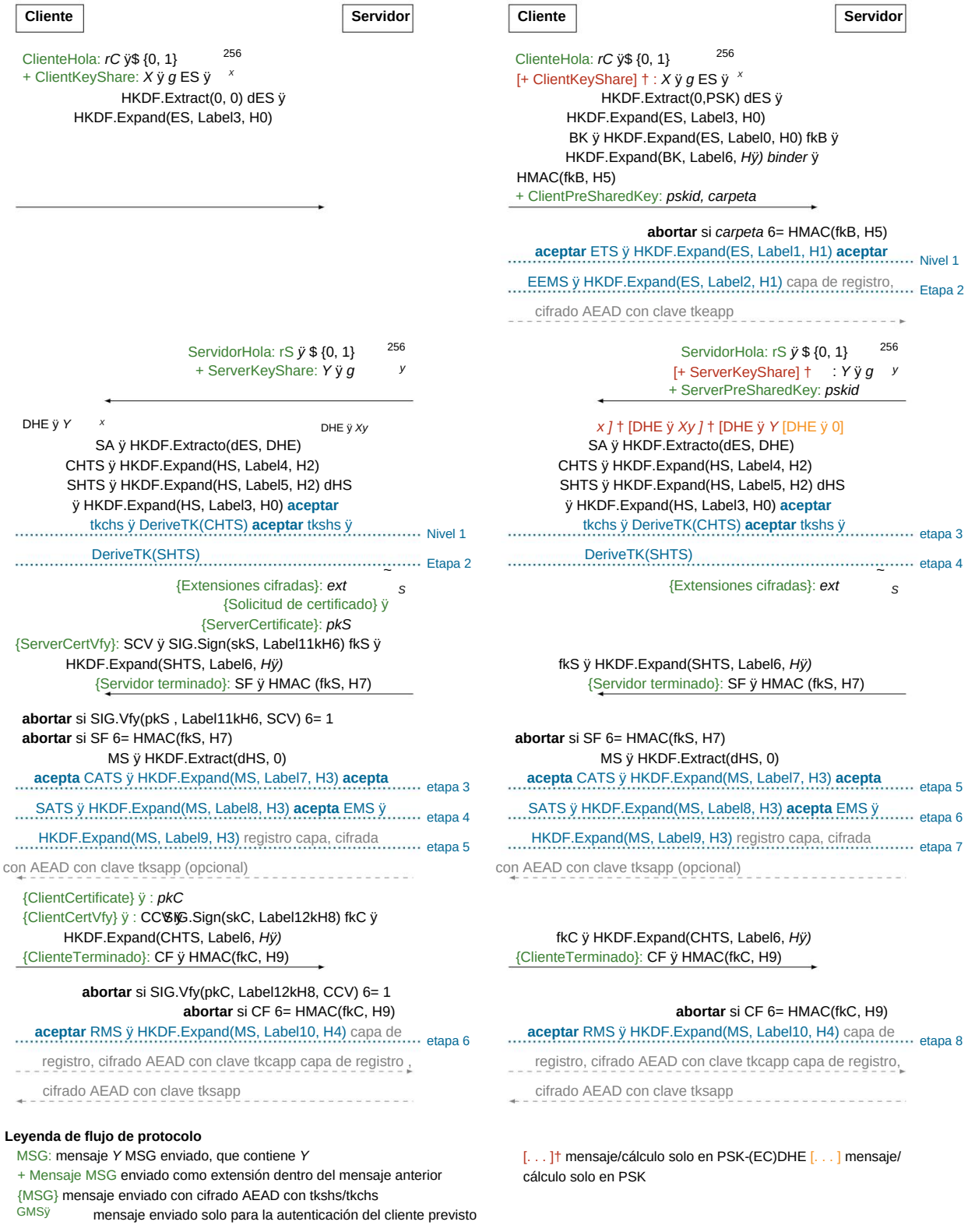


Figura 1: El protocolo de enlace TLS 1.3 completo 1-RTT (izquierda) y el protocolo de enlace PSK/PSK-(EC)DHE con 0-RTT opcional (derecha). Las abreviaturas se explican en la Tabla 1; los valores de las entradas de contexto y etiqueta (Hÿ, resp. Labelÿ) y los detalles sobre el cálculo de las claves de tráfico (tkÿ) se pueden encontrar en la Tabla 2.

Adjunta a ClientHello se encuentra la extensión KeyShare (CKS) que contiene recursos compartidos de clave pública para el intercambio de claves. Existen otras extensiones para la negociación adicional de algoritmos y parámetros.1 Si se ha establecido un secreto precompartido entre el cliente y el servidor (ya sea en un protocolo de enlace previo o mediante

algún mecanismo fuera de banda), el cliente puede incluir la PreSharedKey (CPSK), que indica los modos de protocolo de enlace (como PSK o PSK-(EC)DHE) que admite el cliente, y una lista de identidades simétricas previamente compartidas que se asignan a estos PSK.2 Si se incluye CPSK, el cliente calcula un valor de clave de enlace BK para cada clave previamente compartida PSK en la lista, de ahí una clave *fkB* y un enlace de valor ÿ HMAC(fkB, *H* (CH†)) que enlaza el mensaje CH actual (truncado para excluir el valor del *enlace* en sí) a cada PSK, también incluido en el mensaje CPSK y verificado por el servidor. Esto se captura en el lado derecho de la Figura 1.

Finalmente, si el cliente desea usar el secreto precompartido para enviar datos de tiempo de ida y vuelta cero (0-RTT), el cliente puede indicarlo enviando una extensión EarlyDataIndication. Esto indicará al servidor que el cliente utilizará el primer secreto precompartido indicado en la lista CPSK para derivar un secreto de tráfico anticipado (ETS) y un secreto maestro de exportador anticipado (EEMS), y comenzará a enviar datos cifrados al servidor sin solicitar primero el cliente para recibir la respuesta de ServerHello.

ServidorHola. El siguiente mensaje en la fase de intercambio de claves es el mensaje ServerHello (SH).
Como en CH, el servidor muestreará aleatoriamente un valor nonce de 256 bits *rs.* El servidor elige entre los diversos algoritmos y parámetros ofrecidos por el cliente y responde con sus selecciones. Si se envió CPSK, el servidor decide si acepta un protocolo de enlace basado en PSK. Si es así, entonces el identificador de clave precompartida *pskid* asociado con el PSK seleccionado se envía en la extensión PreSharedKey (SPSK).
Si el servidor ha elegido el modo PSK-(EC)DHE (o ha rechazado el uso de PSK), el servidor generará su propia clave compartida (EC)DHE $Y \ddot{y} g^{y}$, enviando *Y* en la extensión KeyShare (SKS) adjunta a SH.

En este punto, el servidor tiene suficiente información para calcular los valores del secreto de tráfico del protocolo de enlace del cliente (CHTS) y del secreto de tráfico del protocolo de enlace del servidor (SHTS), y los utiliza para derivar las claves de tráfico del protocolo de enlace del cliente y del servidor (tkchs y tkshs, respectivamente). La primera parte de la Figura 2 muestra el programa clave para derivar estas claves. Tenga en cuenta que consideramos que tkchs y tkshs se derivan en el mismo momento (es decir, cuando el protocolo de enlace secreto HS está disponible), aunque, en principio, tkchs solo se necesita un poco más tarde.

El servidor ahora comienza a cifrar todos los mensajes de protocolo de enlace bajo tkshs, y cualquier extensión que no sea necesaria para establecer la clave de tráfico de protocolo de enlace del servidor se envía (y cifra) en los mensajes de extensiones cifradas (EE).

3.2 Fase de autenticación

Ahora comienza la *fase de autenticación .* Todos los mensajes de protocolo de enlace en esta fase están encriptados bajo tkshs o tkchs. En el protocolo de enlace 1-RTT completo, la autenticación se basa en certificados de clave pública; consulte el lado izquierdo de la Figura 1. En los protocolos de enlace de clave precompartida (tanto PSK como PSK-(EC)DHE), el servidor y el cliente se autenticarán mutuamente basándose en un código de autenticación de mensaje aplicado a la transcripción; ver el lado derecho de la Figura 1.

1Tenga en cuenta que nuestro análisis en las Secciones 5 y 6 no considera la negociación de valores criptográficos (como claves precompartidas o grupos (EC)DHE) o modos de protocolo de enlace, sino que nuestro análisis considera cada combinación de modo de protocolo de enlace y suite de cifrado de forma aislada. Esto se puede ver en la Figura 1, por ejemplo, el mensaje CKS contiene solo una única clave compartida (EC)DHE.

2En cuanto a la extensión KeyShare, aquí no consideramos la negociación y solo capturamos la única entrada de PSK que el cliente y el servidor están de acuerdo.

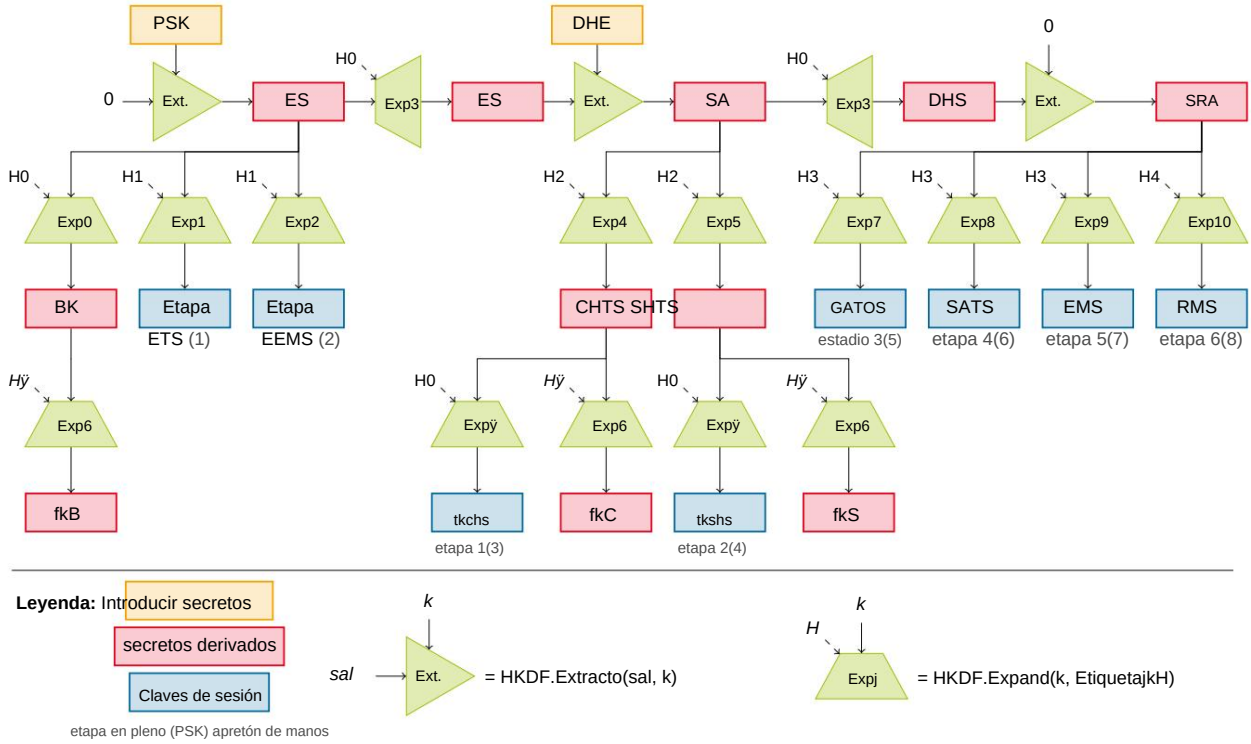


Figura 2: El programa clave de TLS 1.3. Los valores de las entradas de contexto y etiqueta (Hÿ, resp. Labelÿ) y los detalles sobre el cálculo de las claves de tráfico (Expÿ) se pueden encontrar en la Tabla 2.

Autenticación en protocolo de enlace 1-RTT completo. El servidor puede solicitar la autenticación de cliente basada en clave pública mediante el envío de un mensaje CertificateRequest (CR). El servidor se autenticará ante el cliente utilizando las claves públicas a largo plazo del servidor. Aquí, el servidor comienza enviando su certificado (con su clave pública) en el mensaje ServerCertificate (SCRT). Luego, el servidor calcula el valor de autenticación de ServerCertificateVerify firmando el hash de sesión (que es un hash de actualización continua de todos los mensajes hasta este punto en el protocolo), luego lo envía al cliente como el mensaje ServerCertificateVerify.

Confirmación de clave de servidor y derivación de clave. En todos los modos de protocolo de enlace, el mensaje final que el servidor envía al cliente es el mensaje ServerFinished (SF). El servidor primero obtiene una clave finalizada del servidor fkS de SHTS y luego calcula una etiqueta MAC SF sobre el hash de la sesión. Este valor también se cifra bajo tkshs, enviando el texto cifrado de salida al cliente. En este punto, el servidor puede calcular el secreto de tráfico de la aplicación del cliente (CATS), el secreto de tráfico de la aplicación del servidor (SATS) y el secreto maestro del exportador (EMS). La figura 2 muestra el programa de claves para derivar estas claves y todas las demás claves en la fase de autenticación. Ahora que el servidor ha calculado la clave de tráfico de la aplicación del servidor tksapp, puede comenzar a enviar datos de la aplicación encriptados al cliente sin esperar el vuelo final de mensajes del cliente, logrando así un apretón de manos de 0.5-RT

Verificación del cliente, autenticación, confirmación de clave y derivación de clave. El cliente, al recibir estos mensajes, comprueba que la firma SCV (si está en modo 1-RTT completo) y el MAC SF se verifican correctamente. Si el servidor ha solicitado la autenticación del cliente, el cliente comenzará enviando su certificado digital (con su clave pública) en el mensaje ClientCertificate (CCRT),

Mensaje		Clave o valor derivado	
CH	ClienteHola	BK	Clave de carpeta
CKS	ClientKeyShare	Secreto de tráfico de protocolo de enlace de cliente/servidor CHTS/SHTS	
CPSK	ClientPreSharedKey	Secreto de tráfico de aplicaciones de cliente/servidor CATS/SATS	
SH	ServidorHola	DES/DSH	Secreto anticipado/apretón de manos derivado
Clave compartida del servidor SKS		ES/HS/MS	Early/Handshake/Master Secret
Servidor SPSK	PreSharedKey	ETS	Secreto de tráfico temprano
EE.UU.	Extensiones cifradas	Secreto principal del exportador EEMS/EMS (anterior)	
RC	Solicitud de certificado	fkB/fkC /fkS	Carpeta/Cliente/Servidor Clave finalizada
Certificado de servidor SCRT		RMS	Secreto maestro de reanudación
Servidor SCV	CertificateVerify	tkeapp	Clave de tráfico de aplicaciones tempranas
SF	ServidorTerminado	tkchs/tkshs	Clave de tráfico de reconocimiento de cliente/servidor
Certificado de cliente CCRT		Clave de tráfico de aplicaciones de cliente/servidor tkcapp/tksapp	
Certificado de cliente CCV	Verificar		
FC	ClienteTerminado		

Tabla 1: Abreviaturas para mensajes TLS 1.3 (en orden de protocolo) y claves/valores derivados (alfabéticos).

después de lo cual el cliente calculará su propio valor de verificación de certificado CCV firmando el hash de sesión, luego envíelo al servidor como el mensaje CCV. El cliente finalmente deriva la clave finalizada del cliente fkC de CHTS y usa fkC para calcular una etiqueta MAC CF sobre el hash de la sesión.

Verificación del servidor. El servidor verificará el MAC final (SF) y la firma opcional (SCV) mensajes del cliente.

Finalización del apretón de manos. En este punto, ambas partes pueden calcular el secreto maestro de reanudación (RMS) que se puede utilizar como clave precompartida para la reanudación de la sesión en el futuro. Ambas fiestas ahora puede derivar la clave de tráfico de la aplicación del cliente (tkcapp) y usar la capa de registro para cifrado comunicación de los datos de la aplicación con las claves resultantes.

3.3 Nuevo boleto de sesión

El mensaje NewSessionTicket es un mensaje posterior al protocolo de enlace en TLS 1.3 que hace referencia a valores del protocolo de apretón de manos. El mensaje NewSessionTicket puede ser enviado por un servidor al cliente (cifrado bajo una clave de tráfico de aplicación de servidor tksapp) para permitir que el cliente calcule valores asociados con los apretones de manos de reanudación, incluido el PSK utilizado en la reanudación, así como un identificador para indicar al servidor qué clave precompartida se está utilizando. El nuevo boleto de sesión El mensaje contiene dos campos que son interesantes para este propósito:

- ticket_nonce, que el cliente utiliza como valor salt para derivar la clave precompartida para ser utilizado en el protocolo de enlace futuro para la reanudación: PSK ÿ HKDF.Expand (RMS, "reanudación", ticket_nonce).
- ticket, que es una etiqueta opaca utilizada para referirse públicamente a la clave precompartida asociada en futuros mensajes PreSharedKey. En nuestra notación utilizada en la Figura 1, el identificador de clave precompartida *pskid* = boleto.

En nuestro análisis, no capturamos este mensaje NewSessionTicket, ni la derivación de PSK de RMS, y en su lugar suponga que el mapeo entre PSK y *pskid* se establece en algún forma fuera de banda. En particular, no capturamos la transmisión de NewSessionTicket bajo un clave de tráfico de la aplicación del servidor tksapp, ya que afectaría la forma en que consideramos el uso de SATS. En nuestro análisis, actualmente consideramos SATS una "clave externa" utilizada en una clave simétrica arbitraria

Secreto	Entrada de contexto	Entrada de etiqueta
BK	H0 = H(“”)	Label0 = "carpeta ext" / "carpeta res"
fkB	*Hÿ* =	Etiqueta6 = "terminado"
ETS	H1 = H(ClienteHola)	Label1 = "tráfico ce"
EEMS	H1 = H(ClienteHola)	Label2 = "e exp maestro"
dES	H0 = H(“”)	Etiqueta3 = "derivado"
CHTS	H2 = H(ClienteHolaServidorHola)	Label4 = "tráfico c hs"
SHTS	H2 = H(ClienteHolaServidorHola)	Label5 = "tráfico s hs"
fkS	*Hÿ* =	Etiqueta6 = "terminado"
dhs	H0 = H(“”)	Etiqueta3 = "derivado"
GATOS	H3 = H(ClientHellok . . . kServerFinished)	Label7 = “tráfico límite”
SAT	H3 = H(ClientHellok . . . kServerFinished)	Label8 = “tráfico s ap”
ccsme	H3 = H(ClientHellok . . . kServerFinished)	Label9 = “exp master”
fkC	*Hÿ* = Etiqueta6 = “terminado”	
RMS	H4 = H(ClientHellok . . . kClientFinished)	Label10 = “res master”

Aut. Entrada de contexto de valor		Cadena de contexto (solo para firmas)
aglutinante	H5 = H(ClienteHola†)	
SCV	H6 = H(ClienteHellok . . . kServerCert)	Etiqueta11 = “TLS 1.3, servidor CertificateVerify”
SF	H7 = H(ClienteHellok . . . kServerCertVfy)	
CCV	H8 = H(ClienteHellok . . . kCertificadoCliente)	Etiqueta12 = “TLS 1.3, cliente CertificateVerify”
FC	H9 = H(ClienteHellok . . . kClienteCertVfyÿ)	

Cálculo de clave de tráfico

tkeapp/tkchs/tkshs/tkcapp/tksapp = *(clave, iv)* = DeriveTK(ETS/CHTS/SHTS/CATS/SATS)
donde DeriveTK(Secret) = (HKDF.Expand(Secret, “key”, H(“”), Lk), HKDF.Expand(Secret, “iv”, H(“”), *Liv))*
con *Lk/Liv* indicando la longitud de clave/iv del esquema AEAD negociado

Tabla 2: Entradas de secreto, contexto y etiqueta al HKDF.Expand resp. funciones de autenticación, así como el tráfico Cálculo clave en el protocolo de enlace TLS 1.3 (Figura 1) y programación clave (Figura 2). La entrada de etiqueta real para HKDF.Expand es la concatenación de la longitud del hash (en bytes), la cadena "tls13", la etiqueta y el valor de contexto dado. Luego se llama a HKDF.Expand en el secreto correspondiente, esta etiqueta aumentada y la salida deseada longitud. ClientCertVfyÿ solo se incluye en caso de autenticación de cliente. ClientHello† indica una versión truncada de ClientHello que excluye el propio valor del *enlazador* . Las firmas en SCV y CCV se calculan sobre la concatenación de una constante (0x20 repetido 64 veces), la etiqueta como información de contexto, un byte 0 de separación y el valor de contexto.

protocolo. Para capturar la transmisión de NewSessionTicket, necesitaríamos capturar el uso de SATS para derivar tksapp y luego establecer PSK. Elegimos simplificar el análisis omitiendo este mecanismo, y dejar esto como trabajo futuro.

4 Modelo de seguridad de intercambio de claves de varias etapas

Para capturar la seguridad de todas las variantes del protocolo de enlace TLS 1.3 dentro de un solo modelo de seguridad integral, adoptamos el modelo de intercambio de claves de varias etapas en la versión de Gün ther [Gün18] que combina el modelo original de Fischlin y Günther [FG14] con seguimiento extensiones [DFGS15, DFGS16, FG17]. Nos referimos a Günther [Gün18] para una discusión extensa de el modelo, pero recapitular sus conceptos básicos y definiciones, así como las adaptaciones para nuestro análisis en el siguiente.

El modelo sigue el paradigma clásico de los modelos de intercambio de llaves de Bellare y Rogaway [BR94] en el formalismo de Brzuska et al. [BFWW11, Brz13]. Este paradigma captura a un adversario fuerte que controla la red y es capaz tanto de espiar pasivamente como de modificar activamente la comunicación

comunicación a través de múltiples sesiones del protocolo de intercambio de claves (generandolas a través de un oráculo NewSession y dirigiendo la comunicación a través de un oráculo de envío). Además, el adversario puede exponer los secretos a largo plazo de las partes honestas que interactúan (a través de un oráculo Corrupto), así como las claves de sesión en algunas ejecuciones de protocolo (a través de un oráculo Revelar). Entonces, la seguridad básica exige que dicho adversario no pueda distinguir la clave de sesión real establecida en alguna sesión no comprometida ("nueva") de una aleatoria (a través de un oráculo de prueba).

El modelo de intercambio de claves de múltiples etapas ahora amplía la configuración básica de intercambio de claves mediante la captura de protocolos que derivan una serie de claves de sesión en múltiples *etapas.* Cada etapa está asociada con propiedades de seguridad particulares, determinando la admisibilidad de ciertas acciones adversarias para esa etapa y bajo qué condiciones la clave de esta etapa se considera fresca. Estas propiedades de seguridad modelan los siguientes aspectos:

Autenticación. Nuestro modelo distingue entre etapas *no autenticadas ,* etapas autenticadas *unilateralmente* donde solo el respondedor (el servidor en TLS 1.3) se autentica y etapas *mutuamente autenticadas* donde ambos pares se autentican. Tratamos la autenticación de cada etapa *individualmente* y consideramos ejecuciones *simultáneas* de diferentes modos de autenticación del mismo protocolo. Las identidades de los socios de comunicación pueden conocerse solo durante la ejecución del protocolo (por ejemplo, a través de certificados intercambiados), que implementamos a través *de pares especificados* posteriormente siguiendo a Canetti y Krawczyk [CK02]. Nuestro modelo exige una fuerte noción de seguridad para las sesiones con pares no autenticados, es decir, que tales sesiones logren confidencialidad clave al recibir sus mensajes de una sesión honesta (identificada a través de un *identificador contributivo),* independientemente de si esa sesión honesta de pares se asocia más tarde.

Además, el nivel de autenticación de alguna etapa puede elevarse con la aceptación de una etapa posterior, por ejemplo, de no autenticado a unilateral o incluso mutuamente autenticado. Esto puede suceder, por ejemplo, si una parte firma posteriormente los datos transmitidos previamente, como en el caso de TLS 1.3.
Capturamos esto al permitir que un protocolo especifique el nivel de autenticación para cada etapa de aceptación, así como en qué etapa(s) posterior(es) aumenta ese nivel.

Tenga en cuenta que capturamos la autenticación *implícitamente* a través de la confidencialidad de la clave (es decir, las claves solo las conoce la sesión de pares prevista) pero no probamos la autenticación explícita (es decir, la existencia de una sesión asociada). El diseño SIGMA [Kra03], en el que se basa el protocolo de enlace TLS 1.3 principal, garantiza la autenticación explícita. de Saint Guilhem et al. [dFW19] brinda un argumento genérico de que la autenticación explícita se deriva del secreto de clave implícita (que se muestra para TLS 1.3 en este artículo) y la confirmación de clave [FGSW16].

Adelante secreto. Capturamos la noción habitual de secreto directo, que garantiza que las claves de sesión aceptadas permanezcan seguras después de un compromiso secreto a largo plazo. Sin embargo, en un protocolo de intercambio de claves de varias etapas, la confidencialidad hacia adelante puede alcanzarse solo a partir de una cierta etapa (por ejemplo, debido a la mezcla de material de clave secreta hacia adelante). Por lo tanto, el modelo trata *el secreto hacia adelante de la etapa j,* lo que indica que las claves de la etapa *j* en adelante son secretas hacia adelante.

Uso de clave. Algunas claves de etapa pueden usarse internamente en el protocolo de intercambio de claves, por ejemplo, en el caso de TLS 1.3, la clave de reconocimiento se usa para cifrar parte de la comunicación de intercambio de claves. Distinguimos el uso de claves como *interno* cuando se usa dentro del intercambio de claves y *externo* cuando se usa exclusivamente fuera del intercambio de claves (por ejemplo, para cifrar datos de aplicaciones). En el primer caso, nuestro modelo garantiza que las claves reales o aleatorias probadas se utilicen en consecuencia en los pasos posteriores de intercambio de claves y detiene la ejecución del protocolo para permitir la prueba de esas claves. Observamos que la declaración de si una clave es interna o externa es un parámetro del modelo y se convierte en parte de la descripción del protocolo y sus garantías de seguridad.

Claves públicas o precompartidas. Nuestro modelo de múltiples etapas viene en dos sabores que capturan tanto el caso de clave pública regular (abreviado como pMSKE) de claves a largo plazo que son pares de claves públicas/secretas (como en el protocolo de enlace completo TLS 1.3) así como el pre- caso de secreto compartido (abreviado sMSKE) caso donde las claves simétricas previamente compartidas actúan como secretos a largo plazo (como en el protocolo de enlace de reanudación de TLS 1.3).

Rejugabilidad. Para el establecimiento de claves 0-RTT, los protocolos de intercambio de claves (incluido TLS 1.3) normalmente renuncian a fuertes garantías de protección de reproducción, en el sentido de que los mensajes del cliente (iniciador) pueden reproducirse en varias sesiones del servidor (respondedor). Capturamos esto en nuestro modelo al distinguir entre *etapas* reproducibles (0-RTT) y *no* reproducibles regulares , teniendo en cuenta las posibles repeticiones para las primeras y aún requiriendo secreto de clave. Determinar el tipo de reproducción de una etapa es nuevamente un parámetro para el modelo y debe especificarse como parte de la descripción del protocolo resp. el reclamo de seguridad.

Observamos que las variantes anteriores de los modelos de intercambio de claves de múltiples etapas, incluido [Gün18] , diferenciaron aún más si el compromiso de la clave de alguna etapa afecta la seguridad de las claves de otras etapas bajo la noción de *(in) dependencia de clave.* Aquí, siempre exigimos que tal compromiso nunca afecte las claves de otras etapas como el objetivo deseable, es decir, postulamos la independencia de las claves y reducimos la complejidad del modelo al incorporar esta propiedad directamente en el modelo. Como veremos, TLS 1.3 siempre logra esta propiedad debido a la separación clara de claves en la programación de claves, y ya lo hizo en versiones preliminares anteriores [DFGS15, DFGS16, FG17].

Paradigma de compromiso secreto. Seguimos el paradigma del modelo Bellare-Rogaway [BR94], centrándonos en la fuga de entradas secretas a largo plazo y salidas clave de sesión del intercambio de claves, pero no en valores internos dentro de la ejecución de una sesión. Esto contrasta en cierta medida con el modelo de Canetti y Krawczyk [CK01] resp. La Macchia et al. [LLM07] que incluyen una "revelación del estado de la sesión" resp. Consulta de “revelación de secreto efímero” que permite acceder a variables internas de la ejecución de la sesión.

En el contexto de TLS 1.3, esto significa que consideramos la fuga de:

- *Claves a largo plazo* (como las claves de firma del servidor o cliente, pero también sus claves precompartidas), ya que los valores a largo plazo tienen el potencial de verse comprometidos, y esto es necesario para modelar el secreto hacia adelante; está capturado en nuestro modelo por la consulta corrupta.
- *Claves de sesión* (como las diversas claves de cifrado de tráfico o la reanudación derivada o los secretos del exportador), ya que son salidas del intercambio de claves y se utilizan más allá de este protocolo para el cifrado, la reanudación posterior o la exportación de material de claves; esto está modelado por Revelar consulta.

No permitimos la fuga de:

- *Secretos efímeros / aleatoriedad* (como la aleatoriedad en un algoritmo de firma o exponentes efímeros de Diffie-Hellman); esto no está permitido ya que TLS 1.3 no está diseñado para ser seguro si estos valores se ven comprometidos.
- *Valores internos/ estado de la sesión* (p. ej., claves maestras o claves MAC calculadas internamente); esto no está permitido ya que TLS 1.3 no está diseñado para ser seguro si estos valores se ven comprometidos.

Comparación con modelos anteriores de intercambio de llaves de varias etapas. En comparación con el modelo MSKE original de Fischlin y Günther [FG14], los cambios más notables en nuestro modelo son la adición que modela la autenticación actualizable y acomoda claves simétricas públicas y previamente compartidas para la autenticación. Tampoco rastreamos si las claves son independientes o no, ya que todas las claves establecidas en TLS 1.3 satisfacen la independencia de claves (a diferencia del análisis de QUIC en [FG14]).
[FG17] introdujo el uso de claves (internas versus externas) y la capacidad de reproducción en MSKE.

4.1 Sintaxis

En nuestro modelo, separamos explícitamente algunas propiedades *específicas del protocolo* (como, por ejemplo, varios tipos de autenticación) de las propiedades *específicas de la sesión* (como, por ejemplo, el estado de una sesión en ejecución). Representamos las propiedades específicas del protocolo como un vector (M, AUTH, FS, USE, REPLAY) que captura lo siguiente:

- M ÿ N: el número de etapas (es decir, el número de claves derivadas).3
- AUTORIZACIÓN ÿ *{((u1,* m1), . . . ,*(uM,* mM)) | *uj , mj* ÿ {1, . . . , *M,* ÿ}}: un conjunto de vectores de pares, cada vector codifica un esquema compatible para autenticación y actualizaciones de autenticación, para cada etapa. Por ejemplo, la i-ésima entrada *(ui , mi)* en un vector dice que la clave de sesión en la etapa *i* inicialmente tiene el nivel *no autenticado* predeterminado , es decir, no proporciona autenticación para ninguno de los socios de comunicación, luego en la etapa *ui* se *autentica unilateralmente* y, por lo tanto, autentica solo al respondedor (servidor) y se *autentica mutuamente* para autenticar a ambos socios de comunicación en la etapa *mj .* Tenga en cuenta que permitimos, por ejemplo, *ui* = *i* (o incluso *ui* = *mi* = i) de modo que la clave de sesión se autentique inmediatamente de forma unilateral (resp. mutuamente) cuando se deriva.

  Las entradas en cada par deben ser no decrecientes, y *ui* = ÿ o *mi* = ÿ denota que unilateral, resp. mutua, la autenticación nunca se alcanza para la etapa i.
- FS ÿ {1, . . . , *M,* ÿ}: la etapa *j* = FS a partir de la cual las claves on son secretas hacia adelante (o ÿ en caso de que no haya secreto hacia adelante).4
- USE ÿ {interno, externo} M: el indicador de uso para cada etapa, donde USEi indica el uso de la tecla stage-i. Aquí, una clave interna se usa dentro del protocolo de intercambio de claves (pero posiblemente también externamente), mientras que una clave externa no debe usarse dentro del protocolo, lo que hace que este último sea potencialmente susceptible de composición genérica (cf. Sección 7.3). Como notación abreviada, por ejemplo, escribimos USE = (interno: {1, 4}, externo: {2, *3,* 5}) para indicar que el uso de teclas en las etapas 1 y 4 es interno y externo para las otras etapas. .
- REPLAY ÿ {replayable, nonreplayable} M: el indicador de rejugabilidad para cada etapa, donde REPLAYi indica si la i-ésima etapa se puede reproducir en el sentido de que un adversario puede forzar fácilmente una comunicación idéntica y, por lo tanto, identificadores de sesión y claves idénticos en esta etapa (por ejemplo, reenviando los mismos datos en etapas 0-RTT). Tenga en cuenta que el adversario, sin embargo, aún no debería ser capaz de distinguir una clave reproducida de una aleatoria. Destacamos que, desde el punto de vista de la seguridad, el uso de etapas reproducibles idealmente debería ser limitado, aunque tales etapas suelen tener un beneficio de eficiencia. Usamos la misma abreviatura

3Fijamos una etapa máxima M solo para facilitar la notación. Tenga en cuenta que M puede ser arbitrariamente grande para cubrir protocolos donde el número de etapas no está limitado a priori. También tenga en cuenta que las etapas y las derivaciones de clave de sesión pueden ser "back to back", sin más interacciones de protocolo entre las partes.

4Un modelo de intercambio de claves de varias etapas más general podría tener un vector de seguimiento específico de qué subconjunto de etapa las claves tienen secreto hacia adelante. No necesitamos tal generalidad ya que el secreto directo es monótono en TLS 1.3.

notación como para USO; por ejemplo, REPETIR = (no reproducible: {1, *2,* 3}) indica que las tres etapas no se pueden reproducir.

Denotamos por U el conjunto de *identidades* (o *usuarios)* utilizados para modelar los participantes en el sistema, cada uno identificado por alguna *U* ÿ U. *Las sesiones* de un protocolo se identifican de forma única (en el nivel administrativo del modelo) utilizando una *etiqueta* etiqueta ÿ ETIQUETAS = U × U × N, donde etiqueta = *(U, V,* n) indica la n-ésima sesión local de la identidad *U* (el *propietario de la sesión)* con *V* como el *socio* de comunicación previsto .

En la variante de clave pública del modelo (pMSKE), cada identidad *U* está asociada con una clave pública certificada a *largo plazo* pkU y una clave secreta skU . En la configuración de secreto previamente compartido (sMSKE), una sesión contiene un identificador psid ÿ {0, 1} ÿ para el secreto previamente compartido pss ÿ P (de algún espacio de secreto previamente compartido P) utilizado. El retador mantiene asignaciones *U* ÿ {0, 1} ÿ P al secreto correspondiente compartido por las partes *U* y *V* donde (*U* usa ese secreto (solo) en el rol de iniciador y *V* (solo) en el de respondedor[5]) y para cualquier usuario U, un identificador de secreto precompartido psid identifica de forma única la identidad del par *V* con la que *U* comparte.

Para cada sesión, se mantiene una tupla con la siguiente información como entrada en la *lista de sesiones* ListS, donde los valores entre corchetes [ ] indican el valor inicial predeterminado. Algunas variables tienen valores para cada etapa *i* ÿ {1, . . . , M}.

- etiqueta ÿ ETIQUETAS: la etiqueta de sesión única (administrativa)
- id ÿ U: la identidad del propietario de la sesión
- pid ÿ U ÿ{ÿ}: la identidad del socio de comunicación previsto, donde el símbolo comodín distintivo 'ÿ' significa "identidad actualmente desconocida", pero luego el protocolo puede establecer una identidad específica en U una vez
- role ÿ {initiator,responder}: el rol del propietario de la sesión en esta sesión
- auth ÿ AUTH: el vector de tipo de autenticación previsto del conjunto de propiedades de autenticación admitidas AUTH, donde authi indica el par de niveles de autenticación para la etapa i, y authi,j su j-ésima entrada
- pssid ÿ {0, 1} ÿ ÿ {ÿ}: en la variante de secreto precompartido (sMSKE), el identificador del secreto precompartido (es decir, pssid,pid si rol = iniciador, de lo contrario psspid,id) que se utilizará en la sesión; se puede inicializar con ÿ si se desconoce pid = ÿ y luego se debe establecer (una vez) cuando se establece pid
- stexec ÿ (EN EJECUCIÓN ÿ ACEPTADO ÿ RECHAZADO): el estado de ejecución [en ejecución], donde CORRIENDO = {corriendoi | *i* ÿ N ÿ {0}}, ACEPTADO = {aceptadodi | *i* ÿ N}, RECHAZADO = {rechazadodi | *yo* ÿ N}; se establece en accepti en el momento en que una sesión acepta la i-ésima clave, en disabledi cuando la sesión rechaza esa clave (una sesión puede continuar después de rechazar en una etapa6), y en runningi cuando una sesión continúa después de aceptar la i-ésima clave
- etapa ÿ {0, . . . , M}: la etapa actual [0], donde la etapa se incrementa a *i* cuando stexec alcanza aceptado resp. rechazado
- sid ÿ {0, 1} ÿ ÿ {ÿ}: M etapa [ÿ] indica el identificador de sesión en la etapa i, se configura una (y única) al aceptar ÿ

5Requerir un rol fijo en el que cualquiera de los pares pueda usar una clave precompartida evita el ataque Selfie [DG21b, AASS19].

6Esto modela, por ejemplo, servidores que rechazan datos 0-RTT de un cliente, pero continúan con el protocolo de enlace restante.

varias veces hasta la ÿ(ÿ) M: cid [ÿ] indica el identificador contributivo en la etapa i, puede establecerse • cid ÿ ({0, 1} ÿ

1} ÿ aceptación en esa ÿ(ÿ) M: keyi [ÿ] indica la clave de sesión establecida en la etapa i, configurada una vez • tecla ÿ ({0,

- stkey ÿ {fresh,revealed} M: stkey,i [fresh] indica el estado de la clave de sesión en la etapa *i*
- probado ÿ {verdadero, falso} M: indicador de prueba probadoi [falso], donde verdadero significa que keyi ha sido probado
- corrupto ÿ {0, . . . , *M,* ÿ}: indicador de corrupción [ÿ] que muestra la etapa en la que se encontraba la sesión cuando se emitió un corrupto a su propietario o socio previsto, incluido el valor 0 si la corrupción se había producido antes de que comenzara la sesión, y ÿ si ninguno de los las fiestas estan corrompidas

Por convención, agregar una tupla no completamente especificada (etiqueta, id, pid, rol, auth) resp. (etiqueta, id, pid, rol, auth, psid) en ListS establece todas las demás entradas en su valor predeterminado. Como abreviaturas, para alguna tupla con etiqueta (única) label en ListS , además escribimos label.X para el elemento *X* de esa tupla y label.(X, *Y,* Z) para el vector *(X, Y,* Z) de los elementos X de esa tupla , *Y* y Z. ,

Definimos dos sesiones distintas label y label0 *para* que se asocien en la etapa *i* si ambas sesiones tienen el mismo identificador de sesión en esa etapa, es decir, *label.sidi* = label0 .sidi 6= ÿ, y requerimos para la corrección que dos sesiones que no tengan un la ejecución conjunta manipulada se asocian en todas las etapas a partir de la aceptación.

Nuestro modelo de seguridad trata la corrupción de secretos a largo plazo (claves secretas para pMSKE, secretos previamente compartidos para sMSKE). Si bien los efectos de tales compromisos en las sesiones pueden diferir en cada configuración, en términos generales consideramos que las claves derivadas de alguna sesión se revelarán si, en la configuración de clave pública (pMSKE), la clave secreta del propietario o del par está comprometida, o en el configuración de secreto precompartido (sMSKE), si el secreto precompartido utilizado para esa sesión se ve comprometido. El secreto hacia adelante entra en juego al determinar si las claves obtenidas antes de la corrupción del secreto a largo plazo también se ven afectadas. En una notación más precisa, decimos que una etiqueta de sesión está *dañada* si

- para pMSKE, el propietario de la sesión label.id o el socio de comunicación previsto label.pid está dañado (es decir, {label.id, *label.pid}* ÿ C 6= ÿ), resp.
- para sMSKE, el secreto precompartido usado está dañado (es decir, (label.id, *label.pid,* label.pssid) ÿ C, el conjunto de usuarios dañados) si *label.role* = iniciador, resp. (etiqueta.pid, etiqueta.id, etiqueta.pssid) ÿ C si *etiqueta.rol* = respondedor.

Autenticación actualizable. Captamos que el nivel de autenticación de alguna etapa puede aumentar, posiblemente al doble, con la aceptación de una etapa posterior a través de un vector por etapa en la matriz del nivel de autenticación. Sin embargo, al capturar la seguridad, nuestro modelo debe considerar cuidadosamente la interacción de la autenticación y la corrupción (algo similar a lo que uno podría usar para el secreto directo). Más precisamente, la garantía de autenticación de alguna etapa *i después de su aceptación* solo puede intensificarse (en alguna etapa posterior *j* > i) si las partes involucradas no están corrompidas por el momento en que la etapa *j* acepta. De lo contrario, el adversario puede haber suplantado al partido hasta la etapa *i* no autenticada y ahora se autentica posteriormente como el partido después de la corrupción en la etapa j. Esto significaría efectivamente que el adversario ha tenido el control total de la sesión y, por lo tanto, puede conocer la clave de sesión de la etapa i.

Capturamos la actualización definiendo el *nivel de autenticación rectificado* rect_authi de alguna etapa *i* en una sesión con autenticación de vector de autenticación prevista, que consta de pares (authi, 1, *authi, 2)* que describen

la etapa en la que la i-ésima clave de sesión se autentica unilateral y mutuamente, con el indicador de corrupción dañado y con la etapa de ejecución actual de la siguiente manera:

$$\text{rect\_autorización} := \begin{cases} \text{si estadio ÿ authi,2 y corrupto ÿ authi,2} & \text{unilateral} \\ \text{si estadio ÿ authi,1 y corrupto ÿ authi,1} & \text{ÿÿÿ} \\ \text{no autenticar} & \text{de lo contrario} \end{cases}$$

Esto codifica que el nivel de autenticación de la etapa *i* se actualiza (a unilateral o mutuo) al alcanzar la etapa *authi,1,* resp. *authi,2,* solo si ninguna corrupción afectó esta sesión antes de estas etapas (authi,1, resp. *authi,2).*

4.2 Modelo adversario

Consideramos un adversario A de tiempo polinomial probabilístico (PPT) que controla la comunicación entre todas las partes, lo que permite la interceptación, inyección y eliminación de mensajes. Nuestro modelo adversario refleja aún más los aspectos de seguridad avanzados en el intercambio de claves de varias etapas, como se describe anteriormente. Capturamos convenientemente la admisibilidad de las interacciones adversarias y las condiciones en las que el adversario pierde trivialmente (como revelar y probar la clave de sesión en sesiones asociadas) a través de una marca perdida (inicializada en falso).

El adversario interactúa con el protocolo a través de las siguientes consultas.

- NewSecret(U, *V,* pssid): esta consulta solo está disponible en la variante de secreto precompartido (sMSKE). para Genera un nuevo secreto con identificador psid compartido entre las partes *U* y *V* por *U* en el rol , ser utilizado de iniciador y por *V* en el rol de respondedor. Si *pssU,V* (pssid) ya está configurado, devuelva ÿ para garantizar la unicidad de los identificadores de pssid entre dos partes en estos roles. De lo contrario, muestree pss ÿ$ P uniformemente al azar del espacio secreto precompartido P del protocolo y defina *pssU,V* (pssid) := pss.

- NewSession(U, V,role, auth[, psid]): crea una nueva sesión con una nueva etiqueta (única) para la identidad del propietario id = *U* con rol role, con pid = *V* como socio previsto (potencialmente no especificado, indicado por *V* = ÿ) y apuntando al tipo de autenticación auth.

  En la variante de secreto precompartido (sMSKE), el parámetro adicional pssid identifica el secreto precompartido que se utilizará, a saber , *pssU,V* (pssid) si rol = iniciador, resp. *pssV,U* (pssid) si rol = respondedor. El identificador puede no estar especificado en este punto (indicado por psid = ÿ) y luego puede ser establecido por el protocolo una vez más.

  Agregar (etiqueta, *U,* V, rol, autenticación), resp. (etiqueta, *U,* V, rol, autenticación, psid), a ListS. Si la etiqueta está dañada, establezca label.corrupted ÿ 0. Esto codifica la información de que la sesión está dañada desde el principio. Etiqueta de devolución.

- Enviar(etiqueta, m): Envía un mensaje *m* a la sesión con etiqueta etiqueta.

  Si no hay tupla con etiqueta label en ListS, devuelve ÿ. De lo contrario, ejecute el protocolo en nombre de *U* en el mensaje *m* y devuelva la respuesta y el estado actualizado de ejecución label.stexec.
  Como caso especial, si *label.role* = iniciador y *m* = init, el protocolo se inicia (sin ningún mensaje de entrada).

  Si, durante la ejecución del protocolo, el estado de ejecución cambia a accepti , la ejecución del protocolo se suspende inmediatamente y se devuelve accepti como resultado al adversario. El adversario puede activar más tarde la reanudación de la ejecución del protocolo mediante la emisión de un especial

ÿ mutuo

Enviar (etiqueta, continuar) consulta. Para tal consulta, el protocolo continúa como se especifica, con la parte creando el siguiente mensaje de protocolo y entregándoselo al adversario junto con el estado de ejecución resultante stexec. Notamos que esto es necesario para permitir que el adversario pruebe una clave interna, antes de que pueda usarse inmediatamente en la respuesta y, por lo tanto, ya no pueda probarse para evitar ataques de distinción triviales. Además, permite que el adversario corrompa las claves a largo plazo de manera detallada después de la aceptación de una clave.

Si el estado de ejecución cambia a label.stexec = accepti para alguna *i* y hay una sesión asociada label0 6= label en ListS (es decir, *label.sidi* = label0 .sidi) con label0 .testedi = true, entonces establezca label. testi ÿ verdadero y (solo si USEi = interno) *label.keyi* ÿ label0 .keyi . Esto garantiza que, si la sesión asociada se ha probado antes, las consultas de prueba posteriores para la sesión se respondan en consecuencia y, en caso de que se use internamente, la clave keyi de esta sesión se establece de forma coherente.7 Si el estado de ejecución cambia a etiqueta. stexec = accepti para algunos *i* y la etiqueta de la sesión está dañada, luego establezca label.stkey,i ÿ revelada.

• Revelar(etiqueta, i): Revela la clave de sesión *etiqueta.keyi* de la etapa *i* en la sesión con etiqueta etiqueta.

Si no hay una sesión con etiqueta label en ListS o *label.stage* < i, devuelva ÿ. De lo contrario, configure label.stkey,i para revelar y proporcione al adversario *label.keyi .*

• Corrupto (U) o Corrupto (U, *V,* psid): la primera consulta solo se usa en la variante de clave pública (pMSKE), la segunda consulta solo en la variante de secreto precompartido (sMSKE). Proporcionar al adversario el secreto a largo plazo correspondiente, es decir, skU (pMSKE), resp. *pssU,V* (psid) (sMSKE).
Agregue al conjunto de entidades corruptas C el usuario *U* (por pMSKE), resp. (para sMSKE) el identificador secreto precompartido global *(U, V,* pssid).

Registre el tiempo de corrupción en cada etiqueta de sesión con label.id = *U* o label.pid = *U* (pMSKE), resp. con etiqueta.(rol, id, pid, pssid) ÿ {(iniciador, *U, V,* pssid),(respondedor, *V, U,* pssid)} (sMSKE), configurando label.corrupted ÿ *label.stage* (a menos que la etiqueta .corrupted 6= ÿ ya, en cuyo caso la corrupción se había producido antes, por lo que dejamos el valor sin cambios).

En el caso de secreto no directo, para cada etiqueta de sesión y para todo *i* ÿ {1, . . . , M}, establezca label.stkey,i en revelado. Es decir, todas las claves de sesión (anteriores y futuras) se consideran deshabilitadas.
cerrado.

En el caso de la confidencialidad directa de etapa-j , stkey,i de cada una de esas etiquetas de sesión se establece en cambio como revelado solo si *i* < *j* o si *i* > etapa. Esto significa que las claves de sesión antes de la j-ésima etapa (donde se activa el secreto directo), así como las claves que aún no se han establecido, son potencialmente divulgadas.

• Test(label, i): Prueba la clave de sesión de la etapa *i* en la sesión con label label. En el juego de seguridad, este oráculo recibe un bit de prueba aleatorio uniforme btest como estado que se fija a lo largo del juego.

Si no hay sesión con label label en ListS o si label.stexec 6= accepti o label.testedi = true, devuelve ÿ. Si la etapa *i* es interna (es decir, USEi = interna) y hay una sesión asociada label0 en ListS (es decir, *label.sidi* = label0 .sidi) con label0 .stexec 6= accepti , establezca el indicador 'perdido' en perdido ÿ verdadero . Esto garantiza que la prueba se pueda obtener una vez y, en el caso de las claves internas, si se acaban de aceptar pero aún no se han utilizado, lo que también garantiza que cualquier sesión asociada que ya haya establecido esta clave no la haya utilizado. Si label.rect_authi = unauth, o

7Tenga en cuenta que para las claves internas esto asume implícitamente la siguiente propiedad de la seguridad Match definida más adelante: Siempre que dos sesiones asociadas acepten una clave en alguna etapa, estas claves serán iguales.

si label.rect_authi = unilateral and *label.role* = respondedor, pero no hay sesión label0 (para label 6= label0) en ListS with *label.cidi* = label0 .cidi ing un socio colaborador, honesto es un requisito previo para probar etapas no Luego establezca perdido ÿ verdadero. Esto asegura que hay autenticadas, respectivamente . el respondedor sesiona en una etapa autenticada unilateralmente.8 La verificación se basa en el nivel de autenticación no corrupto rect_authi para tener en cuenta las corrupciones entre las actualizaciones de autenticación.

De lo contrario, establezca label.testedi en verdadero. Si el bit de prueba btest es 0, muestree una clave *K* ÿ$ D al azar de la distribución de claves de sesión D. Si btest = 1, sea *K* ÿ *label.keyi* la clave de sesión real. Si USEi = interno (es decir, la clave i-ésima probada se indica como utilizada internamente), establezca *label.keyi* ÿ K; en otras palabras, cuando btest = 0, reemplazamos una clave de sesión utilizada *internamente* por la clave de prueba aleatoria e independiente *K* que también se usa para un uso futuro consistente *dentro* del protocolo de intercambio de claves. Por el contrario, las claves de sesión *utilizadas externamente* no se reemplazan por claves aleatorias, el adversario solo recibe la clave real (en caso btest = 1) o aleatoria (en caso btest = 0). Esta distinción entre claves internas y externas para las consultas de prueba enfatiza que no se supone que las claves externas se usen dentro del intercambio de claves (y, por lo tanto, no es necesario registrar la clave aleatoria probada en el campo de clave de sesión del protocolo), mientras que se usarán las claves internas. (y, por lo tanto, la clave aleatoria probada debe implementarse en los pasos restantes del protocolo para mantener la coherencia).

Además, si existe una sesión asociada label0 que también acaba de aceptar la i-ésima clave (es decir, *label.sidi* = label0 .sidi y label.stexec = label0 .stexec = accepti), también establezca label0 .testedi ÿ true y (solo si USEi = interno) label0 .keyi ÿ *label.keyi* para garantizar la consistencia (de pruebas posteriores y uso de claves (internas)) en el caso especial de que tanto label como label0 estén en estado accepti y, por lo tanto, cualquiera de ellos puede ser probado primero.

Vuelve K.

4.3 Seguridad de los protocolos de intercambio de claves de varias etapas

Al igual que en la formalización del modelo de intercambio de claves Bellare-Rogaway por Brzuska et al. [BFWW11, Brz13], modelamos la seguridad según dos juegos, uno para la indistinguibilidad de claves y otro para la coincidencia de sesiones. La primera es la noción clásica de claves de apariencia aleatoria, refinada bajo el término Seguridad de múltiples etapas de acuerdo con los aspectos de seguridad avanzada para el intercambio de claves de múltiples etapas: (etapa-j) confidencialidad directa, diferentes modos de autenticación y capacidad de reproducción. La propiedad Match complementa esta noción al garantizar que los identificadores de sesión especificados sid coincidan efectivamente con las sesiones asociadas, y también se adapta a la configuración de varias etapas.

4.3.1 Seguridad del partido

La noción de Match security garantiza la solidez de los identificadores de sesión sid, es decir, que identifiquen correctamente las sesiones asociadas en el sentido de que

1. las sesiones con el mismo identificador de sesión para alguna etapa tienen la misma clave en esa etapa,
2. las sesiones con el mismo identificador de sesión para alguna etapa tienen roles opuestos, excepto para posibles respondedores múltiples en etapas rejugables,
3. las sesiones con el mismo identificador de sesión para alguna etapa acuerdan la autenticación de esa etapa nivel,

8Tenga en cuenta que las entradas ListS solo se crean para sesiones honestas, es decir, sesiones generadas por consultas NewSession.

4. las sesiones con el mismo identificador de sesión para alguna etapa comparten el mismo identificador contributivo en esa etapa,

5. las sesiones se asocian con el participante previsto (autenticado) y, para la autenticación mutua basada en secretos previamente compartidos, comparten el mismo identificador de clave,

6. los identificadores de sesión no coinciden en las diferentes etapas, y

7. como máximo dos sesiones tienen el mismo identificador de sesión en cualquier etapa no reproducible.

El juego de seguridad Match GMatch se define así de la siguiente manera. ^{KE,A}

Definición 4.1 (Seguridad del partido). *Sea* KE *un protocolo de intercambio de claves de múltiples etapas con propiedades* (M, AUTH, FS, USE, REPLAY) *y* A *sea un adversario PPT que interactúa con* KE a *través de las consultas definidas en la Sección 4.2 en el siguiente juego* GMatch ^{KE,A} *:*

Configuración. *En la variante de clave pública (pMSKE), el retador genera claves públicas/privadas a largo plazo. pares para cada participante U* ÿ *U.*

Consulta. *El adversario* A *recibe las claves públicas generadas (pMSKE) y tiene acceso a las consultas* NewSecret, NewSession, Send, Reveal, Corrupt *y* Test.

Detener. *En algún momento, el adversario se detiene sin salida.*

Decimos que A *gana el juego, denotado por* GMatch ^{KE,A} *= 1, si se cumple al menos una de las siguientes condiciones:*

1. *Existen dos etiquetas distintas* label, label0 *tales que label.sidi* = label0 .sidi 6= ÿ *para alguna etapa i* ÿ {1, . . . , M}, *pero etiqueta.keyi* 6= etiqueta0 .keyi . *(Diferentes claves de sesión en alguna etapa de sesiones asociadas).*

2. *Existen dos etiquetas distintas* label, label0 *tales que label.sidi* = label0 .sidi 6= ÿ *para alguna etapa i* ÿ {1, . . . , M}, *pero label.role* = label0 .role *y* REPLAYi = no reproducible, *o label.role* = label0 .role = iniciador *y* REPLAYi = reproducible. *(Roles no opuestos de sesiones asociadas en etapa no reproducible).*

3. *Existen dos etiquetas distintas* label, label0 *tales que label.sidi* = label0 .sidi 6= ÿ *para alguna etapa i* ÿ {1, . . . , M}, *pero label.authi* 6= label0 .authi. *(Diferentes tipos de autenticación en alguna etapa de las sesiones asociadas).*[9]

4. *Existen dos etiquetas distintas* label, label0 *tales que label.sidi* = label0 .sidi 6= ÿ *para alguna etapa i* ÿ {1, . . . , M}, *pero label.cidi* 6= label0 .cidi *o label.cidi* = label0 .cidi = *ÿ. (Identificadores contributivos diferentes o no establecidos en alguna etapa de las sesiones en pareja).*

5. *Existen dos etiquetas distintas* label, label0 *tales que label.sidi* = label0 .sidi 6= ÿ *y label.sidj* = label0 .sidj 6= ÿ *para las etapas i, j* ÿ {1, . . . , M} *donde j* ÿ *i, con label.role* = iniciador *y* label0 .role = respondedor *tal que*

   - label.authj,1 ÿ *i (autenticación unilateral), pero* label.pid 6= label0 .id, *o*

9Observe que la seguridad de Match garantiza el acuerdo sobre los niveles de autenticación *previstos* (incluidas las posibles actualizaciones); el nivel de autenticación *rectificado ,* por el contrario, es un elemento técnico del modelo de seguridad que captura el nivel real alcanzado a la luz de las primeras corrupciones al evaluar las consultas de prueba.

• label.authj,2 ÿ *i (autenticación mutua), pero* label.id 6= label0 .pid *o (solo para sMSKE)* *etiqueta.pssid* 6= etiqueta0 .pssid.

(Diferente socio autenticado previsto o (solo sMSKE) diferentes identificadores de clave en la autenticación mutua).

6. Existen dos etiquetas (no necesariamente distintas) label, label0 *tales que label.sidi* = label0 .sidj 6= ÿ *para algunas etapas i, j* ÿ {1, . . . , M} *con i* 6= *j. (Distintas etapas comparten el mismo identificador de sesión).*

7. Existen tres etiquetas distintas por pares label, label0 , label00 *tal que label.sidi* = label0 .sidi = label00 .sidi 6= ÿ *para alguna etapa i* ÿ {1, . . . , M} *con* REPLAYi = no reproducible. *(Más de dos sesiones comparten el mismo identificador de sesión en una etapa no reproducible).*

Decimos que KE *es* Match-secure *si para todos los adversarios* A de PPT *la siguiente función de ventaja es insignificante en el parámetro de seguridad:*

$$\mathrm{AdvMatch}_{KE,A} := \Pr\left[G^{\mathrm{Partido}}_{KE,A} = 1i\right].$$

4.3.2 Seguridad de varias etapas

La segunda y principal noción, la seguridad de múltiples etapas, captura el secreto de la clave similar a Bellare-Rogaway en la configuración de múltiples etapas de la siguiente manera.

Definición 4.2 (Seguridad multietapa). *Sea* KE *un protocolo de intercambio de claves de múltiples etapas con propiedades* (M, AUTH, FS, USE, REPLAY) *y distribución de claves D, y* A *un adversario PPT que interactúa con* KE a *través de las consultas definidas en la Sección 4.2 en el siguiente juego G* Multi -Escenificado $_{KE,A}$ *:*

Configuración. *El retador elige el bit de prueba* btest ÿ$ {0, 1} *al azar y establece los* perdidos ÿ falso. *En la variante de clave pública (pMSKE), además genera pares de clave pública/ privada a largo plazo para cada participante U* ÿ *U.*

Consulta. *El adversario* A *recibe las claves públicas generadas (pMSKE) y tiene acceso a las consultas* NewSecret, NewSession, Send, Reveal, Corrupt *y* Test. *Recuerde que tales consultas pueden definirse* como *verdaderas .*

Suponer. *En algún momento,* A *se detiene y genera una suposición b.*

Finalizar. *El retador establece el indicador 'perdido' en* perdido ÿ verdadero *si existen dos etiquetas (no necesariamente distintas)* label, label0 *y alguna etapa i* ÿ {1, . . . , M} *tal que label.sidi* = label0 .sidi, label.stkey,i = unlocked, *y* label0 .testedi = true. *(El adversario ha probado y revelado la clave de alguna etapa en una sola sesión o en dos sesiones en pareja).*

Decimos que A *gana el juego, denotado por G* Multi-Stage D = *1, si b* = btest y lost = false. *Tenga en cuenta que KE A, la condición ganadora, es independiente de las propiedades de autenticación y confidencialidad directa de* KE, *ya que están directamente integradas en las consultas afectadas* (Revelar *y* Corruptar) *y el paso de finalización del juego; por ejemplo,* Corrupt *se define de manera diferente para non-forward-secrecy versus stage-j forward secreto.*

Decimos que KE *es* seguro en varias etapas *con propiedades* (M, AUTH, FS, USE, REPLAY) *si* KE *es seguro de* coincidencia y para todos los adversarios de PPT A , *la siguiente función de ventaja es insignificante en el parámetro de seguridad:*

$$\mathrm{AdvMulti\text{-}etapa,D}_{KE,A} := \Pr\left[G^{\mathrm{Multietapa,D}}_{KE,A} = 1i\right] - \frac{1}{2}.$$

5 Análisis de seguridad de TLS 1.3 Full 1-RTT Handshake

Ahora pasamos a analizar el protocolo de enlace 1-RTT completo de TLS 1.3 en el modelo de intercambio de claves de varias etapas de clave pública (pMSKE).

Propiedades del protocolo. El protocolo de enlace completo apunta a las siguientes propiedades específicas del protocolo (M, AUTH, FS, USE, REPLAY):

- M = 6: El protocolo de enlace 1-RTT completo consta de seis etapas que derivan, en orden: las claves de tráfico de protocolo de enlace de cliente y servidor tkchs y tkshs, los secretos de tráfico de aplicaciones de cliente y servidor CATS y SATS, el secreto maestro exportador EMS y el secreto maestro de reanudación RMS. Como se muestra en la Figura 1, consideramos que las claves de todas las etapas se derivan en cualquier lado tan pronto como el secreto principal relevante (ES, HS, MS) esté disponible, a pesar de que las claves de cliente/servidor derivadas en paralelo pueden activarse con algún retraso en función de la dirección del flujo.

- AUTORIZACIÓN = ((3, m),(3, m),(3, m),(4, m),(5, m),(6, m)) | *m* ÿ {6, ÿ}: las claves de tráfico de protocolo de enlace tkchs/tkshs no están autenticadas inicialmente y todas las claves se autentican unilateralmente después de alcanzar la etapa 3. Con la autenticación del cliente (opcional), todas las claves además se autentican mutuamente con la etapa *m* = 6; de lo contrario, nunca alcanzan este nivel, *m* = ÿ.

- FS = 1: el protocolo de enlace 1-RTT completo garantiza la confidencialidad hacia adelante para todas las claves derivadas.

- USO = (interno: {1, 2}, externo: {3, *4, 5,* 6}): las claves de tráfico de protocolo de enlace se utilizan internamente para cifrar la segunda parte del apretón de manos; todas las demás claves son externas.

- REPETIR = (no reproducible: {1, *2, 3, 4, 5,* 6}): las teclas de todas las etapas no se pueden reproducir en el protocolo de enlace 1-RTT completo.

Identificadores de sesión y contributivos. Como parte del análisis en el modelo pMSKE, debemos definir cómo se establecen los identificadores de sesión y de contribución para cada etapa durante la ejecución del protocolo de enlace 1-RTT completo de TLS 1.3.

Los identificadores de sesión se establecen al aceptar cada etapa e incluyen una etiqueta y todos los mensajes de protocolo de enlace hasta este punto (ingresando la derivación de la clave):

sid1 = ("CHTS", CH, CKS, SH, SKS), sid2 = ("SHTS", CH, CKS, SH, SKS), sid3 = ("GATOS", CH, CKS, SH, SKS, EE, CR ÿ , SCRT, SCV, SF),
sid4 = (“SATS”, CH, CKS, SH, SKS, EE, CRÿ , SCRT, SCV, SF),
sid5 = (“EMS”, CH, CKS, SH, SKS, EE, CRÿ , SCRT, SCV, SF),
sid6 = (“RMS”, CH, CKS, SH, SKS, EE, CRÿ , SCRT, SCV, SF, CCRTÿ , CCVÿ , CF).

Aquí, los componentes marcados con asterisco (ÿ) están presentes solo en el modo de autenticación mutua. Tenga en cuenta que definimos identificadores de sesión sobre los mensajes de protocolo de enlace no cifrados .

Para los identificadores contributivos en las etapas 1 y 2, el cliente (resp. servidor) al enviar (resp. recibir) los mensajes ClientHello y ClientKeyShare establece cid1 = ("CHTS", CH, CKS), cid2 = ("SHTS", CH, CKS) y luego, al recibir (resp. enviar) los mensajes ServerHello y ServerKeyShare, extenderlo a cid1 = (“CHTS”, CH, CKS, SH, SKS), cid2 = (“SHTS”, CH, CKS, SH, SKS). Todos los demás identificadores contributivos se establecen en cidi = sidi (para etapas *i* ÿ {3, *4, 5,* 6}) cuando se establece el identificador de sesión respectivo.

5.1 Seguridad del partido

Ahora estamos listos para brindar nuestros resultados de seguridad formales para el protocolo de enlace de 1 RTT completo de TLS 1.3, comenzando con la seguridad de Match.

Teorema 5.1 (Seguridad de coincidencia de TLS1.3-full-1RTT). *El protocolo de enlace 1-RTT completo de TLS 1.3 es* Match *seguro con las propiedades* (M, AUTH, FS, USE, REPLAY) *indicadas anteriormente. Para cualquier adversario eficiente* A *nosotros tener*

$$\mathrm{Adv}^{\mathrm{Match}}_{\mathrm{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq n_s^2 \cdot \frac{1}{q} \cdot 2^{-|nonce|},$$

donde ns es el número máximo de sesiones, q es el orden del grupo y |nonce| = 256 *es la longitud en bits de los nonces.*

Recuerde que la seguridad de Match es una propiedad de solidez de los identificadores de sesión. De nuestra definición de identificadores de sesión anterior, se deduce inmediatamente que las sesiones asociadas acuerdan la clave derivada, los roles opuestos, las propiedades de autenticación, los identificadores contributivos y las etapas respectivas. El límite de seguridad surge como el límite de cumpleaños para dos sesiones honestas que eligen el mismo nonce y elemento de grupo; que esto no suceda garantiza que, como máximo, dos socios compartan el mismo identificador de sesión.

Prueba. Necesitamos mostrar las siete propiedades de la seguridad de Match (cf. Definición 4.1).

1. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen la misma clave en esa etapa. y* (a través de identificadores de sesión en cada etapa incluyen las acciones Diffie-Hellman g^x y g^y los mensajes CKS y SKS, fijando la única entrada clave DHE = g^{xy} a todas las claves de etapa derivadas (recuerde que PSK = 0 en el TLS 1.3 completo 1-RTT handshake). Además, el identificador de sesión para cada etapa incluye todos los mensajes de handshake que ingresan a la derivación de clave: para las etapas 1 y 2 mensajes hasta SKS, para las etapas 3–5 mensajes hasta SF y para la etapa 6 todos mensajes (hasta CF).En cada etapa, el identificador de sesión determina *todas las* entradas a la derivación de la clave, y el acuerdo sobre él asegura el acuerdo sobre la clave de la etapa.

2. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen funciones opuestas, excepto para posibles respondedores múltiples en etapas rejugables.*
   Suponiendo que, como máximo, dos sesiones comparten el mismo identificador de sesión (que mostramos a continuación), dos sesiones de iniciador (cliente) o respondedor (servidor) nunca tienen el mismo identificador de sesión, ya que nunca aceptan mensajes entrantes con roles incorrectos, y los mensajes de saludo iniciales son escrito con el rol del remitente. No hay etapas reproducibles en el protocolo de enlace de 1 RTT completo de TLS 1.3.

3. *Las sesiones con el mismo identificador de sesión para alguna etapa acuerdan la autenticación de esa etapa nivel.*
   Por definición, la autenticación para las etapas 1 a 2 y 3 a 5 está fijada en no autenticar y unilateral (desde la etapa 3 en adelante), respectivamente, por lo tanto, es acordada por todas las sesiones. Para la última etapa, la presencia de CR, CCRT y CCV en sid6 determina sin ambigüedad si, a partir de la etapa 6, las claves se autentican mutuamente (y de otra manera unilateralmente).

4. *Las sesiones con el mismo identificador de sesión para alguna etapa comparten el mismo identificador contributivo.*
   Esto se debe a que, para cada etapa i, el identificador contributivo cidi es final e igual a sidi una vez que se establece el identificador de sesión.

5. *Las sesiones se asocian con el participante previsto (autenticado).*
   Este caso solo se aplica a las etapas autenticadas unilateral o mutuamente, lo que se logra, posiblemente de forma retroactiva, al alcanzar las etapas 3, resp. etapa 6 (solo si el cliente se autentica).

En el protocolo de enlace 1-RTT completo de TLS 1.3, las identidades de los pares se aprenden a través de los mensajes del certificado. Como solo nos preocupan las sesiones honestas de cliente y servidor para la seguridad de Match, que solo enviará certificados que acrediten su propia identidad, el acuerdo sobre SCRT garantiza el acuerdo sobre la identidad del servidor (respondedor), y viceversa para CCRT y la identidad del cliente (iniciador). . Dicho acuerdo se garantiza mediante la inclusión de SCRT en el identificador de sesión para la etapa 3 para autenticación unilateral, y SCRT y CCRT para autenticación mutua en sid6: una vez que dos sesiones alcanzan estas etapas y están de acuerdo en sid3, resp. sid6, ellos (retroactivamente) también están de acuerdo con el par previsto (respondedor, resp. iniciador).

6. *Los identificadores de sesión son distintos para diferentes etapas.*
   Esto es trivial, ya que el identificador de sesión de cada etapa tiene una etiqueta única.

7. *Como máximo, dos sesiones tienen el mismo identificador de sesión en cualquier etapa no reproducible.*
   Recuerde que todos los identificadores de sesión mantenidos por alguna sesión incluyen el nonce aleatorio de esa sesión y el recurso compartido Diffie-Hellman. Por lo tanto, para una colisión triple entre los identificadores de sesión de las partes honestas, alguna sesión necesitaría elegir el mismo elemento de grupo y el mismo nonce que otra sesión (que luego se puede asociar a través de un protocolo regular que se ejecuta en una tercera sesión). La probabilidad de que ocurra tal colisión puede estar acotada desde arriba por $\dot{z}\, 1/q \cdot 2\ddot{y}|\text{nonce}|$ donde ns es el número máximo de sesiones, q es el orden de grupo de n con límite de cumpleaños y $|\text{nonce}| = 256$ la longitud de bits de los nonces.

□

5.2 Seguridad de múltiples etapas

Ahora llegamos al resultado principal de seguridad de varias etapas para el protocolo de enlace de 1 RTT completo de TLS 1.3.

Teorema 5.2 (Seguridad multietapa de TLS1.3-full-1RTT). *El protocolo de enlace 1-RTT completo de TLS 1.3 es* seguro en varias etapas *con las propiedades* (M, AUTH, FS, USE, REPLAY) *indicadas anteriormente. Formalmente, para cualquier adversario* A eficiente *contra la* seguridad *Multi-Stage existen algoritmos eficientes* B1, . . . , B7 *tal que*

$$\begin{aligned} \text{AdvMulti-etapa,D}_{\text{TLS1.3-full-1RTT,A}} \ddot{y}\, 6ns &\leq \text{AdvCOLL}_{\text{H,B1}} + nu \cdot \text{AdvEUF-CMA}_{\text{SIG,B2}} \\ &+ ns \cdot \text{Advdual-spPRF-ODH}_{\text{HKDF.Extract,G,B3}} + \text{AdvPRF-sec}_{\text{HKDF.Expandir,B4}} \\ &+ 2 \cdot \text{AdvPRF-sec}_{\text{HKDF.Expand,B5}} + \text{AdvPRF-sec}_{\text{HKDF.Extracto,B6}} \\ &+ \text{AdvPRF-sec}_{\text{HKDF.Expandir,B7}} \end{aligned}$$

donde ns es el número máximo de sesiones y nu es el número máximo de usuarios.

Para el protocolo de enlace 1-RTT completo de TLS 1.3, la seguridad de varias etapas se deriva esencialmente de dos líneas de razonamiento. En primer lugar, las firmas (infalsificables) que cubren (un hash resistente a colisiones de) los mensajes completos de Hello aseguran que las etapas de la sesión con un par compartido autenticado intercambien valores Diffie-Hellman que se originen en una sesión de pareja honesta. Luego, todas las claves se derivan de manera que se asegure que (a) de un secreto Diffie-Hellman desconocido para las sesiones del adversario se derivan claves indistinguibles de las aleatorias (bajo los supuestos PRF-ODH y PRF en los pasos HKDF.Extract y HKDF.Expand) que (b) son independientes, lo que permite revelar y probar claves de sesión en diferentes etapas.

Prueba. A continuación, procedemos a través de una secuencia de juegos. A partir del juego de etapas múltiples, limitamos la diferencia de ventaja del adversario A entre dos juegos cualesquiera mediante supuestos teóricos de complejidad hasta llegar a un juego en el que el adversario A no puede ganar, es decir, su ventaja es como máximo 0.

Juego 0. Este es el juego Multi-Stage original, es decir,

$$\text{AdvMulti-Stage}_{D} = \text{AdvG0}^{\text{TLS1.3-full-1RTT,A}}_{\text{TLS1.3-full-1RTT,A.}}$$

Juego 1. En un primer paso, restringimos al adversario A en el juego de Etapas Múltiples a realizar una sola consulta de Prueba. Es decir, podemos convertir formalmente cualquier adversario A de múltiples consultas en un adversario A1 que solo realiza una única consulta de prueba. Esto reduce su ventaja, en base a un cuidadoso argumento híbrido, por un factor como máximo de 1/6ns para las seis etapas en cada una de las sesiones de *ns* . Tenga en cuenta que en el argumento híbrido A1 adivina aleatoriamente una de las sesiones por adelantado y solo realiza la consulta de prueba única para esta sesión. Las otras consultas de prueba de un atacante de consultas múltiples se sustituyen gradualmente por consultas Reveal cuidadosamente diseñadas, donde el atacante de consulta única A1 necesita conocer la asociación correcta de sesiones a través de los identificadores de sesión sid para una simulación correcta, por ejemplo, para evitar pérdidas debido a malas combinaciones Revelar-Prueba en los socios de sesión debido a las nuevas consultas Revelar. Los identificadores de sesión sid1 y sid2 solo contienen información pública, por lo que la asociación es fácil de verificar. Pero luego el cifrado de protocolo de enlace se activa de tal manera que sid3, . . . ,sid6 se basan en datos cifrados. Afortunadamente, si la consulta de prueba única se refiere a un secreto de tráfico de protocolo de enlace (cliente o servidor), entonces la asociación es fácil de decidir en función de sid1 resp. sid2. Si la consulta de prueba se refiere a una clave posterior, podemos revelar las claves de tráfico de protocolo de enlace de etapas anteriores, usarlas para descifrar la comunicación posterior y, por lo tanto, determinar sid3, . . . ,sid6 también. Proporcionamos los detalles completos de este argumento híbrido en el Apéndice A.

Incorporando la transformación de A en A1 en el juego, es decir, haciendo que el retador adivinar la sesión correcta y haciendo las adaptaciones, obtenemos

$$\text{AdvG0}^{\text{TLS1.3-full-1RTT,A}} \leq 6ns \cdot \text{AdvG1}^{\text{TLS1.3-completo-1RTT,A.}}$$

De ahora en adelante, podemos referirnos a *la* etiqueta de sesión probada en la etapa i, y podemos asumir que conocemos este número de sesión (según el orden de las sesiones iniciadas) al comienzo del experimento.

Juego 2. En este juego, el retador aborta si dos sesiones honestas calculan el mismo valor hash para diferentes entradas en cualquier evaluación de la función hash H. Podemos romper la resistencia a la colisión de H en caso de este evento dejando una reducción B1 enviar los dos valores de entrada distintos a H. Por lo tanto:

$$\text{AdvG1}^{\text{TLS1.3-full-1RTT,A}} \leq \text{AdvG2}^{\text{TLS1.3-full-1RTT,A}} + \text{AdvCOLL}_{H,\,B1}.$$

A partir de aquí, nuestro análisis de seguridad considera por separado los dos casos (disjuntos) que

A. la etiqueta de sesión probada no tiene un socio contribuyente honesto en la primera etapa (es decir, existe sin etiqueta0 6= etiqueta con etiqueta.cid1 = etiqueta0 .cid1) , y

B. la etiqueta de sesión probada tiene un socio contribuyente honesto en la primera etapa (es decir, existe etiqueta0 con etiqueta.cid1 = etiqueta0 .cid1).

Esto nos permite considerar la ventaja del adversario por separado para estos dos casos A (denominado “prueba sin compañero”) y B (“prueba con compañero”):

$$\text{AdvG2}^{\text{TLS1.3-full-1RTT,A}} \leq \text{AdvG2}^{\text{TLS1.3-full-1RTT,A}}_{\text{prueba sin pareja}} + \text{AdvG2}^{\text{TLS1.3-completo-1RTT,A}}_{\text{prueba con pareja}}.$$

Caso A. Prueba sin pareja

En primer lugar, consideramos el caso de que la etiqueta de sesión probada no tenga un socio contribuyente en la etapa 1, lo que implica que no tiene un socio contribuyente en ninguna etapa. Por definición, un adversario no puede ganar si la consulta de prueba emitida para dicha sesión se encuentra en una etapa que, en el momento de la consulta de prueba, tiene un par no autenticado. Aquí, la autenticación se refiere al nivel *rectificado ,* porque el oráculo de prueba comprueba esta propiedad refinada. Por lo tanto, para una sesión de cliente probada, la prueba (para cualquier etapa) no se puede emitir antes de que se alcance la etapa 3 y luego solo si la corrupción del cliente o del servidor asociado no se ha producido antes de la etapa 3. De lo contrario, el adversario pierde el juego. Para una sesión de servidor, la prueba solo se puede emitir cuando se alcanza la etapa 6 y se realiza la autenticación del cliente. Aquí, nuevamente, el cliente no puede corromperse antes, de lo contrario, el nivel de autenticación rectificado no se autenticaría

Juego A.0. Es igual a G2 con adversario restringido para probar una sesión sin un socio contribuyente honesto en la primera etapa.

$$Adv^{G2,\text{ prueba sin pareja}}_{\text{TLS1.3-full-1RTT},A} = Adv^{GA.0}_{\text{TLS1.3-full-1RTT},A}.$$

Juego A.1. En este juego, dejamos que el retador adivine la identidad de pares *U* ÿ U de la etiqueta de sesión probada (observe que se debe establecer una para que la Prueba sea admisible, como se discutió anteriormente), y aborte si esa suposición fue incorrecta (es decir, etiqueta.pid 6= U). Esto puede reducir la ventaja de A en un factor como máximo el número de usuarios *nu:*

$$Adv^{GA.0}_{\text{TLS1.3-full-1RTT},A} \text{ ÿ } nu \cdot Adv^{GA.1}_{\text{TLS1.3-completo-1RTT},A}.$$

Juego A.2. Ahora permitimos que el retador cancele el juego si la etiqueta de la sesión probada recibe, dentro del mensaje CertificateVerify de su igual label.pid = U, una firma válida en algún (valor hash de un) mensaje que no ha sido calculado por ninguna sesión honesta de usuario U. Tenga en cuenta que este mensaje debe incluir los datos de la transcripción ClientHellok . . . kClientCert resp. ClienteHola . . . kServerCert (ver Tabla 2). Tenga en cuenta que, como se discutió anteriormente, cuando se envía la consulta de prueba a la etiqueta, dicho mensaje debe haberse recibido, en el caso de un cliente, antes de aceptar la etapa 3 y sin corrupción previa del servidor; o, en el caso de un servidor, antes de la etapa 6 cuando el servidor está hablando con un cliente de autenticación que aún no está dañado.

Podemos acotar la probabilidad de que el Juego *GA.2 aborte* por este motivo por la ventaja de un adversario B2 frente a la seguridad EUF-CMA del esquema de firma SIG. En la reducción, B2 recibe una clave pública *pkU* de un esquema de firma, calcula las claves a largo plazo de todas las partes *U* U \ *{U}* excepto *U* y simula *GA.1* para A1. ÿ Cada vez que en esa simulación B2 tiene que calcular una firma bajo *skU ,* lo hace a través de su oráculo de firma. Cuando la etiqueta recibe una firma válida en el valor hash del mensaje, el adversario B2 emite *(H(m),* ÿ) como su falsificación. Tenga en cuenta que, en este punto, la sesión asociada no puede corromperse de modo que el falsificador de la firma no necesite revelar la clave de firma secreta antes de generar la falsificación.

Queda por argumentar que el par *(H(m),* ÿ) constituye una falsificación exitosa. Para ver esto, tenga en cuenta que la etiqueta de sesión probada calcula el valor hash *H(m)* del mensaje *m* para verificar la corrección, pero ninguna otra sesión honesta ha calculado una firma para este mensaje. Según Game G2, esto también significa que ninguna otra sesión honesta ha obtenido el mismo valor hash *H(m0) = H(m)* para algún otro mensaje m0 . Concluimos que el valor hash *H(m)* no ha sido firmado por el usuario *U* antes.

$$Adv^{GA.1}_{\text{TLS1.3-full-1RTT},A} \text{ ÿ } Adv^{GA.2}_{\text{TLS1.3-full-1RTT},A} + Adv^{\text{EUF-CMA}}_{\text{SEÑAL},B2}$$

Para el caso A, se deduce que el adversario no puede hacer una consulta de prueba legítima en absoluto, a menos que falsifique firmas. O las sesiones no tienen socio coadyuvante, o las sesiones en etapas posteriores han sido rechazadas por firmas inválidas. Si el adversario no puede probar ninguna sesión sin un compañero que contribuya, claramente no tiene ninguna ventaja al predecir el desafío secreto bit b:

$$Adv^{GA.2}_{TLS1.3\text{-}full\text{-}1RTT,A} = 0.$$

Caso B. Prueba con Compañero

Juego B.0. Este es G2 donde el adversario está restringido a emitir una consulta de prueba a una sesión con un socio contribuyente honesto en la primera etapa.

$$Adv^{G2,\ prueba\ con\ socio}_{TLS1.3\text{-}full\text{-}1RTT,A} = Adv^{GB.0}_{TLS1.3\text{-}full\text{-}1RTT,A}.$$

Juego B.1. En este juego, adivinamos una sesión label0 6= label (de un máximo de *ns* sesiones en el juego) y abortamos el juego si *label.cid1* 6= label0 .cid1 es decir, label0 no es el socio contribuyente honesto en la etapa 1 de la prueba. sesión (recuerde que asumimos que tal socio existe en este caso de prueba). Esto reduce la ventaja del adversario en un factor de *1/ns como máximo.*

$$Adv^{GB.0}_{TLS1.3\text{-}full\text{-}1RTT,A} \ddot{y} ns \cdot Adv^{GB.1}_{TLS1.3\text{-}completo\text{-}1RTT,A}.$$

Juego B.2. En este juego, reemplazamos el secreto del apretón de manos HS derivado en la sesión probada y su sesión asociada contributiva con una cadena uniformemente aleatoria e independiente HSf ÿ$ {0, 1} empleamos la suposición dual-snPRF-ODH (Definición 2.3) para ser capaz de simular el cálculo de HS en una sesión de cliente asociada para un mensaje ServerKeyShare modificado. Más precisamente, podemos convertir a cualquier adversario capaz de distinguir este cambio en un adversario B3 contra la seguridad dual-snPRF-ODH de la función HKDF.Extract (tomando dES como primera entrada y DHE como segunda entrada). Para esto , B3 solicita un desafío de PRF en dES calculado en la sesión de prueba y su socio contribuyente honesto. Utiliza las acciones Diffie-Hellman *g* y ServerKeyShare obtenidas de las sesiones probadas y contributivas, y el valor de desafío PRF con ClientKeyShare, *g* probada. Si es necesario, B3 usa sus consultas PRF-ODH para derivar HS en la sesión asociada en *g* diferentes . Al proporcionar una simulación de sonido de *GB.1* (si el bit muestreado por HKDF.Extract (dES, gxy)), o *GB.2* (si el), esto limita el
el retador dual-snPRF-ODH era 0 y, por lo tanto, HS = bit
muestreado por el retador dual-snPRF-ODH era 1 y, por lo tanto, HSf ÿ$ {0, 1} diferencia de
ventaja de A como:

$$Adv^{GB.1}_{TLS1.3\text{-}full\text{-}1RTT,A} \ddot{y} Adv^{GB.2}_{TLS1.3\text{-}full\text{-}1RTT,A} + Advdual\text{-}snPRF\text{-}ODH_{HKDF.Extracto,G,B3}.$$

Juego B.3. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expand en todas las evaluaciones usando el valor HSf reemplazado en *GB.2.* Esto afecta la derivación del secreto de tráfico de protocolo de enlace del cliente CHTS, el secreto de tráfico de protocolo de enlace del servidor SHTS y el secreto de protocolo de enlace derivado dHS en la sesión de destino y su socio coincidente, y el secreto de protocolo de enlace derivado dHS en todas las sesiones que utilizan el mismo secreto de protocolo de enlace HSf . Tenga en cuenta que para CHTS y SHTS, estos valores son distintos de cualquier otra sesión que utilice el mismo valor secreto de protocolo de enlace HSf como entrada del valor hash H2 = H(CHkSH), (donde CH y SH contienen los valores aleatorios de cliente y servidor *rc, rs* respectivamente) y por Game G2 excluimos las colisiones hash. Reemplazamos la derivación

de CHTS, SHTS y dHS en dichas sesiones con valores aleatorios CHTS ^, SHTS ^, dHS ÿ$ {0, 1} ^{ÿ} . Para asegurar la consistencia, reemplazamos las derivaciones de dHS con el dHS reemplazado muestreado por la primera sesión para evaluar HKDF. Expandir usando HSf . Podemos acotar la diferencia que este paso introduce en el ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por el juego anterior, HSf es un valor uniformemente aleatorio y el reemplazo es sólido. Por lo tanto:

$$Adv^{GB.2}_{TLS1.3\text{-}full\text{-}1RTT,A} \text{ ÿ } Adv^{GB.3}_{TLS1.3\text{-}full\text{-}1RTT,A} + AdvPRF\text{-}seg_{HKDF.Expandir,B4} .$$

En este punto, CHTS y SHTS son independientes de cualquier valor calculado en cualquier sesión no asociada (en la etapa 1 o 2) con la sesión probada: identificadores de sesión distintos y sin colisiones de hash (a partir del Juego G2) asegúrese de que las entradas de la etiqueta PRF para derivar CHTS y SHTS sean únicas.

Juego B.4. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones usando los valores CHTS ^, SHTS reemplazados en *GB.3.* Esto afecta la derivación del apretón de manos del cliente. clave de tráfico tkchs, y el protocolo de enlace del servidor tkshs de la clave de tráfico en la sesión de destino y su contribución compañero. Reemplazamos la derivación de tkchs y tkshs con valores aleatorios tk]chs ÿ$ {0, 1} ^{L} y tk]shs ÿ$ {0, 1} L, donde *L* indica la suma de la longitud de la clave y la longitud de iv para el AEAD negociado esquema. Podemos acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de dos evaluaciones de las funciones pseudoaleatorias HKDF.Expand. Tenga en cuenta que por el juego anterior CHTS y SHTS son valores uniformemente aleatorios, y estos reemplazos son correctos. Por lo tanto:

$$Adv^{GB.3}_{TLS1.3\text{-}full\text{-}1RTT,A} \text{ ÿ } Adv^{GB.4}_{TLS1.3\text{-}full\text{-}1RTT,A} + 2 \cdot AdvPRF\text{-}seg_{HKDF.Expandir,B5} .$$

Juego B.5. En este juego, reemplazamos la función pseudoaleatoria HKDF.Extract en todas las evaluaciones del valor dHS reemplazado en *GB.3.* Esto afecta la derivación del MS secreto maestro en cualquier sesión usando el mismo secreto de apretón de manos derivado dHS. Reemplazamos la derivación de MS en dichas sesiones. con el valor aleatorio MS ÿ$ {0, 1} ^{ÿ} . Podemos acotar la diferencia que este paso introduce en el ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Extract. Tenga en cuenta que por *GB.3,* dHS es un valor uniformemente aleatorio y este reemplazo es sólido. Por lo tanto:

$$Adv^{GB.4}_{TLS1.3\text{-}full\text{-}1RTT,A} \text{ ÿ } Adv^{GB.5}_{TLS1.3\text{-}full\text{-}1RTT,A} + AdvPRF\text{-}seg_{HKDF.Extracto,B6} .$$

Juego B.6. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones del valor MS reemplazado en *GB.5* en la sesión de destino y su sesión coincidente. Esto afecta el derivación del secreto de tráfico de la aplicación cliente CATS, el secreto de tráfico de la aplicación servidor SATS el secreto maestro del exportador EMS y el secreto maestro de reanudación RMS. Para CATS, SATS y EMS, estas evaluaciones son distintas de cualquier sesión no asociada con la sesión probada, ya que la evaluación de HKDF.Expand también toma como entrada H4 = H(CHk . . . kSF) (donde CH y SH contienen el valores aleatorios de cliente y servidor *rc* y *rs* respectivamente), y por Game G2 excluimos las colisiones hash. Para RMS, esta evaluación es distinta de cualquier sesión no asociada con la sesión probada, ya que la evaluación de HKDF.Expand también toma como entrada H5 = H(CHk . . . kCF). Reemplazamos la derivación de CATS, SATS, EMS y RMS con valores aleatorios CATS ^, SATS ^, EMS], RMS ÿ$ {0, 1} ^{ÿ} . Nosotros puede acotar la diferencia que este paso introduce en la ventaja de A por el secreto de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por el juego anterior es uniformemente aleatorio MS y el valor independiente, y estos reemplazos son sólidos. Por lo tanto:

$$Adv^{GB.5}_{TLS1.3\text{-}full\text{-}1RTT,A} \text{ ÿ } Adv^{GB.6}_{TLS1.3\text{-}full\text{-}1RTT,A} + AdvPRF\text{-}seg_{HKDF.Expandir,B7} .$$

Notamos que en este juego ahora hemos reemplazado las claves de todas las etapas en la sesión probada con valores uniformemente aleatorios que, en la ejecución del protocolo, son independientes de los valores en cualquier sesión no asociada a la sesión probada. Por lo tanto:

$$Adv^{GB\_6}_{TLS1.3\text{-}full\text{-}1RTT,A} = 0.$$

La combinación de los límites simples dados produce la siguiente declaración de seguridad:

$$Adv^{G2}_{TLS1.3\text{-}full\text{-}1RTT,A} \le ns \cdot \Big(Adv^{dual\text{-}sspPRF\text{-}ODH}_{HKDF.Extract,G,B3} + Adv^{PRF\text{-}sec}_{HKDF.Expand,B4} + 2 \cdot Adv^{PRF\text{-}sec}_{HKDF.Expandir,B5} + Adv^{PRF\text{-}sec}_{HKDF.Extract,B6} + Adv^{PRF\text{-}sec}_{HKDF.Expandir,B7} \Big) \; !\square$$

6 Análisis de seguridad de los protocolos de enlace TLS 1.3 PSK/ PSK-(EC)DHE (con 0-RTT opcional)

Pasamos ahora a analizar los protocolos de enlace de clave precompartida TLS 1.3, con y sin intercambio de claves Diffie-Hellman (PSK-(EC)DHE, resp. PSK) y con claves 0-RTT opcionales, en el multi secreto precompartido. modelo de intercambio de claves de etapa (sMSKE).

Propiedades del protocolo. Los protocolos de enlace PSK/PSK-(EC)DHE (0-RTT) tienen como objetivo las siguientes propiedades específicas del protocolo (M, AUTH, FS,USE, REPLAY):

- M = 8: Los protocolos de enlace PSK con 0-RTT opcional constan de ocho etapas que derivan, en orden: el secreto de tráfico inicial ETS y el secreto maestro exportador inicial EEMS (ambos solo en modo 0-RTT), el tráfico de protocolo de enlace del cliente y del servidor las claves tkchs y tkshs, los secretos de tráfico de aplicaciones de cliente y servidor CATS y SATS, el secreto maestro del exportador EMS y el secreto maestro de reanudación RMS.

- Las propiedades de autenticación AUTH difieren entre PSK(-only) y PSK-(EC)DHE (0-RTT) apretones de manos:

  – para PSK (0-RTT), AUTH = ((1, 1),(2, 2), . . . ,(8, 8)): Todas las claves se cambian inmediatamente aliado autenticado (a partir de la clave precompartida). – para

  PSK-(EC)DHE (0-RTT), AUTH = ((1, 1),(2, 2),(5, 8),(5, 8),(5, 8),(6, 8),(7, 8),(8, 8)): Las claves 0-RTT ETS/EEMS siempre se autentican mutuamente, las claves de tráfico de protocolo de enlace tkchs/tkshs inicialmente no están autenticadas, todas las claves que no son 0-RTT alcanzan un alcance unilateral. autenticación con etapa 5 y autenticación mutua con etapa 8.[10]

- El secreto hacia adelante del protocolo de enlace PSK depende de si se realiza un intercambio efímero de claves Diffie-Hellman:

  – para PSK solo, FS = ÿ: el protocolo de enlace solo PSK no proporciona ningún secreto hacia adelante. – para PSK-(EC)DHE,

  FS = 3: El protocolo de enlace PSK-(EC)DHE proporciona confidencialidad hacia adelante para todas las claves que no sean 0-RTT.

10 No es sencillo ver por qué algunas claves PSK-(EC)DHE no se consideran autenticadas mutuamente de inmediato, en contraste con las claves del protocolo de enlace de solo PSK. Considere las claves de tráfico de protocolo de enlace en el protocolo de enlace PSK-(EC)DHE: en el modelo, el adversario A podría enviar su propio recurso compartido *g* a una sesión de servidor, observar el recurso compartido de la sesión de tráfico de protocolo de enlace de PSK y DHE. Esas claves ahora deben considerarse secretas hacia adelante (debido a los recursos compartidos DH efímeros), sin embargo, cuando A corrompe PSK, puede calcular las claves de tráfico de protocolo de enlace. Por lo tanto, estas claves no pueden tratarse como un secreto directo y autenticarse mutuamente al mismo tiempo.

- USO = (interno: {3, 4}, externo: {1, *2, 5, 6, 7,* 8}): las claves de tráfico de protocolo de enlace se utilizan entre finalmente para cifrar la segunda parte del apretón de manos; todas las demás claves son externas.

- REPETIR = (reproducible: {1, 2}, no reproducible: {3, *4, 5, 6, 7,* 8}): las teclas 0-RTT ETS y EEMS se pueden reproducir, todas las teclas de otras etapas no.

Identificadores de sesión y contributivos. En cuanto al protocolo de enlace 1-RTT completo (cf. Sección 5), definimos los identificadores de sesión sobre los mensajes de protocolo de enlace no cifrados; el identificador de cada etapa incluye una etiqueta y todos los mensajes de apretón de manos hasta que esa etapa acepta:

sid1 = ("ETS", CH, CKS† , CPSK), sid2 = ("EEMS", CH, CKS† , CPSK), sid3 = ("CHTS", CH, CKS† , CPSK, SH, SKS† , SPSK), sid4 = ("SHTS", CH, CKS† , CPSK, SH, SKS† , SPSK), sid5 = ("GATOS", CH, CKS† , CPSK, SH, SKS† , SPSK, EE, SF), sid6 = ("SATS", CH, CKS† , CPSK, SH, SKS† , SPSK, EE, SF), sid7 = ("EMS", CH, CKS† , CPSK, SH, SKS† , SPSK, EE, SF),
sid8 = ("RMS", CH, CKS† , CPSK, SH, SKS† , SPSK, EE, SF, CF).

Los componentes indicados con † están presentes solo en la variante PSK-(EC)DHE.

Para los identificadores contributivos en las etapas 3 y 4, como para el protocolo de enlace completo, queremos asegurarnos de que se puedan probar las sesiones del servidor con una contribución honesta del cliente, incluso si la respuesta del servidor nunca llega al cliente. Por lo tanto, permitimos que el cliente (resp. servidor) al enviar (resp. recibir) los mensajes ClientHello, ClientKeyShare† y ClientPreSharedKey establezca cid3 = ("CHTS", CH, CKS† , CPSK), cid4 = ("SHTS", CH, CKS† , CPSK) y luego, al recibir (resp. enviar) los mensajes ServerHello, ServerKeyShare† y ServerPreSharedKey extender a cid3 = ("CHTS", CH, CKS† , CPSK, SH, SKS† , SPSK), cid4 = ("SHTS", CH, CKS† , CPSK, SH, SKS† , SPSK). Todas las demás etapas (i ∈ {1, 2, 5, 6, 7, 8}) establecen cidi = sidi cuando se establece el identificador de sesión respectivo.

6.1 TLS 1.3 PSK solo (0-RTT opcional)

Podemos comenzar a dar nuestros resultados de seguridad para el protocolo de enlace 0-RTT solo para PSK de TLS 1.3. Comenzamos con la seguridad de Match.

6.1.1 Seguridad del partido

Teorema 6.1 (Seguridad de coincidencia de TLS1.3-PSK-0RTT). *El protocolo de enlace 0-RTT solo para PSK de TLS 1.3 es Match-secure con las propiedades* (M, AUTH, FS, USE, REPLAY) *indicadas anteriormente. Para cualquier adversario eficiente* A *existe un algoritmo eficiente* B *tal que*

$$\mathrm{Adv}^{\mathrm{Match}}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq \mathrm{Adv}^{\mathrm{COLL}}_{\mathrm{HMAC},\mathcal{B}} + \frac{n_p^2}{|\mathcal{P}|} + n_s^2 \cdot 2^{-|nonce|},$$

donde ns es el número máximo de sesiones, np es el número máximo de secretos previamente compartidos, |P| *es el tamaño del espacio secreto precompartido y* |*nonce*| = 256 *es la longitud en bits de los nonces.*

Recuerde que la seguridad de Match es una propiedad de solidez de los identificadores de sesión. De nuestra definición de identificadores de sesión anterior, se deduce inmediatamente que las sesiones asociadas acuerdan la clave derivada, los roles opuestos, las propiedades de autenticación, los identificadores contributivos y las etapas respectivas. Como en la prueba de seguridad Match para TLS1.3-full-1RTT, el límite de seguridad surge como el límite de cumpleaños para dos sesiones honestas que eligen el mismo nonce; que esto no suceda garantiza que, como máximo, dos socios compartan el mismo identificador de sesión.

Prueba. Necesitamos mostrar las siete propiedades de la seguridad de Match (cf. Definición 4.1).

1. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen la misma clave en esa etapa.*
   Los identificadores de sesión en cada etapa incluyen el identificador precompartido pssid = *pskid* (a través de los mensajes CPSK y SPSK, fijando la única clave de entrada PSK (ya que ambas partes acuerdan una asignación *pssU,V* (pssid) = pss = PSK a todas las claves de etapa derivadas) (recuerde que DHE = 0 en el protocolo de enlace 0-RTT solo PSK de TLS 1.3). Además, para cada etapa, el identificador de sesión incluye todos los mensajes de protocolo de enlace que ingresan a la derivación de la clave: para las etapas 1 y 2 mensajes hasta CPSK para las etapas 3 y 4 mensajes hasta SPSK, para las etapas 5, 6, 7 mensajes hasta SF, y para la etapa 8 todos los mensajes (hasta CF).En cada etapa, el identificador de sesión determina *todas las* entradas para la derivación de claves y el acuerdo sobre por lo tanto, asegura el acuerdo sobre la clave de escena.

2. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen funciones opuestas, excepto para posibles respondedores múltiples en etapas rejugables.*
   Suponiendo que, como máximo, dos sesiones comparten el mismo identificador de sesión (que mostramos a continuación), dos sesiones de iniciador (cliente) o respondedor (servidor) nunca tienen el mismo identificador de sesión, ya que nunca aceptan mensajes entrantes con roles incorrectos, y los mensajes de saludo iniciales son escrito con el rol del remitente. Esto excluye las etapas 1 y 2, que son etapas reproducibles en el protocolo de enlace 0-RTT solo para PSK de TLS 1.3.

3. *Las sesiones con el mismo identificador de sesión para alguna etapa acuerdan la autenticación de esa etapa nivel.*
   Todas las etapas del protocolo de enlace 0-RTT de TLS 1.3 PSK solo se autentican mutuamente, por lo que esto es trivialmente cierto.

4. *Las sesiones con el mismo identificador de sesión para alguna etapa comparten el mismo identificador contributivo.*
   Esto se debe a que, para cada etapa i, el identificador contributivo cidi es final e igual a sidi una vez que se establece el identificador de sesión.

5. *Las sesiones se asocian con el participante previsto (autenticado) y comparten la misma clave identificador*
   Todos los identificadores de sesión incluyen los valores de pssid y *binder* enviados como parte de ClientHello.
   El psid, por lo tanto, se acuerda trivialmente. El valor del *aglutinante* se deriva de ese PSK a través de una secuencia de cálculos de HKDF/HMAC. Si tratamos HMAC como una función hash resistente a colisiones sin clave sobre ambas entradas, la clave y el espacio del mensaje, el acuerdo sobre el *aglutinante* implica el acuerdo sobre PSK. Este paso es necesario, ya que A puede establecer varios valores de PSK para compartir el mismo psid y, por lo tanto, un psid no necesariamente determina de manera única un PSK secreto precompartido desde la perspectiva de cada compañero.
   En su lugar, usamos el *aglutinante* para determinar de forma única el acuerdo sobre PSK entre pares. Como todos los valores de PSK se eligen uniformemente al azar dentro de NewSecret p/|P|, donde P es el espacio secreto previamente consulta, colisionan solo con una probabilidad insignificante, limitada por el límite de cumpleaños $\frac{n^2}{2}$ compartido y *np* el número máximo de secretos previamente compartidos.
   Por lo tanto, el acuerdo sobre el *aglutinante* y PSK finalmente implica que pssid, tal como lo interpreta la sesión del servidor y el cliente asociados, se origina en la misma llamada NewSecret. Esto, desde el

perspectiva tanto del cliente como del servidor, identifica de manera única la identidad del par respectivo y, por lo tanto, garantiza el acuerdo sobre los pares previstos.

6. *Los identificadores de sesión son distintos para diferentes etapas.*
   Esto es trivial, ya que el identificador de sesión de cada etapa tiene una etiqueta única.

7. *Como máximo, dos sesiones tienen el mismo identificador de sesión en cualquier etapa no reproducible.*
   Recuerda que las etapas 1 y 2 son rejugables, por lo que solo necesitamos considerar las etapas *i* ÿ {3, *4, 5, 6, 7,* 8}. Observe que todos los identificadores de sesión de estas etapas incluyen un nonce aleatorio de cliente y servidor *(rc* y *rs* respectivamente), a través de los mensajes ClientHello y ServerHello. Por lo tanto, para una colisión triple entre los identificadores de sesión de las partes honestas, alguna sesión tendría que elegir el mismo nonce que otra sesión (que luego se puede asociar a través de un protocolo regular que se ejecuta en una tercera sesión). La probabilidad de que ocurra tal colisión puede ser donde *ns* es el número máximo delimitado desde arriba por el límite de cumpleaños *n* de sesiones, y |*nonce*| = 256 la longitud de bits de los nonces. $\frac{2}{s} \cdot 2^{\ddot{y}|\textit{una vez}|}$, □

6.1.2 Seguridad de varias etapas

Teorema 6.2 (Seguridad multietapa de TLS1.3-PSK-0RTT). *El protocolo de enlace TLS 1.3 PSK 0-RTT es* seguro en varias etapas *con las propiedades* (M, AUTH, FS, USE, REPLAY) *indicadas anteriormente. Formalmente, para cualquier adversario* A eficiente *contra la* seguridad *Multi-Stage existen algoritmos eficientes* B1, . . . , B8 *tal que*

$$\mathrm{Adv}^{\text{Multi-etapa,D}}_{\text{TLS1.3-PSK-0RTT,A}} \ddot{y}\ 8ns \cdot \Big(\mathrm{Adv}^{\text{AdvCOLL}}_{\text{H, B1 + np}} \ddot{y}\ddot{y}\ddot{y}\ddot{y}\ddot{y}\ddot{y} + 2 \cdot \mathrm{Adv}^{\text{dual-PRF-sec}}_{\text{HKDF.Extract,B2}} + \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expandir,B3}} + \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract,B4}} + \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expandir,B5}} + 2 \cdot \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand,B6}} + \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extracto,B7}} + \mathrm{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand,B8}} \Big)$$

donde ns es la cantidad máxima de sesiones, nu es la cantidad máxima de usuarios y np es la cantidad máxima de secretos previamente compartidos.

Para el protocolo de enlace TLS 1.3 PSK 0-RTT, la seguridad de varias etapas se deriva de la seguridad de la clave precompartida: todas las claves se derivan de un PSK secreto precompartido desconocido para el adversario (dado que el modo PSK no es secreto directo, es posible que PSK no sea dañado en la sesión probada) Como tal, las claves derivadas son indistinguibles de las aleatorias (según los supuestos de PRF en los pasos HKDF.Extract y HKDF.Expand) que son independientes, lo que permite revelar y probar las claves de sesión en diferentes etapas.

Prueba. Como antes, procedemos a través de una secuencia de juegos, acotando las diferencias entre juegos a través de una serie de suposiciones hasta que demostramos que la ventaja de A para ganar el juego final es 0.

Juego 0. Este es el juego Multi-Stage original, es decir,

$$\mathrm{Adv}^{\text{Multi-etapa,D}}_{\text{TLS1.3-PSK-0RTT,A}} = \mathrm{AdvG0}_{\text{TLS1.3-PSK-0RTT,A}}.$$

Juego 1. Restringimos A a una única consulta de prueba, lo que reduce su ventaja en un factor de *1/8ns como máximo.* Formalmente, construimos un adversario de A haciendo solo una única consulta de Prueba a través de un argumento híbrido, de manera análoga a la demostración del Teorema 5.2 en la página 27, detallada en el Apéndice A.

$$\mathrm{AdvG0}_{\text{TLS1.3-PSK-0RTT,A}} \ddot{y}\ 8ns \cdot \mathrm{AdvG1}_{\text{TLS1.3-PSK-0RTT,A}}.$$

De ahora en adelante, podemos referirnos a *la* etiqueta de sesión probada en la etapa i, y asumir que sabemos esto sesión con antelación.

Juego 2. En este juego, el retador aborta si dos sesiones honestas calculan el mismo hash valor para diferentes entradas en cualquier evaluación de la función hash H. Si ocurre este evento, esto puede ser utilizado para romper la resistencia a la colisión de H dejando una reducción B1 (con aproximadamente el mismo tiempo de ejecución como A) envía los dos valores de entrada distintos a H. Por lo tanto:

$$\mathrm{AdvG1}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} \ddot{y} \mathrm{AdvG2}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} + \mathrm{AdvCOLL}_{\mathrm{H,B1}}.$$

Juego 3. En este juego, el retador adivina el PSK secreto precompartido utilizado en la sesión probada, y aborta el juego si esa conjetura fue incorrecta. Esto reduce la ventaja de A por un factor de como máximo *1/np* siendo *np* el número máximo de secretos precompartidos registrados, así:

$$\mathrm{AdvG2}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} \ddot{y} np \cdot \mathrm{AdvG3}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A.}}$$

Juego 4. En este juego, reemplazamos las salidas de la función pseudoaleatoria HKDF.Extract en todas las evaluaciones utilizan el PSK secreto precompartido adivinado de la sesión probada como clave por valores aleatorios. Esto afecta la derivación del ES secreto temprano en cualquier sesión que utilice el mismo PSK compartido. Nosotros reemplazar la derivación de ES en tales sesiones con un valor aleatorio ESf ÿ$ {0, 1} $^{\ddot{y}}$. Podemos atar el diferencia que este paso introduce en la ventaja de A por la doble seguridad PRF de HKDF.Extract. Tenga en cuenta que cualquier adversario exitoso no puede emitir una consulta corrupta para revelar el PSK utilizado en el sesión probada, y por lo tanto el secreto precompartido es un valor desconocido y uniformemente aleatorio, y el la simulación es sonido. Por lo tanto:

$$\mathrm{AdvG3}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} \ddot{y} \mathrm{AdvG4}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} + \mathrm{Advdual\text{-}PRF\text{-}sec}_{\mathrm{HKDF.Extracto,B2}}.$$

Juego 5. En este juego reemplazamos la función pseudoaleatoria HKDF. Expandir en todas las evaluaciones utilizando el valor ESf sustituido en G4. Esto afecta la derivación del secreto temprano derivado dES, la clave de enlace BK, el secreto de tráfico temprano ETS y el secreto maestro de exportador temprano EEMS en cualquier sesión que use el mismo valor secreto temprano ESf debido a que la etapa se puede reproducir. reemplazamos la derivación de dES, BK, ETS y EEMS en tales sesiones con valores aleatorios $\mathrm{dES}_{\text{grano}}$, $\mathrm{BKg}_{\text{grano}}$, ETS, EEMS ÿ$ {0, 1} $^{\ddot{y}}$. Podemos acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por Game G4, ESf es un valor desconocido y uniformemente aleatorio, y este reemplazo es sólido. Por lo tanto:

$$\mathrm{AdvG4}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} \ddot{y} \mathrm{AdvG5}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT,A}} + \mathrm{AdvPRF\text{-}seg}_{\mathrm{HKDF.Expandir,B3}}.$$

En este punto, hemos reemplazado las claves de la etapa 1 y la etapa 2 (ETS). $^{\text{grano}}$ y EEMS ^, respectivamente). Notamos que si A emite una consulta Reveal (etiqueta, i) a una sesión etiqueta0 de modo que la sesión probada etiqueta.sidi = etiqueta0 .sidi, entonces A perdería el juego. Dado que estas etapas son reproducibles, puede haber varias sesiones de este tipo, de modo que *label.sidi* = label0 .sidi , sin embargo, si *alguna* de estas etapas se revela, A pierde el juego.

Juego 6. En este juego, reemplazamos la función pseudoaleatoria HKDF.Extract en todas las evaluaciones utilizando el valor $\mathrm{dES}_{\text{grano}}$ sustituido en G5. Esto afecta la derivación del secreto de apretón de manos HS en cualquier sesión utilizando el mismo valor secreto temprano derivado $\mathrm{dES}_{\text{grano}}$, como la derivación de HS no incluye adicional

entropía Reemplazamos la derivación de HS en tales sesiones con un valor aleatorio HSf ÿ$ {0, 1}^{ÿ}. Nosotros puede acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Extract. Tenga en cuenta que en el juego anterior, el es uniformemente aleatorio valor dES y la simulación son sólidos. Por lo tanto:

$$\mathrm{Adv}^{G5}_{TLS1.3\text{-}PSK\text{-}0RTT,A} \ ÿ \ \mathrm{Adv}^{G6}_{TLS1.3\text{-}PSK\text{-}0RTT,A} + \mathrm{Adv}^{PRF\text{-}seg}_{HKDF.Extracto,B4}.$$

Juego 7. En este juego reemplazamos la función pseudoaleatoria HKDF. Expandir en todas las evaluaciones usando el valor HSf reemplazado en G6. Esto afecta la derivación del secreto de tráfico de protocolo de enlace del cliente CHTS, el secreto de tráfico de protocolo de enlace del servidor SHTS en la sesión de destino y (si existe) su compañero coincidente y el secreto de apretón de manos derivado dHS en todas las sesiones que usan el mismo apretón de manos HSf secreto . Tenga en cuenta que para CHTS y SHTS, estos valores son distintos de cualquier otra sesión que use el mismo valor secreto de protocolo de enlace HSf,, ya que la evaluación también toma como entrada el valor hash H2 = H(CHkCPSKkSHkSPSK), donde CH y SH contienen los valores aleatorios de cliente y servidor *rc, rs* respectivamente, y por Game G2 excluimos las colisiones hash. Sin embargo, dHS se puede derivar en múltiples sesiones, ya que no incluye entropía adicional en su cálculo. Reemplazamos la derivación de CHTS, SHTS y dHS en dichas sesiones con valores aleatorios CHTS ^, SHTS ^, dHS ÿ$ {0, 1}^{ÿ}. Para asegurar la consistencia, reemplazamos las derivaciones de dHS con el dHS reemplazado muestreado por la primera sesión para evaluar HKDF. Expandir usando HSf . Podemos acotar la diferencia que este paso introduce en el ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por el juego anterior, HSf es un valor uniformemente aleatorio y el reemplazo es sólido. Por lo tanto:

$$\mathrm{Adv}^{G6}_{TLS1.3\text{-}PSK\text{-}0RTT,A} \ ÿ \ \mathrm{Adv}^{G7}_{TLS1.3\text{-}PSK\text{-}0RTT,A} + \mathrm{Adv}^{PRF\text{-}seg}_{HKDF.Expandir,B5}.$$

Juego 8. En este juego reemplazamos la función pseudoaleatoria HKDF. Expandir en todas las evaluaciones usando los valores CHTS ^, SHTS reemplazados en G7. Esto afecta la derivación del apretón de manos del cliente. clave de tráfico tkchs, y la clave de tráfico de protocolo de enlace del servidor tkshs en la sesión de destino y su coincidencia compañero. Reemplazamos la derivación de tkchs y tkshs con valores aleatorios tk]chs ÿ$ {0, 1}^{L} y tk]shs ÿ$ {0, 1}^{L}, donde *L* indica la suma de la longitud de la clave y la longitud de iv para el AEAD negociado esquema. Podemos acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de dos evaluaciones de las funciones pseudoaleatorias HKDF.Expand. Tenga en cuenta que por el juego anterior CHTS y SHTS son valores uniformemente aleatorios, y estos reemplazos son correctos. Por lo tanto:

$$\mathrm{Adv}^{G7}_{TLS1.3\text{-}PSK\text{-}0RTT,A} \ ÿ \ \mathrm{Adv}^{G8}_{TLS1.3\text{-}PSK\text{-}0RTT,A} + 2 \cdot \mathrm{Adv}^{PRF\text{-}seg}_{HKDF.Expandir,B6}.$$

Juego 9. En este juego, reemplazamos la función pseudoaleatoria HKDF.Extract en todas las evaluaciones del valor dHS reemplazado en G8. Esto afecta la derivación del MS secreto maestro en cualquier sesión usando el mismo secreto de apretón de manos derivado dHS. Reemplazamos la derivación de MS en dichas sesiones. con el valor aleatorio MS ÿ$ {0, 1}^{ÿ}. MS puede derivarse en varias sesiones, ya que no incluye entropía adicional en su cálculo. Podemos acotar la diferencia que introduce este paso en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Extract. Tenga en cuenta que por Juego G7, dHS es un valor uniformemente aleatorio y este reemplazo es sólido. Por lo tanto:

$$\mathrm{Adv}^{G8}_{TLS1.3\text{-}PSK\text{-}0RTT,A} \ ÿ \ \mathrm{Adv}^{G9}_{TLS1.3\text{-}PSK\text{-}0RTT,A} + \mathrm{Adv}^{PRF\text{-}seg}_{HKDF.Extracto,B7}.$$

Juego 10. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expand en todas las evaluaciones del valor MS reemplazado en G9 en la sesión objetivo y su sesión correspondiente. Esto afecta la derivación del secreto de tráfico de la aplicación cliente CATS, el secreto de tráfico de la aplicación del servidor SATS, el secreto maestro del exportador EMS y el secreto maestro de reanudación RMS. Para CATS, SATS y EMS, estas evaluaciones son distintas de cualquier otra sesión, ya que la evaluación de HKDF.Expand también toma como entrada H4 = H(CHkCPSKkSHkSPSKkSF), donde CH y SH contienen los valores aleatorios de cliente y servidor *rc* y *rs* respectivamente, y por Game G2 excluimos las colisiones hash. Para RMS, esta evaluación es distinta de cualquier otra sesión, ya que la evaluación de HKDF.Expand también toma como entrada H5 = H(CHkCPSKkSHkSPSKkSFkCF). Reemplazamos la derivación de CATS, SATS, EMS y RMS con valores aleatorios CATS ^, SATS ^, EMS], RMS ÿ$ {0, 1} paso que introduce la ventaja de A por el secreto de la función pseudoaleatoria HKDF.Expand. Podemos acotar la diferencia que esto

Tenga en cuenta que en el juego anterior, MS es un valor uniformemente aleatorio e independiente, y estos reemplazos son sólidos. Por lo tanto:

$$\mathrm{Adv}^{G9}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT},A} \le \mathrm{Adv}^{G10}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT},A} + \mathrm{Adv}^{\mathrm{PRF\text{-}seg}}_{\mathrm{HKDF.Expand},B8}.$$

En Game G10 ahora hemos reemplazado las teclas de todas las etapas en la sesión probada con uniformemente aleatorias valores independientes de la ejecución del protocolo, y así:

$$\mathrm{Adv}^{G10}_{\mathrm{TLS1.3\text{-}PSK\text{-}0RTT},A} = 0.$$

La combinación de los límites simples dados produce la declaración de seguridad general. □

6.2 TLS 1.3 PSK-(EC)DHE (0-RTT opcional)

Ahora podemos pasar a los resultados de seguridad para el protocolo de enlace TLS 1.3 PSK-(EC)DHE 0-RTT, comenzando de nuevo con Match security.

6.2.1 Seguridad del partido

Teorema 6.3 (Seguridad de coincidencia de TLS1.3-PSK-(EC)DHE-0RTT). *El protocolo de enlace TLS 1.3 PSK-(EC)DHE 0-RTT es Match-secure con las propiedades* (M, AUTH, FS,USE, REPLAY) *indicadas anteriormente. Para cualquier adversario eficiente* A *existe un algoritmo eficiente* B *tal que*

$$\mathrm{Adv}^{\mathrm{Match}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},A} \le \mathrm{Adv}^{\mathrm{COLLHMAC}}_{B} + |P| \cdot \frac{np^2}{q} + ns^2 \cdot \frac{1}{2^{|nonce|}},$$

donde ns es el número máximo de sesiones, q es el orden del grupo, np es el número máximo de secretos precompartidos, |P| *es el tamaño del espacio secreto precompartido y* |*nonce*| = 256 *es la longitud en bits de los nonces.*

Como antes, las propiedades de solidez de la seguridad de Match se derivan inmediatamente de nuestra definición de identificadores de sesión, con el límite de seguridad que surge como el límite de cumpleaños para dos sesiones honestas que eligen el mismo nonce y elemento de grupo. Por lo tanto, la demostración sigue de cerca la del teorema 6.1.

Prueba. Necesitamos mostrar las siete propiedades de la seguridad de Match (cf. Definición 4.1).

1. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen la misma clave en esa etapa.*
   Los identificadores de sesión en cada etapa incluyen tanto el identificador precompartido *pskid* como el Hellman comparte g^x y g^y Diffie– *y* (a través de los mensajes CPSK y SPSK, y CKS, SKS respectivamente),

arreglando tanto la entrada de clave precompartida PSK como la entrada de clave Diffie-Hellman DHE = *g xy* para todas las claves de etapa derivadas. Además, para cada etapa, el identificador de sesión incluye todos los mensajes de protocolo de enlace que ingresan a la derivación de clave: para las etapas 1 y 2 mensajes hasta CPSK, para las etapas 3 y 4 mensajes hasta SPSK, para las etapas 5, 6, 7 mensajes hasta SF , y para la etapa 8 todos los mensajes (hasta CF). En cada etapa, el identificador de sesión determina, por lo tanto, *todas las* entradas a la derivación de la clave, y el acuerdo sobre él asegura el acuerdo sobre la clave de etapa.

2. *Las sesiones con el mismo identificador de sesión para alguna etapa tienen funciones opuestas, excepto para posibles respondedores múltiples en etapas rejugables.*
   Suponiendo que, como máximo, dos sesiones comparten el mismo identificador de sesión (que mostramos a continuación), dos sesiones de iniciador (cliente) o respondedor (servidor) nunca tienen el mismo identificador de sesión, ya que nunca aceptan mensajes entrantes con roles incorrectos, y los mensajes de saludo iniciales son escrito con el rol del remitente. Esto excluye las etapas 1 y 2, que son etapas reproducibles en el protocolo de enlace TLS 1.3 PSK-(EC)DHE 0-RTT.

3. *Las sesiones con el mismo identificador de sesión para alguna etapa acuerdan la autenticación de esa etapa nivel.*
   Por definición, el vector que determina la autenticación (actualizable) se fija en ((1, 1), (2, 2), (5, 8), (5, 8), (5, 8), (6, 8), (7, 8), (8, 8)), con lo que todas las sesiones están de acuerdo, por lo tanto, trivialmente.

4. *Las sesiones con el mismo identificador de sesión para alguna etapa comparten el mismo identificador contributivo.*
   Esto se debe a que, para cada etapa i, el identificador contributivo cidi es final e igual a sidi una vez que se establece el identificador de sesión.

5. *Las sesiones se asocian con el participante previsto (autenticado) y comparten la misma clave identificador*
   Todos los identificadores de sesión incluyen los valores de pssid y *binder* enviados como parte de ClientHello.
   Por lo tanto, el psid se acuerda de manera trivial y determina de manera única un PSK secreto previamente compartido desde la perspectiva de cada par. El valor del *aglutinante* se deriva de ese PSK a través de una secuencia de cálculos de HKDF/HMAC. Si tratamos HMAC como una función hash resistente a colisiones sin clave sobre ambas entradas, la clave y el espacio del mensaje, el acuerdo sobre el *aglutinante* implica el acuerdo sobre PSK. Este paso es necesario, ya que A puede establecer varios valores de PSK para compartir el mismo psid y, por lo tanto, un psid no necesariamente determina de manera única un PSK secreto precompartido desde la perspectiva de cada par. En su lugar, usamos el *aglutinante* para determinar de forma única el acuerdo sobre PSK entre pares. Como todos los valores de PSK se eligen uniformemente al azar dentro de NewSecret p/|P|, donde P es el espacio secreto previamente compartido y *np* el número máximo consulta, colisionan solo con una probabilidad insignificante, limitada por el límite de cumpleaños n^2 de secretos previamente compartidos.
   Por lo tanto, el acuerdo sobre el *aglutinante* y PSK finalmente implica que pssid, tal como lo interpreta la sesión del servidor y el cliente asociados, se origina en la misma llamada NewSecret. Esto, desde la perspectiva tanto del cliente como del servidor, identifica de manera única la identidad del par respectivo y, por lo tanto, garantiza el acuerdo sobre los pares previstos.

6. *Los identificadores de sesión son distintos para diferentes etapas.*
   Esto es trivial, ya que el identificador de sesión de cada etapa tiene una etiqueta única.

7. *Como máximo, dos sesiones tienen el mismo identificador de sesión en cualquier etapa no reproducible.*
   Recuerda que las etapas 1 y 2 son rejugables, por lo que consideramos solo las etapas *i* ÿ {3, *4, 5, 6, 7,* 8}.
   Recuerde que todos los identificadores de sesión de estas etapas en poder de alguna sesión incluyen un nonce aleatorio de cliente y servidor y un recurso compartido Diffie-Hellman, ya que todos los identificadores de sesión contienen los mensajes ClientHello y ServerHello. Por lo tanto, para una triple colisión entre sesión

identificadores de partes honestas, alguna sesión necesitaría elegir el mismo nonce y elemento de grupo que otra sesión (que luego se puede asociar a través de un protocolo regular que se ejecuta en una tercera sesión). La probabilidad de que ocurra tal colisión se puede acotar desde arriba, donde *ns* es el número máximo de sesiones, *q* está limitado por el cumpleaños en el $\frac{2}{s} \cdot 1/q \cdot 2$ *ÿ|una vez|* , orden del grupo y | *nonce* | = 256 la longitud de bits de los nonces.

□

6.2.2 Seguridad de múltiples etapas

Teorema 6.4 (Seguridad multietapa de TLS1.3-PSK-(EC)DHE-0RTT). *El protocolo de enlace TLS 1.3 PSK-(EC)DHE 0-RTT es* seguro en varias etapas *con las propiedades* (M, AUTH, FS, USE, REPLAY) *indicadas anteriormente.*
Formalmente, para cualquier adversario A eficiente *contra la* seguridad *Multi-Stage existen algoritmos eficientes* B1, . . . , B16 *tal que*

$$\begin{aligned} \mathrm{Adv}^{\mathrm{MultietapaD}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} \leq 8ns \cdot \mathrm{Adv}^{\mathrm{COLL}}_{\mathrm{H,B1}} + npns \cdot \Big(& \mathrm{Adv}^{\mathrm{dual\text{-}PRF\text{-}sec}}_{\mathrm{HKDF.Extract,B2}} + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Expandir,B3}} \\ & + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Extract,B4}} + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Expandir,B5}} \\ & + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Expand,B6}} + \mathrm{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathrm{HMAC,B7}} \\ & + \mathrm{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathrm{HKDF.Extract,B8}} + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HMAC,B9}} \\ & + \mathrm{Adv}^{\mathrm{dual\text{-}PRF\text{-}sec}}_{\mathrm{HKDF.Expandir,B11}} \\ & + \mathrm{Adv}^{\mathrm{dual\text{-}snPRF\text{-}ODH}}_{\mathrm{HKDF.Extract,G,B12}} + \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Expandir,B13}} \\ & + 2 \cdot \mathrm{Adv}^{\mathrm{PRF\text{-}sec}}_{\mathrm{HKDF.Expand,B14}} + \mathrm{Adv}^{\mathrm{PRF\text{-}seg}}_{\mathrm{HKDF.Extracto,B15}} \\ & + \mathrm{Adv}^{\mathrm{PRF\text{-}seg}}_{\mathrm{HKDF.Expandir,B16}} \Big) \end{aligned}$$

donde ns es el número máximo de sesiones, np el número máximo de secretos precompartidos establecidos entre dos partes y nu es el número máximo de usuarios.

Para el protocolo de enlace TLS 1.3 PSK-(EC)DHE 0-RTT, la seguridad de varias etapas se deriva esencialmente de dos líneas de razonamiento. En primer lugar, las etiquetas MAC (infalsificables) que cubren (un hash resistente a colisiones de) los mensajes completos de Hello aseguran que las etapas de sesión con un intercambio de acciones entre pares autenticado retienen las acciones Diffie-Hellman intercambiadas que se originan en una sesión de socio honesta. Luego, todas las claves se derivan de manera que se asegure que (a) para las etapas de secreto directo, las claves se derivan de un secreto Diffie-Hellman desconocido para el adversario y son indistinguibles de las aleatorias (bajo PRF-ODH y dual-PRF-sec/ PRF-sec en los pasos HKDF.Extract y HKDF.Expand), y para las etapas sin secreto directo, las claves se derivan de un secreto precompartido desconocido para el adversario, y también son indistinguibles de las aleatorias (según las suposiciones de PRF en el pasos HKDF.Expand y HKDF.Extract) que (b) son independientes, lo que permite revelar y probar las claves de sesión en diferentes etapas.

Prueba. Nuevamente, procedemos a través de una secuencia de juegos que comienza en el juego de Etapas Múltiples y limita la ventaja (diferencias) del adversario A.

Juego 0. Este es el juego Multi-Stage original, es decir,

$$\mathrm{Adv}^{\mathrm{MultietapaD}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} = \mathrm{Adv}^{\mathrm{G0}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} .$$

Juego 1. Nuevamente restringimos A a una sola consulta de Prueba, reduciendo su ventaja por un factor de 1/8ns como máximo a través de un argumento híbrido análogo al de la demostración del Teorema 5.2 en la página 27, detallado en el Apéndice A.

$$\mathrm{Adv}^{\mathrm{G0}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} \leq 8ns \cdot \mathrm{Adv}^{\mathrm{G1}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} .$$

De ahora en adelante, podemos referirnos a *la* etiqueta de sesión probada en la etapa i, y asumir que conocemos la sesión de antemano.

Juego 2. En este juego, el retador aborta si dos sesiones honestas calculan el mismo valor hash para diferentes entradas en cualquier evaluación de la función hash H. Podemos romper la resistencia a la colisión de H en caso de este evento dejando una reducción B1 enviar los dos valores de entrada distintos a H. Por lo tanto:

$$\mathrm{Adv}^{\mathrm{G1}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} \ddot{y} \mathrm{Adv}^{\mathrm{G2}} + \mathrm{Adv}^{\mathrm{COLL}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}}{}_{H,B1}.$$

Desde este punto, nuestro análisis considera por separado los siguientes tres casos (disjuntos):

A. que la etiqueta de sesión probada no tiene un socio contribuyente honesto en la tercera etapa (es decir, no hay no existe label0 6= label with *label.cid3* = label0 .cid3), y,

B. la etiqueta de sesión probada tiene un socio contribuyente honesto en la tercera etapa (es decir, existe label0 con *label.cid3* = label0 .cid3) y A emite una consulta de prueba a las etapas que no son secretas (es decir, A emite Test(etiqueta, i) donde *i* ÿ {1, 2}.

C. la etiqueta de sesión probada tiene un socio contribuyente honesto en la tercera etapa (es decir, existe label0 con *label.cid3* = label0 .cid3) y A emite una consulta de prueba a las etapas de secreto directo (es decir, A emite Test(label, i) donde *i* ÿ {3, . . . , 8}.

Esto nos permite considerar la ventaja del adversario por separado para los casos A (indicado como "prueba sin socio"), B (indicado como "prueba NFS con socio") y C ("prueba FS con socio"):

$$\mathrm{Adv}^{\mathrm{G2}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} \ddot{y} \mathrm{Adv}^{\mathrm{G2}}_{\text{prueba sin pareja}} + \mathrm{Adv}^{\mathrm{G2}}_{\text{prueba NFS con socio}} + \mathrm{Adv}^{\mathrm{G2}}_{\text{prueba FS con socio}}$$
$$\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} \qquad \mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} \qquad \mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}.$$

Caso A. Prueba sin pareja

Como antes, primero consideramos el caso de que la etiqueta de sesión probada no tenga un socio contribuyente de etapa 3. Para las sesiones de iniciador probadas, esto significa que no existe ninguna sesión honesta que haya generado los mensajes SH, SKS y SPSK recibidos. Para la sesión de respuesta probada, esto significa que no existe una sesión de iniciador honesto que haya emitido los mensajes CH, CKS o CPSK recibidos. Dado que estos mensajes se incluyen en todos los identificadores de sesión de etapa subsiguientes, esto implica que la sesión probada no tiene un socio contribuyente en ninguna etapa. Por definición, un adversario no puede ganar si la consulta de prueba emitida para dicha sesión se encuentra en una etapa que, en el momento de la consulta de prueba, tiene un par no autenticado (donde la autenticación se refiere a la noción *rectificada).* Para una sesión de respondedor probada sin un socio contribuyente honesto en la etapa 3, solo se puede enviar una consulta de prueba a la sesión cuando llega a la etapa 8. Para una sesión de iniciador probada sin un socio contribuyente honesto en la etapa 3, solo se puede enviar una consulta de prueba emitido a la sesión cuando alcanza la etapa 5.

Juego A.0. Esto es idéntico al Juego G2 con el adversario restringido a probar una sesión sin un socio honesto que contribuya en la tercera etapa.

$$\mathrm{Adv}^{\mathrm{G2}}_{\text{prueba sin pareja}} = \mathrm{Adv}^{\mathrm{GA.0}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}}\ \mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}.$$

Juego A.1. En este juego, el retador adivina el PSK secreto precompartido que se usó en la sesión probada y aborta el juego si esa suposición fue incorrecta. Esto reduce la ventaja de A por un factor de como máximo *1/np* (para *np* el número máximo de secretos previamente compartidos), por lo tanto: ÿ *np* · *AdvGA.1*

$$\mathrm{Adv}^{\mathrm{GA.0}}_{\mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}} \qquad \mathrm{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}.$$

Juego A.2. En este juego, el retador aborta inmediatamente si el iniciador (respectivamente respondedor) de la sesión con etiqueta etiqueta acepta en la quinta (respectivamente octava) etapa sin un socio honesto que contribuya en la etapa 3. Deje que *abortGA.2,A* denote el evento que esto ocurre en *GA.2.* Por lo tanto:

$$\mathit{AdvGA.1} \ddot{y} \mathit{AdvGA.2}_{\text{TLS1.3-PSK-(EC)DHE-0RTT,A}} \ddot{y} \Pr[\text{abortarGA}_{\text{cuenta}}.2,A]$$

Tenga en cuenta que el caso A restringe a A a emitir una consulta de prueba a una sesión sin un socio contribuyente honesto en la etapa 3. Debido al tipo de autenticación de TLS1.3-PSK-(EC)DHE-0RTT, esta consulta de prueba solo se puede emitir a la sesión del iniciador (resp. respondedor) *después de* que alcance la etapa 5 (resp. 8). Dado que *GA.2* se aborta cuando la sesión llega a esas etapas, un adversario exitoso no puede emitir dicha consulta y, por lo tanto:

$$\mathit{AdvGA.2} = 0.$$

Ahora pasamos a acotar la probabilidad de que *abortGA.2,A* ocurre.

Juego A.3. En este juego, el retador adivina una sesión (de un máximo de *ns* sesiones en el juego) y aborta si la sesión adivinada no es la sesión del *primer* iniciador (resp. respondedor) que acepta en la quinta (resp. octava) etapa sin un honesto compañero contribuyente en la etapa 3. Si el retador adivina correctamente (lo que sucede con una probabilidad de al menos *1/ ns),* entonces este juego aborta exactamente al mismo tiempo que el juego anterior y, por lo tanto:

$$\Pr[\text{abortarGA}_{\text{cuenta}}.2,A] \ddot{y} \mathit{ns} \cdot \Pr[\text{abortarGA}_{\text{cuenta}}.3,A].$$

Restringimos a A para que no realice una consulta Corrupt(U, *V,* k) tal que label.id = U, label.pid = *V label.pssid* = , k, y mostramos que esto no afecta la ventaja de A para ganar este caso. Según la definición del caso, no existe una sesión label0 tal que label0 .cid3 = *label.cid3* donde se emite la consulta de prueba de A a la etiqueta. Dado que el modo PSK-(EC)DHE se autentica unilateralmente en la etapa 5 y se autentica mutuamente en la etapa 8, si el adversario emite una consulta corrupta (U, *V,* k) antes de la etiqueta de sesión probada (sin un socio contribuyente honesto en la etapa 3) llega a aceptar en la etapa de autenticación de su compañero, cuando A emite una consulta Test(label, i) (donde *i* ÿ {1, . . . , 8}) se establece la bandera perdida y A perderá el juego. Según los juegos anteriores, abortamos cuando la etiqueta de la sesión del iniciador (resp. la sesión del respondedor) alcanza la etapa 5 (resp. la etapa 8) sin un socio contribuyente honesto y, por lo tanto, A nunca emitirá una consulta corrupta (U, *V,* k). En los siguientes juegos, esto nos permitirá reemplazar el pss secreto precompartido en la sesión probada (y en todas las sesiones con el mismo valor de pss) sin que sea inconsistente o detectable con respecto a la consulta corrupta. En lo que sigue, sea *pssU,V,k* el secreto precompartido adivinado.

Juego A.4. En este juego, reemplazamos las salidas de la función pseudoaleatoria HKDF.Extract en todas las evaluaciones usando el secreto precompartido adivinado *pssU,V,k* de la sesión probada como clave por valores aleatorios. Esto afecta la derivación del ES secreto temprano en cualquier sesión que utilice el mismo PSK compartido. Reemplazamos la derivación de ES en tales sesiones con un valor aleatorio ESf ÿ$ {0, 1} puede acotar la diferencia que este paso introduce en la ventaja de A por la seguridad (dual) de la función pseudoaleatoria HKDF.Extract. Tenga en cuenta que cualquier adversario exitoso no puede emitir una consulta corrupta para revelar *pssU, V, k* utilizado en la sesión probada y, por lo tanto, el secreto precompartido es un valor desconocido y uniformemente aleatorio, y la simulación es sólida. Por lo tanto:

$$\Pr[\text{abortGA}_{\text{Juego3}},A] \ddot{y} \Pr[\text{abortGA.4}_{\text{cuenta}},A] + \text{Advdual-PRF-sec}_{\text{HKDF.Extracto,B2}}.$$

Juego A.5. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones utilizando el valor ESf sustituido en *GA.4.* Esto afecta la derivación del secreto temprano derivado dES, la clave de enlace BK, el secreto de tráfico temprano ETS y el secreto maestro de exportador temprano EEMS en cualquier sesión que use el mismo valor secreto temprano ESf debido a que la etapa se puede reproducir. reemplazamos la derivación de dES, BK, ETS y EEMS en tales sesiones con valores aleatorios dES , BKg, ETS , EEMS ÿ$ {0, 1} . Podemos acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por Game *GA.4,* ESf es un valor desconocido y uniformemente aleatorio, y este reemplazo es sólido. Por lo tanto:

$$\Pr[\text{abortar}^{GA.4}_{\text{cuenta}},A] \ddot{y} \Pr[\text{abortar}^{GA.5}_{\text{cuenta}},A] + \text{AdvPRF-seg}_{\text{HKDF.Expandir},B3}.$$

Juego A.6. En este juego, reemplazamos la función pseudoaleatoria HKDF.Extract en todas las evaluaciones utilizando el valor dES sustituido en *GA.5.* Esto afecta la derivación del secreto de apretón de manos HS en cualquier sesión que utilice el mismo valor secreto temprano derivado dES debido a que la etapa se puede reproducir. Nosotros reemplazar la derivación de HS en tales sesiones con un valor aleatorio HSf ÿ$ {0, 1} . Podemos atar la diferencia que este paso introduce en la ventaja de A por la seguridad de la pseudoaleatoria función HKDF. Extracto. Tenga en cuenta que por el juego anterior, dES es un valor uniformemente aleatorio, y la simulación es sólida. Por lo tanto:

$$\Pr[\text{abortar}^{GA.5}_{\text{cuenta}},A] \ddot{y} \Pr[\text{abortar}^{GA.6}_{\text{cuenta}},A] + \text{AdvPRF-seg}_{\text{HKDF.Extracto},B4}.$$

Juego A.7. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones usando el valor HSf reemplazado en *GA.6.* Esto afecta la derivación del secreto de tráfico del protocolo de enlace del cliente. CHTS, el secreto de tráfico de protocolo de enlace del servidor SHTS en la sesión de destino y su socio coincidente, y el secreto de protocolo de enlace derivado dHS en todas las sesiones que utilizan el mismo secreto de protocolo de enlace HSf . Tenga en cuenta que para CHTS y SHTS, estos valores son distintos de cualquier otra sesión que use el mismo protocolo de enlace valor secreto HSf ya que la evaluación también toma como entrada el valor hash H2 = H(CHkSH), (donde CH y SH contienen los valores aleatorios de cliente y servidor *rc, rs* respectivamente) y por Game G2 excluimos colisiones hash. Reemplazamos la derivación de CHTS, SHTS y dHS en tales sesiones con aleatorio valores CHTS ^, SHTS ^, dHS ÿ$ {0, 1} . Para garantizar la coherencia, reemplazamos las derivaciones de dHS con el dHS reemplazó muestreado por la primera sesión para evaluar HKDF.Expand usando HSf . Podemos atar la diferencia que este paso introduce en la ventaja de A por la seguridad del pseudoaleatorio función HKDF.Expandir. Tenga en cuenta que por el juego anterior, HSf es un valor uniformemente aleatorio, y el el reemplazo es sólido. Por lo tanto:

$$\Pr[\text{abortar}^{GA.6}_{\text{cuenta}},A] \ddot{y} \Pr[\text{abortar}^{GA.7}_{\text{cuenta}},A] + \text{AdvPRF-seg}_{\text{HKDF.Expandir},B5}.$$

Juego A.8. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones usando el secreto de tráfico de protocolo de enlace del cliente CHTS reemplazado en *GA.7.* Esto afecta la derivación de la clave de tráfico de protocolo de enlace del cliente tkchs, y el cliente finalizó la clave fkC en la sesión de destino. Nosotros reemplace la derivación de tkchs y fkC con valores aleatorios tk]chs ÿ$ {0, 1} L, fkgC ÿ$ {0, 1} , donde *L* indica la suma de la longitud de la clave y la longitud iv para el esquema AEAD negociado. Podemos atar la diferencia que este paso introduce en la ventaja de A por la seguridad de la pseudoaleatoria función HKDF.Expandir. Tenga en cuenta que en el juego anterior CHTS es un valor uniformemente aleatorio, y estos reemplazos son sólidos. Por lo tanto:

$$\Pr[\text{abortar}^{GA.7}_{\text{cuenta}},A] \ddot{y} \Pr[\text{abortar}^{GA.8}_{\text{cuenta}},A] + \text{AdvPRF-seg}_{\text{HKDF.Expandir},B6}.$$

cuenta

cuenta

cuenta

cuenta

cuenta

HMAC,B7

Tenga en cuenta que para el resto de este caso, limitamos la probabilidad de que un adversario active el *abortoGA.9,A*

cuenta

ÿ

cuenta

cuenta

cuenta

cuenta

cuenta

HMAC,B9

La combinación de los límites simples dados produce la siguiente declaración de seguridad:

$$Adv^{NFS}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) \le \left[Adv^{dual\text{-}PRF\text{-}sec}_{HKDF.Extract,B2} + Adv^{PRF\text{-}sec}_{HKDF.Expand,B3} + Adv^{PRF\text{-}sec}_{HKDF.Extract,B4} + Adv^{PRF\text{-}sec}_{HKDF.Expand,B5} + Adv^{PRF\text{-}sec}_{HKDF.Expand,B6} + Adv^{EUF\text{-}CMA}_{HMAC,B7} + Adv^{PRF\text{-}sec}_{HKDF.Expand,B8} + Adv^{EUF\text{-}CMA}_{HMAC,B9} \right]$$

AdvG2, prueba, sin pareja

Caso B. Prueba NFS con Partner

Ahora pasamos al caso en el que la sesión probada tiene un socio contribuyente honesto en la tercera etapa, y A emite una consulta Test(label, i) tal que *i* ÿ {1, 2}.

Juego B.0. Esto es idéntico al Juego G2 con el adversario probando una sesión con un compañero honesto que contribuye en la tercera etapa.

$$Adv^{G2}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) = Adv^{GB.0}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns).$$

Juego B.1. En este juego, adivinamos el PSK secreto precompartido utilizado en la sesión probada y abortamos en caso de una suposición incorrecta. Esto reduce la ventaja de A por un factor de como máximo *1/np,* por lo tanto:

$$Adv^{GB.0}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) \le np \cdot Adv^{GB.1}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns).$$

Juego B.2. En este juego, dejamos que el retador adivine una sesión (como máximo de *ns* en el juego) y aborte si la sesión adivinada no es el socio contribuyente honesto en la etapa 3 de la sesión probada. Esto reduce la ventaja de A por un factor de como máximo *1/ns,* y por lo tanto:

$$Adv^{GB.1}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) \le ns \cdot Adv^{GB.2}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns).$$

Juego B.3. En este juego, reemplazamos las salidas de la función pseudoaleatoria HKDF.Extract en todas las evaluaciones usando el secreto precompartido adivinado *pssU,V,k* de la sesión probada como clave por valores aleatorios. Esto afecta la derivación del ES secreto temprano en cualquier sesión que utilice el mismo PSK compartido. Reemplazamos la derivación de ES en tales sesiones con un valor aleatorio ESf ÿ$ {0, 1} $. Podemos limitar la diferencia que este paso introduce en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Extract. Tenga en cuenta que cualquier adversario exitoso no puede emitir una consulta corrupta para revelar *pssU, V, k* utilizada en la sesión probada (ya que A emitirá una consulta Prueba (etiqueta, i) tal que *i* ÿ {1, 2} según la definición de este caso , y A hará que se establezca el indicador perdido si se emite Corrupt(U, *V,* k)), y por lo tanto el secreto precompartido es un valor desconocido y uniformemente aleatorio, y la simulación es sólida. Por lo tanto:

$$Adv^{GB.2}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) \le Adv^{GB.3}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}(npns) + Adv^{dual\text{-}PRF\text{-}sec}_{HKDF.Extract,B10}.$$

Juego B.4. En este juego, reemplazamos la función pseudoaleatoria HKDF. Expandir en todas las evaluaciones usando el valor ESf reemplazado en *GB.3.* Esto afecta la derivación del secreto temprano derivado dES, la clave de enlace BK, el secreto de tráfico temprano ETS y el secreto maestro exportador temprano EEMS en cualquier sesión que utilice el mismo valor secreto temprano ESf debido a que la etapa se puede reproducir. Reemplazamos , BKg, ETS la derivación de dES, BK, ETS y EEMS en tales sesiones con valores aleatorios dES EEMS ÿ$ {0, 1} . Podemos acotar la diferencia que este paso introduce en la ventaja de A

por la seguridad de la función pseudoaleatoria HKDF.Expand. Tenga en cuenta que por Game *GB.3,* ESf es un valor desconocido y uniformemente aleatorio, y este reemplazo es sólido. Por lo tanto:

$$Adv^{GB.3}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} ÿ Adv^{GB.4}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} + Adv^{PRF\text{-}sec}_{HKDF.Expandir,B11} .$$

observamos que en este punto, hemos reemplazado las teclas de etapa 1 y etapa 2 (ETS respectivamente), y EEMS^, Notamos que si A emite una consulta Reveal(label, i) a una sesión label0 tal que la sesión probada *label.sidi* = label0 .sidi entonces A perdería el juego.,Dado que estas etapas son rejugables, entonces puede haber varias sesiones de este tipo, de modo que *label.sidi* = label0 .sidi . Dado que ETS y EEMS^ ahora son valores uniformemente aleatorios independientes de la ejecución del protocolo, tenemos:

$$Adv^{GB.4}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} = 0.$$

La combinación de los límites simples dados produce la siguiente declaración de seguridad:

$$Adv^{G2,\ prueba\ NFS\ con\ socio}_{(EC)DHE\text{-}0RTT,A} Adv^{dual\text{-}PRF\text{-}sec}_{HKDF.Extract,B10} + Adv^{PRF\text{-}sec}_{HKDF.Expandir,B11}\ ÿ\ nsnp\ TLS1.3\text{-}PSK$$

Caso C. Prueba FS con pareja

Pasamos ahora al tercer caso, "Prueba FS con socio", donde la sesión probada tiene un socio contribuyente honesto en la tercera etapa, y A emite una consulta Prueba (etiqueta, i) tal que *i* ÿ {3, . . . , 8}.

Juego C.0. Esto es idéntico al Juego G2 con el adversario probando una sesión con un compañero honesto que contribuye en la tercera etapa.

$$Adv^{G2,\ prueba\ FS\ con\ socio}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} = Adv^{GC.0}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} .$$

Juego C.1. En este juego, dejamos que el retador adivine una sesión (como máximo de *ns* en el juego) y aborte si la sesión adivinada no es el socio contribuyente honesto en la etapa 3 de la sesión probada. Esto reduce la ventaja de A por un factor de *1/ ns* como máximo y, por lo tanto:

$$Adv^{GC.0}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}\ ÿ\ ns \cdot Adv^{GC.1}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} .$$

Juego C.2. En este juego, reemplazamos el secreto del apretón de manos HS derivado en la sesión probada y su sesión asociada contributiva con una cadena uniformemente aleatoria e independiente HSf ÿ$ {0, 1} ÿ Nosotros empleamos la suposición dual-snPRF-ODH para poder simular el cálculo de HS en una sesión de cliente asociado para un mensaje ServerKeyShare modificado. Más precisamente, podemos convertir a cualquier adversario capaz de distinguir este cambio en un adversario B12 contra la seguridad dual-snPRF-ODH de la función HKDF.Extract (tomando dES como primera entrada y DHE como segunda entrada). Para esto, B12 solicita un desafío PRF en dES. Utiliza los recursos compartidos de Diffie-Hellman obtenidos *g* ClientKeyShare y ServerKeyShare g^x y g^y dentro de la sesión probada y su sesión de socio contributiva, y el valor de desafío PRF como HS en la sesión de prueba. Si es necesario, B12 usa sus consultas PRF-ODH para derivar HS en la sesión asociada en *g* diferentes . Proporcionar una simulación de sonido de *GC.1* (si el bit muestreado por el retador de dual-snPRF-ODH era 0 y, por lo tanto, HS = PRF(dES, *gxy*)) o *GC.2* (si el bit muestreado por el dual-snPRF-ODH) El retador de ODH era 1 y, por lo tanto, HSf ÿ$ {0, 1} ÿ), esta limita la diferencia de ventaja de A como:

$$Adv^{GC.1}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A}\ ÿ\ Adv^{GC.2}_{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT,A} + Adv^{dual\text{-}snPRF\text{-}ODH}_{HKDF.Extracto,G,B12} .$$

Juego C.3. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones utilizando el valor HSf sustituido en *GC.2.* Esto afecta la derivación del secreto de tráfico del protocolo de enlace del cliente. CHTS, el secreto de tráfico de protocolo de enlace del servidor SHTS en la sesión de destino y su socio coincidente, y el secreto de protocolo de enlace derivado dHS en todas las sesiones que utilizan el mismo secreto de protocolo de enlace HSf . Tenga en cuenta que para CHTS y SHTS, estos valores son distintos de cualquier otra sesión que use el mismo protocolo de enlace valor secreto HSf ya que la evaluación también toma como entrada el valor hash H2 = H(CHkSH), (donde CH y SH contienen los valores aleatorios de cliente y servidor *rc, rs* respectivamente) y por Game G2 excluimos colisiones hash. Reemplazamos la derivación de CHTS, SHTS y dHS en tales sesiones con aleatorio valores CHTS ^, SHTS ^, dHS ÿ$ {0, 1} ^{ÿ} . Para garantizar la coherencia, reemplazamos las derivaciones de dHS con el dHS reemplazó muestreado por la primera sesión para evaluar HKDF.Expand usando HSf . Podemos atar la diferencia que este paso introduce en la ventaja de A por la seguridad del pseudoaleatorio función HKDF.Expandir. Tenga en cuenta que por el juego anterior, HSf es un valor uniformemente aleatorio, y el el reemplazo es sólido. Por lo tanto:

$$Adv^{GC.2}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} \ddot{y} Adv^{GC.3}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} + AdvPRF\text{-}seg_{HKDF.Expandir,B13} .$$

En este punto, CHTS y SHTS son independientes de cualquier valor calculado en cualquier sesión no asociada (en la etapa 1 o 2) con la sesión probada: identificadores de sesión distintos y sin colisiones de hash (a partir del Juego G2) asegúrese de que las entradas de la etiqueta PRF para derivar CHTS y SHTS sean únicas.

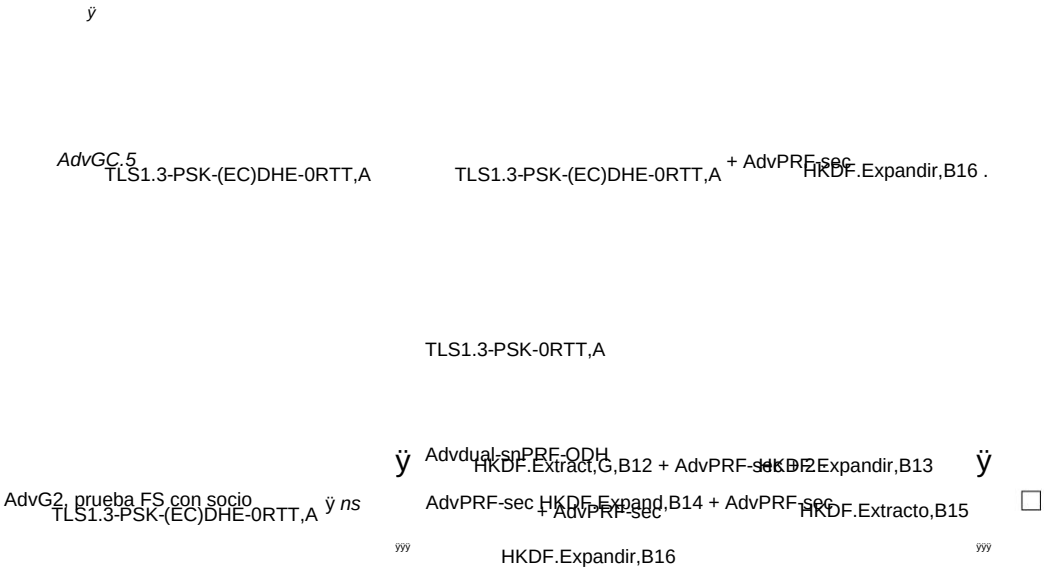
Juego C.4. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones usando los valores CHTS ^, SHTS reemplazados en *GC.3.* Esto afecta la derivación del apretón de manos del cliente. clave de tráfico tkchs, y la clave de tráfico de protocolo de enlace del servidor tkshs en la sesión de destino y su coincidencia compañero. En la derivación, reemplazamos tkchs y tkshs con valores aleatorios tk]chs ÿ$ $\{0, 1\}^{L}$ y tk]shs ÿ$ {0, 1} L, donde *L* indica la suma de la longitud de la clave y la longitud de iv para el AEAD negociado esquema. Podemos acotar la diferencia que este paso introduce en la ventaja de A por la seguridad de dos evaluaciones de las funciones pseudoaleatorias HKDF.Expand. Tenga en cuenta que por el juego anterior CHTS y SHTS son valores uniformemente aleatorios, y estos reemplazos son correctos. Por lo tanto:

$$Adv^{GC.3}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} \ddot{y} Adv^{GC.4}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} + 2 \cdot AdvPRF\text{-}seg_{HKDF.Expandir,B14} .$$

Juego C.5. En este juego, reemplazamos la función pseudoaleatoria HKDF.Extract en todas las evaluaciones del valor dHS sustituido en el Juego *GC.4.* Esto afecta la derivación del secreto maestro MS en cualquier sesión que utilice el mismo secreto de protocolo de enlace derivado dHS. Reemplazamos la derivación de MS en tales sesiones con el valor aleatorio MS ÿ$ {0, 1} ^{ÿ} . Podemos acotar la diferencia que este paso introducen en la ventaja de A por la seguridad de la función pseudoaleatoria HKDF.Extract. Nota que por el Juego *GC.3,* dHS es un valor uniformemente aleatorio y este reemplazo es correcto. Por lo tanto:

$$Adv^{GC.4}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} \ddot{y} Adv^{GC.5}_{TLS1.3\text{-}PSK\text{-}(CE)DHE\text{-}0RTT,A} + AdvPRF\text{-}seg_{HKDF.Extracto,B15} .$$

Juego C.6. En este juego, reemplazamos la función pseudoaleatoria HKDF.Expandir en todas las evaluaciones del valor MS reemplazado en *GC.5* en la sesión de destino y su sesión coincidente. Esto afecta el derivación del secreto de tráfico de la aplicación cliente CATS, el secreto de tráfico de la aplicación servidor SATS el el secreto maestro del exportador EMS y el secreto maestro de reanudación RMS. Para CATS, SATS y EMS, estas evaluaciones son distintas de cualquier otra sesión, ya que la evaluación de HKDF.Expand también toma como entrada H4 = H(CHkSHkSF) (donde CH y SH contienen los valores aleatorios de cliente y servidor *rc* y *rs*



7 Discusión y Conclusiones

7.1 Diferencias técnicas con nuestro trabajo anterior

apretón de manos: secreto de tráfico temprano ETS; exportador maestro secreto EEMS temprano; claves de tráfico de protocolo de enlace tkchs y tkshs; secretos de tráfico de aplicaciones CATS y SATS; y EMS y RMS. Una vez más, la razón principal de las etapas adicionales en este documento es el cambio antes mencionado en el cronograma clave.

Identificadores de sesión. En el protocolo de enlace principal, los identificadores de sesión para las claves de tráfico del protocolo de enlace son los mismos en [DFGS15, DFGS16] y en este documento. Para las claves de aplicación, los identificadores de sesión cambiaron en función de los cambios en el flujo de mensajes que provocaron cambios en la transcripción incluida en el hash de sesión utilizado para la derivación de claves. En particular, draft-05-(EC)DHE y draft-10-(EC)DHE incluyeron ClientCertificate en los identificadores de sesión clave de la aplicación pero no ServerFinished, mientras que TLS 1.3 analizado en este documento no incluye CCRT pero sí SF. De manera similar, los identificadores de sesión para los protocolos de enlace de PSK cambiaron en los documentos debido a cambios en el orden de los mensajes y qué mensajes estaban disponibles para incluirse en el hash de la sesión.

Supuestos criptográficos: protocolo de enlace principal. Las suposiciones criptográficas utilizadas en las pruebas para draft-05-(EC)DHE, draft-dh, draft-10-(EC)DHE y este documento siguen siendo las mismas. (En los primeros artículos usamos la notación PRF-ODH en lugar de la nueva notación snPRF-ODH introducida por [BFGJ17], pero la suposición real era la misma).

Supuestos criptográficos: protocolo de enlace PSK. En el borrador 05, no estaba presente ninguna variante (EC)DHE del protocolo de enlace PSK (entonces llamado "intercambio de protocolo de reanudación de sesión"), por lo que la prueba del borrador 05-SR se basó únicamente en suposiciones de clave simétrica. draft-10-PSK se basó en las mismas suposiciones que draft-05-SR, mientras que draft-10-PSK-(EC)DHE agregó una suposición EUF-CMA en HMAC, así como la suposición PRF-ODH. draft-14-PSK-0RTT y draft-14-PSK-(EC)DHE-0RTT agregaron una suposición de aleatoriedad en HMAC, que en el análisis de los RFC finales TLS1.3-PSK-0RTT y TLS1.3-PSK- (EC)DHE-0RTT en este documento se reemplaza por una suposición de doble PRF-sec en HKDF.Extract en los límites de seguridad de múltiples etapas. Este último indica de manera más explícita aquellos lugares donde HKDF.Extract está tecleado a través del segundo argumento, que se trataron de manera más implícita en las declaraciones de teoremas de versiones anteriores.

Match-security de TLS1.3-PSK-0RTT y TLS1.3-PSK-(EC)DHE-0RTT en este documento agrega una suposición de resistencia a la colisión en HMAC debido a la introducción del aglutinante PSK.

7.2 Comentarios sobre el diseño de TLS 1.3

Valor de la separación de claves. Las versiones anteriores de TLS usaban la misma clave de sesión para cifrar los datos de la aplicación, así como los mensajes Finalizados al final del protocolo de enlace. Esto hizo imposible demostrar que la clave de sesión TLS cumplía con el estándar de seguridad de indistinguibilidad de clave de estilo Bellare-Rogaway [BR94] , como se indica en [JK02, MSW08, Gaj08], lo que motivó el análisis combinado de protocolo de enlace + capa de registro en el protocolo autenticado y confidencial. modelo de establecimiento de canal de [JKSS12]. Confirmamos que el cambio en las claves para el cifrado de mensajes de protocolo de enlace permite que las claves establecidas durante el protocolo de enlace TLS 1.3 logren la seguridad estándar de indistinguibilidad de claves.

Independencia clave. Todas las formas del protocolo de enlace TLS 1.3 logran la independencia de clave para todas las claves de etapa: uno puede revelar la clave de sesión de una etapa sin poner en peligro la seguridad de las claves de etapa posterior. Esto se deriva del hecho de que cada clave exportada o utilizada para el cifrado es un nodo de hoja en el gráfico dirigido que representa el programa de claves en la Figura 2. Más allá de permitir una composición genérica, la independencia de claves protege el uso de claves derivadas contra los efectos entre protocolos. de averías de seguridad. (Algunos borradores iniciales tenían menos independencia clave: por ejemplo, en el borrador-05, cada

la clave exportada se derivó directamente del maestro secreto MS. Dado que MS también se utilizó para derivar otras claves, no podía considerarse como una clave de etapa de salida, por lo que cada clave exportada debía incluirse directamente en el análisis principal. Compare esto con el enfoque final en el que un EMS secreto maestro exportador se deriva de MS, y luego todas las claves exportadas se derivan de EMS: podemos tratar EMS como una clave de etapa de salida y considerar la derivación de claves exportadas como un protocolo simétrico usando EMS que se compone con el protocolo de protocolo de enlace TLS 1.3.)

Una "abolladura" en el cronograma clave. En cuanto a la derivación de claves, destacamos que hay una nota digna de "abolladura" en el programa de claves de TLS 1.3 (cf. Figura 2): todos los secretos de segundo nivel derivados de los secretos principales (temprano/intercambio/principal) se utilizan únicamente para derivar claves de cifrado de tráfico (en el caso de secretos de tráfico) u otros fines (reanudación y exportación), *excepto* los secretos de tráfico de protocolo de enlace CHTS/SHTS que, además de derivar las claves de tráfico de protocolo de enlace, también se utilizan para calcular las claves terminadas. Esto nos permitió definir todos, excepto los secretos de tráfico de protocolo de enlace, como claves de sesión de salida en el sentido de intercambio de claves de varias etapas, al tiempo que requería descender un nivel más para capturar las claves de tráfico de protocolo de enlace.

Un programa de claves más uniforme podría haber derivado las claves terminadas en una rama separada del HS secreto de protocolo de enlace, lo que permitiría que CHTS/SHTS se convirtieran en claves de sesión de primer orden en el mismo nivel que todas las demás. Esto, a su vez, permitiría una interfaz más uniforme para la composición con un protocolo de clave simétrica arbitraria y posiblemente respaldaría mejor el tratamiento de las actualizaciones de clave (cf. [GM17]).
Si bien este es solo un problema menor para el análisis de TLS 1.3, resultó complicar un análisis modular de la integración del protocolo de enlace TLS 1.3 en el protocolo QUIC [TT20] , como lo señaló Delignat-Lavaud et al. [DFP+20].

Incluyendo el hash de sesión en firmas y derivación de claves. En el protocolo de enlace completo TLS 1.3, las partes autenticadoras (el servidor y, a veces, el cliente) firman (el hash de) todos los mensajes de protocolo de enlace hasta el momento en que se emite la firma (el "hash de sesión"). Esto es diferente de TLS 1.2 y versiones anteriores, donde la firma del servidor solo se encuentra sobre los nonces aleatorios del cliente y el servidor y la clave pública efímera del servidor.

En cuanto a la derivación de claves, cada clave de etapa se deriva utilizando una aplicación PRF que incluye el hash de todos los mensajes intercambiados hasta el momento en que se deriva la clave de etapa.

En nuestro análisis, el identificador de sesión para cada etapa está configurado para ser la transcripción de los mensajes hasta ese punto. Por lo tanto, asumiendo la resistencia a la colisión de la función hash, diferentes identificadores de sesión dan como resultado diferentes claves. (Este fue el objetivo del hash de sesión que se introdujo en respuesta al ataque de triple protocolo de enlace [BDF+14] en TLS 1.2 y versiones anteriores). El servidor que firma la transcripción también facilita nuestras pruebas de las propiedades de autenticación en el protocolo de enlace completo.

Además, si las claves de salida están destinadas a usarse como un identificador de canal o para el enlace de canal (con el fin de aprovechar las propiedades de autenticación y protección de sesión establecidas por TLS en un protocolo de capa de aplicación), es apropiado incluir el hash de sesión. Si bien los métodos de enlace de canal TLS tls-unique [AWZ10] estandarizado y tls-unique-prf [Jos15] propuesto no usan claves directamente para el enlace, el bajo costo de incluir el hash de sesión parece valer la pena en caso de que un desarrollador de aplicaciones decida usar material clave directamente para encuadernación.

En el protocolo de enlace de PSK sin (EC)DHE, no hay un secreto compartido efímero y el secreto principal se calcula como una serie de cálculos de HKDF. Extraiga sobre una cadena 0 utilizando la clave precompartida como clave. Todas las sesiones que comparten el mismo secreto precompartido calculan el mismo secreto maestro. Sin embargo, dado que la derivación de las claves de salida aún usa el hash de la sesión como contexto, las claves de salida son únicas asumiendo la unicidad de los mensajes de protocolo (que se asegura, por ejemplo, mediante nonces únicos).

Cifrado de mensajes de protocolo de enlace. Un objetivo de diseño importante de TLS 1.3 era mejorar la privacidad (frente a adversarios pasivos) mediante el cifrado de la segunda parte del protocolo de enlace (que contiene certificados de identidad) mediante las claves de tráfico del protocolo de enlace inicial tkchs y tkshs. Nuestro análisis muestra que las claves de tráfico de protocolo de enlace sí tienen seguridad contra adversarios pasivos (e incluso adversarios activos cuando se usa la clave de tráfico de protocolo de enlace de cliente tkchs) y, por lo tanto, esta función de TLS 1.3 aumenta la privacidad del protocolo de enlace. Sin embargo, el secreto de las claves de etapa restantes no depende de que el apretón de manos esté encriptado y permanecería seguro incluso si el apretón de manos se hiciera en claro.

Mensajes terminados. Los mensajes Finalizados enviados por el cliente y el servidor al final del protocolo de enlace TLS 1.3 son valores MAC calculados mediante la aplicación de HMAC a la transcripción del protocolo de enlace (hash de la), codificada por secretos de finalización de cliente/servidor dedicados fkC/fkS.

Curiosamente, según nuestras pruebas, los mensajes Finalizados no contribuyen a la autenticación y confidencialidad implícitas de las claves de salida en el protocolo de enlace completo o en el protocolo de enlace de solo PSK, en el sentido de que el intercambio de claves lograría la misma noción de seguridad sin estos mensajes. . Esto se debe principalmente a que, en el protocolo de enlace completo, las firmas ya autentican las transcripciones y, en el protocolo de enlace de solo PSK, todas las claves se derivan del PSK, que proporciona autenticación implícita. Si bien los mensajes Finalizados no son necesarios para proporcionar autenticación implícita en protocolos de enlace de solo PSK, jugarían un papel en proporcionar autenticación explícita, pero nuestro modelo no incluye una propiedad de autenticación explícita. En el protocolo de enlace PSK-(EC)DHE, los mensajes Finalizados contribuyen a la autenticación de las claves públicas efímeras Diffie-Hellman bajo (una clave derivada de) el PSK. Los mensajes Finalizados todavía pueden interpretarse generalmente como que proporcionan algún tipo de confirmación y autenticación de clave de sesión (explícita) [FGSW16, Gün18, dFW19].

Compárelos con el caso del transporte de claves RSA en el protocolo de enlace completo TLS 1.2: los análisis de Krawczyk et al. [KPW13] y Bhargavan et al. [BFK+14] tenga en cuenta las debilidades potenciales o requiera suposiciones de seguridad más sólidas si se omiten los mensajes Finalizados.

Hashing ascendente en firmas, MAC y derivación de claves. Al firmar (resp. MAC-ing) la transcripción para la autenticación, así como al derivar claves a través de HKDF, TLS 1.3 usa el *hash* de la transcripción actual como entrada; si, por ejemplo, el algoritmo de firma es un algoritmo hash-then-sign, entonces realizará un hash adicional. Desde un punto de vista criptográfico, sería preferible insertar la transcripción completa (sin cifrar) y dejar que los respectivos algoritmos de firma, MAC o KDF se encarguen de procesar este mensaje de forma opaca. Sin embargo, para fines de ingeniería, puede ser conveniente codificar la transcripción de forma iterativa, almacenando solo los valores intermedios en lugar de la transcripción completa.
En nuestra prueba de seguridad, este hashing ascendente introduce la suposición de resistencia a la colisión para la función hash (y, por lo tanto, una posible fuente adicional de debilidades, cf. [BFG19a]), que de otro modo sería atendida por la firma, MAC, resp. esquema KDF.

Repeticiones 0-RTT y confidencialidad hacia adelante. A través de nuestro análisis, capturamos los efectos de las repeticiones en el sentido de la seguridad criptográfica, y lo que es más importante, confirmamos que la capacidad de reproducción de las claves 0-RTT no tiene efectos negativos en la seguridad criptográfica de las claves derivadas posteriormente. Sin embargo, desde una perspectiva práctica de la capa de aplicación, el potencial de las repeticiones 0-RTT sigue siendo una elección de diseño fundamental en TLS 1.3 y ha sido objeto de un debate controvertido (consulte, por ejemplo, [Mac17]). El estándar TLS 1.3 [Res18, Sección 8] reconoce que "TLS no proporciona protecciones de reproducción inherentes para datos 0-RTT", y al mismo tiempo insta a las implementaciones a implementar al menos un cierto nivel básico de protección anti-reproducción (como un solo -utilizar tickets de sesión, grabación de ClientHello o comprobaciones de actualización). Los modos 0-RTT del protocolo QUIC de Google y TLS 1.3 generaron una

de tratamientos académicos de intercambio de claves 0-RTT [FG14, LJBN15, HJLS17] y nuevos diseños de cifrado seguro hacia adelante [CHK03, GM15] para lograr un intercambio de claves 0-RTT secreto hacia adelante y no reproducible [GHJL17, DJSS18] y TLS reanudación de la sesión [AGJ21].

Además, desde una perspectiva criptográfica, la variante del modo 0-RTT basada en Diffie-Hellman ofrecía un mayor nivel de seguridad (directa) ya que no requería que el cliente mantuviera el estado secreto para la reanudación y, por lo tanto, solo los compromisos del servidor afectarían el secreto. de comunicación 0-RTT [Kra16a, FG17]. Esta variante de apretón de manos se abandonó con el borrador 13 en favor del rendimiento y la simplificación estructural.

7.3 Preguntas abiertas de investigación

Composición. Los protocolos de intercambio de claves serían de uso limitado si se aplicaran de forma aislada; en general, las claves derivadas están destinadas a implementarse en un protocolo de seguimiento (o general). El cifrado (y la autenticación) de los datos de la aplicación a través de un protocolo de canal (criptográfico) es, por supuesto, un enfoque común, siendo el protocolo de registro TLS un excelente ejemplo, pero otro uso en la configuración de TLS incluye la exportación de material clave o la reanudación de los protocolos de enlace (a través de la exportador o secreto maestro de reanudación).

Los protocolos de intercambio de claves seguros en el sentido de Bellare-Rogaway [BR94] son de hecho susceptibles de una composición segura genérica con protocolos simétricos de seguimiento arbitrarios, como lo muestran Brzuska et al. [BFWW11, Brz13]. Las versiones anteriores de nuestro trabajo [DFGS15, Gün18] incluyeron adaptaciones de estos resultados de composición a la configuración de múltiples etapas, lo que demuestra que las claves de etapa se pueden usar de manera segura en protocolos de clave simétrica. Esos resultados aún se aplican a nuestro modelo actual, cuando se restringen a claves de etapa que están marcadas para uso externo, no se pueden reproducir y cuando se trata la característica de autenticación como fija en el momento de la aceptación, no actualizable. Sin embargo, no es obvio cómo traducir las nociones de autenticación actualizable o capacidad de reproducción genéricamente a un protocolo de clave simétrica. Dado que nuestro enfoque está en el protocolo de protocolo de enlace TLS 1.3 como un protocolo de intercambio de claves autenticado, dejamos un resultado de composición que traduce la capacidad de reproducción y la autenticación actualizable para trabajos futuros.

Como parte de un tratamiento compuesto del protocolo TLS 1.3 general (es decir, protocolo de enlace y capa de registro), una alternativa conceptual a nuestro tratamiento del protocolo de enlace podría ser considerar *todas* las claves, incluidas las claves de tráfico del protocolo de enlace, como externas (desde la perspectiva del protocolo de enlace).) y confiar en el protocolo de registro para el cifrado de protocolo de enlace. Este punto de vista se adopta especialmente en los análisis basados en implementaciones verificadas [DFK+17] y, en el entorno computacional, requeriría una combinación adecuada de modelos de canal que capturen la naturaleza bidireccional, multiclave, multiplexada y de transmisión del protocolo de registro TLS 1.3. [FGMP15, MP17, BH17, GM17, PS18].

Intercambio de claves poscuánticas. Si bien nuestros teoremas son en su mayoría genéricos en términos de supuestos criptográficos, se basan directamente en un supuesto de Diffie-Hellman en un grupo. Sin embargo, el intercambio de claves poscuántico suele formularse de forma genérica como un mecanismo de encapsulación de claves (KEM). Si se va a ampliar TLS 1.3 para admitir el intercambio de claves poscuántico o híbrido (es decir, tradicional más poscuántico) [CPS19], será necesario revisar nuestros resultados en los modos completos 1-RTT y PSK-(EC)DHE en el contexto de KEM poscuánticos específicos o propiedades genéricas de KEM. Dado que confiamos en la suposición PRF-ODH [BFGJ17], una suposición interactiva que proporciona cierta noción de "seguridad activa", puede darse el caso de que traducir nuestras pruebas a la configuración KEM requiera el uso de un KEM IND-CCA. Brendel et al. [BFG+19b] analiza los desafíos que surgen al trasladar los intercambios de claves al estilo Diffie-Hellman al entorno poscuántico y Schwabe et al. [SSW20] presenta una alternativa basada en KEM al protocolo de enlace TLS 1.3 con un flujo de mensajes modificado.

7.4 Conclusiones

En este trabajo, hemos actualizado nuestros análisis previos de la seguridad criptográfica de varios protocolos de enlace TLS 1.3 preliminares a la versión final estandarizada de TLS 1.3 en RFC 8446 [Res18]. Analizamos el modo de protocolo de enlace 1-RTT completo, así como los modos de protocolo de enlace de reanudación basados en PSK, con claves 0-RTT opcionales, en el marco reduccionista de un modelo mejorado de seguridad de intercambio de claves de varias etapas que captura las diversas propiedades de seguridad de varios claves derivadas en TLS 1.3. Nuestro análisis confirmó que el protocolo de enlace TLS 1.3 sigue principios de diseño criptográfico sólidos y establece claves de sesión con las propiedades de seguridad deseadas bajo suposiciones criptográficas estándar.

El grupo de trabajo de TLS del IETF desarrolló TLS 1.3 a través de un proceso de estandarización novedoso y proactivamente transparente (cf. [PvdM16]) que solicitó activamente tanto a la industria como a la academia. En nuestra opinión, esto ha llevado a un éxito sin precedentes al contar con análisis de seguridad de amplio alcance para un importante protocolo de seguridad de Internet *antes* de su estandarización e implementación. Si bien los modelos de seguridad o las herramientas de métodos formales nunca pueden capturar la totalidad de las amenazas del mundo real a dichos protocolos, creemos que, a través de este proceso, los límites de la comprensión formal se han ampliado hasta el punto de fortalecer significativamente la confianza en la solidez de TLS 1.3 diseño de Como tal, el proceso de estandarización de TLS 1.3 ejemplifica un paradigma encomiable que legítimamente se está adoptando para los procesos de estandarización de otros protocolos importantes de seguridad de Internet, y que alentamos a otros organismos de estándares a adoptar.

Expresiones de gratitud

Agradecemos a Markulf Kohlweiss por los debates perspicaces sobre la necesidad de la suposición PRF-ODH para las pruebas de los protocolos de enlace TLS 1.3. Agradecemos a Håkon Jacobsen por los comentarios sobre la prueba del apretón de manos de clave precompartida, a Joseph Jaeger y Damiano Abram por los comentarios sobre la formalización de la seguridad de Match y a Denis Diemert por los comentarios sobre las cifras del protocolo de apretón de manos. También agradecemos a los revisores de esta y versiones anteriores de este trabajo por sus valiosos comentarios. Benjamin Dowling recibió el apoyo de la subvención EPSRC EP/L018543/1. El trabajo de Marc Fischlin ha sido financiado en parte por el Ministerio Federal de Educación e Investigación de Alemania y el Ministerio de Educación Superior, Investigación y Artes del Estado de Hessen dentro de su apoyo conjunto al Centro Nacional de Investigación de Ciberseguridad Aplicada. Felix Günther ha sido apoyado en parte por la beca de investigación GU 1859/1-1 de la Fundación Alemana de Investigación (DFG) y las subvenciones de la Fundación Nacional de Ciencias (NSF) CNS-1526801 y CNS-1717640. Douglas Stebila recibió el apoyo de la subvención DP130104304 del Proyecto Discovery del Consejo Australiano de Investigación (ARC), la subvención RGPIN-2016-05146 del Consejo de Investigación de Ciencias Naturales e Ingeniería de Canadá (NSERC) y la subvención RGPIN-2016-05146 del suplemento Discovery Accelerator del NSERC. Este trabajo ha sido cofinanciado por la DFG como parte del proyecto S4 dentro del CRC 1119 CROSSING.

Referencias

[AASS19] Liliya Akhmetzyanova, Evgeny Alekseev, Ekaterina Smyshlyaeva y Alexandr Sokolov. Continuando con la reflexión sobre TLS 1.3 con PSK externo. Cryptology ePrint Archive, Informe 2019/421, 2019. https://eprint.iacr.org/2019/421. (Citado en las páginas 4 y 18.)

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Ben jamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin y Paul Zimmerman. Imperfecto

secreto hacia adelante: cómo Diffie-Hellman falla en la práctica. En *ACM CCS 15,* mayo de 2015. (Citado en la página 3).

[ABF+19] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu y Cristina Onete. La privacidad del protocolo TLS 1.3. *PoPETs,* 2019(4):190–210, octubre de 2019. doi:10.2478/ popets-2019-0065. (Citado en la página 4.)

[ABP+13] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering y Jacob CN Schuldt. Sobre la seguridad de RC4 en TLS. En *Proc. 22º Simposio de seguridad de USENIX,* páginas 305–320. USENIX, 2013. (Citado en la página 3).

[ABR01] Michel Abdalla, Mihir Bellare y Phillip Rogaway. Los supuestos del oráculo Diffie-Hellman y un análisis de DHIES. En David Naccache, editor, *CT-RSA 2001,* volumen 2020 de *LNCS,* páginas 143–158. Springer, Heidelberg, abril de 2001. doi:10.1007/3-540-45353-9_12. (Citado en la página 8.)

[AGJ21] Nimrod Aviram, Kai Gellert y Tibor Jager. Protocolos de reanudación de sesión y seguridad de reenvío eficiente para TLS 1.3 0-RTT. *Journal of Cryptology,* 2021. Por aparecer. Disponible como Cryptol ogy ePrint Archive, Informe 2019/228. https://eprint.iacr.org/2019/228. (Citado en las páginas 4 y 52.)

[AP13] Nadhem J. AlFardan y Kenneth G. Paterson. Lucky trece: rompiendo los protocolos récord TLS y DTLS. En *Simposio IEEE sobre seguridad y privacidad de 2013,* páginas 526–540. IEEE Computer Society Press, mayo de 2013. doi:10.1109/SP.2013.42. (Citado en la página 3.)

[AWZ10] J. Altman, N. Williams y L. Zhu. Enlaces de canales para TLS. RFC 5929 (Estándar propuesto), julio de 2010. URL: http://www.ietf.org/rfc/rfc5929.txt. (Citado en la página 50.)

[BBD+15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub y Jean Karim Zinzindohoue. Un estado desordenado de la unión: domar las máquinas de estado compuesto de TLS. En *Simposio IEEE sobre seguridad y privacidad de 2015,* páginas 535–552. IEEE Computer Society Press, mayo de 2015. doi:10.1109/SP.2015.39. (Citado en la página 4.)

[BBDL+15] Benjamin Beurdouche, Katheikeyan Bhargavan, Antoine Delignat-Levaud, Cedric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub y Jean Karim Zinzindohoue. Un estado desordenado de la unión: domar las máquinas de estado compuesto de TLS. En *Proc. Simposio IEEE on Security & Privacy (S&P) 2015,* páginas 535–552. IEEE, 2015. (Citado en la página 3).

[BBF+16] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss y Santiago Zanella-Béguelin. Degradar la resiliencia en los protocolos de intercambio de claves. En *Simposio IEEE sobre seguridad y privacidad de 2016,* páginas 506–525. IEEE Computer Society Press, mayo de 2016. doi:10.1109/SP.2016.37. (Citado en la página 4.)

[BCK96] Mihir Bellare, Ran Canetti y Hugo Krawczyk. Funciones hash de codificación para autenticación de mensajes. En Neal Koblitz, editor, *CRYPTO'96,* volumen 1109 de *LNCS,* páginas 1–15. Springer, Heidelberg, agosto de 1996. doi:10.1007/3-540-68697-5_1. (Citado en la página 8.)

[BDF+14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti y Pierre Yves Strub. Apretones de manos triples y cortadores de galletas: rompiendo y arreglando la autenticación sobre TLS. En *Simposio IEEE sobre seguridad y privacidad de 2014,* páginas 98–113. IEEE Computer Society Press, mayo de 2014. doi:10.1109/SP.2014.14. (Citado en las páginas 3 y 50.)

[Bel06] Mihir Bellare. Nuevas pruebas para NMAC y HMAC: Seguridad sin resistencia a colisiones. En Cyn thia Dwork, editora, *CRYPTO 2006,* volumen 4117 de *LNCS,* páginas 602–619. Springer, Heidelberg, agosto de 2006. doi:10.1007/11818175_36. (Citado en la página 8.)

[BFG19a] Jacqueline Brendel, Marc Fischlin y Felix Günther. Desglose de la resiliencia de los protocolos de intercambio de claves: NewHope, TLS 1.3 e híbridos. En Kazue Sako, Steve Schneider y Peter YA Ryan, editores, *ESORICS 2019, Parte II,* volumen 11736 de *LNCS,* páginas 521–541. Springer, Hei delberg, septiembre de 2019. doi:10.1007/978-3-030-29962-0_25. (Citado en las páginas 4 y 51.)

[BFG+19b] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson y Douglas Stebila. Desafíos en la prueba de intercambios de claves poscuánticas basados en mecanismos de encapsulación de claves. Criptología ePrint Archive, Informe 2019/1356, 2019. https://eprint.iacr.org/2019/1356. (Citado en la página 52.)

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther y Christian Janson. PRF-ODH: Relaciones, instanciaciones y resultados de imposibilidad. En Jonathan Katz y Hovav Shacham, editores, *CRYPTO 2017, Parte III,* volumen 10403 de *LNCS,* páginas 651–681. Springer, Heidelberg, agosto de 2017. doi:10.1007/978-3-319-63697-9_22. (Citado en las páginas 6, 8, 49 y 52.)

[BFK+13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti y Pierre-Yves Strub. Implementación de TLS con seguridad criptográfica verificada. En *Simposio IEEE sobre seguridad y privacidad de 2013,* páginas 445–459. IEEE Computer Society Press, mayo de 2013. doi:10.1109/ SP.2013.37. (Citado en la página 4.)

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub y Santiago Zanella Béguelin. Demostrando que el apretón de manos TLS es seguro (tal como es). En Juan A. Garay y Rosario Gennaro, editores, *CRYPTO 2014, Parte II,* volumen 8617 de *LNCS,* páginas 235–255. Springer, Heidelberg, agosto de 2014. doi:10.1007/978-3-662-44381-1_14. (Citado en las páginas 4 y 51.)

[BFK16] Karthikeyan Bhargavan, Cédric Fournet y Markulf Kohlweiss. miTLS: Verificación de implementaciones de protocolos contra ataques del mundo real. *Seguridad y privacidad de IEEE,* 14(6):18–25, 2016. doi: 10.1109/ MSP.2016.123. (Citado en la página 4.)

[BFS+13] Christina Brzuska, Mark Fischlin, Nigel P. Smart, Bogdan Warinschi y Stephen C. Williams. Menos es más: nociones de seguridad relajadas pero componibles para el intercambio de claves. *International Journal of Information Security,* 12(4):267–297, agosto de 2013. doi:10.1007/s10207-013-0192-y. (Citado en la página 4.)

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi y Stephen C. Williams. Composibilidad de los protocolos de intercambio de claves Bellare-Rogaway. En Yan Chen, George Danezis y Vitaly Shmatikov, editores, *ACM CCS 2011,* páginas 51–62. ACM Press, octubre de 2011. doi: 10.1145/2046707.2046716. (Citado en las páginas 5, 14, 22 y 52.)

[BH17] Colin Boyd y Britta Hale. Canales seguros y terminación: la última palabra sobre TLS. En Tanja Lange y Orr Dunkelman, editores, *LATINCRYPT 2017,* volumen 11368 de *LNCS,* páginas 44–65. Springer, Heidelberg, septiembre de 2017. doi:10.1007/978-3-030-25283-0_3. (Citado en la página 52.)

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway y Björn Tackmann. Canales seguros aumentados y el objetivo de la capa de registro TLS 1.3. En Man Ho Au y Atsuko Miyaji, editores, *ProvSec 2015,* volumen 9451 de *LNCS,* páginas 85–104. Springer, Heidelberg, noviembre de 2015. doi:10.1007/978-3-319-26059-4_5. (Citado en la página 4.)

[BR94] Mihir Bellare y Philip Rogaway. Autenticación de entidades y distribución de claves. En Douglas R. Stinson, editor, *CRYPTO'93,* volumen 773 de *LNCS,* páginas 232–249. Springer, Heidelberg, agosto de 1994. doi:10.1007/3-540-48329-2_21. (Citado en las páginas 4, 5, 14, 16, 49 y 52.)

[Brz13] Cristina Brzuska. *Sobre los fundamentos del intercambio de claves.* Tesis doctoral, Technische Universität Darmstadt, Darmstadt, Alemania, 2013. http://tuprints.ulb.tu-darmstadt.de/3414/. (Citado en las páginas 5, 14, 22 y 52.)

[BT16] Mihir Bellare y Björn Tackmann. La seguridad multiusuario del cifrado autenticado: AES-GCM en TLS 1.3. En Matthew Robshaw y Jonathan Katz, editores, *CRYPTO 2016, Parte* I, volumen 9814 de *LNCS,* páginas 247–276. Springer, Heidelberg, agosto de 2016. doi: 10.1007/978-3-662-53018-4_10. (Citado en la página 4.)

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen y Tibor Jager. Protocolos de intercambio de claves altamente eficientes con una estanqueidad óptima. En Alexandra Boldyreva y Daniele Micciancio, editores, *CRYPTO 2019, Parte III,* volumen 11694 de *LNCS,* páginas 767–797. Springer, Heidelberg, agosto de 2019. doi:10.1007/978-3-030-26954-8_25. (Citado en las páginas 4 y 7.)

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott y Thyla van der Merwe. Un análisis simbólico completo de TLS 1.3. En Bhavani M. Thuraisingham, David Evans, Tal Malkin y Dongyan Xu, editores, *ACM CCS 2017,* páginas 1773–1788. ACM Press, octubre/noviembre de 2017. doi:10.1145/3133956.3134063. (Citado en la página 4.)

[CHK03] Ran Canetti, Shai Halevi y Jonathan Katz. Un esquema de cifrado de clave pública seguro hacia adelante. En Eli Biham, editor, *EUROCRYPT 2003,* volumen 2656 de *LNCS,* páginas 255–271. Springer, Heidelberg, mayo de 2003. doi:10.1007/3-540-39200-9_16. (Citado en la página 52.)

[CHSv16] Cas Cremers, Marko Horvat, Sam Scott y Thyla van der Merwe. Análisis y verificación automatizados de TLS 1.3: 0-RTT, reanudación y autenticación retrasada. En *Simposio IEEE sobre seguridad y privacidad de 2016,* páginas 470–485. IEEE Computer Society Press, mayo de 2016. doi:10.1109/SP.2016.35. (Citado en la página 4.)

[CJJ+19] Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva y Cristina Nita-Rotaru. Establecimiento de un canal de comunicación seguro: TLS 1.3 (sobre TCP de apertura rápida) frente a QUIC. En Kazue Sako, Steve Schneider y Peter YA Ryan, editores, *ESORICS 2019, Parte* I, volumen 11735 de *LNCS,* páginas 404–426. Springer, Heidelberg, septiembre de 2019. doi:10.1007/978-3-030-29959-0_20. (Citado en la página 4.)

[CK01] Ran Canetti y Hugo Krawczyk. Análisis de protocolos de intercambio de claves y su uso para la construcción de canales seguros. En Birgit Pfitzmann, editora, *EUROCRYPT 2001,* volumen 2045 de *LNCS,* páginas 453–474. Springer, Heidelberg, mayo de 2001. doi:10.1007/3-540-44987-6_28. (Citado en las páginas 5 y 16.)

[CK02] Ran Canetti y Hugo Krawczyk. Análisis de seguridad del protocolo de intercambio de claves basado en firmas de IKE. En Moti Yung, editor, *CRYPTO 2002,* volumen 2442 de *LNCS,* páginas 143–161. Springer, Heidelberg, agosto de 2002. http://eprint.iacr.org/2002/120/. doi:10.1007/ 3-540-45708-9_10. (Citado en la página 15.)

[Bacalao14] Codenomicón. El bicho Heartbleed. http://heartbleed.com, Abril de 2014. (Citado en la página 3).

[CPS19] Eric Crockett, Christian Paquin y Douglas Stebila. Creación de prototipos de autenticación e intercambio de claves post-cuánticas e híbridas en TLS y SSH. En *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019,* agosto de 2019. (Citado en la página 52).

[DA99] Tim Dierks y Christopher Allen. El Protocolo TLS Versión 1.0. RFC 2246 (Estándar propuesto), enero de 1999. Obsoleto por RFC 4346, actualizado por RFC 3546, 5746, 6176, 7465, 7507, 7919. URL: https://www.rfc-editor.org/rfc/rfc2246.txt, doi:10.17487/RFC2246. (Citado en la página 3.)

[DFGS15] Benjamin Dowling, Marc Fischlin, Felix Günther y Douglas Stebila. Un análisis criptográfico de los candidatos al protocolo de protocolo de enlace TLS 1.3. En Indrajit Ray, Ninghui Li y Christopher Kruegel, editores, *ACM CCS 2015,* páginas 1197–1210. ACM Press, octubre de 2015. doi:10.1145/2810103.2813653. (Citado en las páginas 4, 6, 14, 16, 48, 49 y 52.)

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther y Douglas Stebila. Un análisis criptográfico del protocolo TLS 1.3 draft-10 full and pre-shared key handshake. Cryptology ePrint Archive, Informe 2016/081, 2016. http://eprint.iacr.org/2016/081. (Citado en las páginas 4, 6, 14, 16, 48 y 49.)

[DFK+17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Ras togi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan y Jean Karim Zinzindohoue. Implementación y prueba de la capa de registro TLS 1.3. En *2017 IEEE*

Simposio sobre seguridad y privacidad, páginas 463–482. IEEE Computer Society Press, mayo de 2017. doi:10.1109/SP.2017.58. (Citado en las páginas 4 y 52.)

[DFP+20] Antoine Delignat-Lavaud, Cédric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ra mananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina y Yi Zhou. Un modelo de seguridad y una implementación completamente verificada para la capa de registro IETF QUIC. Cryptology ePrint Archive, Informe 2020/114, 2020. https://eprint.iacr.org/2020/114. (Citado en la página 50.)

[dFW19] Cyprien Delpech de Saint Guilhem, Marc Fischlin y Bogdan Warinschi. Autenticación en intercambio de claves: Definiciones, relaciones y composición. Cryptology ePrint Archive, Informe 2019/1203, 2019. https://eprint.iacr.org/2019/1203. (Citado en las páginas 15 y 51.)

[DG21a] Hannah Davis y Felix Günther. Pruebas más estrictas para los protocolos de intercambio de claves SIGMA y TLS 1.3. En *la 19ª Conferencia Internacional sobre Criptografía Aplicada y Seguridad de Redes, ACNS 2021,* 2021. Por comparecer. Disponible como Cryptology ePrint Archive, Informe 2020/1029. https://eprint.iacr.org/2020/1029. (Citado en las páginas 4 y 7.)

[DG21b] Nir Drucker y Shay Gueron. Selfie: reflexiones sobre TLS 1.3 con PSK. *Journal of Cryptology,* 2021. Por aparecer. Disponible como Cryptology ePrint Archive, Informe 2019/347. https://eprint. iacr.org/2019/347. (Citado en las páginas 4, 6 y 18.)

[DJ21] Denis Diemert y Tibor Jager. Sobre la estricta seguridad de TLS 1.3: parámetros criptográficos teóricamente sólidos para implementaciones en el mundo real. *Journal of Cryptology,* 2021. Por aparecer. Disponible como Cryptology ePrint Archive, Informe 2020/726. https://eprint.iacr.org/2020/726. (Citado en las páginas 4 y 7.)

[DJSS18] David Derler, Tibor Jager, Daniel Slamanig y Christoph Striecks. Cifrado de filtro Bloom y aplicaciones para el intercambio eficiente de claves 0-RTT de secreto hacia adelante. En Jesper Buus Nielsen y Vincent Rijmen, editores, *EUROCRYPT 2018, Parte III,* volumen 10822 de *LNCS,* páginas 425– 455. Springer, Heidelberg, abril/mayo de 2018. doi:10.1007/978-3-319-78372-7_14. (Citado en la página 52.)

[Abajo17] Benjamín Dowling. *Seguridad comprobable de los Protocolos de Internet.* Tesis de doctorado, Universidad Tecnológica de Queensland, Brisbane, Australia, 2017. http://eprints.qut.edu.au/108960/. (Citado en las páginas 6 y 48.)

[DR06] Tim Dierks y Eric Rescorla. El protocolo de seguridad de la capa de transporte (TLS) versión 1.1. RFC 4346 (Norma propuesta), abril de 2006. Obsoleto por RFC 5246, actualizado por RFC 4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919. URL: https://www.rfc-editor.org/rfc/ rfc4346.txt, doi:10.17487/RFC4346. (Citado en la página 3.)

[DR08] Tim Dierks y Eric Rescorla. El protocolo de seguridad de la capa de transporte (TLS) versión 1.2. RFC 5246 (Estándar propuesto), agosto de 2008. Actualizado por RFC 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919. URL: https://www.rfc-editor.org/rfc/rfc5246 .TXT, doi:10.17487/RFC5246. (Citado en la página 3.)

[DS15] Benjamin Dowling y Douglas Stebila. Modelado de conjunto de cifrado y negociación de versiones en el protocolo TLS. En Ernest Foo y Douglas Stebila, editores, *ACISP 15,* volumen 9144 de *LNCS,* páginas 270–288. Springer, Heidelberg, junio/julio de 2015. doi:10.1007/978-3-319-19962-7_16. (Citado en la página 4.)

[Dúo11] Tailandés Duong. BESTIA. http://vnhacker.blogspot.com.au/2011/09/beast.html, Septiembre de 2011. (Citado en la página 3).

[FG14] Marc Fischlin y Felix Gunther. Intercambio de claves en varias etapas y el caso del protocolo QUIC de Google. En Gail-Joon Ahn, Moti Yung y Ninghui Li, editores, *ACM CCS 2014,* páginas 1193–1204. ACM Press, noviembre de 2014. doi:10.1145/2660267.2660308. (Citado en las páginas 5, 6, 14, 17 y 52.)

[FG17] Marc Fischlin y Felix Gunther. Ataques de repetición en tiempo de ida y vuelta cero: el caso de los candidatos de protocolo de enlace TLS 1.3. En *el Simposio europeo sobre seguridad y privacidad del IEEE de 2017, EuroS&P 2017,* páginas 60 a 75, París, Francia, del 26 al 28 de abril de 2017. IEEE. (Citado en las páginas 4, 6, 14, 16, 17, 48 y 52).

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson y Kenneth G. Paterson. Los datos son un flujo: seguridad de los canales basados en flujo. En Rosario Gennaro y Matthew JB Robshaw, editores, *CRYPTO 2015, Parte II,* volumen 9216 de *LNCS,* páginas 545–564. Springer, Heidelberg, agosto de 2015. doi:10.1007/978-3-662-48000-7_27. (Citado en la página 52.)

[FGSW16] Marc Fischlin, Felix Günther, Benedikt Schmidt y Bogdan Warinschi. Confirmación de clave en el intercambio de claves: un tratamiento formal e implicaciones para TLS 1.3. En *Simposio IEEE sobre seguridad y privacidad de 2016,* páginas 452–469. IEEE Computer Society Press, mayo de 2016. doi:10.1109/SP.2016.34. (Citado en las páginas 4, 15 y 51.)

[Gaj08] Sebastián Gajeck. Un marco de composición universal para el análisis de protocolos de seguridad basados en navegador. En Joonsang Baek, Feng Bao, Kefei Chen y Xuejia Lai, editores, *ProvSec 2008,* volumen 5324 de *LNCS,* páginas 283–297. Springer, Heidelberg, octubre/noviembre de 2008. (Citado en las páginas 4 y 49).

[GHJL17] Felix Günther, Britta Hale, Tibor Jager y Sebastian Lauer. Intercambio de claves 0-RTT con total confidencialidad directa. En Jean-Sébastien Coron y Jesper Buus Nielsen, editores, *EUROCRYPT 2017, Part III,* volumen 10212 de *LNCS,* páginas 519–548. Springer, Heidelberg, abril/mayo de 2017. doi:10.1007/978-3-319-56617-7_18. (Citado en la página 52.)

[GKS13] Florian Giesen, Florian Kohlar y Douglas Stebila. Sobre la seguridad de la renegociación de TLS. En Ahmad-Reza Sadeghi, Virgil D. Gligor y Moti Yung, editores, *ACM CCS 2013,* páginas 387–398. ACM Press, noviembre de 2013. doi:10.1145/2508859.2516694. (Citado en la página 4.)

[GM15] Matthew D. Green e Ian Miers. Reenvíe mensajes asincrónicos seguros desde el cifrado perforable. En *Simposio IEEE sobre seguridad y privacidad de 2015,* páginas 305–320. IEEE Computer Society Press, mayo de 2015. doi:10.1109/SP.2015.26. (Citado en la página 52.)

[GM17] Félix Günther y Sogol Mazaheri. Un tratamiento formal de los canales multiclave. En Jonathan Katz y Hovav Shacham, editores, *CRYPTO 2017, Parte III,* volumen 10403 de *LNCS,* páginas 587–618. Springer, Heidelberg, agosto de 2017. doi:10.1007/978-3-319-63697-9_20. (Citado en las páginas 4, 50 y 52.)

[Gün18] Félix Gunther. *Modelado de aspectos de seguridad avanzados de intercambio de claves y protocolos de canal seguro.* Tesis doctoral, Technische Universität Darmstadt, Darmstadt, Alemania, 2018. http://tuprints.ulb.tu-darmstadt.de/7162/. (Citado en las páginas 5, 6, 14, 16, 48, 51 y 52.)

[HJLS17] Britta Hale, Tibor Jager, Sebastian Lauer y Jörg Schwenk. Definiciones de seguridad simples y construcciones de intercambio de claves 0-RTT. En Dieter Gollmann, Atsuko Miyaji y Hiroaki Kikuchi, editores, *ACNS 17,* volumen 10355 de *LNCS,* páginas 20–38. Springer, Heidelberg, julio de 2017. doi:10.1007/978-3-319-61204-1_2. (Citado en la página 52.)

[JK02] Jakob Jonsson y Burton S. Kaliski Jr. Sobre la seguridad del cifrado RSA en TLS. En Moti Yung, editor, *CRYPTO 2002,* volumen 2442 de *LNCS,* páginas 127–142. Springer, Heidelberg, agosto de 2002. doi:10.1007/3-540-45708-9_9. (Citado en las páginas 4 y 49.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge y Jörg Schwenk. Sobre la seguridad de TLS-DHE en el modelo estándar. En Reihaneh Safavi-Naini y Ran Canetti, editores, *CRYPTO 2012,* volumen 7417 de *LNCS,* páginas 273–293. Springer, Heidelberg, agosto de 2012. doi:10.1007/ 978-3-642-32009-5_17. (Citado en las páginas 4, 6, 8 y 49.)

[Jos15] Simón Josefsson. Enlaces de canales para TLS basados en PRF. https://tools.ietf.org/html/draft-josefsson-sasl-tls-cb-03 , Marzo de 2015. (Citado en la página 50).

[JSS15] Tibor Jager, Jörg Schwenk y Juraj Somorovsky. Sobre la seguridad de TLS 1.3 y QUIC frente a debilidades en el cifrado PKCS#1 v1.5. En Indrajit Ray, Ninghui Li y Christopher Kruegel, editores, *ACM CCS 2015,* páginas 1185–1196. ACM Press, octubre de 2015. doi:10.1145/ 2810103.2813657. (Citado en la página 7.)

[KBC97] Hugo Krawczyk, Mihir Bellare y Ran Canetti. HMAC: Keyed-Hashing para autenticación de mensajes. RFC 2104 (informativo), febrero de 1997. Actualizado por RFC 6151. URL: https://www.rfc-editor.org/rfc/ rfc2104.txt, doi:10.17487/RFC2104. (Citado en la página 8.)

[KE10] Hugo Krawczyk y Pasi Eronen. Función de derivación de clave de extracción y expansión basada en HMAC (HKDF). RFC 5869 (informativo), mayo de 2010. URL: https://www.rfc-editor.org/rfc/ rfc5869.txt, doi:10.17487/ RFC5869. (Citado en la página 8.)

[KMO+15] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann y Daniele Venturi. (De-)construir TLS 1.3. En Alex Biryukov y Vipul Goyal, editores, *INDOCRYPT 2015,* volumen 9462 de *LNCS,* páginas 85–102. Springer, Heidelberg, diciembre de 2015. doi:10.1007/ 978-3-319-26617-6_5. (Citado en la página 4.)

[KPW13] Hugo Krawczyk, Kenneth G. Paterson y Hoeteck Wee. Sobre la seguridad del protocolo TLS: un análisis sistemático. En Ran Canetti y Juan A. Garay, editores, *CRYPTO 2013, Parte* I, volumen 8042 de *LNCS,* páginas 429–448. Springer, Heidelberg, agosto de 2013. doi: 10.1007/978-3-642-40041-4_24. (Citado en las páginas 4 y 51.)

[Kra01] Hugo Krawczyk. El orden de encriptación y autenticación para proteger las comunicaciones (o: ¿Qué tan seguro es SSL?). En Joe Kilian, editor, *CRYPTO 2001,* volumen 2139 de *LNCS,* páginas 310–331. Springer, Heidelberg, agosto de 2001. doi:10.1007/3-540-44647-8_19. (Citado en la página 4.)

[Kra03] Hugo Krawczyk. SIGMA: El enfoque “SIGn-and-MAc” para la autenticación Diffie-Hellman y su uso en los protocolos IKE. En Dan Boneh, editor, *CRYPTO 2003,* volumen 2729 de *LNCS,* páginas 400–425. Springer, Heidelberg, agosto de 2003. doi:10.1007/978-3-540-45146-4_24. (Citado en las páginas 3 y 15.)

[Kra10] Hugo Krawczyk. Extracción criptográfica y derivación de claves: El esquema HKDF. En Tal Rabin, editor, *CRYPTO 2010,* volumen 6223 de *LNCS,* páginas 631–648. Springer, Heidelberg, agosto de 2010. doi:10.1007/978-3-642-14623-7_34. (Citado en la página 8.)

[Kra16a] Hugo Krawczyk. [Lista de correo de IETF TLS] Re: Solicitud de consenso: Eliminación de 0-RTT basado en DHE. https://mailarchive.ietf.org/arch/msg/tls/xmnvrKEQkEbD-u8HTeQkyitmclY, Marzo de 2016. (Citado en la página 52).

[Kra16b] Hugo Krawczyk. Un compilador de autenticación unilateral a mutua para el intercambio de claves (con aplicaciones para la autenticación de clientes en TLS 1.3). En Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers y Shai Halevi, editores, *ACM CCS 2016,* páginas 1438–1450. ACM Press, octubre de 2016. doi:10.1145/2976749.2978325. (Citado en las páginas 4 y 7.)

[KSS13] Florian Kohlar, Sven Schäge y Jörg Schwenk. Sobre la seguridad de TLS-DH y TLS-RSA en el modelo estándar. Cryptology ePrint Archive, Informe 2013/367, 2013. http://eprint. iacr.org/2013/367. (Citado en la página 4.)

[KW16] Hugo Krawczyk y Hoeteck Wee. El protocolo OPTLS y TLS 1.3. En *el Simposio europeo IEEE sobre seguridad y privacidad de 2016,* páginas 81–96. IEEE, marzo de 2016. doi:10.1109/EuroSP. 2016.18. (Citado en las páginas 4 y 6.)

[LJBN15] Robert Lychev, Samuel Jero, Alexandra Boldyreva y Cristina Nita-Rotaru. ¿Qué tan seguro y rápido es QUIC? Análisis de rendimiento y seguridad comprobable. En *Simposio IEEE sobre seguridad y privacidad de 2015,* páginas 214–231. IEEE Computer Society Press, mayo de 2015. doi:10.1109/ SP.2015.21. (Citado en la página 52.)

[LLM07] Brian A. LaMacchia, Kristin Lauter y Anton Mityagin. Mayor seguridad de intercambio de claves autenticadas. En Willy Susilo, Joseph K. Liu y Yi Mu, editores, *ProvSec 2007,* volumen 4784 de *LNCS,* páginas 1–16. Springer, Heidelberg, noviembre de 2007. (Citado en las páginas 5 y 16).

[LP17] Atul Luykx y Kenneth G. Paterson. Límites en el uso de cifrado autenticado en TLS, agosto de 2017. URL: http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf. (Citado en la página 4.)

[LSY+14] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar y Jörg Schwenk. Sobre la seguridad de los conjuntos de cifrado de claves precompartidas de TLS. En Hugo Krawczyk, editor, *PKC 2014,* volumen 8383 de *LNCS,* páginas 669–684. Springer, Heidelberg, marzo de 2014. doi:10.1007/978-3-642-54631-0_ 38. (Citado en la página 4.)

[LXZ+16] Xinyu Li, Jing Xu, Zhenfeng Zhang, Dengguo Feng y Honggang Hu. Múltiples protocolos de seguridad de los candidatos a TLS 1.3. En *Simposio IEEE sobre seguridad y privacidad de 2016,* páginas 486–505. IEEE Computer Society Press, mayo de 2016. doi:10.1109/SP.2016.36. (Citado en la página 4.)

[Mac17] Colm MacCarthaigh. [Lista de correo de IETF TLS] Revisión de seguridad de TLS1.3 0-RTT. https://mailarchive.ietf.org/arch/msg/tls/mHxi-O3du9OQHkc6CBWBpc_KBpA , Mayo de 2017. (Citado en la página 51).

[MDK14] Bodo Möller, Thai Duong y Krzysztof Kotowicz. Este CANICHE muerde: Explotando el respaldo de SSL 3.0. https://www.openssl.org/~bodo/ssl-poodle.pdf, Septiembre de 2014. (Citado en la página 3).

[MP17] Giorgia Azzurra Marson y Bertram Poettering. Nociones de seguridad para canales bidireccionales. *IACR Trans. Sim. Cryptol.,* 2017(1):405–426, 2017. doi:10.13154/tosc.v2017.i1.405-426. (Citado en la página 52.)

[MSW08] Paul Morrissey, Nigel P. Smart y Bogdan Warinschi. Un análisis de seguridad modular del protocolo de protocolo de enlace TLS. En Josef Pieprzyk, editor, *ASIACRYPT 2008,* volumen 5350 de *LNCS,* páginas 55–73. Springer, Heidelberg, diciembre de 2008. doi:10.1007/978-3-540-89255-7_5. (Citado en las páginas 4 y 49.)

[PRS11] Kenneth G. Paterson, Thomas Ristenpart y Thomas Shrimpton. El tamaño de la etiqueta sí importa: Ataques y pruebas para el protocolo de registro TLS. En Dong Hoon Lee y Xiaoyun Wang, editores, *ASIACRYPT 2011,* volumen 7073 de *LNCS,* páginas 372–389. Springer, Heidelberg, diciembre de 2011. doi:10.1007/978-3-642-25385-0_20. (Citado en la página 4.)

[PS18] Christopher Patton y Thomas Shrimpton. Canales parcialmente especificados: la capa de registro TLS 1.3 sin elisión. En David Lie, Mohammad Mannan, Michael Backes y XiaoFeng Wang, editores, *ACM CCS 2018,* páginas 1415–1428. ACM Press, octubre de 2018. doi:10.1145/3243734.3243789. (Citado en las páginas 4 y 52.)

[PvdM16] Kenneth G. Paterson y Thyla van der Merwe. Estandarización reactiva y proactiva de TLS. En Lidong Chen, David A. McGrew y Chris J. Mitchell, editores, *Security Standardization Research: Third International Conference (SSR 2016),* volumen 10074 de *Lecture Notes in Computer Science,* páginas 160–186, Gaithersburg, MD, EE. UU., 5 y 6 de diciembre de 2016. Springer.
(Citado en las páginas 3 y 53.)

[Res18] E. Rescorla. El protocolo de seguridad de la capa de transporte (TLS) versión 1.3. RFC 8446 (Estándar propuesto), agosto de 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.txt, doi:10.17487/RFC8446. (Citado en las páginas 3, 6, 51 y 53.)

[Rog06] Felipe Rogaway. Formalizando la ignorancia humana. En Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06,* volumen 4341 de *LNCS,* páginas 211–228. Springer, Heidelberg, septiembre de 2006. (Citado en la página 7).

[RTM19] Eric Rescorla, Hannes Tschofenig y Nagendra Modadugu. El protocolo de seguridad de la capa de transporte de datagramas (DTLS) versión 1.3 – draft-ietf-tls-dtls13-33. https://tools.ietf.org/html/draft-ietf-tls-dtls13-33 , Octubre de 2019. (Citado en la página 7).

[SSW20] Peter Schwabe, Douglas Stebila y Thom Wiggers. TLS poscuántico sin firmas de protocolo de enlace. En Jay Ligatti, Xinming Ou, Jonathan Katz y Giovanni Vigna, editores, *ACM CCS 20,* páginas 1461–1480. ACM Press, noviembre de 2020. doi:10.1145/3372297.3423350. (Citado en las páginas 4 y 52.)

[TT20] Martín Thomson y Sean Turner. Uso de TLS para proteger QUIC: draft-ietf-quic-tls-29. https: //tools.ietf.org/html/borrador-ietf-quic-tls-29, Junio de 2020. (Citado en la página 50).

A Reducción de consultas de prueba múltiples a únicas

En esta sección, brindamos más detalles sobre el argumento híbrido para reducir los adversarios Amulti que realizan múltiples consultas de prueba en el juego Multi-Stage para los protocolos de enlace TLS 1.3 a los adversarios ASingle que se restringen a una única consulta de prueba. Tenga en cuenta que cualquier adversario de consultas múltiples no puede realizar más consultas de prueba (razonables) que el número *ns* de sesiones generales multiplicado por el número máximo M de etapas. Cualquier adversario que realice más consultas debe repetir las consultas para algunas claves, lo que arroja la respuesta ÿ, y dichas consultas se pueden resolver fácilmente.

El paso principal es el argumento híbrido donde el adversario Asingle simula el ataque de Amulti , haciendo una única consulta de prueba. Para hacerlo, para un índice *n* elegido aleatoriamente entre 1 y el número máximo *ns* de consultas de prueba, el adversario Asingle devuelve las claves genuinas en las primeras *n* ÿ 1 consultas de Amulti, plantea la consulta de prueba para la n-ésima consulta como propia consulta y devuelve claves aleatorias a partir de la consulta *n* + 1 en adelante. Para obtener las claves genuinas para las primeras consultas, Asingle llama al oráculo Reveal.

Lo anterior funciona junto con el argumento común en los juegos híbridos si podemos asegurarnos de que Asingle no pierda debido a las consultas adicionales de Revelación que realiza, es decir, si revela una clave para el compañero de la (única) sesión de Prueba. Una opción para garantizar esto es exigir que Amulti nunca pruebe una sesión y su pareja. Afortunadamente, el modelo de seguridad de múltiples etapas en la Sección 4 admite esto sin problemas. Es decir, probar una sesión en la etapa *i* para la cual el indicador testi ya se ha establecido en verdadero devolverá inmediatamente ÿ. Esta configuración de la bandera ocurre en uno de los siguientes casos:

- Cuando la sesión en sí se prueba por primera vez en esta etapa, o
- si la sesión acepta en esta etapa después de una llamada de envío y ya hay un socio con testi = verdadero (activado en la ejecución de envío para el cual la sesión acepta), o • si está asociado a una sesión probada y acaba de aceptar en la misma etapa (desencadenada a través de la prueba del socio en la ejecución de Test oracle).

Además, una llamada de prueba a una etapa de sesión *i* para la que ya pasó la sesión, es decir, stexec 6= accepti , también devuelve ÿ. En otras palabras, cualquier consulta de prueba de Amulti a un socio de una consulta de prueba anterior devuelve ÿ. Por lo tanto, podemos evitar tales consultas y dejar que Asingle responda ÿ para tales consultas directamente. En este caso, las consultas Reveal restantes, que sustituyen las primeras consultas Test en el argumento híbrido, no pueden hacer que Asingle pierda, porque ahora seguro que no están asociadas con la (única) consulta Test de Asingle.

Hay dos advertencias en el razonamiento anterior. En primer lugar, el adversario ASingle necesita saber si las dos etapas de la sesión están asociadas para responder correctamente ÿ a algunas consultas de prueba. Si bien esto es trivial de deducir de los datos de comunicación pública para sid1 y sid2, los identificadores de sesión sid3, . . . ,sid6 contienen mensajes transmitidos de forma confidencial, protegidos a través de las claves de tráfico de protocolo de enlace derivadas en las etapas 1 y 2 del protocolo de enlace TLS 1.3. Pero si le hacemos saber a Asingle estas dos claves internas a través de consultas Reveal cuidadosamente seleccionadas al probar una etapa *i* ÿ 3, entonces puede descifrar la comunicación y decidir asociarse con otras sesiones para esta etapa. Dado que estas consultas de revelación adicionales son para etapas anteriores, esencialmente no pueden interferir con la etapa de la sesión de prueba.

Es conveniente almacenar la información sobre las sesiones probadas en una matriz interna simTestedi [etiqueta] que se establece en verdadero si la etiqueta de la sesión se hubiera marcado como probada en el juego, si Amulti realmente hubiera realizado esa consulta. Escribimos esto como una matriz para distinguir esta lista interna de Asingle de las entradas en la etiqueta de sesiones. En cualquier momento, la matriz simTested en la simulación de Asingle contendrá la misma información que las entradas probadas si Amulti realmente hubiera realizado todas las consultas de prueba.

El segundo problema surge del hecho de que las claves internas (es decir, las claves en las etapas *i* con USEi = interna) se sobrescriben en los socios de las sesiones probadas. Esto puede suceder en el comando Send(label, m) si una etiqueta de socio de una etapa de sesión probada para label0 (con label0 .testedi = true) va a accepti . Luego, el juego de seguridad establece *label.keyi* ÿ label0 .keyi . El otro caso puede ocurrir en la consulta Test(label, i) si una sesión asociada label0 a la sesión probada ya está en el estado label0 .stexec = accepti . Luego, la clave interna de esa sesión label0 se reemplaza por la respuesta de la sesión probada. Pero nuestro adversario ASingle con una sola consulta de prueba, por supuesto

solo establece una etapa de sesión para probar, lo que influye como máximo en una sesión adicional, mientras que las múltiples consultas de prueba de Amulti pueden sobrescribir varias claves.

Dado que no existe otro mecanismo para modificar claves en el modelo de seguridad, debemos solucionar el problema manualmente en la simulación de Amulti a través de Asingle. Afortunadamente, las claves internas en el protocolo de enlace TLS 1.3 solo se usan para proteger los datos en el transporte, envolviendo y desenvolviendo los datos inmediatamente al enviarlos o recibirlos. Por lo tanto, dejamos que Asingle mantenga los arreglos internos actualKeyi [etiqueta] y simKeyi [etiqueta] para las claves internas (en el ataque real de Asingle, resp. en la simulación de Amulti) en la etapa *i* ÿ {1, 2} y dejamos que Asingle se adapte autenticado encriptaciones con respecto a tales claves cuando las retransmite entre la simulación Amulti y Send.

Lema A.1. *Deje* que *Amulti sea un adversario que realiza como máximo* nTest ÿ M · *ns llamadas a* Test *atacando TLS 1.3 full 1-RTT handshake en el* juego *Multi-Stage . Entonces existe un adversario Asingle que hace una sola* consulta de prueba *tal que*

$$\mathrm{Adv}^{\text{Multi-etapa}}_{\text{TLS1.3-full-1RTT, Amulti}}.D \;\text{ÿ}\; \text{nPrueba} \cdot \mathrm{Adv}^{\text{Multietapa}}_{\text{TLS1.3-full-1RTT, Un solo}}.D .$$

Además, Asingle inicia el mismo número máximo de sesiones que Amulti .

El lema es válido de manera análoga para las otras variantes de protocolo de enlace de TLS 1.3, ya que el argumento usa solo detalles que comparten todas las variantes.

Prueba. Construimos nuestro adversario Asingle a partir de Amulti a través de una simulación de caja negra. Adversario Un solo procede de la siguiente manera. Inicialmente elige *n* ÿ {1, *2, . . . ,* nTest} al azar e inicializa matrices vacías actualKeyi [ ] ÿ ÿ, simKeyi [ ] ÿ ÿ para *i = 1,* 2 y simTestedi [ ] ÿ false para *i* ÿ {1, *2, . . . ,* M}. El algoritmo Asingle luego invoca a Amulti, retransmitiendo todas las consultas de Oracle excepto las consultas de envío y prueba.

Simulación de consultas de envío . Una consulta Enviar (etiqueta, m) se responde posiblemente cambiando los cifrados, si la sesión se ha marcado como una sesión (virtualmente) probada, de modo que las claves deben adaptarse. Sea *i* = *label.stage* la etapa actual, lo que significa que la sesión ya ha aceptado en la etapa i:

- Si *i* ÿ 1 o *m* = init o *m* = continuar, entonces pase el comando al propio oráculo Enviar. tales mensajes no están encriptados y no necesitamos volver a encriptar los datos de comunicación.
- Si *i* ÿ 2 y simTestedi [etiqueta] = verdadero, es decir, los datos se cifran con una clave que posiblemente haya cambiado debido a la prueba (virtual), luego vuelva a cifrar con el resp. del cliente. clave de tráfico de protocolo de enlace del servidor en el experimento de Asingle. Tenga en cuenta que los identificadores de sesión (especialmente para las etapas *i* ÿ 3) no se ven afectados por este nuevo cifrado, ya que se definen sobre los textos claros:
  - Si *label.role* = iniciador, es decir, esperamos que el cliente reciba un mensaje protegido bajo el secreto de intercambio de tráfico del servidor (la clave de la etapa 2), luego descifre *m* con la clave simKey2 [etiqueta] y vuelva a cifrar el resultado con actualKey2 [label] a m0 antes de pasar (label, m0) al propio oráculo de Send. Aquí, y en lo que sigue, asumimos que el cifrado siempre tiene éxito para mensajes diferentes de ÿ, y que ÿ se cifra en algo que nuevamente se descifra en ÿ.
  - Si *label.role* = respondedor, es decir, esperamos que el servidor reciba un mensaje protegido bajo el secreto de intercambio de tráfico del cliente (la clave de la etapa 1), luego descifre *m* con la clave simKey1 [etiqueta] y vuelva a cifrar el resultado con actualKey1 [label] a m0 antes de pasar (label, m0) al propio oráculo de Send.
- En cualquier otro caso basta con reenviar (etiqueta, m) al propio oráculo Enviar.

Para la respuesta *m* del oráculo Enviar, haga lo siguiente:

- Si *i* ÿ 1, devolver la respuesta sin cambios.
- Si *i* ÿ 2 y simTestedi [label] = true, entonces adapte el cifrado a las claves esperadas por Amulti:
  - Si *label.role* = iniciador, descifre *m* con la clave actualKey1 [label] y vuelva a cifrar el resultado con simKey1 [label] en m0 antes de devolver m0 .
  - Si *label.role* = respondedor, descifre *m* con la clave actualKey2 [label] y vuelva a cifrar el resultado con simKey2 [label] en m0 antes de devolver m0 .

• En cualquier otro caso devolver m.

Además, verifique si es necesario establecer el estado de simTested. Si la llamada Enviar cambia el estado a aceptadoi+1 —sobre el cual se informa al adversario ASingle— entonces haga lo siguiente:

• Para i+1 ÿ 2, si hay una sesión label0 6= label with label0 .sidi+1 = label.sidi+1 and *simTestedi+1[label0]* = true, entonces configure el estado de prueba para la sesión aquí, *simTestedi +1[etiqueta]* ÿ verdadero. Tenga en cuenta que dado que los identificadores de sesión en las dos primeras etapas consisten en mensajes de texto claro, esto es fácil de verificar en este caso. Copie también las claves internas, *actualKeyi+1[label]* ÿ *actualKeyi+1[label0]* y simKeyi+1[label] ÿ simKeyi+1[label0].

• Para *i* + 1 ÿ 3, si simTested1[label] = true, busque la clave actualKey1 [label], de lo contrario, realice una consulta Reveal (label, 1), y análogamente para la clave para la etapa 2. Dado que la sesión en consideración tiene ya aceptado en etapa *i* + 1 ÿ 3 en este punto, nuestro adversario Asingle obtiene las dos claves de tráfico de protocolo de enlace y utiliza estas claves para descifrar la comunicación (en su ataque) para recuperar *sidi+1* en la etiqueta de sesión. Compare este valor con los identificadores de sesión en todas las sesiones label0 con *simTestedi+1[label0]* = true para la misma etapa *i* + 1. Tenga en cuenta que una sesión label0 solo puede asociarse en la etapa *i* + 1 ÿ 3 si ya está asociada en las dos primeras etapas, porque *sidi+1* contiene los identificadores sid1 y sid2 como prefijo (excepto la etiqueta). Esto también implica que tales sesiones solo pueden derivar las mismas claves de tráfico de protocolo de enlace. Por lo tanto, podemos usar las mismas claves de protocolo de enlace que para label para descifrar label0 . Si hay una coincidencia, actualice *simTestedi+1[label]* ÿ true y copie las claves de la sesión label0 como antes, *actualKeyi+1[label]* ÿ *actualKeyi+1[label0]* y simKeyi+1[label] ÿ simKeyi+1[etiqueta0].

Excepto por la copia de la clave actual, esto ahora corresponde exactamente al paso de actualización en Enviar consulta.

Simulación de consultas de prueba . La consulta de prueba t-ésima (etiqueta, i) se responde de la siguiente manera:

• Si no hay una etiqueta de sesión, o la etiqueta de sesión aún no se ha aceptado en la etapa *i* , lo cual es conocido por el adversario porque aprende label.stexec al completar con éxito la i-ésima etapa, o simTestedi [label] = verdadero, luego devuelve inmediatamente ÿ.

• De lo contrario, proceda de la siguiente manera:

– Si *t* < *n* , haga una llamada Reveal(label, i) para obtener la clave *K* y devolverla a Amulti. Configure simTestedi [etiqueta] ÿ verdadero y, si *i* ÿ 2 y la clave es interna, configure también actualKeyi [etiqueta] ÿ *K* y simKeyi [etiqueta] ÿ K.

– Si *t* = *n* , entonces haga una llamada Test(label, i) y devuelva la respuesta *K* a Amulti. Establezca simTestedi [etiqueta] ÿ verdadero y, si *i* ÿ 2, también configure actualKeyi [etiqueta] ÿ *K* y simKeyi [etiqueta] ÿ K.

– Si *t* > *n* , elija una tecla *K* ÿ$ D al azar y devuélvala a Amulti. Establezca simTestedi [etiqueta] ÿ verdadero y, si *i* ÿ 2, esta vez defina actualKeyi [etiqueta] ÿ Reveal (etiqueta, i) y simKeyi [etiqueta] ÿ K.

• Finalmente, debemos verificar como en la consulta de prueba original si ya existe una sesión asociada en estado aceptado para la misma etapa y, de ser así, modificar su estado. Si existe una sesión label0 6= label asociada, label0 .sidi = *label.sidi* simTestedi [label0] ÿ true. Si *i* ÿ 2, copie también las claves actualKeyi [label0] ÿ actualKeyi [label] y simKeyi [label0] ÿ establecer y donde label0 .stexec = label.stexec = accept, simKeyi [label]. Notamos que la verificación contra una coincidencia con label0 se realiza de manera análoga a la consulta Enviar.

Hacemos notar que no alteramos el oráculo Reveal simulado pero permitimos consultas sin modificaciones.
Ahora hay casos en los que el adversario de consultas múltiples Amulti puede obtener una clave interna diferente a la esperada. Pero esto solo puede suceder para las sesiones que han sido (virtualmente) probadas, de modo que cualquier consulta Reveal o una sesión asociada en la que se haya cambiado la clave haría que Amulti perdiera. Por lo tanto, ignoramos estos casos y simplemente continuamos con la respuesta desalineada.

Análisis. Tenga en cuenta que las consultas Reveal adicionales, que Asingle realiza para claves internas en la simulación de Enviar y Prueba anterior, no pueden interferir con su única consulta de Prueba. Recuerde que Asingle puede realizar consultas Reveal(label, 1) y Reveal(label, 2) al simular las consultas Send y Test.

En la consulta de envío simulada, debemos verificar que las posibles consultas Reveal(label, 1) y Reveal(label, 2) para las claves internas en las etapas 1 y 2 no entren en conflicto con la (única) consulta de prueba para la sesión labeltested which Asingle hace para la etapa i. Tenga en cuenta que estas consultas solo se realizarán si la etiqueta de la sesión ha aceptado en una etapa ÿ 3. Suponga que, de hecho, *i* ÿ {1, 2} y que la etiqueta probada está asociada con la etiqueta en esa etapa i. Para ello, distinga el momento en el que se realiza la llamada Test a labeltested :

- Si la llamada de prueba para labeltested se realiza más tarde, después de que la etiqueta de sesión haya continuado después de aceptar en la etapa *i* ÿ 2, entonces el adversario Amulti pierde, la razón es que en este caso hay una etiqueta de sesión asociada que ha continuado más allá de la etapa *i* ÿ {1, 2} y ya ha usado la clave interna. Una llamada de prueba de este tipo establece perdido y verdadero según el modelo.

- Si la llamada de prueba para labeltested ya se realizó para la etapa *i* ÿ {1, 2} antes de que la sesión label haya continuado después de aceptar en esa etapa, entonces la sesión (label, i) se asoció con (labeltested, i) para *i* ÿ { 1, 2} debe haberse marcado como probado, simTestedi [label] = true, en una simulación de Test o Send (sin usar consultas Reveal para esta etapa con identificadores de sesión de texto claro). En este caso, sin embargo, nuestro algoritmo Asingle no realiza una consulta Reveal para esta etapa, sino que lee la clave de la matriz actualKeyi [etiqueta].

Por lo tanto, en el primer caso solo podemos aumentar la probabilidad de éxito y en el segundo caso evitamos una consulta Reveal conflictiva de inmediato. Tenga en cuenta que lo mismo es cierto para la verificación final en la simulación de Prueba.

Queda por discutir la compatibilidad de las otras consultas potenciales de Reveal en la consulta de prueba simulada. Si la consulta n para la sesión labeltested y la etapa *i* (que Asingle reenvía a su oráculo Test) se asociaría con la consulta t (label, i), entonces la llamada de Amulti a su oráculo Test (simulado) para labeltested luego

- haría que Amulti perdiera si la sesión etiquetada estaba en el punto de la consulta ya más allá del estado aceptadoi para la clave interna (según la descripción del oráculo de prueba), o
- la sesión labeltested ya está en estado accepti cuando se realiza la consulta de prueba aquí, en cuyo caso el oráculo de prueba (simulado) marcaría esa sesión labeltested como probada, simTestedi [labeltested] = true, porque está asociada con el ahora (virtualmente) etiqueta de sesión probada, o
- la sesión labeltested aún no está en estado accepti cuando se realiza la consulta de prueba aquí, en cuyo caso más tarde el oráculo de envío (simulado) marcaría esa sesión labeltested como probada cuando finalmente acepta en la etapa i, simTestedi [labeltested] = true, porque luego se asocia a la etiqueta de sesión (virtualmente) probada aquí.

Ignoramos el primer caso porque no puede contribuir a la probabilidad de éxito de Amulti . Para los dos últimos casos, se deduce que la n-ésima consulta de Amulti en realidad no se reenviará a la prueba del oráculo de Asingle , porque para tales sesiones marcadas con simTestedi [labeltested] = true, el oráculo de prueba simulado devuelve inmediatamente ÿ.
Por lo tanto, Asingle no realiza ninguna consulta de prueba en estos casos y, en particular, no puede perder debido a una consulta de revelación para una sesión asociada a la de la consulta de prueba.

De lo anterior se deduce que Asingle sólo se echa a perder en su ataque si Amulti lo hace en la simulación. Para el paso final en el análisis del argumento híbrido, observe que si *n* = 1 y btest = 0 (para el bit de desafío en el juego de Asingle), entonces nuestro adversario Asingle solo devuelve claves aleatorias a A0 (o mensajes de error ÿ) en consultas de prueba simuladas . Además, a menos que $A0_{multi}$ pierde el juego, la simulación es perfectamente sólida en el sentido de que $_{multi}$ tiene la misma distribución que en un ataque real; en particular, este argumento no es violado por el nuevo cifrado. Por lo tanto, en este caso tenemos que Asingle predice su valor btest con la misma probabilidad que A0 cuando recibe solo claves aleatorias en todas las consultas de prueba (válidas). Análogamente, si $n = n_{Test}$ y btest = 1, Asingle siempre devuelve claves genuinas (o errores) a $A0_{multi}$ pierde. Por lo tanto, esto corresponde al caso de que $A0_{multi}$ Para el análisis, sea *b* la salida de Asingle y btest sea su bit de desafío. De manera similar, sea $b'_{multi,}$ de nuevo, en una simulación de sonido a menos que $A0_{multi}$ $_{multi}$ solo recibe claves genuinas en todas las consultas de prueba (válidas). de Amulti en un ataque real para el bit de prueba $b'_{prueba.}$ Denotamos por *b* = btest resp. *B* $b' = b'_{prueba}$ eventos que el bit

es correcto y la bandera perdida no está configurada. Luego,

$$\Pr[b = btest] = \sum_{n0=1}^{nXTest} \Pr[b = bprueba \ddot{y} n = n0]$$

$$= \frac{1}{nPrueba} \cdot \sum_{n0=1}^{prueba\ nXT} \Pr[b = bprueba \mid n = n0]$$

$$= \frac{1}{nPrueba} \cdot \sum_{n0=1}^{prueba\ nXT} \bar{}_{1\,2} \cdot \Pr[b = 0 \mid bprueba = 0 \ddot{y} n = n0] + \bar{}_{1\,2} \cdot 1 \ddot{y} \Pr[b = 0 \mid bprueba = 1 \ddot{y} n = n0]$$

$$= \frac{1}{1 + 2} \cdot \frac{1}{nPrueba} \cdot \sum_{n0=1}^{prueba\ nXT} \bar{}_{1\,2} \cdot \Pr[b = 0 \mid bprueba = 0 \ddot{y} n = n0] \ddot{y} \Pr[b = 0 \mid bprueba = 1 \ddot{y} n = n0]$$

y observando que la simulación condicionada a btest = 1 y *n* = n0 es equivalente a la simulación para btest = 0 y *n* = n0 + 1, la suma del telescopio se simplifica a

$$= \frac{1\,1 + 2\ nPrueba}{nPrueba\ 1\,1 + 2} \cdot \bar{}_{1\,2} \Pr[b = 0 \mid bprueba = 0 \ddot{y} n = 1] \ddot{y} \Pr[b = 0 \mid bprueba = 1 \ddot{y} n = nPrueba]$$

$$= \bar{} \cdot \frac{}{} \cdot \bar{}_{1\,2} \Pr[b = 0 \mid bprueba = 0 \ddot{y} n = 1] \ddot{y} 1 + \Pr[b = 1 \mid bprueba = 1 \ddot{y} n = nPrueba]$$

$$\ddot{y} \frac{1 + 2}{nPrueba} \cdot \Pr[b^{0} {}^{= segundo} {}^{0} prueba] - \bar{}_{1\,2}$$

donde usamos en el último paso que la simulación es perfectamente sólida (si Amulti no pierde) y, por lo tanto, al menos la probabilidad en un ataque real. Hacemos notar que nuestro adversario Asingle puede desencadenar los perdidos ÿ verdaderos con menos frecuencia que Amulti, por ejemplo, debido a las consultas de prueba omitidas y los posibles conflictos con las consultas de revelación.
Por lo tanto, obtenemos

$$\mathrm{Adv}^{\mathrm{Multi\text{-}etapa,D}}_{\mathrm{TLS1.3\text{-}full\text{-}1RTT,A0\ multi}} \ddot{y}\ nPrueba \cdot \mathrm{Adv}^{\mathrm{Multietapa,D}}_{\mathrm{TLS1.3\text{-}full\text{-}1RTT,\ Un\ solo\ ,}}$$

probar la afirmación del lema. □