Desencriptando RC4 en WPA-TKIP y TLS

Mathy Vanhoef
KU Leuven
Mathy.Vanhoef@cs.kuleuven.be

Frank Piessens
KU Leuven
Frank.Piessens@cs.kuleuven.be

Resumen

Se presentan nuevos enfoques (biases) en RC4 para romper el **Protocolo Wi-Fi de Acceso Temporal Protegido por Claves** (WPA-TKIP) y un diseño práctico de ataque de recuperación de texto sencillo contra el **Protocolo de Transporte de Capas de Seguridad** (TLS). Para los nuevos enfoques en el flujo de claves RC4 se emplean pruebas de hipótesis estadísticas, lo cual permite desarrollar nuevas posibilidades en los *bytes* iniciales de flujo de clave (initial keystream *bytes*), al igual que varios enfoques a largo plazo (long-term biases). Nuestros renovadores algoritmos de recuperación de texto sencillo (plaintext recovery algorithms) son capaces de utilizar múltiples tipos de enfoques y devolver una lista de candidatos de texto sencillo (plaintext candidates) en orden de probabilidad decreciente.

Para desencriptar el WPA-TKIP introducimos un método generador de amplios números de paquetes idénticos. Estos se desencriptan a través de la generación de una lista de candidatos de texto sencillo, y el uso de una estructura de paquetes redundantes para excluir los candidatos falsos. Del paquete desencriptado se deriva la clave TKIP MIC, que puede usarse para inyectar y desencriptar otros paquetes. En la práctica, el ataque puede ser ejecutado en una hora. También atacamos el TLS tal como es usado por HTTPS, aquí mostramos cómo desencriptar una *cookie* segura con índice de éxito del 94%, mediante $9 \cdot 2^{27}$ textos cifrados (ciphertexts). Esto se hace inyectando datos conocidos alrededor de la *cookie* utilizando el enfoque *ABSABB* de Mantin y atacando la *cookie* mediante una lista de candidatos de texto sencillo. Con nuestra técnica de generación de tráfico el ataque se ejecuta en solo 75 horas.

1 Introducción

RC4 es todavía uno de los encriptadores de flujos (stream cyphers) más usados. Se conoce bien su empleo en SSL y WEP, así como en sus sucesores TLS[8] y WPA-TKIP[19]. Particularmente, se comenzó a utilizar más luego de que fueran publicados los ataques de BEAST [9], Lucky 13[1] y el de *padding oracle*[7] contra los esquemas de encriptación de modo CBC en TLS. Como atenuante se recomendó el RC4. De ahí que en un momento dado, casi el 50 % de las conexiones TLS se utilizara el RC4[2], con un estimado actual del 30%[18]. Ello motivó la creación de nuevos ataques, siendo los ejemplos más relevantes [2, 20, 31, 15,30]. De especial interés es el propuesto por AlFardan et al., en el que aproximadamente se requieren $13 \cdot 2^{30}$ textos cifrados para desencriptar una *cookie* enviada a través de HTTPS[2]. Lo cual se corresponde con cerca de 2000 horas de datos en su configuración, por tanto, este ataque se considera muy cerca de ser factible. Nuestro objetivo es ver en qué medida estos ataques pueden desarrollarse explorando tres áreas: Primeramente, con la búsqueda de nuevos enfoques en el flujo de clave. En segundo lugar, a través de la optimización de los algoritmos de recuperación de texto sencillo, y por último, con la demostración de nuestras técnicas para realizar los ataques en la práctica.

En un primer momento, la búsqueda empírica de nuevos enfoques en el flujo de clave se hizo generando una gran cantidad de flujo de clave y almacenando esas estadísticas en varios lotes de datos, para luego ser analizados mediante pruebas de hipótesis estadísticas.

Nuestra hipótesis nula es que un *byte* de flujo de clave está distribuido uniformemente, o que dos *bytes* son independientes. El rechazo de esta hipótesis nula equivale a detectar un enfoque. En comparación con los gráficos de inspección manual, esta nueva forma permite un análisis a mayor escala. De esa manera, encontramos varios nuevos enfoques en los *bytes* iniciales del flujo de datos, al igual que varios nuevos enfoques a largo plazo.

Se accede al WPA-TKIP desencriptando un paquete completo mediante el RC4 y derivando la clave TKIP MIC. La misma puede ser empleada para inyectar y desencriptar paquetes[48]. En particular, modificamos el ataque de recuperación de texto sencillo de Paterson et al. [31,30] para obtener una lista de candidatos con probabilidad decreciente. Los candidatos falsos son detectados y desechados sobre la base de CRC (desencriptado) del paquete, incrementando el margen de éxito para desencriptar simultáneamente todos los *bytes* desconocidos. En su implementación empleamos un nuevo método para inyectar rápidamente paquetes de datos idénticos en una red. En la práctica este ataque se ejecuta en una hora.

También atacamos el RC4 tal como es usado en TLS y HTTPS, desencriptando una *cookie* asegurada en condiciones reales. Ello se hace combinando los enfoques *ABSAB* y de Fluhrer-McGrew mediante las variantes de los ataques de Isobe et al. y AlFardan et al [20,2]. Nuestra técnica se puede extender fácilmente hacia otros enfoques igualmente. Según el enfoque *ABSAB* de Mantin, inyectamos texto sencillo conocido alrededor de la *cookie* y explotamos esto para calcular las probabilidades de texto sencillo Bayesianas (Bayesian plaintext likelihoods) de la *cookie* desconocida. Luego se genera una lista de candidatos de *cookie*s con probabilidades decrecientes, y a través de ella se desencripta la *cookie* en poco tiempo. El algoritmo para generar candidatos difiere del WPA-TKIP por su dependencia del doble *byte* en vez de las probabilidades del *byte* único. Combinándolo todo, se necesitan $9 \cdot 2^{27}$ encriptaciones de una *cookie* para desencriptarla con índice de éxito del 94%. Finalmente, mostramos cómo ejecutar este ataque en la práctica en tan solo 75 horas.

En resumen, nuestras principales contribuciones son:

- Empleo de pruebas estadísticas para detectar empíricamente enfoques en el flujo de clave, mostrando varios enfoques nuevos.
- Se diseñan algoritmos de recuperación de texto sencillo capaces de utilizar múltiples enfoques, devolviendo una lista de candidatos de texto sencillo con probabilidad decreciente.
- Se demuestran técnicas prácticas para desencriptar RC4, tanto en WPA-TKIP como en TLS.

El presente artículo se organiza de la siguiente manera: la sección 2 ofrece un contexto sobre RC4, TKIP y TLS. En la tercera, introducimos las pruebas de hipótesis y reportamos los nuevos enfoques. Las técnicas de recuperación de texto sencillo se dan en la sección 4. Los ataques prácticos sobre TKIP y TLS se presentan en la sección 5 y 6 respectivamente. Luego resumimos el trabajo en la 7 y se concluye en la sección 8.

2 Contexto

Se introduce el algoritmo RC4 y su utilización en WPA-TKIP y TLS.

2.1 El algoritmo RC4

El algoritmo RC4 es breve, interesante y reconocido por funcionar rápidamente en software. Consiste en un Algoritmo de Programación de Claves (KSA) y un Algoritmo de Generación Pseudo Aleatorio (PRGA), ambos se muestran en la figura 1. Está compuesto por una permutación *S* del lote {0,…,255}, un contador público *i*, y un índice privado *j*. El KSA toma como entrada una clave de longitud de variable e inicializa *S*. En cada vuelta r =1,2…del PRGA, resulta un *byte* Z_r de flujo de clave. Todas las adiciones son realizadas mediante modulo 256. Un *byte* de texto sencillo P_r es encriptado a un *byte* de texto cifrado C_r mediante $C_r = P_r \oplus Z_r$.

Listing (1) RC4 Key Scheduling (KSA).

```
j, S = 0, range(256)
for i in range(256):
    j += S[i] + key[i % len(key)]
    swap(S[i], S[j])
return S
```

Listing (2) RC4 Keystream Generation (PRGA).

```
S, i, j = KSA(key), 0, 0
while True:
    i += 1
    j += S[i]
    swap(S[i], S[j])
    yield S[S[i] + S[j]]
```

Figura 1: Implementación RC4 en un pseudo código similar a Python. Todas las adiciones se hacen en modulo 256.

2.1.1 Los enfoques a corto plazo (Short-term biases)

Varios enfoques se han encontrado en los *bytes* iniciales de flujo de clave RC4, ellos se denominan enfoques a corto plazo. El más significativo fue encontrado por Mantin y Shamir. Ellos mostraron que el segundo *byte* de flujo de clave tiene doble probabilidad de ser cero en vez de uniforme.[25] O más formalmente, que $P_r[\, Z_2 = 0\,] \approx 2 \cdot 2^{-8}$, donde la probabilidad depende de la elección aleatoria de la clave. Debido a que el valor cero aparece más de lo esperado, lo nombramos enfoques positivos. De forma similar, si un valor aparece menos de lo esperado, se denominarán enfoques negativos. Este resultado fue difunfido por Maitra et al.[2] y perfeccionado aún más por Sen Gupta et al.[38], quienes demostraron que en la mayoría de los *bytes* iniciales de flujo de clave existe un enfoque tendiente a cero. Sen Gupta et al. también detectaron enfoques dependientes de la longitud de clave (key-length dependent biases): si ℓ es la longitud de la clave, el *byte* de flujo de clave Z_ℓ tiene un enfoque positivo hacia 256 – ℓ.[38] AlFardan et al. mostraron que todos los 256 *bytes* iniciales de flujo de clave están enfocados, estimando empíricamente sus probabilidades con claves de 16 *bytes*.[2] También encontraron enfoques válidos adicionales, un ejemplo es el tendiente hacia el valor r para todas la posiciones $1 \leq r \leq 256$. Este también fue descubierto independientemente por Isobe et al.[20]

El enfoque $P_r [Z_1 = Z_2] = 2^{-8} (1-2^{-8})$ fue detectado por Paul y Preneel.[33] Isobe et al. perfeccionaron este resultado para el valor cero $P_r [Z_1 = Z_2 = 0] \approx 3 \cdot 2^{-16}$ [20]. Estos autores buscaron también enfoques de similar validez entre los *bytes* iniciales, aunque infructuosamente.[20] Sin embargo, en este artículo sí detectamos enfoques nuevos (véase sección 3.3).

2.1.2 Los enfoques a largo plazo (Long-term biases)

En contraste son los enfoques a corto plazo, que ocurren solo en los *bytes* iniciales del flujo de clave, también existen otros que se mantienen a través del flujo de clave completo. Ellos se denominan enfoques a largo plazo. Por ejemplo, Fluhrer y McGrew (FM) determinaron la probabilidad de ciertos dígrafos, es decir, *bytes* de flujo de clave consecutivos (Z_r, Z_{r+1}), que se desvían de la uniformidad durante el flujo de clave completo.[13] Estos enfoques dependen del contador público *i* del PRGA, y están listados en la Tabla 1 (ignorando la condición en *r* por el momento). En su análisis, Fluhrer y McGrew asumieron que el estado interno del algoritmo RC4 era uniformente aleatorio.

Digraph	Condition	Probability
(0,0)	$i = 1$	$2^{-16}(1+2^{-7})$
(0,0)	$i \neq 1,255$	$2^{-16}(1+2^{-8})$
(0,1)	$i \neq 0,1$	$2^{-16}(1+2^{-8})$
(0,$i+1$)	$i \neq 0,255$	$2^{-16}(1-2^{-8})$
($i+1$,255)	$i \neq 254 \wedge r \neq 1$	$2^{-16}(1+2^{-8})$
(129,129)	$i = 2, r \neq 2$	$2^{-16}(1+2^{-8})$
(255,$i+1$)	$i \neq 1,254$	$2^{-16}(1+2^{-8})$
(255,$i+2$)	$i \in [1,252] \wedge r \neq 2$	$2^{-16}(1+2^{-8})$
(255,0)	$i = 254$	$2^{-16}(1+2^{-8})$
(255,1)	$i = 255$	$2^{-16}(1+2^{-8})$
(255,2)	$i = 0,1$	$2^{-16}(1+2^{-8})$
(255,255)	$i \neq 254 \wedge r \neq 5$	$2^{-16}(1-2^{-8})$

Tabla 1: Enfoques Fluhrer y McGrew (FM) generalizados. Aquí *i* es el contador público en el PRGA y *r* la posición del primer *byte* del dígrafo. Se muestran las probabilidades de enfoques a largo plazo (para los enfoques a corto plazo véase figura 4).

Este supuesto es verdadero solo después de varias vueltas del PRGA [13,26,38]. Por consiguiente, estos enfoques generalmente no se esperaban en los *bytes* iniciales del flujo de clave. De cualquier forma, en la sección 3.3.1 mostramos que la mayoría de los enfoques sí ocurren en los *bytes* iniciales del flujo de clave, aunque con probabilidades diferentes respecto a las variantes a largo plazo.

Otro enfoque a largo plazo fue encontrado por Mantin:[24] en su patrón *ABSAB*, *A* y *B* representan valores de *bytes* y *S* una secuencia breve de *bytes* nombrada "el intervalo". Con la longitud del intervalo *S* denotada por *g*, el enfoque se escribe:

$$\Pr[(Z_r, Z_{r+1}) = (Z_{r+g+2}, Z_{r+g+3})] = 2^{-16}(1+2^{-8}e^{\frac{-4-8g}{256}}) \quad (1)$$

De ahí que entre mayor sea el intervalo, más débiles serán los enfoques. Finalmente, Sen Gupta et al. detectaron el enfoque a largo plazo.[38]

$$\Pr[(Z_{w256}, Z_{w256+2}) = (0,0)] = 2^{-16}(1+2^{-8})$$

donde w ≥ 1. Descubrimos que un enfoque hacia (128,0) también está presente en esas posiciones (véase sección 3.4).

2.2 La encapsulación criptográfica TKIP

El objetivo del diseño de WPA-TKIP es ser un reemplazo temporal del WEP [19, §11.4.2]. Aunque ha sido retirado paulatinamente por la alianza WiFi, un estudio reciente demuestra que todavía se usa mucho[48]. De 6803 redes, se encontró que el 71% de las redes protegidas todavía empleaba el TKIP, mientras que un 19% lo tenía como protocolo de seguridad exclusivo.

Nuestro ataque sobre el TKIP se basa en dos aspectos: su débil **Chequeo de Integridad de Mensajes** (MIC) [44,48] y su construcción de claves por paquetes defectuosa [2,15,31,30]. Introducimos brevemente ambos tópicos asumiendo una **Clave Transitoria Atenta a los Pares** de 512-bit (PTK), negociada previamente entre el **Punto de Acceso** (AP) y el cliente. De la PTK se derivaron una **Clave de Encriptación Temporal** de 128-bit (TK) y dos claves de Chequeo de Integridad de Mensajes (MIC) de 64-bit. La primera clave MIC se emplea para la comunicación AP-Cliente y la segunda en la dirección inversa. Algunas investigaciones afirman que la PTK y sus claves derivadas son renovadas después de un intervalo definido por el usuario, generalmente dispuesto a 1 hora [44,48]. De cualquier forma, hemos encontrado que solo la **Clave Transitoria Atenta a Grupos** (GTK) es renovada periódicamente. Como nuestro ataque puede ser ejecutado en una hora, incluso las redes que cambian la PTK cada hora pueden ser desencriptadas.

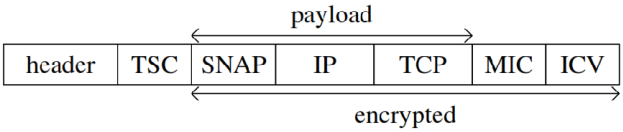

Figura 2: Marco TKIP simplificado con una carga útil TCP.

Cuando el cliente quiere trasmitir una carga útil (payload), se calcula primero un valor MIC utilizando la clave MIC apropiada y el algoritmo Micheal (véase figura 2). Desafortunadamente, este algoritmo es sencillo de invertir: la clave MIC se puede derivar eficientemente una vez dados los datos de texto sencillo y su valor MIC.[44] Después de añadir la clave MIC, también se adiciona un CRC cheksum denominado **Valor de Chequeo de Integridad** (ICV). El paquete resultante, que incluye una cabecera MAC y una carga útil TCP de ejemplo se muestra en la figura 2. La carga útil, la MIC, y el ICV están encriptados mediante RC4 con una clave por paquete. Esta se puede calcular a través de una función compuesta que toma como entrada la TK, el contador de secuencia TKIP (TSC) y el trasmisor de dirección MAC (TA). Lo cual se plantea como K=KM (TA,TK,TSC). El TSC es un contador de 6-*bytes* que se incrementa después de trasmitir el paquete y se incluye desencriptado en la cabecera del MAC. En la práctica, el resultado de KM puede ser modelado como uniformemente aleatorio [2,31]. En un intento de evitar los ataques de claves débiles (weak-key attacks) contra el WEP [12], los 3 primeros *bytes* de K están dispuestos como [19, § 11.4.2.1.1]:

$$K_0 = TSC_1 \quad K_1 = (TSC_1 \mid 0x20) \,\&\, 0x7f \quad K_2 = TSC_0$$

Aquí TSC_0 y TSC_1 son los dos *bytes* menos significativos del TSC. Como el TSC es público, también lo serán los 3 primeros *bytes* de K. A través de simulaciones y formalmente, se ha demostrado que realmente esto debilita la seguridad [2,15,31,30].

2.3 El protocolo de registro TLS

Nos concentramos en el protocolo de registro TLS cuando el RC4 se selecciona como la clave simétrica (symmetric cypher).[8] En particular, asumimos que ha sido completada la fase de protocolo de intercambio, y un terminal secreto de TLS de 48-*bytes* (48-byte **TLS master secret**) ha sido negociado.

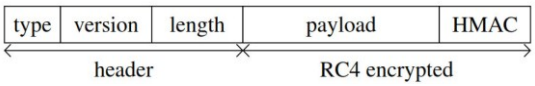

Figura 3: Estructura del registro TLS cuando se selecciona el RC4.

Para enviar una carga útil encriptada, un registro TLS de datos de tipo de aplicación (type application data) es creado. El mismo contiene la versión del protocolo, la longitud del contenido encriptado, la carga útil en sí misma, y finalmente un HMAC. El diagrama resultante se muestra en la figura 3. El HMAC se calcula por la cabecera, un número de secuencia incrementado por cada registro trasmitido, y la carga útil de texto sencillo. Tanto esta carga útil como el HMAC están encriptados. Al comienzo de la conexión, el RC4 se inicializa con una clave derivada del terminal secreto TLS. Dicha clave puede ser moderada como uniformemente aleatoria.[2] No se desechan ninguno de los *bytes* iniciales del flujo de clave.

En el contexto de HTTPS, una conexión TLS puede utilizarse para manejar múltiples solicitudes HTTP. Esto se conoce como conexión persistente. Para activar este tipo de opción en un servidor, basta con configurar la **Cabecera de la Conexión HTTP** (HTTP Connection header) con la opción "keep-alive" (mantener conectado). Ello implica que el RC4 se inicializa solo una vez para enviar todas las solicitudes HTTP, permitiendo el empleo de enfoques a largo plazo en los ataques. Finalmente, las *cookie*s pueden marcarse como seguras, garantizando que sean trasmitidas solo a través de la conexión TLS.

3 La detección de nuevos enfoques empíricamente

En esta sección explicamos cómo detectar empíricamente enfoques nuevos válidos. Aunque encontramos muchos enfoques no todos se utilizan en los ataques. Esto simplifica la descripción de los ataques. Se conoce que el empleo de enfoques nuevos puede mejorar los ataques, no obstante, el uso de los ya existentes ha probado ser significativamente suficiente para optimizarlos. De ahí que nos concentremos en los enfoques nuevos de mayor interés principalmente.

3.1 La detección de enfoques válidos

Para detectar empíricamente enfoques nuevos nos basamos en pruebas de hipótesis. Ellas consisten en generar estadísticas de flujo de clave sobre claves RC4 aleatorias, y emplear pruebas estadísticas para descubrir las desviaciones de la norma (de los valores uniformes). Esto permite que el análisis sea automatizado y a gran escala. Para detectar un enfoque de *byte* sencillo, nuestra hipótesis nula es que los valores de *bytes* de flujo de clave están distribuidos uniformemente. Para determinar enfoques entre dos *bytes*, pudiéramos estar tentados a usar como hipótesis que el par está distribuido uniformemente, al igual que la anterior. Sin embargo no es así, mientras existan ya enfoques de *byte* sencillo. En este caso, el enfoque de *byte* sencillo implica que el par también está enfocado, aunque en realidad ambos *bytes* puedan ser independientes. Por tanto, para determinar enfoques de *bytes* doble, nuestra hipótesis nula es que ambos son independientes. Con esta prueba, se detectaron pares que son en realidad más

uniformes que lo esperado. Por tanto, la negación de la hipótesis nula sería lo mismo que la detección de un enfoque.

Además, basándonos en investigaciones previas, se espera que solo poco valores entre los *bytes* de flujo de clave muestren una clara dependencia entre ellos [13,24,20,38,4]. Tomando como ejemplo el enfoque de Fluher-McGrew, en cualquier posición, solo 8 de un total 65536 pares de valores muestran un enfoque válido.[13] La prueba M de Fuchs y Kenet se utilizó para detectar dependencias entre los *bytes* de flujo de clave. Para determinar qué valores son acertados entre los *bytes* dependientes, realizamos una prueba de proporción sobre todos los pares de valores.

Se rechaza la hipótesis nula solo si el valor p es menos que 10^{-4}. El método de Holm se usa para controlar la proporción de errores posibles durante la comprobación de hipótesis, lo cual permite detectar falsos positivos en todos los procesos. Siempre se emplea la variante de doble cara en las pruebas de hipótesis debido a que los enfoques pueden ser positivos o negativos.

Determinar solo la probabilidad de dos *bytes* independientes no es aconsejable, puesto que ella incluye también la posibilidad de enfoques de un solo *byte*, y lo que se quiere es reportar la fortaleza de la dependencia entre ambos *bytes*. Para resolver esto, multiplicamos la dos posibilidades de *byte* único de un par, lo cual se espera que ocurra con la probabilidad de valor p. Dado que ese par realmente aparece con la probabilidad s, trazamos el valor $| q |$ de la fórmula $s = p \cdot (1 + q)$. De manera que este enfoque relativo indica cuánta información se obtuvo, no solo considerando el enfoque de un solo *byte*, sino también la posibilidad real del par de *byte* doble.

3.2 Generación de lotes de datos

Con el objetivo de generar una estadística detallada de *bytes* de flujo de clave se creó una configuración distribuida. Se utilizaron aproximadamente 80 computadoras de escritorio normales y tres servidores poderosos como trabajadores (**workers**). La generación de estadísticas se hizo en C. El lenguaje Python se empleó para manejar los lotes de datos generados y controlar a los trabajadores. Durante el inicio cada trabajador generó una clave AES aleatoria criptográficamente. Claves RC4 aleatorias de 128-bit se derivaron de aquella mediante AES en modo contador. Finalmente se usó R para todo el análisis estadístico.[34]

Nuestros resultados principales se basan en dos lotes de datos denominados `first16` y `consec512`. El primer lote `first16` estimó $P_r [Z_a = x \wedge Z_b = y]$ para $1 \leq a \leq 16$, $1 \leq b \leq 256$, $0 \leq x,y < 256$ utilizando 2^{44} claves. La generación tomó aproximadamente 9 años de CPU. Esto permitió detectar enfoques entre los primeros 16 *bytes* y los restantes 256 *bytes* iniciales. El lote `consec512` estimó $P_r [Z_r = x \wedge Z_{r+1} = y]$ para $1 \leq r \leq 512$ y $0 \leq x,y < 256$ utilizando 2^{45} claves, el proceso tomó 16 años de CPU. El mismo, permitió realizar un estudio detallado de los *bytes* de flujo de clave hasta la posición 512.

La generación de ambos lotes de datos se optimizó. Uno de los métodos fue que una parte de los trabajadores generaran casi 2^{30} flujos de clave. Lo cual posibilitó el empleo de enteros de 16-bit para todos los contadores que recogían estadísticas, incluso con la presencia de enfoques significativos. Solamente durante la combinación de los resultados de los trabajadores se requirieron enteros mayores. Esto significó poco uso de memoria y la reducción de las pérdidas de cache. Para continuar reduciendo las pérdidas de cache generamos varios flujos de claves antes de actualizar los contadores. En una investigación independiente, Paterson et al. utilizaron las mismas optimizaciones.[30] Para el lote `first16` aplicamos una optimización adicional, primero se generaron varios flujos de clave y después actualizamos los contadores, basados de

alguna forma en el valor de Z_a. Esta optimización viabilizó una aceleración más significativa del lote `first16.`

3.3 Nuevos enfoques a corto plazo

Durante el análisis de los lotes de datos generados detectamos muchos enfoques nuevos de corto plazo y se clasificaron en varios lotes.

3.3.1 Los enfoques en *bytes* no consecutivos

En el análisis del lote `consec512` descubrimos numerosos enfoques entre *bytes* de flujo de clave consecutivos. Nuestra primera observación es que los enfoques Fluher-McGrew también estuvieron presentes en los *bytes* iniciales de flujo de clave. Las excepciones se encontraron en las posiciones 1, 2 y 5, tal como se listaron en la Tabla 1 (nótese las condiciones extras en la posición *r*). Esto es sorprendente, pues no se esperaban enfoques de Fluher-McGrew en los *bytes* iniciales de flujo de clave.[13] La figura 4 muestra los enfoques relativos absolutos de la mayoría de los dígrafos Fluher-McGrew, comparados con sus probabilidades esperadas de *bytes* únicos (véase sección 3.1). Para todos los dígrafos, la señal del enfoque relativo *q* es la misma que su variante a largo plazo, como se observó en la Tabla 1.

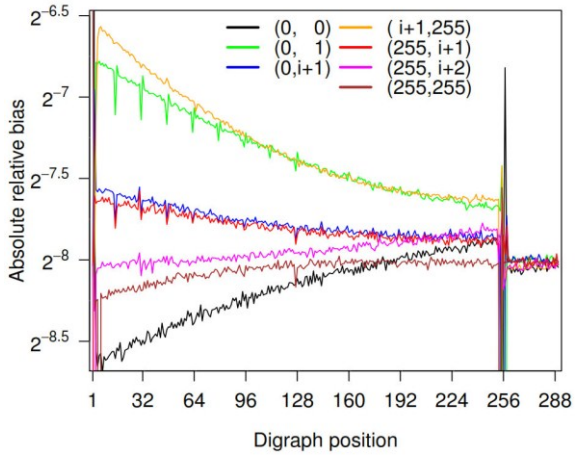

Figura 4: Enfoques relativos absolutos de varios dígrafos Fluher-McGrew en los *bytes* iniciales de flujo de clave, en comparación con sus probabilidades esperadas de *bytes* únicos.

Observamos que los enfoques relativos convergen con sus valores a largo plazo, especialmente después de la posición 257. Las líneas verticales alrededor de las posiciones 1 y 256 son causadas por dígrafos que no se mantienen (o que no se sostienen consistentemente) en estas posiciones.

Un segundo lote de enfoques sólidos tiene la forma de:

$$\Pr[Z_{w16-1} = Z_{w16} = 256 - w16] \qquad (2)$$

con $1 \leq w \leq 7$.

En la Tabla 2 siguiente listamos sus probabilidades. Como nuestra extensión de clave es igual a 16, estos son probablemente enfoques dependientes de la longitud de clave (key-length dependent biases).

First byte	Second byte	Probability
Consecutive biases:		
$Z_{15} = 240$	$Z_{16} = 240$	$2^{-15.94786}(1 - 2^{-4.894})$
$Z_{31} = 224$	$Z_{32} = 224$	$2^{-15.96486}(1 - 2^{-5.427})$
$Z_{47} = 208$	$Z_{48} = 208$	$2^{-15.97595}(1 - 2^{-5.963})$
$Z_{63} = 192$	$Z_{64} = 192$	$2^{-15.98363}(1 - 2^{-6.469})$
$Z_{79} = 176$	$Z_{80} = 176$	$2^{-15.99020}(1 - 2^{-7.150})$
$Z_{95} = 160$	$Z_{96} = 160$	$2^{-15.99405}(1 - 2^{-7.740})$
$Z_{111} = 144$	$Z_{112} = 144$	$2^{-15.99668}(1 - 2^{-8.331})$
Non-consecutive biases:		
$Z_3 = 4$	$Z_5 = 4$	$2^{-16.00243}(1 + 2^{-7.912})$
$Z_3 = 131$	$Z_{131} = 3$	$2^{-15.99543}(1 + 2^{-8.700})$
$Z_3 = 131$	$Z_{131} = 131$	$2^{-15.99347}(1 - 2^{-9.511})$
$Z_4 = 5$	$Z_6 = 255$	$2^{-15.99918}(1 + 2^{-8.208})$
$Z_{14} = 0$	$Z_{16} = 14$	$2^{-15.99349}(1 + 2^{-9.941})$
$Z_{15} = 47$	$Z_{17} = 16$	$2^{-16.00191}(1 + 2^{-11.279})$
$Z_{15} = 112$	$Z_{32} = 224$	$2^{-15.96637}(1 - 2^{-10.904})$
$Z_{15} = 159$	$Z_{32} = 224$	$2^{-15.96574}(1 + 2^{-9.493})$
$Z_{16} = 240$	$Z_{31} = 63$	$2^{-15.95021}(1 + 2^{-8.996})$
$Z_{16} = 240$	$Z_{32} = 16$	$2^{-15.94976}(1 + 2^{-9.261})$
$Z_{16} = 240$	$Z_{33} = 16$	$2^{-15.94960}(1 + 2^{-10.516})$
$Z_{16} = 240$	$Z_{40} = 32$	$2^{-15.94976}(1 + 2^{-10.933})$
$Z_{16} = 240$	$Z_{48} = 16$	$2^{-15.94989}(1 + 2^{-10.832})$
$Z_{16} = 240$	$Z_{48} = 208$	$2^{-15.92619}(1 - 2^{-10.965})$
$Z_{16} = 240$	$Z_{64} = 192$	$2^{-15.93357}(1 - 2^{-11.229})$

Tabla 2: Enfoques entre *bytes* no consecutivos.

Otro grupo de enfoques tiene la forma $P_r[Z_r = Z_{r+1} = x]$. En dependencia del valor de x, estos serán positivos o negativos. De ahí que si se suman todos los valores x, y se calcula $P_r[Z_r = Z_{r+1}]$, se perdería información estadística. En principio, estos enfoques también incluyen los pares Fluher-McGrew (0,0) y (255, 255). De cualquier manera, como el enfoque para ambos pares es mucho mayor que para los demás valores, no los incluimos aquí. Nuestro nuevo enfoque, con la forma de $P_r[Z_r = Z_{r+1}]$, fue detectado hasta la posición 512.

También descubrimos enfoques entre *bytes* no consecutivos que no entraron en ninguna categoría. Una reseña de esto se ofreció en la Tabla 2. Podemos comentar que los enfoques inducidos por $Z_{16} = 240$ tienen generalmente una posición o valor múltiplo de 16. Lo cual indica que tienen la probabilidad de ser enfoques dependientes de la longitud de clave.

3.3.2 La influencia de Z_1 y Z_2

Nuestro hallazgo más interesante es posiblemente la cantidad de información que se filtra de los dos primeros *bytes* del flujo de clave. En concreto, Z_1 y Z_2 influyeron en todos los 256 *bytes* iniciales de flujo de clave. Detectamos los siguientes 6 lotes de enfoques:

$$
\begin{aligned}
&1)\ Z_1 = 257 - i \wedge Z_i = 0 && 4)\ Z_1 = i - 1 \wedge Z_i = 1\\
&2)\ Z_1 = 257 - i \wedge Z_i = i && 5)\ \ Z_2 = 0 \wedge Z_i = 0\\
&3)\ Z_1 = 257 - i \wedge Z_i = 257 - i && 6)\ \ Z_2 = 0 \wedge Z_i = i
\end{aligned}
$$

Sus enfoques relativos absolutos, en comparación con los enfoques de *bytes* único se muestran en la Figura 5.

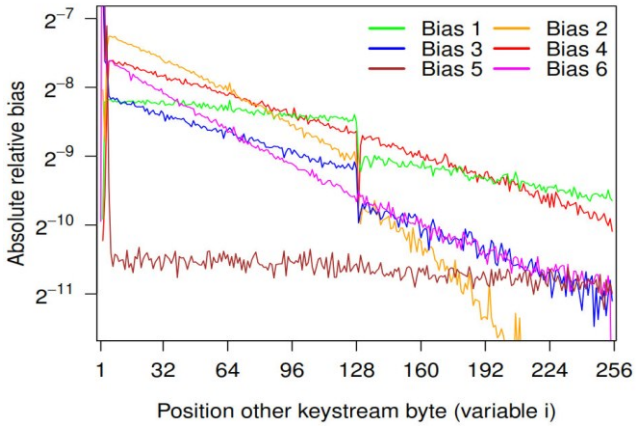

Figura 5: Enfoques inducidos por los primeros dos *bytes*. El número de los enfoques se corresponde con los de la sección 3.3.2

Los enfoques relativos de los pares 5 y 6, o sea, aquellos que implican Z_2, son generalmente negativos. Los pares que implican Z_1 son generalmente positivos, excepto el par 3, que siempre tiene un enfoque relativo negativo. También detectamos dependencias entre Z_1 y Z_2 aparte del enfoque $P_r[Z_1 = Z_2]$ de Paul y Preneel. [33] Esto significa que los siguientes pares tiene un enfoque válido:

$$\text{A) } Z_1 = 0 \wedge Z_2 = x \qquad \text{C) } Z_1 = x \wedge Z_2 = 0$$
$$\text{B) } Z_1 = x \wedge Z_2 = 258 - x \qquad \text{D) } Z_1 = x \wedge Z_2 = 1$$

Los enfoques A y C son negativos para todas las $x \neq 0$, y ambos parecen ser causados principalmente por el enfoque positivo válido $P_r[Z_1 = Z_2 = 0]$ detectado por Isobe et al. Los enfoques B y D son positivos. Además se descubrieron los tres enfoques siguientes:

$$\Pr[Z_1 = Z_3] = 2^{-8}(1 - 2^{-9.617}) \qquad (3)$$
$$\Pr[Z_1 = Z_4] = 2^{-8}(1 + 2^{-8.590}) \qquad (4)$$
$$\Pr[Z_2 = Z_4] = 2^{-8}(1 - 2^{-9.622}) \qquad (5)$$

Nótese que cualquiera de ellos supone una igualdad con Z_1 o Z_2.

3.3.3 Los enfoques de *byte* único

Analizamos los enfoques de *byte* único agregando el lote de datos `consec512`, y generando estadísticas adicionales específicamente para las probabilidades de *byte* único. La adición se representa de la siguiente manera:

$$\Pr[Z_r = k] = \sum_{y=0}^{255} \Pr[Z_r = k \wedge Z_{r+1} = y] \qquad (6)$$

Finalizamos con 2^{47} claves empleadas para estimar las probabilidades de un *byte* único. Para todos los 513 *bytes* iniciales pudimos rechazar la hipótesis de que estuvieran uniformemente distribuidos. En otras palabras, todos los 513 *bytes* iniciales constituyen enfoques.

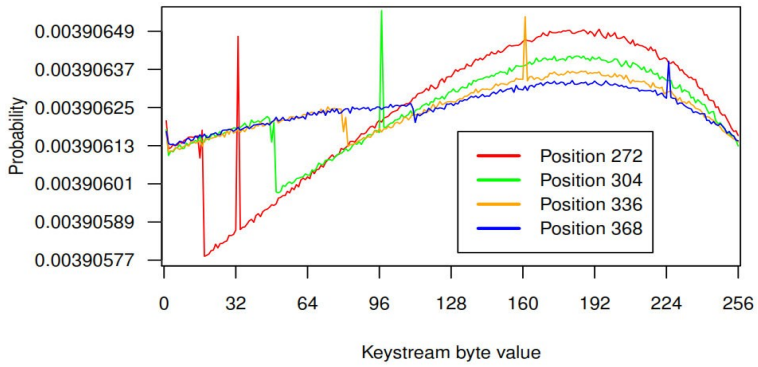

Figura 6: Enfoques de *bytes* único más allá de la posición 256.

La figura 6 muestra la probabilidad de distribución para algunas posiciones. La inspección manual de esas distribuciones reveló enfoques significativos entre $Z_{256 + k \cdot 16} = k \cdot 32$ para $1 \leq k \leq 7$. Estos son probablemente enfoques dependientes de la longitud de clave. Siguiendo a I. Mironov, [26] presumimos que también existen enfoques de *bytes* único más allá de estas posiciones, aunque menos consistentes.

3.4 Los nuevos enfoques a largo plazo

Para encontrar nuevos enfoques a largo plazo creamos una variante del lote de datos `first16`. Se estima

$$\Pr[Z_{256w+a} = x \wedge Z_{256w+b} = y] \qquad (7)$$

para $0 \leq a \leq 16$, $0 \leq b < 256$, $0 \leq x,y < 256$ y $w \geq 4$. Este enfoque se genera utilizando 2^{12} claves RC4, donde cada clave se empleó para generar 2^{40} *bytes* de flujo de clave. Esto tomó aproximadamente 8 años de CPU. La condición sobre w significa que siempre desechamos los 1023 *bytes* iniciales de flujo de clave. Usando este lote de datos podemos detectar enfoques cuya periodicidad será un divisor apropiado de 256 (o sea, detecta todos los enfoques de Fluher-McGrew). Nuestros enfoques a corto plazo no estuvieron presentes en este lote de datos, lo cual indica que ellos solo ocurren realmente en los *bytes* iniciales de flujo de clave, al menos con las probabilidades que tuvimos en cuenta. También detectamos el enfoque a largo plazo siguiente:

$$\Pr[(Z_{w256}, Z_{w256+2}) = (128,0)] = 2^{-16}(1 + 2^{-8}) \qquad (8)$$

para $w \geq 1$. Sorprendentemente este no se había descubierto antes, aunque ya se conocía un enfoque entre (0,0) en estas posiciones.[38] También buscamos específicamente enfoques con la forma $P_r[Z_r = Z_{r'}]$ mediante la adición de nuestro lote de datos. Ello develó que muchos *bytes* son dependientes entre sí. Lo cual quiere decir que encontramos enfoques a largo plazo con la forma

$$\Pr[Z_{256w+a} = Z_{256w+b}] \approx 2^{-8}(2 \pm 2^{-16}) \qquad (9)$$

Debido a que los enfoques relativos de 2^{-16} son pequeños, estos son difíciles de detectar fiablemente. Por tanto, no está claro su patrón de ocurrencia, ni cuándo sus enfoques relativos son positivos o negativos. Por lo que consideramos la detección precisa y fiable de todos los *bytes* de flujo de clave dependientes de estas cuestiones, una línea de investigación futura de mucho interés.

4 La recuperación de texto sencillo

Diseñaremos técnicas de recuperación de texto sencillo en las dos áreas siguientes: la desencriptación de paquetes TKIP y las *cookies* HTTPS. Variantes de nuestros métodos pueden utilizarse en otros escenarios.

4.1 El cálculo de estimados de probabilidad

Nuestro objetivo es convertir una secuencia de textos cifrados C en predicciones sobre texto sencillo. Esto se hace insertando enfoques en las distribuciones de flujos de clave $p_k = P_r [Z_r = k]$. Igual se puede lograr siguiendo los pasos de la sección 3.2. Todos los enfoques en p_k se utilizan para calcular la probabilidad de que un *byte* de texto sencillo sea igual a un valor μ determinado. Para llevar a cabo esto, contamos con los cálculos de probabilidades de AlFardan et al.[2] La idea es calcular, para cada valor μ de texto sencillo, las distribuciones de flujo de clave (inducidas) requeridas para revelar los textos cifrados capturados. Mientras más se acerquen al valor de la distribución de flujo de clave p_k, más cerca estaremos de obtener el *byte* de texto sencillo correcto. Asumiendo una posición determinada r, para simplificar, las distribuciones de flujos de clave se definirán por el vector $N^{\mu} = (N^{\mu}_0, \ldots, N^{\mu}_{255})$. Cada N^{μ}_k representa las veces que el *byte* de flujo de clave fue igual a k, asumiendo que el *byte* de texto sencillo sea μ:

$$N_k^{\mu} = |\{C \in \mathcal{C} \mid C = k \oplus \mu\}| \qquad (10)$$

Véase que los vectores N^{μ} y $N^{\mu'}$ son permutaciones entre sí. Basándonos en las probabilidades de flujo de clave p_k, podemos calcular la probabilidad de que esta distribución inducida pueda ocurrir en la práctica. Esto se modela empleando una distribución multinomial con un número de rutas (pistas) igual a $|C|$ y las categorías serán los 256 valores posibles del *byte* de flujo de clave. Como necesitamos las probabilidades de esta *secuencia* de *bytes* de flujo de clave, obtenemos: [30]

$$\Pr[\mathcal{C} \mid P = \mu] = \prod_{k \in \{0,\ldots,255\}} (p_k)^{N_k^{\mu}} \qquad (11)$$

Utilizando el teorema de Bayes podemos convertir esa ecuación en la probabilidad λ_{μ} de que el *byte* de texto sencillo sea μ:

$$\lambda_{\mu} = \Pr[P = \mu \mid C] \sim \Pr[C \mid P = \mu] \qquad (12)$$

Para nuestros propósitos, podemos considerar esta ecuación como una igualdad.[2] El *byte* de texto sencillo μ más probable es el que maximiza λ_{μ}. Esto se extendió a un par de *bytes* de flujo de clave dependiente de manera obvia:

$$\lambda_{\mu_1,\mu_2} = \prod_{k_1,k_2 \in \{0,\ldots,255\}} (p_{k_1,k_2})^{N_{k_1,k_2}^{\mu_1,\mu_2}} \qquad (13)$$

Encontramos que esta fórmula puede ser optimizada si la mayoría de los *bytes* de flujo de clave k_1 y k_2 son independientes y uniformes. Para mayor precisión, asumamos que todos los pares de valores de flujos clave en el lote $\mathcal{I}$ sean independientes y uniformes:

$$\forall (k_1, k_2) \in \mathcal{I}\colon\ p_{k_1,k_2} = p_{k_1} \cdot p_{k_2} = u \qquad (14)$$

Aquí u representa la probabilidad de un valor de *byte* doble de flujo de clave no enfocado. Entonces re-escribimos la fórmula[13] así:

$$\lambda_{\mu_1,\mu_2} = (u)^{M^{\mu_1,\mu_2}} \cdot \prod_{k_1,k_2 \in \mathcal{I}^c} (p_{k_1,k_2})^{N^{\mu_1,\mu_2}_{k_1,k_2}} \qquad (15)$$

donde

$$M^{\mu_1,\mu_2} = \sum_{k_1,k_2 \in \mathcal{I}} N^{\mu_1,\mu_2}_{k_1,k_2} = |\mathcal{C}| - \sum_{k_1,k_2 \in \mathcal{I}^c} N^{\mu_1,\mu_2}_{k_1,k_2} \qquad (16)$$

y con $\mathcal{I}^c$ como lote de valores de flujo de clave dependiente. Si el lote $\mathcal{I}^c$ es pequeño, resultará en una complejidad temporal menor. Por ejemplo, cuando se aplica a la configuración a largo plazo del enfoque Fluher-McGrew, aproximadamente se requieren 2^{19} operaciones para calcular todas las probabilidades estimadas, en vez de 2^{32}. Una optimización similar (aunque menos drástica) puede hacerse cuando están presentes enfoques de un solo *byte*.

4.2 Las probabilidades del enfoque Mantin

A continuación mostramos cómo calcular una probabilidad de *byte* doble de texto sencillo empleando el enfoque *ABSAB* de Mantin. Más formalmente, queremos calcular la probabilidad $\lambda_{\mu 1,\mu 2}$ de que los *bytes* de texto sencillo en las posiciones r y $r + 1$ son μ_1 y μ_2 respectivamente. Para lograr este objetivo se emplea texto sencillo conocido (known plaintext). Nuestra idea principal es primero calcular la probabilidades de las *diferencias* entre los textos sencillos conocidos y desconocidos. Definimos el diferencial $\widehat{Z}^g_r$ como:

$$\widehat{Z}^g_r = (Z_r \oplus Z_{r+2+g}, Z_{r+1} \oplus Z_{r+3+g}) \qquad (17)$$

De manera similar, empleamos $\widehat{C}^g_r$ y $\widehat{P}^g_r$ para denotar los diferenciales sobre el texto cifrado y los *bytes* de texto sencillo respectivamente. El enfoque *ABSAB* puede escribirse como:

$$\Pr[\widehat{Z}^g_r = (0,0)] = 2^{-16}(1 + 2^{-8}\mathrm{e}^{\frac{-4-8g}{256}}) = \alpha(g) \qquad (18)$$

Cuando se XORean ambos lados de $\widehat{Z}^g_r = (0,0)$ con $\widehat{P}^g_r$ obtenemos

$$\Pr[\widehat{C}^g_r = \widehat{P}^g_r] = \alpha(g) \qquad (19)$$

Por tanto, el enfoque de Mantin implica que el diferencial del texto cifrado está enfocado hacia el diferencial de texto plano. Esta idea se usa para calcular la probabilidad $\lambda_{\widehat{\mu}}$ de un diferencial $\widehat{\mu}$. Para aligerar la notación, asumimos una posición determinada r y un intervalo *ABSAB*

determinado de *g*. Definamos $\widehat{\mathcal{C}}$ como la secuencia de diferenciales de texto cifrado capturadas, y μ_1' y μ_2' como los *bytes* de texto plano conocidos en las posiciones $r + 2 + g$ y $r + 3 + g$, respectivamente.

Al igual que en nuestros estimados de probabilidades previos, calculamos la probabilidad de obtener diferenciales de texto cifrado $\widehat{\mathcal{C}}$ asumiendo que el diferencial de texto sencillo es $\widehat{\mu}$:

$$\Pr[\widehat{\mathcal{C}} \mid \widehat{P} = \widehat{\mu}] = \prod_{\widehat{k} \in \{0,\ldots,255\}^2} \Pr[\widehat{Z} = \widehat{k}]^{N_{\widehat{k}}^{\widehat{\mu}}} \qquad (20)$$

donde

$$N_{\widehat{k}}^{\widehat{\mu}} = \left| \left\{ \widehat{C} \in \widehat{\mathcal{C}} \mid \widehat{C} = \widehat{k} \oplus \widehat{\mu} \right\} \right| \qquad (21)$$

Empleando esta notación observamos que en realidad es idéntica a una estimación de probabilidades normal (ordinaria). Mediante el teorema de Bayes se obtiene $\lambda_{\widehat{\mu}} = \Pr[\widehat{\mathcal{C}} \mid \widehat{P} = \widehat{\mu}]$. Como el único par diferencial está enfocado, podemos aplicar y simplificar la fórmula.[15]

$$\lambda_{\widehat{\mu}} = (1 - \alpha(g))^{|\mathcal{C}| - |\widehat{\mu}|} \cdot \alpha(g)^{|\widehat{\mu}|} \qquad (22)$$

Alteraremos ligeramente en ella la notación definiendo $|\widehat{\mu}|$ como

$$|\widehat{\mu}| = \left| \left\{ \widehat{C} \in \widehat{\mathcal{C}} \mid \widehat{C} = \widehat{\mu} \right\} \right| \qquad (23)$$

Por último, aplicamos lo que sabemos de los *bytes* de texto plano conocidos para obtener el estimado de probabilidades deseado:

$$\lambda_{\mu_1,\mu_2} = \lambda_{\widehat{\mu} \oplus (\mu_1', \mu_2')} \qquad (24)$$

Para estimar en cuál tamaño de intervalo el enfoque *ABSAB* es aún detectable, generamos 2^{48} bloques de 512 *bytes* de flujo de clave. Sobre esta base queda empíricamente confirmado que el enfoque *ABSAB* de Mantin ocupa tamaños de intervalos de al menos 135 *bytes*. El estimado teórico de la fórmula (1) infravalora ligeramente el enfoque empírico verdadero. En nuestros ataques usamos un tamaño de intervalo máximo de 128.

4.3 La combinación de estimados de probabilidad

Nuestro objetivo es combinar múltiples tipos de enfoques en un cálculo de probabilidades. Por desgracia, si los enfoques cubren posiciones coincidentes, no será factible hacer una estimación de probabilidades sobre todos los *bytes*. En el peor de los casos, el cálculo no podrá ser optimizado contando con los enfoques independientes. De ahí que, un estimado sobre las posiciones de flujos de clave *n* tendría una complejidad temporal de $\mathcal{O}(2^{2 \cdot 8 \cdot n})$. Para superar este problema, combinaremos y realizaremos múltiples estimados de probabilidades por separado.

Vamos a combinar varios tipos de enfoques multiplicando sus estimados de probabilidades individuales. Por ejemplo, sea $\lambda'_{\mu 1,\mu 2}$ la probabilidad de *bytes* de texto plano μ_1 y μ_2 basados en los enfoques de Fluher-McGrew. De manera similar, sean $\lambda'_{g,\mu 1,\mu 2}$ las probabilidades derivadas de los enfoques *ABSAB* del intervalo *g*. Entonces su combinación será sencilla:

$$\lambda_{\mu_1,\mu_2} = \lambda'_{\mu_1,\mu_2} \cdot \prod_g \lambda'_{g,\mu_1,\mu_2} \qquad (25)$$

Aunque este método pudiera no ser óptimo cuando se combinan probabilidades de *bytes* dependientes, parece ser una posibilidad general y poderosa. Un problema abierto es la determinación de cuáles enfoques pueden ser combinados en un solo cálculo de probabilidades, manteniendo los requisitos computacionales aceptables. Probabilidades basabas en otros enfoques, como por ejemplo los de Sen Gupta y nuestros enfoques nuevos a largo plazo, pueden añadirse como otro factor (aunque se necesita de cuidado para que las posiciones coincidan correctamente).

Para verificar la efectividad de esta aproximación, se realizaron simulaciones donde se intentó desencriptar dos *bytes* empleando un estimado de probabilidad de *byte* doble. Primero se hizo aplicando solo los enfoques de Fluher-McGrew y un solo enfoque *ABSAB*. Luego se combinaron 2·129 enfoques *ABSAB* y los enfoques Fluher-McGrew, usando el método de la sección 4.2 para derivar las probabilidades de los enfoques *ABSAB*. Asumimos que los *bytes* desconocidos están rodeados de ambos lados por texto sencillo conocido, y utilizamos un intervalo *ABSAB* máximo de 128 *bytes*.

La figura 7 muestra los resultados de este experimento. Véase que un solo enfoque *ABSAB* es más débil que el empleo todos los enfoques Fluher-McGrew en una posición específica. De cualquier forma, la combinación de varios enfoques *ABSAB* resulta en una mejoría considerable. Concluimos que nuestro método de combinar enfoques reduce significativamente el número de textos cifrados necesarios.

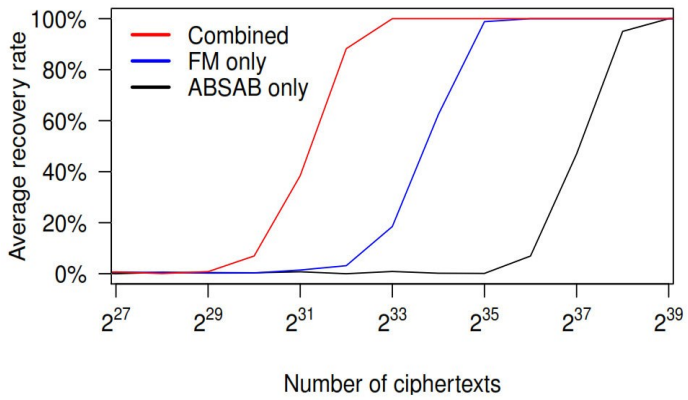

Figura 7: Promedio de éxito en la desencriptación de dos *bytes* empleando: (1) un enfoque *ABSAB*; (2) enfoques Fluher-McGrew (FM); y (3) una combinación de los enfoques FM con 258 enfoques *ABSAB*. Los resultados están basados en 2048 simulaciones.

4.4 Listados de candidatos de texto sencillo

En la práctica es útil tener una lista de candidatos de texto sencillo con probabilidad decreciente. Por ejemplo, con ella se desencriptan las palabras claves (*passwords*), *cookies*, etc. (véase sección 6). En otras situaciones, el texto sencillo pudiera tener una estructura rígida permitiendo

la eliminación de candidatos (véase sección 5). Se generará una lista de candidatos de texto plano en orden decreciente cuando se dé una probabilidad de *byte* único o de doble *byte* indistintamente.

Primero mostramos cómo construir una lista de candidatos cuando se den las probabilidades de texto plano de *byte* único. Es insignificante generar los dos candidatos más probables, pues más allá de este punto el cálculo se hace tedioso. Nuestra solución es calcular incrementalmente la cantidad de candidatos más probables N basándonos en su longitud. Es decir, primero calculamos la cantidad de candidatos más probables N de longitud 1, luego los de longitud 2 y así sucesivamente. El algoritmo 1 ofrece una implementación de alto nivel de esta idea. La variable $P_r[i]$ denota la probabilidad i de texto plano de longitud r, con probabilidad $E_r[i]$. Las dos operaciones `min` son necesarias porque en las vueltas iniciales no se puede generar aun los candidatos N, es decir, solo existen 256^r textos sencillos de longitud r. La elección de μ' que maximiza $pr(\mu')$ puede hacerse eficientemente utilizando una cola de espera de prioridades (priority queue).

En la práctica, solo las últimas dos versiones de las listas E y P necesitan ser guardadas. Para mantener una mejor estabilidad numérica y hacer el cálculo más eficiente, realizamos operaciones usando los logaritmos de las probabilidades. Implementamos el Algoritmo 1 y hacemos un reporte de su funcionamiento en la sección 5, donde lo empleamos para atacar una red inalámbrica protegida por WPA-TKIP.

Algorithm 1: Generate plaintext candidates in decreasing likelihood using single-byte estimates.

Input: L : Length of the unknown plaintext
$\lambda_{1\le r\le L,\,0\le\mu\le 255}$: single-byte likelihoods
N: Number of candidates to generate
Returns: List of candidates in decreasing likelihood

$P_0[1] \leftarrow \varepsilon$
$E_0[1] \leftarrow 0$

for $r = 1$ **to** L **do**
  for $\mu = 0$ **to** 255 **do**
    $pos(\mu) \leftarrow 1$
    $pr(\mu) \leftarrow E_{r-1}[1] + \log(\lambda_{r,\mu})$
  for $i = 1$ **to** $\min(N, 256^r)$ **do**
    $\mu \leftarrow \mu'$ which maximizes $pr(\mu')$
    $P_r[i] \leftarrow P_{r-1}[pos(\mu)] \,||\, \mu$
    $E_r[i] \leftarrow E_{r-1}[pos(\mu)] + \log(\lambda_{r,\mu})$
    $pos(\mu) \leftarrow pos(\mu) + 1$
    $pr(\mu) \leftarrow E_{r-1}[pos(\mu)] + \log(\lambda_{r,\mu})$
    if $pos(\mu) > \min(N, 256^{r-1})$ **then**
      $pr(\mu) \leftarrow -\infty$
return P_N

Longitud del texto sencillo desconocido
Probabilidades de *byte* único
Número de candidatos a generar
Lista de candidatos en probabilidad decreciente

Algoritmo 1: Generador de candidatos de texto plano con probabilidad decreciente usando estimados de *byte* único.

Para generar una lista de candidatos con probabilidades de *byte* doble, primero mostramos como modelar las probabilidades sobre la base del modelo de Markov oculto (HMM). Esto permite presentar una versión más intuitiva de nuestro algoritmo y remitir a una búsqueda extensiva en esta área, en caso de que sean necesarias más implementaciones. El algoritmo que presentamos

puede ser visto como una combinación del clásico algoritmo Viterbi y el algoritmo 1 ya planteado. Aunque no es el óptimo, es suficiente para mejorar la recuperación de texto sencillo (véase sección 6). Para una introducción a los modelos HMM el lector puede acudir al tutorial de L. Rabiner.[35] Un HMM esencialmente modela un sistema donde los estados internos no son visibles, y luego de la transición de cada estado, el resultado se produce (probablemente) en dependencia del estado nuevo.

Modelamos los estimados de probabilidad de texto sencillo como modelos HMM homogéneos de primer orden. El espacio de estado *S* del HMM está definido por un lote de valores de textos sencillo posibles {0,…, 255}. Las posiciones de los *bytes* se modelaron empleando las probabilidades de estados de transición dependientes temporalmente (homogéneas). Intuitivamente, "el tiempo real" en el HMM se corresponde con la posición actual del texto sencillo. Esto significa que las probabilidades de transición para moverse de un estado a otro, normalmente basadas en el tiempo real, dependerán ahora de la posición del *byte*. Véase más formalmente:

$$\Pr[S_{t+1} = \mu_2 \mid S_t = \mu_1] \sim \lambda_{t,\mu_1,\mu_2} \qquad (26)$$

En esta ecuación *t* representa el tiempo. Para nuestros propósitos, podemos considerarla una igualdad. En un modelo HMM se asume que su estado actual no es visible. Ello se debe al hecho que no se conoce el valor de ningún *byte* de texto sencillo. No obstante, en ese modelo hay una vía de mostrar los resultados que dependen del estado actual. En nuestra configuración, un valor particular de texto sencillo filtra (leaks) información no visible (canal lateral). Esto se modela permitiendo siempre que cada estado produzca el mismo resultado nulo con probabilidad 1.

Utilizando el modelo HMM anterior, la búsqueda del texto sencillo más probable se reduce a encontrar la secuencia de estado más probable. Ello se resuelve a través del conocido algoritmo Viterbi. Debe comentarse que el algoritmo presentado por AlFardan et al. se asemeja grandemente al de Viterbi. [2] De igual forma, detectar variable de los textos sencillos *N* más probables es lo mismo que buscar las secuencias de estado *N* más probables. De ahí que cualquier variante *N* mejor del algoritmo Viterbi (también llamada Lista del algoritmo Viterbi) puede ser empleada, véanse los ejemplos [42, 36, 40, 28]. La forma más simple de ese algoritmo recoge la lista de mejores candidatos de texto sencillo *N*, finalizando con un valor particular μ, tal como se muestra en el algoritmo 2 de la página siguiente. Al igual que Alfardan et al. y Patterson et al., asumimos que el primer *byte* m_1 y el último m_L de texto sencillo son conocidos. Durante el último intento, la vuelta sobre μ_2 tiene que ser ejecutada solo para el valor m_L. En la sección 6 utilizamos este algoritmo para generar una lista de *cookies*.

Algorithm 2: Generate plaintext candidates in decreasing likelihood using double-byte estimates.

Input: L : Length of the unknown plaintext plus two
m_1 and m_L: known first and last byte
$\lambda_{1\leq r<L,\, 0\leq\mu_1,\mu_2\leq 255}$: double-byte likelihoods
N: Number of candidates to generate
Returns: List of candidates in decreasing likelihood

for $\mu_2 = 0$ **to** 255 **do**
  $E_2[\mu_2, 1] \leftarrow \log(\lambda_{1,m_1,\mu_2})$
  $P_2[\mu_2, 1] \leftarrow m_1 \,||\, \mu_2$

for $r = 3$ **to** L **do**
  for $\mu_2 = 0$ **to** 255 **do**
    for $\mu_1 = 0$ **to** 255 **do**
      $pos(\mu_1) \leftarrow 1$
      $pr(\mu_1) \leftarrow E_{r-1}[\mu_1, 1] + \log(\lambda_{r,\mu_1,\mu_2})$
    for $i = 1$ **to** $\min(N, 256^{r-1})$ **do**
      $\mu_1 \leftarrow \mu$ which maximizes $pr(\mu)$
      $P_r[\mu_2, i] \leftarrow P_{r-1}[\mu_1, pos(\mu_1)] \,||\, \mu_2$
      $E_r[\mu_2, i] \leftarrow E_{r-1}[\mu_1, pos(\mu_1)] + \log(\lambda_{r,\mu_1,\mu_2})$
      $pos(\mu_1) \leftarrow pos(\mu_1) + 1$
      $pr(\mu_1) \leftarrow E_{r-1}[\mu_1, pos(\mu_1)] + \log(\lambda_{r,\mu_1,\mu_2})$
      if $pos(\mu_1) > \min(N, 256^{r-2})$ **then**
        $pr(\mu_1) \leftarrow -\infty$

return $P_N[m_L, :]$

Algoritmo 2: Generador de candidatos de texto plano con probabilidad decreciente usando estimados de doble *byte*.

El algoritmo 2 consume considerablemente más memoria que el algoritmo 1. Ello se debe a que debe guardar la cantidad de candidatos más probables N para cada posible valor terminal μ. Recordamos a los lectores que nuestro propósito no es presentar el algoritmo óptimo, sino mostrar cómo modelar el problema como un HMM, basándonos en investigaciones relacionadas en esta área para encontrar algoritmos más eficientes [42, 36, 40, 28]. Como el modelo HMM se puede ilustrar como un gráfico, todos los algoritmos de trayectoria k más corta también son aplicables.[10] Finalmente, puede comentarse que incluso la variante más simple fue suficiente para mejorar significativamente los índices de recuperación de texto sencillo (véase sección 6).

5 El ataque al protocolo WPA-TKIP

Empleamos nuestras técnicas de recuperación de texto sencillo para desencriptar un paquete completo. Del paquete desencriptado se deriva la clave MIC, permitiendo al atacante inyectar y desencriptar los demás paquetes. El ataque se ejecuta en la práctica en solo una hora.

5.1 El cálculo de las probabilidades de texto sencillo

Nos apoyamos en el ataque de Paterson et al. para calcular los estimados de probabilidad de texto sencillo [31,30]. Estos investigadores detectaron que los primeros tres *bytes* de cada clave RC4 por paquete son públicos. Como se explicó en la sección 2.2., los primeros tres *bytes* se determinan completamente a través del contador de secuencia TKIP (TSC). Se observó que esta dependencia causa muchos enfoques dependientes del TSC en el flujo de clave [31,15, 30], los cuales pueden utilizarse para mejorar los estimados de probabilidad de texto sencillo. Para cada valor TSC se calcularon las probabilidades de texto sencillo basadas en distribuciones de texto sencillo por TSC empíricas. Las 256^2 probabilidades resultantes para un *byte* de texto sencillo,

se combinaron multiplicándolas por todos los pares TSC. De cierta manera, esto es lo mismo que combinar varios tipos de enfoques, tal como se hizo en la sección 4.3, aunque aquí los diferentes tipos de enfoques se reconocen como independientes. Empleamos la variante de *bytes* sencillos del ataque [30,§ 4.1] para obtener posibilidades $\lambda_{r,\mu}$ de cada *byte* desconocido en una posición determinada *r*.

La desventaja de este procedimiento es que requiere estadísticas detalladas de los flujos de clave por TSC. Paterson et al. generaron estadísticas para los primeros 512 *bytes*, lo cual tomó 30 años de CPU[30]. Como quiera, en nuestro ataque solo las necesitamos para los pocos primeros *bytes* de flujo de clave. Usamos 2^{32} claves por valor TSC para estimar la distribución de flujo de clave para los 128 *bytes* iniciales. Con nuestra configuración distribuida, la generación de esas estadísticas tomo 10 años de CPU.

Mediante las distribuciones de flujo de clave por TSC obtuvimos resultados similares a Paterson et al. [31, 30]. A través de simulaciones confirmamos que las posiciones inusuales de *bytes*[30], en vez de las esperadas[31], pueden ser recuperadas con una probabilidad mayor que otras. De igual forma, los *bytes* en las posiciones 49-51 y 63-67 son recuperados generalmente con mayor probabilidad también. Ambas observaciones se tendrán en cuenta para optimizar el ataque en la práctica.

5.2 La inyección de paquetes idénticos

A continuación mostramos como cumplir el primer requisito de un ataque exitoso: la generación de paquetes idénticos. Si se conoce el IP de la víctima y las conexiones de entrada no están bloqueadas, simplemente se envían los paquetes idénticos. Si no es el caso, se induce a la víctima a que abra una conexión TCP hacia el servidor que controla el ataque. La misma se utilizará para trasmitir los paquetes idénticos. Otra forma sencilla de lograr esto es invitando a la víctima a que visite un sitio *web* administrado por el agresor. El navegador abrirá una conexión TCP para cargar el sitio *web*.

Sin embargo, pueden emplearse métodos más sofisticados con un rango de objetivos más amplio. Uno sería la inclusión forzada de recursos de terceros (inseguros) en sitios *web* populares.[27] Por ejemplo, el atacante puede registrar un dominio mal escrito, utilizado accidentalmente en una dirección de recursos (como pudiera ser un URL imagen) de un sitio *web* popular. Cada vez que la víctima visite el sitio *web* y cargue el recurso, una conexión TCP se establece con el servidor del atacante. Estos tipos de vulnerabilidades se encontraron en muchos sitios *webs* populares.[27] Adicionalmente, cualquier tipo de vulnerabilidad que obligue a la víctima a ejecutar código JavaScript puede ser efectiva. En este sentido, nuestros requisitos son más sencillos que los necesarios en los recientes ataques sobre SSL y TLS, los cuales demandan la habilidad de ejecutar JavaScript en el navegador de la víctima [9,1,2]. Otro método es secuestrar una conexión TCP de la víctima, lo cual bajo ciertas condiciones es posible sin necesidad de una posición intermedia (man-in-the middle position). Se concluye que aunque no existan métodos universales para ejecutar este ataque, se puede asumir el control de la conexión TCP de la víctima. A través de ella se retrasmiten repetidamente paquetes TCP idénticos. Como las retrasmisiones son válidas en el comportamiento de TCP, esto puede funcionar incluso si la víctima se protege con *firewall* (cortafuegos).

Ahora determinaremos la estructura óptima del paquete inyectado. Una aproximación fácil sería usar el paquete más corto posible, es decir, que no incluya carga útil TCP. Como el tamaño total del IP, LLC/SNAP y la cabecera TCP es de 48 *bytes*, el MIC e ICV estarían ubicados en la

posición 49 en adelante, incluyendo la 60 (véase figura 2). En estas locaciones 7 *bytes* están enfocados acertadamente. En contraste, si usamos una carga útil TCP de 7 *bytes*, el MIC e ICV se ubicarían de la posición 56 en adelante, incluyendo la 60. En este rango 8 *bytes* estarán enfocados acertadamente, resultando en mejores estimados de probabilidad de flujo de clave. Mediante simulaciones confirmamos que empleando una carga útil de 7 *bytes* aumenta la probabilidad de desencriptar exitosamente el MIC e ICV. En la práctica, la adición de 7 *bytes* de carga útil también hace única la longitud del paquete inyectado. Resultando en una identificación y captura más fácil de esos paquetes. Por estas ventajas es que usamos paquetes de 7 *bytes* de carga útil de datos TCP.

5.3 Desencriptando un paquete completo

El objetivo es desencriptar el paquete TCP inyectado, incluyendo sus campos MIC e ICV. Nótese que todos los paquetes estarán encriptados con una clave RC4 diferente. Por ahora, asumimos que todos los campos en el IP y el paquete TCP son conocidos, más tarde mostraremos el porqué de esta afirmación. Solo el MIC de 8 *bytes* y el ICV de 4 *bytes* del paquete permanecerán desconocidos. Empleamos las estadísticas de flujo de clave por TSC para calcular las probabilidades de texto sencillo de un *byte* único para todos los 12 *bytes*. De cualquier forma, esto daría pocas probabilidades de éxito para desencriptar correctamente todos los *bytes* al mismo tiempo. La solución está en darse cuenta que el ICV TKIP es un CRC checksum simple que puede verificarse fácilmente. Por tanto, podemos detectar candidatos falsos inspeccionando sus CRC checksum. Luego generamos una lista de candidatos de texto sencillo y la analizamos hasta encontrar un paquete con el CRC correcto. Esto mejora grandemente la probabilidad de desencriptar simultáneamente todos los *bytes*. Del paquete desencriptado podemos derivar la clave MIC TKIP,[44] que luego puede utilizarse para inyectar y desencriptar paquetes arbitrarios.[48]

La figura 8 muestra el índice de éxito de la detección de un paquete con buen ICV y la derivación de clave MIC correcta. Para comparar, se incluye también el mismo índice pero obtenido solo con los dos candidatos más probables. La figura 9 presenta la posición media del primer candidato con ICV correcto. Trazamos la media, en vez del promedio, para reducir la influencia de errores, pues por ejemplo, a veces el candidato correcto se encuentra inesperadamente lejos (o cerca) en la lista resultante.

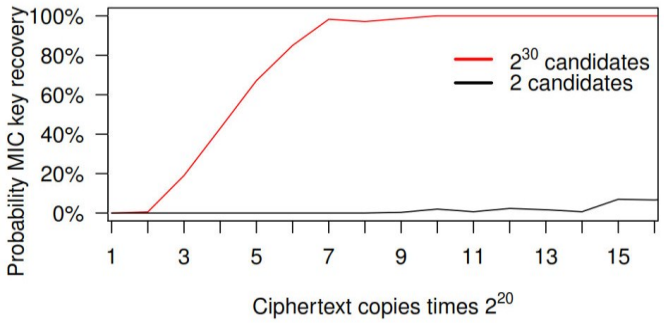

Figura 8: Índice de éxito en la detección de clave MIC TKIP usando 2^{30} candidatos, y utilizando solos los dos mejores candidatos. Los resultados se basan en 256 simulaciones.

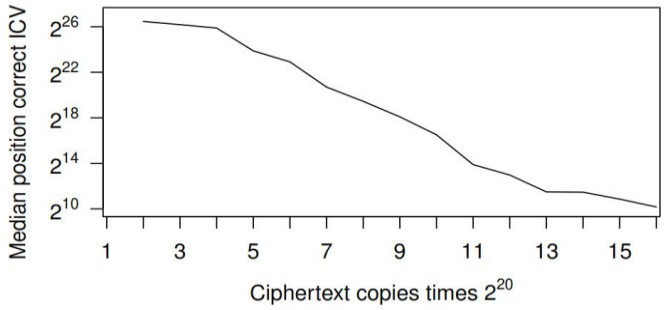

Figura 9: Posición media de un candidato con ICV correcto entre aproximadamente 2^{30} candidatos. Los resultados se basan en 256 simulaciones.

No obstante, se mantiene la pregunta de cómo determinar los contenidos de los campos desconocidos en el IP y el paquete TCP. Para mayor precisión, los campos desconocidos son el IP interno y el puerto del cliente, y el campo TTL del IP **(IP time-to-live field)**. Debe aclararse que la cabecera del IP y el TCP contienen checksums. Por tanto, se puede aplicar la misma técnica anterior (generación y desestimación de candidatos) para derivar los valores de estos campos con altos índices de éxito. Puede hacerse de manera independiente de cada uno, e independientemente de la desencriptación del MIC e ICV.

Otro método para obtener el IP interno es apoyándonos en las ventajas del HTML5. Si la conexión TCP inicial se crea en el navegador, puede primero enviarse código JavaScript para obtener el IP interno de la víctima mediante WebRTC.[37] También apreciamos que nuestro puerto NAT no modificó el puerto fuente de la víctima (sencillamente podemos leer este valor en el servidor). El campo TTL puede determinarse también sin depender del checksum del IP. Con el comando `traceroute` se puede contar el número de saltos (hops) entre el servidor y el cliente, permitiendo derivar el valor TTL de la víctima.

5.4 Evaluación empírica

Para evaluar la fase de recuperación de texto sencillo de nuestro ataque creamos una herramienta que genera un archivo `pcap` contenedor de los paquetes de Wi-Fi capturados. Esta herramienta busca los paquetes inyectados, extrae las estadísticas de los textos cifrados, calcula las probabilidades de texto sencillo y encuentra el candidato con ICV correcto. De este último, es decir, del paquete inyectado desencriptado, derivamos la clave MIC.

Para la fase de generación de texto cifrado empleamos un Open VZ VPS como servidor malicioso. La señal entrante de la víctima se manipuló con una herramienta escrita en Scapy, que cuenta con una versión crackeada de `Tcpreplay` para inyectar rápidamente los paquetes TCP idénticos. La máquina víctima es una Latitude E6500 conectada a un router ASUS RT-N10 ejecutando Tomato 1.28. La víctima abre la conexión TCP al servidor malicioso visitando un sitio *web* administrado por él. La máquina atacante fue una Compaq 8510p con un AWUS036nha para capturar el tráfico inalámbrico. Bajo esta configuración pudimos generar aproximadamente 2500 paquetes por segundo. Esta cantidad se alcanzó inclusive mientras la víctima cargaba videos de YouTube activamente. Gracias a la carga útil de 7 *bytes* pudimos detectar el paquete inyectado en todos los experimentos sin falsos positivos.

Ejecutamos varias pruebas en las que generamos y capturamos tráfico alrededor de una hora. Ello permitió, como promedio, capturar $9.5 \cdot 2^{20}$ encriptaciones diferentes del paquete inyectado. Las retrasmisiones fueron filtradas sobre la base del TSC del paquete. En casi todos los casos desencriptamos exitosamente el paquete y derivamos la clave MIC. Tal como se explicó en la

sección 2.2, la clave MIC es válida hasta tanto la víctima no renove su PTK, y puede ser usada para inyectar y desencriptar desde la AP hasta la víctima. Durante una captura nuestra herramienta detectó un paquete con un ICV correcto, pero ese candidato no se correspondió con el texto sencillo verdadero. Aunque nuestra evaluación actual se limita al número de capturas realizadas, se muestra que el ataque es prácticamente factible, con una probabilidad de éxito que en general coincide con los resultados simulados en la figura 8.

6 Desencriptando las *cookies* HTTPS

Se inyectan datos conocidos alrededor de una *cookie*, permitiendo el uso de los enfoques *ABSAB*. Luego demostramos que la *cookie* HTTPS puede ser atacada utilizando solo 75 horas de texto cifrado.

6.1 La inyección de texto sencillo conocido

Necesitamos predecir la posición de la *cookie*-objetivo en la solicitud encriptada de HTTP y atacarla con texto sencillo conocido. Para asegurarnos, enviamos una comprobación de la *cookie* segura a https://site.com. Al igual que en los ataques anteriores a SSL y TLS, asumimos que el agresor puede ejecutar código JavaScript en el navegador de la víctima [9,1,2]. En nuestro caso, ello significa que se emplea una posición intermedia activa (MiTM) (**man-in-the-middle position**), donde los canales HTTP de texto sencillo puedan ser manipulados. Primero nos dimos cuenta que el atacante puede predecir la longitud y el contenido de las cabeceras HTTP precedentes al campo de la *cookie*. A través del monitoreo de las solicitudes HTTP de texto sencillo las cabeceras pueden ser descubiertas. Si la *cookie*-objetivo es el primer valor en la cabecera de la *cookie*, ello implica que conocemos su posición en la solicitud HTTP. Por eso, nuestra meta es confeccionar un diseño tal como se muestra en la lista 3.

```
1 GET / HTTP/1.1
2 Host: site.com
3 User-Agent: Mozilla/5.0 (X11; Linux i686; rv:32.0)
      Gecko/20100101 Firefox/32.0
4 Accept: text/html,application/xhtml+xml,application/
      xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Cookie: auth=XXXXXXXXXXXXXXXX; injected1=known1;
      injected2=knownplaintext2; ...
```

Lista 3: Solicitud de HTTP manipulada, con texto sencillo conocido rodeando ambos lados de la *cookie*.

Aquí la *cookie*-objetivo es el primer valor de la *cookie*, precedido por cabeceras conocidas, y seguidas por *cookies* inyectadas por el atacante.

Para obtener el diseño de la lista 3, empleamos nuestra posición MiTM para redireccionar la víctima a https://site.com, o lo que es lo mismo, al sitio web creado sobre un canal HTTP inseguro. Si el sitio *web* utiliza **Seguridad de Transporte Estricta HTTP** (HSTS), pero no el atributo `includeSubDomains`, aún es posible redirigir a la víctima a un (falso) subdominio.[6] Ello se debe a que pocos sitios *web* usan HSTS y aún menos saben emplearlo correctamente,[47] por lo que es muy probable que el redireccionamiento sea exitoso. Como las *cookies* seguras

garantizan la confidencialidad pero no la integridad, el canal inseguro HTTP puede utilizarse para sobreescribir, borrar e inyectar *cookies* seguras [3, 4.1.2.5]. Esto permite borrar todas las *cookies* excepto la correcta, que se ubicará en el primer lugar de la lista. Luego podemos seguir inyectando *cookies* que se incluirán después de la primera. Un ejemplo de solicitud HTTP(S) manipulada de esta forma se mostró en la lista 3. Allí la *cookie*-objetivo asegurada se rodea a ambos lados con texto sencillo conocido. Lo cual permite usar los enfoques *ABSAB* de Mantin para calcular las probabilidades de texto sencillo.

6.2 El ataque a la *cookie*

Lo común es que los sitios *web* empleen palabras claves como única medida de seguridad, por lo que casi nunca se protegen contra los ataques a sus *cookies*. Ello se debe a que las palabras claves de los usuarios promedios tienen una entropía mucho más baja que la de una *cookie* aleatoria. Es decir, tiene más sentido tratar de desencriptar las palabras claves que las *cookies*, pues la posibilidad de desencriptar una *cookie* (correctamente generada) es casi igual a cero. No obstante, si se utiliza el RC4 para conectarse al servidor *web*, nuestro candidato de algoritmo de generación anula esa posibilidad. La lista de candidatos de texto sencillo es una vía factible para atacar las *cookies*.

Como el objetivo es desencriptar una *cookie*, podemos excluir determinados valores de texto plano. Tal como dice RFC 6265, el valor de una *cookie* consiste por lo general en 90 caracteres únicos [3, § 4.1.1]. Una observación similar, aunque menos específica, fue hecha por AlFardan et al.[2] Nuestro acercamiento permite ofrecer una aproximación más exacta sobre la cantidad de texto cifrado requerida para desencriptar la *cookie*. En la práctica, el ataque con un lote reducido de caracteres se ejecutó modificando el algoritmo 2 para que los valores μ_1 y μ_2 solamente se "muevan" entre caracteres permitidos.

La figura 10 muestra el índice de éxito del ataque a una *cookie* de 16 caracteres mediante 2^{23} intentos. Para poder establecer comparaciones, también incluimos en la figura la probabilidad de desencriptar la *cookie* usando solo el texto sencillo más probable. Esto permite compararla a la vez más fácilmente con la investigación de AlFardan et al [2]. Nótese que estos solo utilizan los enfoques Fluher-McGrew, mientras que aquí combinamos varios enfoques *ABSAB*, junto a los de Fluher-McGrew. Puede concluirse que nuestro intento de ataque, mediante la inclusión de los enfoques ABSAB, mejora significativamente la posibilidad de éxito. Incluso ejecutando solo 2^{23} intentos, se obtuvo más del 94% de éxito una vez que las $9 \cdot 2^{27}$ encriptaciones de la *cookie* fueron capturadas. Podemos suponer que generando más candidatos podrían incrementarse los índices de éxito.

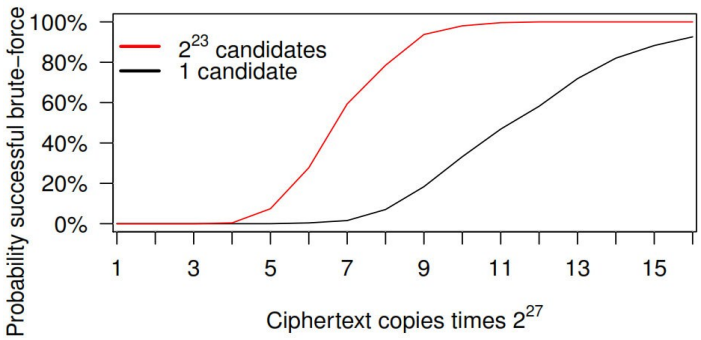

Figura 10: Ataque exitoso a una *cookie* de 16 caracteres usando aproximadamente 2^{23} candidatos, y el candidato más probable, dependiendo del número de texto cifrado recopilado. El resultado se basa en 256 simulaciones.

6.3 Evaluación empírica

El requisito principal de este ataque es poder recopilar suficientes encriptaciones de la *cookie*, o lo que es lo mismo, conseguir la mayor cantidad de textos cifrados posible. Esto pudo cumplirse forzando a la víctima a generar un gran número de solicitudes HTTPS. Como en los ataques previos sobre TLS [9,1,2], logramos esta tarea asumiendo que el agresor puede ejecutar JavaScript en el navegador de la víctima. Por ejemplo, mientras se ejecuta un ataque MiTM, puede inyectarse JavaScript en cualquier conexión HTTP. Luego, empleamos la opción `XMLHttpRequest` para generar una Solicitud de Origen Cruzado (**Cross-Origin Request**) en el sitio *web* deseado. El navegador añadirá automáticamente la *cookie* segura a estas solicitudes encriptadas. Debido a la restricción del origen similar (**same-origin policy**) no podemos leer las respuestas, pero esto no es problema, solamente se necesita que la *cookie* esté incluida en la solicitud. Las solicitudes serán enviadas dentro de los HTML5 WebWorkers. Esto significa en esencia, que nuestro JavaScript se estará ejecutando de manera oculta en el navegador, y cualquier página abierta responderá positivamente. A través de solicitudes GET, concebimos cuidadosamente los valores de nuestras *cookie*s inyectadas, de manera que la *cookie* principal esté siempre en una posición determinada del flujo de clave (modulo 256). Recuérdese que este ajuste es necesario para poder utilizar de manera óptima los enfoques Fluher-McGrew. El atacante puede saber la cantidad requerida de paquetes (*padding*) dejando que sea primero el cliente quien haga una solicitud sin paquetes. Como el RC4 es un encriptador de flujo, y el protocolo TLS no añade ningún paquete, el atacante puede observar fácilmente la longitud de la solicitud.

Para comprobar nuestro ataque en la práctica empleamos una herramienta escrita en C, ella monitorea el tráfico de red y recopila las estadísticas de texto cifrado necesarias. Esto requiere un re-montaje de los flujos TCP y TLS, para luego detectar las solicitudes HTTP de 512 *bytes* (encriptadas). De la misma manera que optimizamos la generación de lotes de datos en la sección 3.2, guardamos algunas solicitudes antes de actualizar los contadores. También se creó una herramienta para atacar las *cookies*, basada en una lista de candidatos generada. Con ella se envía a través de la conexión HTTP múltiples solicitudes sin necesidad de esperar por cada una de las respuestas.

En nuestro experimento la víctima utilizó un CPU Core i5-2400 a 3.1 GHz y 8 GB de RAM, ejecutando Windows 7. El navegador fue Internet Explorer 11. El servidor corrió en un CPU Core i7-3770 a 3.4 GHz y 8GB de RAM. Se empleó `nginx` como servidor *web* configurado solo con una suite de cifrado RSA y RC4-SHA1. Esto asegura que se utilice el RC4 en todas las pruebas. Tanto el cliente como el servidor usaron una tarjeta de red Intel 82579LM, con velocidad de conexión ajustada a 100 Mbps. Con el navegador desocupado, esta configuración consiguió un promedio de 4450 solicitudes por segundo. Mientras la víctima cargaba activamente videos de YouTube, esa cantidad decreció aproximadamente a 4100. Para alcanzar esas cifras es esencial que el navegador tenga conexiones persistentes para trasmitir las solicitudes HTTP. Si no es así, para cada solicitud debiera generarse un nuevo protocolo de intercambio TCP y TLS, ralentizando significativamente el tráfico. El navegador respondió positivamente todo el tiempo durante la generación de solicitudes. Por último, nuestra herramienta fue capaz de comprobar más de 20000 *cookies* por segundo. Para ejecutar el ataque con un índice de éxito del 94% se necesitaron aproximadamente $9 \cdot 2^{27}$ textos cifrados. Con 4450 solicitudes por segundo se requirieron 75 horas de datos. Si se comparan con las 2000 horas de la prueba de AlFardan et al. [2, § 5.3.3], nuestro experimento representa una mejora sustancial. Debe enfatizarse que, al igual que en la prueba de los investigadores anteriores, nuestro ataque

tolera también cambios en las claves de encriptación. Por tanto, como las *cookie*s pueden tener un tiempo de vida largo, la generación de tráfico puede extenderse temporalmente. Con 20000 ataques por segundo, todos los 2^{23} candidatos para la *cookie* pueden ser probados en menos de 7 minutos.

Se ejecutó el ataque en la práctica y se ha desencriptó exitosamente una *cookie* de 16 caracteres. En nuestra instancia, la captura de tráfico durante 52 horas resultó ser suficiente. Se recopilaron $6.2 \cdot 2^{27}$ textos cifrados. Luego de procesarlos, se encontró la *cookie* en la posición 46229 de la lista de candidatos. Esto constituye un ejemplo acertado de que, si el atacante tiene suerte, se necesitaría menos que nuestro estimado de 9.2^{27} de texto cifrado. Estos resultados posicionan nuestro ataque casi al punto de la factibilidad.

7 Investigaciones relacionadas

El RC4 por su popularidad ha sido sometido a un amplio análisis criptográfico. Los ataques de recuperación de claves contra WEP [12,50,45,44,43] son particularmente bien conocidos. También se han estudiado mucho los enfoques basados en claves y otras mejoras partiendo del ataque original contra WEP [21,39,32,22].

Remitimos a la sección 2.1 de este artículo para una perspectiva de los múltiples enfoques detectados en el flujo de clave [25,23,38,2,20,33,13,24,38,15,31,30]. Además de estos, el enfoque a largo plazo $P_r[Z_r = Z_{r+1} \mid 2 \cdot Z_r = i_r] = 2^{-8}(1+2^{-15})$ fue descubierto por Basu et al.[4] Aunque este último se parece al nuestro de corto plazo $P_r[Z_r = Z_{r+1}]$, en Basu et al. se asumió que el estado interno S es una permutación aleatoria, lo cual es cierto solo después de varias vueltas del PRGA. Isobe et al. buscaron interdependencias entre los *bytes* iniciales del flujo de clave, estimando empíricamente $P_r[Z_r = y \wedge Z_{r-a} = x]$ para $0 \leq x,y \leq 255$, $2 \leq r \leq 256$, y $1 \leq a \leq 8$,[20] pero no pudieron encontrar ningún enfoque nuevo. Mironov modeló el RC4 como una cadena Markov y recomendó saltarse los $12 \cdot 256$ *bytes* iniciales del flujo de clave.[26] Paterson et al. generaron estadísticas del flujo de clave sobre *bytes* consecutivos de flujo de clave, empleando la estructura de clave TKIP.[30] Sin embargo, no pudieron reportar cuáles fueron los enfoques nuevos detectados. A través del análisis empírico mostramos que los enfoques entre *bytes* consecutivos está presentes, incluso cuando se usa el RC4 con claves aleatorias de 128 bit.

El primer ataque práctico contra WPA-TKIP fue diseñado por Beck y Tews[44] y mejorado posteriormente por otros investigadores [46,16,48,49]. Varios trabajos recientemente han estudiado la construcción de claves por paquete, tanto analíticamente[15] como a través de simulaciones [2,31,30]. Para nuestro ataque, hemos replicado parte de los resultados de Paterson et al. [31,30] y somos los primeros en demostrar este tipo de ataque en la práctica. AlFardan et al. ejecutaron experimentos donde los dos candidatos de texto sencillo más probables se generaron a través de probabilidades de *byte* único.[2] No obstante, no presentaron un algoritmo para recuperar arbitrariamente muchos de los candidatos, ni extendieron su investigación al estudio de las probabilidades de *bytes* doble.

Los protocolos SSL y TLS han sido sometidos a amplio escrutinio también [9,41,7,1,2,6]. Nuestro trabajo se basa en el ataque de AlFardan et al. quienes estiman que se necesitan $13 \cdot 2^{30}$ textos cifrados para recuperar una *cookie* de 16 caracteres con altos índices de éxito.[2] Pudimos reducir ese número a $9 \cdot 2^{27}$ a través de varias técnicas, la más prominente fue la utilización de probabilidades basadas en el enfoque *ABSAB* de Mantin.[24] Isobe et al. emplearon este último enfoque pero combinado con *bytes* previamente desencriptados, para desencriptar *bytes* después de la posición 257.[20] No obstante, usaron una técnica de conteo en vez de las probabilidades

Bayesianas. Ohigashi et al.[29] presentaron un algoritmo que combina los enfoques *ABSAB* y Fluher-McGrew, requiriendo aproximadamente 2^{34} textos cifrados para desencriptar un *byte* individual con altos índices de éxito.

8 Conclusiones

Aunque los ataques TLS y WPA-TKIP contra RC4 anteriores estuvieron casi al límite la funcionalidad, nuestro trabajo los complementa haciéndolos factibles. Luego de capturar $9 \cdot 2^{27}$ encriptaciones enviadas a través de HTTPS, se pudo atacar exitosamente una *cookie* en un periodo de tiempo insignificante. El ataque se ejecutó en apenas 52 horas ejecutando JavaScript en el navegador de la víctima. Adicionalmente, forzando los enfoques RC4, se atacó una red WPA-TKIP en solo una hora. Es muy posible que estos ataques se mejoren e incrementen en un futuro no lejano. Sobre la base de los resultados expuestos, instamos firmemente a los usuarios a no utilizar más el RC4 como protocolo de seguridad.

9 Agradecimientos

Agradecemos a Kenny Patterson y Tom Van Goethem por sus consejos y ayuda. Esta investigación está parcialmente financiada por el Fondo de Investigaciones KU Leuven. Mathy Vanhoef cursa una beca doctoral en la Fundación de Investigaciones de Flanders (FWO).

Referencias

[1] N. J. Al Fardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In IEEE *Symposium on Security and Privacy*, 2013.

[2] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security Symposium*, 2013.

[3] A. Barth. HTTP state management mechanism. RFC 6265, 2011.

[4] R. Basu, S. Ganguly, S. Maitra, and G. Paul. A complete characterization of the evolution of RC4 pseudo random generation algorithm. *J. Mathematical Cryptology*, 2(3):257–289, 2008.

[5] D. Berbecaru and A. Lioy. On the robustness of applications based on the SSL and TLS security protocols. In *Public Key Infrastructure*, pages 248–264. Springer, 2007.

[6] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub. Triple handshakes and *cookie* cutters: Breaking and fixing authentication over TLS. In *Security and Privacy (SP)*, *2014 IEEE Symposium* on, pages 98–113. IEEE, 2014.

[7] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. In *Advances in Cryptology (CRYPTO)*, 2003.

[8] T. Dierks and E. Rescorla. The transport layer secu- rity (TLS) protocol version 1.2. RFC 5246, 2008.

[9] T. Duong and J. Rizzo. Here come the xor ninjas. In *Ekoparty Security Conference*, 2011.

[10] D. Eppstein. k-best enumeration. *arXiv preprint arXiv:1412.5075*, 2014.

[11] R. Fielding and J. Reschke. Hypertext transfer protocol (HTTP/1.1): Message syntax and routing. RFC 7230, 2014.

[12] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected areas in cryptography*. Springer, 2001.

[13] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In *FSE*, 2000.

[14] C. Fuchs and R. Kenett. A test for detecting outlying cells in the multinomial distribution and two-way contingency tables. *J. Am. Stat. Assoc*., 75:395–398, 1980.

[15] S. S. Gupta, S. Maitra, W. Meier, G. Paul, and S. Sarkar. Dependence in IV-related *bytes* of RC4 key enhances vulnerabilities in WPA. Cryptology ePrint Archive, Report 2013/476, 2013. http: //eprint.iacr.org/.

[16] F. M. Halvorsen, O. Haugen, M. Eian, and S. F. Mjølsnes. An improved attack on TKIP. In *14th Nordic Conference on Secure IT Systems*, NordSec ’09, 2009.

[17] B. Harris and R. Hunt. Review: TCP/IP security threats and attack methods. *Computer Communications*, 22(10):885–897, 1999.

[18] ICSI. The ICSI certificate notary. Retrieved 22 Feb. 2015, from http://notary.icsi.berkeley.edu.

[19] IEEE Std 802.11-2012. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[20] T. Isobe, T. Ohigashi, Y. Watanabe, and M. Morii. Full plaintext recovery attack on broadcast RC4. In *FSE*, 2013.

[21] A. Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, 48(3):269–286, 2008.

[22] S. Maitra and G. Paul. New form of permutation bias and secret key leakage in keystream *bytes* of RC4. In *Fast Software Encryption*, pages 253–269. Springer, 2008.

[23] S. Maitra, G. Paul, and S. S. Gupta. Attack on broadcast RC4 revisited. In Fast Software Encryp- tion, 2011.

[24] I. Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In *EUROCRYPT*, 2005.

[25] I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *FSE*, 2001.

[26] I. Mironov. (Not so) random shuffles of RC4. In *CRYPTO*, 2002.

[27] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large- scale evaluation of remote JavaScript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[28] D. Nilsson and J. Goldberger. Sequentially finding the n-best list in hidden Markov models. In *International Joint Conferences on Artificial Intelligence*, 2001.

[29] T. Ohigashi, T. Isobe, Y. Watanabe, and M. Morii. Full plaintext recovery attacks on RC4 using multiple biases. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 98(1):81–91, 2015.

[30] K. G. Paterson, B. Poettering, and J. C. Schuldt. Big bias hunting in amazonia: Large-scale computation and exploitation of RC4 biases. In *Advances in Cryptology — ASIACRYPT*, 2014.

[31] K. G. Paterson, J. C. N. Schuldt, and B. Poettering. Plaintext recovery attacks against WPA/TKIP. In *FSE*, 2014.

[32] G. Paul, S. Rathi, and S. Maitra. On non-negligible bias of the first output *byte* of RC4 towards the first three *bytes* of the secret key. *Designs, Codes and Cryptography*, 49(1-3):123–134, 2008.

[33] S. Paul and B. Preneel. A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In *FSE*, 2004.

[34] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2014.

[35] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[36] M. Roder and R. Hamzaoui. Fast tree-trellis list Viterbi decoding. *Communications, IEEE Transactions on*, 54(3):453–461, 2006.

[37] D. Roesler. STUN IP address requests for WebRTC. Retrieved 17 June 2015, from https://github.com/diafygi/webrtc-ips.

[38] S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. (Non-)random sequences from (non-)random permutations - analysis of RC4 stream cipher. *Journal of Cryptology*, 27(1):67–108, 2014.

[39] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and exploitation of new biases in RC4. In *Selected Areas in Cryptography*, pages 74–91. Springer, 2011.

[40] N. Seshadri and C.-E. W. Sundberg. List Viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, 42(234):313–323, 1994.

[41] B. Smyth and A. Pironti. Truncating TLS connections to violate beliefs in web applications. In *WOOT'13: 7th USENIX Workshop on Offensive Technologies*, 2013.

[42] F. K. Soong and E.-F. Huang. A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91. 1991 International Conference on*, pages 705–708. IEEE, 1991.

[43] A. Stubblefield, J. Ioannidis, and A. D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Trans. Inf. Syst. Secur.*, 7(2), 2004.

[44] E. Tews and M. Beck. Practical attacks against WEP and WPA. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, 2009.

[45] E. Tews, R.-P. Weinmann, and A. Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In I*nformation Security Applications*, pages 188–202. Springer, 2007.

[46] Y. Todo, Y. Ozawa, T. Ohigashi, and M. Morii. Falsification attacks against WPA-TKIP in a realistic environment. *IEICE Transactions*, 95-D (2), 2012.

[47] T. Van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen. Large-scale security analysis of the web: Challenges and findings. In *TRUST*, 2014.

[48] M. Vanhoef and F. Piessens. Practical verification of WPA-TKIP vulnerabilities. In *ASIACCS*, 2013.

[49] M. Vanhoef and F. Piessens. Advanced Wi-Fi attacks using commodity hardware. In *ACSAC*, 2014.

[50] S. Vaudenay and M. Vuagnoux. Passive–only key recovery attacks on RC4. In *Selected Areas in Cryptography*, pages 344–359. Springer, 2007.