Plaintext Recovery Attacks Against WPA/TKIP*

Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt

Information Security Group
Royal Holloway, University of London

Abstract. We conduct an analysis of the RC4 algorithm as it is used in the IEEE WPA/TKIP wireless standard. In that standard, RC4 keys are computed on a per-frame basis, with specific key bytes being set to known values that depend on 2 bytes of the WPA frame counter (called the TSC). We observe very large, TSC-dependent biases in the RC4 keystream when the algorithm is keyed according to the WPA specification. These biases permit us to mount an effective statistical, plaintext-recovering attack in the situation where the same plaintext is encrypted in many different frames (the so-called "broadcast attack" setting). We assess the practical impact of these attacks on WPA/TKIP.

1 Introduction

The cryptographic mechanisms that aim at protecting transmitted data in modern wireless computer networks have seen an ongoing evolution. Most prominent are the results of the IEEE 802.11 standardization effort; amongst others, IEEE introduced *Wired Equivalent Privacy* (WEP) in 1999, *Wi-Fi Protected Access* (WPA) in 2003, and WPA2 in 2004.

In a nutshell, the WEP protocol [1] works as follows: when a message m is to be transmitted, a CRC32 checksum is appended to it and the resulting string encrypted using RC4; the corresponding packet-specific RC4 key consists of the concatenation of a monotonically increasing sequence number and a shared secret. Practical attacks against integrity, authenticity, and secrecy of transmitted data were reported soon after the publication of WEP, exploiting a wide range of shortcomings of the protocol (including short sequence numbers, lack of randomization, linearity of both RC4 encryption and CRC32, and others) [4]. Refined versions of these attacks [5,21,19] rely on advanced cryptanalysis of RC4 and are based on the fact that the (public) packet sequence number is part of the encryption key. Today, WEP is considered fully broken, and its usage is discouraged even in the IEEE 802.11 standard itself.

To counter these attacks, IEEE decided to redesign the cryptographic components of their wireless standards from scratch. Indeed, the WPA2 standard mandates support for encryption based on the AES block cipher running in CCM

* This is the proceedings version. The full version can be found at `https://eprint.iacr.org/2013/748`.

mode instead of on RC4 [2]. However, as most wireless devices implement their core cryptographic routines directly in silicon, switching from WEP to WPA2 also requires the replacement of hardware in all involved network computers and access points. In order to mitigate these costs, IEEE additionally proposed the *Temporal Key Integrity Protocol* (TKIP) as part of the WPA standard as an intermediate solution[1]. The design of WPA/TKIP is by intention quite close to that of the original WEP, so that all required modifications can be implemented using only firmware updates. Indeed, WPA/TKIP also encrypts packets using RC4, but with (supposedly) better per-packet keys.

The intention of IEEE was that WPA/TKIP should only be a temporary standard during the transition to WPA2. Indeed, it was recently announced that IEEE will deprecate WPA/TKIP in 2014. Yet use of WPA/TKIP is still widespread. For example, the 2013 paper [23] reported that 71% of 6803 different IEEE 802.11 networks surveyed still permit WPA/TKIP, with 19% of those networks using encryption only allowing WPA/TKIP. Given its widespread use, and the spurred on by the WEP fiasco, WPA/TKIP has received a good deal of attention from researchers, including [13,20,7,14,19,22,23].

WPA/TKIP requires the 16-byte RC4 key $\mathtt{K} = (\mathtt{K}_0, \ldots, \mathtt{K}_{15})$ used to encrypt a frame to be generated in a very specific way from the *temporal encryption key* $\mathtt{TK}$ (128 bits), the TKIP sequence counter $\mathtt{TSC}$ (48 bits, incremented for each frame that is transmitted), and the transmitter address $\mathtt{TA}$ (48 bits). Specifically, $\mathtt{K}$ is computed via a so-called 'key mixing' procedure, which we write as $\mathtt{K} \leftarrow \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC})$. Internally, $\mathtt{KM}$ implements key derivation by mixing together its inputs using a custom 8-round Feistel cipher whose round function relies on the AES S-boxes. Key $\mathtt{K}$ is derived from the output of this routine, with some structure added to "preclude the use of known RC4 weak keys" [2]. More precisely, writing $\mathtt{TSC} = (\mathtt{TSC}_0, \mathtt{TSC}_1, \ldots, \mathtt{TSC}_5)$, i.e., the least-significant byte on the left, we have

$$\mathtt{K}_0 = \mathtt{TSC}_1 \qquad \mathtt{K}_1 = (\mathtt{TSC}_1 \mid \mathtt{0x20}) \,\&\, \mathtt{0x7f} \qquad \mathtt{K}_2 = \mathtt{TSC}_0 \tag{1}$$

and $\mathtt{K}_3, \ldots, \mathtt{K}_{15}$ being assigned from the output of the Feistel cipher. Notably here, bytes $\mathtt{K}_0, \mathtt{K}_1, \mathtt{K}_2$ depend only on bytes $\mathtt{TSC}_0$ and $\mathtt{TSC}_1$ of $\mathtt{TSC}$. Moreover, the bits of $\mathtt{TSC}_1$ are used twice. So the bytes of $\mathtt{K}$ have more structure than they would if they were chosen with uniform distribution (as is the case when RC4 is used in TLS, for example); in addition, as $\mathtt{TSC}$ values are public information, partial information about $\mathtt{K}$ might be known to attackers.

Recently, AlFardan *et al.* [3] showed that RC4 with uniformly distributed keys (as in TLS) is biased in its initial keystream bytes, and that these biases can be exploited in passive attacks to recover plaintext. More specifically, they worked in the *broadcast* or *multi-session* setting, wherein the same plaintext is repeatedly encrypted under different, independent keys. This setting is readily realised for TLS protecting web traffic by using JavaScript code running in the client's browser to automatically generate the required encryptions, with the

[1] Note that the WPA2 standard has optional support for TKIP for backward compatibility.

target plaintext being a secure cookie belonging to the client. AlFardan *et al.* [3] presented two attacks on RC4 in TLS, one based on single-byte biases in the initial keystream bytes, the other based on the long-term Fluhrer-McGrew double-byte biases [6]. The first attack proceeds on a simple statistical basis: given enough ciphertext samples encrypting the same plaintext, one may proceed position by position, by simply trying each possibility for plaintext byte P_r (in position r), recovering the corresponding keystream bytes K_r, and then selecting as the output plaintext byte P_r the one for which the induced distribution on the keystream bytes has the highest likelihood when compared to an empirical estimate of the distribution for that position. This attack, therefore, requires the attacker to first build a good estimate of the keystream distribution in each output position r; this was done in [3] using 2^{44} random RC4 keys to get very accurate estimates of the keystream byte distributions.

It is natural to ask whether the same techniques might be deployed against other applications of the RC4 algorithm. In this paper, we address that question for WPA/TKIP.

1.1 Overview of Results

We begin by simply applying the single-byte plaintext recovery attack of AlFardan *et al.* [3] to RC4 with keys generated according to the TKIP specification, as described above. More exactly, we build a keystream estimate using 2^{41} RC4 keys obtained by considering 2^{19} random choices for `TK` and `TA`, and then 2^{22} `TSC` values (implemented as a counter with a random starting value). We then simulate the single-byte bias attack of [3], again generating the RC4 keys and ciphertexts according to the TKIP specification. This not only produces encouraging results in terms of plaintext recovery, but also reveals intriguing behaviour in the biases and in the plaintext recovery rates. The keystream biases that we observe exhibit more complex behaviour than for the random 16-byte RC4 keys used in TLS and that were considered in [3] (see in particular, Figures 2, 3 and 4 in Section 3 below). This makes plaintext recovery easier in some positions, but harder in others when compared to the case of TLS.

In this first approach, the keystream estimates are calculated *averaging over the values of the pair* $(\mathtt{TSC}_0, \mathtt{TSC}_1)$, whereas it might be expected that there would also be keystream biases that depend on the specific values of $\mathtt{TSC}_0$ and $\mathtt{TSC}_1$ because of the way the TKIP key `K` in turn depends on these bytes. Thus it seems reasonable that, even though we have already observed significant and exploitable biases in the course of developing our first attack on TKIP, quite different and/or bigger biases might be found by sampling over keystreams for keys having specific values for the `TSC` pair $(\mathtt{TSC}_0, \mathtt{TSC}_1)$. In fact, we discover that the keystream distributions are not just different for different $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs – they are *radically* different. Moreover, very large biases in the RC4 keystream distributions appear, much larger than were observed in our first analysis. In a sense, these larger and different biases disappear to leave a different set of much smaller biases behind when one averages over the `TSC` pair $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ as in our first attack

These $(\texttt{TSC}_0, \texttt{TSC}_1)$-dependent biases can be exploited to build a second, more powerful attack, which, at a high level, works as follows:

1. Bin the available ciphertexts into 2^{16} bins according to the value of the TSC pair $(\texttt{TSC}_0, \texttt{TSC}_1)$. This binning is possible in TKIP because the TSC field is sent in the clear in each frame's header.
2. Perform a likelihood analysis of plaintext candidates for each of the bins.
3. Combine the resulting plaintext likelihood estimates for the different bins in a statistically sound procedure to get an estimate of the overall likelihood for each plaintext.

Essentially, while our first analysis effectively averages out any $(\texttt{TSC}_0, \texttt{TSC}_1)$-specific behaviour, our second, more delicate analysis exploits it to the full. We refer to this attack as a $(\texttt{TSC}_0, \texttt{TSC}_1)$ *binning attack*.

This second approach once again requires the computation of keystream biases, but now we need a good estimate of the distribution of keystream bytes in each output position r $(1 \leq r \leq 256)$ [2] for each of the 2^{16} $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs, a dataset containing 2^{32} items. To gain a similar level of accuracy for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair as we obtained for our first attack, we would need to compute statistics for around 2^{56} RC4 keystreams. This is currently well beyond our computational reach: generating 2^{40} RC4 keystreams currently takes about 4 days on our 16-core machine, and gives estimates for keystream biases based on only 2^{24} RC4 keystreams for each of the 2^{16} $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs, while the desired computation would be 2^{16} times larger (estimated at 2^{22} core days). The estimates for the biases that we get from 2^{24} RC4 keystreams per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair are somewhat noisy, in the sense that they do not accurately reflect the true keystream distributions except when there are large biases present. And, unfortunately, our experience is that using a noisy set of keystream estimates introduces inaccuracies into our plaintext recovery attacks which substantially reduces their success rates.

We present and investigate two methods to compensate for the problem of not having very accurate estimates of the keystream biases for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair:

1. $\texttt{TSC}_1$ is used to set two of the TKIP RC4 key bytes, so one may suspect that the keystream biases would be particularly dependent on this single byte of TSC. We therefore carry out the approach suggested above, but computing 2^8 keystream estimates, one for each value of $\texttt{TSC}_1$, instead of the original 2^{16} estimates. In our experiments, we use 2^{32} keystreams to obtain each estimate, bringing the total computation up to that of 2^{40} RC4 keystreams. In the attack, we then bin the available ciphertexts into 2^8 bins according to the known value of $\texttt{TSC}_1$, perform our likelihood analysis for each bin, and then combine the results. We refer to this attack as a $\texttt{TSC}_1$ *binning attack*.

[2] We focus on the keystream distributions in the first 256 bytes because we did not observe significant biases beyond these bytes in our experiments, with the exception of byte 257.

2. Based on a computation with 2^{24} RC4 keystreams for each of the 2^{16} ($\mathtt{TSC}_0$, $\mathtt{TSC}_1$) pairs, and confirmed by larger computations for specific ($\mathtt{TSC}_0, \mathtt{TSC}_1$) pairs, we have observed that there are many very large biases present in the ($\mathtt{TSC}_0, \mathtt{TSC}_1$)-specific keystreams. For example, while the typical bias observed in our first (TSC-averaged) analysis is on the order of $\pm 2^{-16}$, we find that there are many thousands of ($\mathtt{TSC}_0, \mathtt{TSC}_1$)-specific biases on the order of $\pm 2^{-12}$ and larger. (See Figure 7(a) for a pictorial representation of the numbers of such large biases across all ($\mathtt{TSC}_0, \mathtt{TSC}_1$) pairs in each position.) Heuristically, these biases might be expected to dominate the plaintext recovery procedure. We therefore "de-noise" our keystream distribution estimates by applying a cut-off procedure to them, setting all probability estimates that fall within a threshold around 2^{-8} to an average value, and leaving those that lie outside that threshold untouched. We then execute our binning attack using these idealised keystream estimates.

Of course, these two methods can be combined, and we examine the effect of doing so on the success rate of our plaintext recovery attacks.

As we shall see in Section 5, our attacks are effective. For example, using just the first method above with 2^{26} ciphertexts, we obtain an average success rate of 65% in recovering each of the first 256 bytes of plaintext. The rate rises to higher than 90% in even positions, this improvement being due to the presence of particularly large and $\mathtt{TSC}_1$-specific RC4 keystream biases in the even positions when TKIP keys are used.

1.2 Related Work

In independent and concurrent work, Sen Gupta *et al.* [16] have identified biases in WPA that are TSC-dependent, and speculated that there may be correlations between keystream bytes and linear combinations of the known key bytes $\mathtt{K}_0, \mathtt{K}_1, \mathtt{K}_2$ (which are computed exclusively from the TSC). This is similar to our approach. However, they did not perform a systematic search for biases, and did not apply them to plaintext recovery except in positions 1, 2, 3, 256 and 257. Their approach to plaintext recovery uses an *ad hoc* approach, with each linear combination being used to compute a keystream estimate, which then suggests a plaintext byte. Our approach is likelihood-based, and takes the reverse approach: for every possibility for the plaintext byte, and each ($\mathtt{TSC}_0, \mathtt{TSC}_1$) pair (or $\mathtt{TSC}_1$ value), we compute the likelihood of the resulting induced keystream estimate, and combine these estimates to select the plaintext having the highest likelihood. Note that this recovery algorithm is optimal.

1.3 Paper Organisation

Section 2 provides further background on the RC4 stream cipher and its use in WPA. Section 3 reports biases in RC4 keystreams when RC4 is keyed according to the WPA specification, comparing biases for random-TSC WPA keys with biases for keys generated according to specific values of $\mathtt{TSC}_1$ and ($\mathtt{TSC}_0, \mathtt{TSC}_1$).

Algorithm 1: Key schedule (KSA)

input : key K of l bytes
output: initial internal state st_0
begin
  for $i = 0$ **to** 255 **do**
    $\mathcal{S}[i] \leftarrow i$
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
    $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$
    swap$(\mathcal{S}[i], \mathcal{S}[j])$
  $i, j \leftarrow 0$
  $st_0 \leftarrow (i, j, \mathcal{S})$
  return st_0

Algorithm 2: Keystream generator (PRGA)

input : internal state st_r
output: keystream byte Z_{r+1}
updated internal state st_{r+1}
begin
  parse $(i, j, \mathcal{S}) \leftarrow st_r$
  $i \leftarrow i + 1$
  $j \leftarrow j + \mathcal{S}[i]$
  swap$(\mathcal{S}[i], \mathcal{S}[j])$
  $Z_{r+1} \leftarrow \mathcal{S}[\mathcal{S}[i] + \mathcal{S}[j]]$
  $st_{r+1} \leftarrow (i, j, \mathcal{S})$
  return (Z_{r+1}, st_{r+1})

Fig. 1. Algorithms implementing the RC4 stream cipher. All additions are performed modulo 256.

Section 4 describes our plaintext recovery attacks on WPA that exploit these biases. We evaluate the attacks in Section 5 via simulation. Finally, Section 6 discusses the impact of and countermeasures to our attacks.

2 Further Background

2.1 The RC4 Stream Cipher

The stream cipher RC4, originally designed by Ron Rivest, became public in 1994 and found application in a wide variety of cryptosystems; well-known examples include SSL/TLS, WEP [1], WPA [2], and some Kerberos-related encryption modes [8]. RC4 has a remarkably short description and is extremely fast when implemented in software. However, these advantages come at the price of lowered security: several weaknesses have been identified in RC4 [6,5,11,10,9,17,19,18,24].

Technically, RC4 consists of two algorithms: a *key scheduling algorithm* (KSA) and a *pseudo-random generation algorithm* (PRGA), which are specified in Algorithms 1 and 2. The KSA takes as input a key K, typically a byte-array of length between 5 and 32 (i.e., 40 to 256 bits), and produces the initial internal state $st_0 = (i, j, \mathcal{S})$, where $\mathcal{S}$ is the canonical representation of a permutation on the set $[0, 255]$ as an array of bytes, and i, j are indices into this array. The PRGA will, given an internal state st_r, output 'the next' keystream byte Z_{r+1}, together with the updated internal state st_{r+1}.

2.2 WPA

We describe the cryptographic operation of WPA when RC4 is selected as the encryption method (referred to as TKIP). Our description is not complete, but provides sufficient detail to enable our subsequent attacks to be understood. We refer the reader to our introduction for an explanation of how TKIP generates

its per-frame key K as a function $\mathtt{K} \leftarrow \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC})$ of the temporal encryption key TK (128 bits), the TKIP sequence counter TSC (48 bits), and the transmitter address TA (48 bits). The per-frame key K is then used to produce an RC4 keystream, following the above description. The initialisation of RC4 in WPA is the standard one for this algorithm. Notably, none of the initial keystream bytes is discarded when RC4 is used in WPA, despite these bytes having known weaknesses.

The TKIP plaintext (consisting of the frame payload, a 64-bit MAC value MIC , and a 32-bit Integrity Check Vector ICV) is then XORed in a byte-by-byte fashion with the RC4 keystream, i.e., the ciphertext bytes are computed as

$$C_r = P_r \oplus Z_r \qquad \text{for } r = 1, 2, 3, \ldots \ ,$$

where P_r are the individual bytes of P, and Z_r are the RC4 keystream bytes. The data transmitted over the air then has the form

$$\mathtt{HDR} || C \ ,$$

where C is the concatenation of the bytes C_r and HDR is the unencrypted frame header.

3 Biases in the RC4 Keystream for WPA Keys

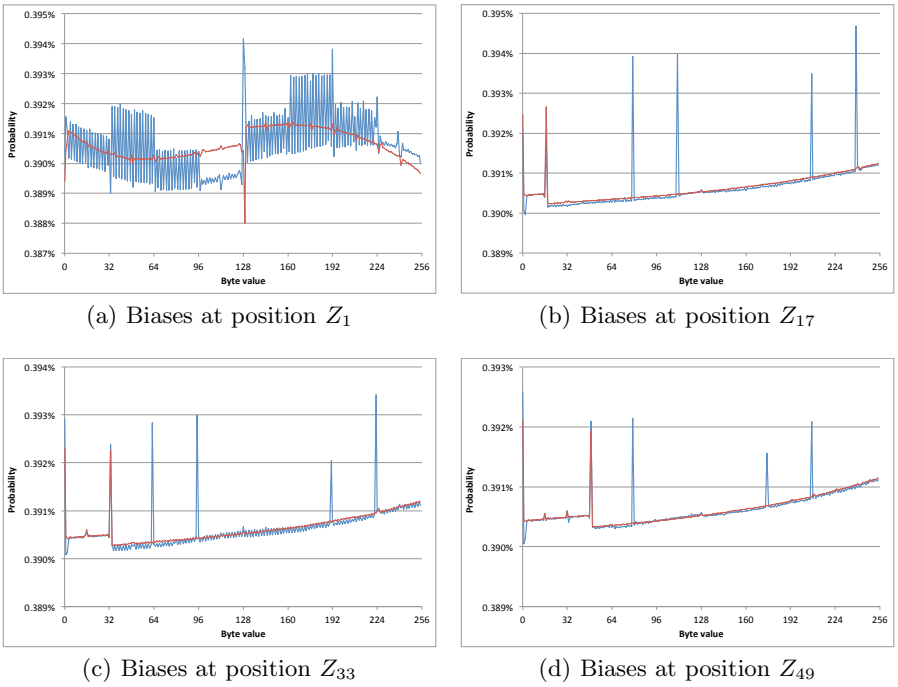
In the context of our analysis, we need to assess the strength of biases in the RC4 output streams for the keys K output by KM. If strong biases exist, then an attack on WPA is likely to be feasible using the ideas sketched in the introduction. That is, in a setting where the same plaintext message is repeatedly transmitted in a WPA-protected wireless network, one can expect that this plaintext is (at least partially) recoverable.

3.1 Fully Aggregated Biases for TKIP

We first experimentally determined single-byte keystream biases in WPA, without regard to TSC values (as would be consumed in a direct application of the attack from [3]). We refer to the biases obtained as being *fully aggregated biases*, since they are computed by using random TSC values and hence can be considered as being generated by aggregating over all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs (in contrast to the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-pair-specific biases that we consider below). More precisely, we implemented the KM key derivation function, verified it against the test vectors from [2, Annex M.1.2], and computed keys

$$\mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC}), \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC}+1), \ldots, \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC}+(2^{22}-1))$$

for 2^{19} random assignments of variables TA, TK, TSC, aiming at modelling a realistic application of TKIP. Considering all resulting 2^{41} RC4 keys, we measured the distribution of keystream bytes at positions 1–256. Independently, from the



(a) Biases at position Z_1

(b) Biases at position Z_{17}

(c) Biases at position Z_{33}

(d) Biases at position Z_{49}

Fig. 2. Measured distribution of the TKIP keystream at positions Z_1, Z_{17}, Z_{33}, and Z_{49} (blue). These estimates were obtained by considering more than 2^{41} `KM`-generated keys. For reference, we overlay the biases of the RC4 keystream with random 128-bit keys (red).

set of all keys `K` consistent with equation (1), we generated 2^{41} random keys (i.e., with random `TSC`, but also setting $\mathtt{K}_3, \ldots, \mathtt{K}_{15}$ randomly instead of using the Feistel cipher). We identified the corresponding keystream distributions at the same positions. We observed that the difference between these two sets of distributions is small, allowing us to make the assumption that the action of the Feistel cipher does not affect the output distribution of RC4. We hence base all of our further observations on statistics obtained from random keys conforming with (1). Here, and throughout, we use AES with a fixed key in counter mode to generate any random values needed (so that they are in fact pseudorandom, and we are relying on AES being a good block cipher to ensure our keys are well distributed).

In contrast to the internal Feistel cipher of `KM`, the structure on RC4 keys implied by equation (1) has a significant influence on the aggregated biases in TKIP. For instance, at positions 17, 33, 49, 65, 81, and 97 (i.e., $16k + 1$ for small k), new peaks in the distribution show up that do not appear in RC4 with random 128-bit keys. For the distributions at positions 17, 33, and 49 see Figures 2(b), 2(c), and 2(d). Even more extreme is the difference at position 1,

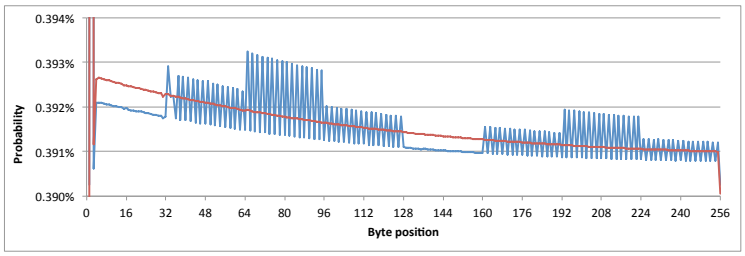


Fig. 3. Strength of the bias towards 0x00 of TKIP keystream bytes at positions 1–256 (blue). The red line corresponds to the biases of RC4 with random 128-bit keys.

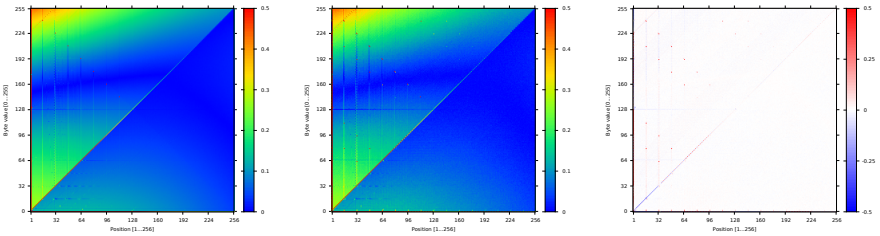
shown in Figure 2(a). It is also interesting to observe how the bias towards 0x00 behaves in the TKIP case: the probability $\Pr(Z_r = \texttt{0x00})$ is persistently smaller than in the case of random 128-bit keys at positions 2–32 and 128–160, whereas for the other positions its value alternates from byte to byte between being significantly larger and being significantly smaller than the corresponding probabilitiy for uniform keys. This is illustrated in Figure 3, which compares the strength of the bias towards 0x00 for TKIP and for random 128-bit RC4 keys. Finally, to make it easier to compare the TKIP-specific biases with the biases for random 128-bit keys in [3], we present Figure 4, in which we depict side-by-side the full set of biases for both cases. Figure 4(c) shows the differences between the two sets of biases; blue and red pixels show the places where the biases differ significantly.

It is an interesting theoretical problem to explain the differences between RC4 biases for random 128-bit keys and TKIP keys (though we stress that having such an explanation does not affect the performance of our attacks to follow).

3.2 $(\texttt{TSC}_0, \texttt{TSC}_1)$-pair-specific Biases for TKIP

As explained in the introduction, we wish to examine how the biases in RC4 keystreams for TKIP keys depend on the $(\texttt{TSC}_0, \texttt{TSC}_1)$ byte pair used in defining the keys. For each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair, we computed 2^{24} RC4 keystreams by assigning the bytes $\texttt{K}_0, \texttt{K}_1, \texttt{K}_2$ according to the $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair (as per the specification, see equation (1)) and assigning the remaining 13 key bytes at random.

Using each set of 2^{24} RC4 keys, we then computed the distribution of keystream bytes at positions 1–256, giving a dataset containing 256 keystream byte distributions for each of the 2^{16} $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs. Of course, it is not possible to represent such a large dataset in full here, but we provide two small samples in Figures 5 and 6. The former shows how the distribution of keystream byte Z_1 depends heavily on the value of the $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair. The latter shows how different is the distribution of keystream bytes in a variety of positions for a specific $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair, $(\texttt{0x00}, \texttt{0x00})$, when compared to the fully aggregated results reported above. This is indicative that plaintext recovery attacks that



(a) Biases in RC4 keystream (random 128-bit keys)

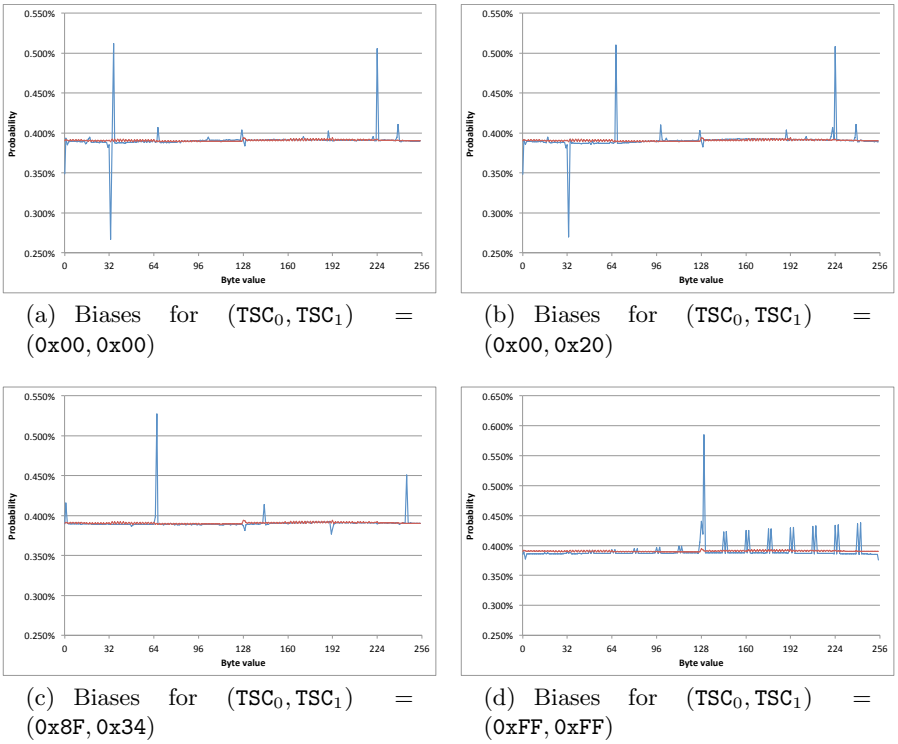
(b) Biases in RC4 keystream (TKIP keys)

(c) Map indicating the difference between Figures 4(a) and 4(b).

Fig. 4. Pictorial representation of biases in RC4 keystreams for random 128-bit keys and for TKIP keys, for different positions (x-axis) and byte values (y-axis). For each position we encode the bias in the keystream for the (position,value) combination as a colour; in Figures 4(a) and 4(b) the colouring scheme encodes the absolute biases, i.e., the absolute difference between the occurring probabilities and the (expected) probability $1/256$, scaled up by a factor of 2^{16}, capped to a maximum of 0.5. In Figure 4(c), the colour encodes the difference between the absolute biases arising for random 128-bit keys and for TKIP keys, scaled up by a factor of 2^{16} and capped to the range $[-0.5, 0.5]$.

focus on exploiting biases arising for individual $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs may perform better than those working with fully aggregated biases.

Our plaintext recovery attacks to be presented in Section 4 proceed on a position-by-position basis, and can be expected to work well in a given position r if there are large biases in that position over the different $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs. Figure 7(a) shows that large biases are indeed plentiful and well-spread over the keystream positions. For instance, it reveals that the 256 strongest biases at positions 1–128 have a value of about 2^{-11}, with a couple of exceptions where strengths of more than 2^{-10} can be reported. Even more impressive are the many thousands of biases of strength $> 2^{-10}$ at positions 1–3. Also for position 256 many thousands of relatively strong biases do exist. The interesting structure in intervals 32–128 and 160–256, where positions with strong biases alternate with positions having no strong biases, will considerably affect the recovery rate of our attacks, as we will see. To conclude, the numbers of large biases seen is significantly larger than one would expect if the keystream bytes were uniformly random. For example, with 2^{24} keystreams per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair, we would expect the count for each byte value in each position to follow a (roughly) Normal distribution with mean 2^{16} and standard deviation σ approximately 2^8. We would then expect the number of counts outside the range $\left[2^{16} - 2^{10}, 2^{16} + 2^{10}\right]$ (i.e., outside the 4σ range) to be roughly 0.2%, whereas the actual rate of such counts is about 1.5–2% for at least half of the positions, and for some positions even larger.



(a) Biases for $(\texttt{TSC}_0, \texttt{TSC}_1) = (\texttt{0x00}, \texttt{0x00})$

(b) Biases for $(\texttt{TSC}_0, \texttt{TSC}_1) = (\texttt{0x00}, \texttt{0x20})$

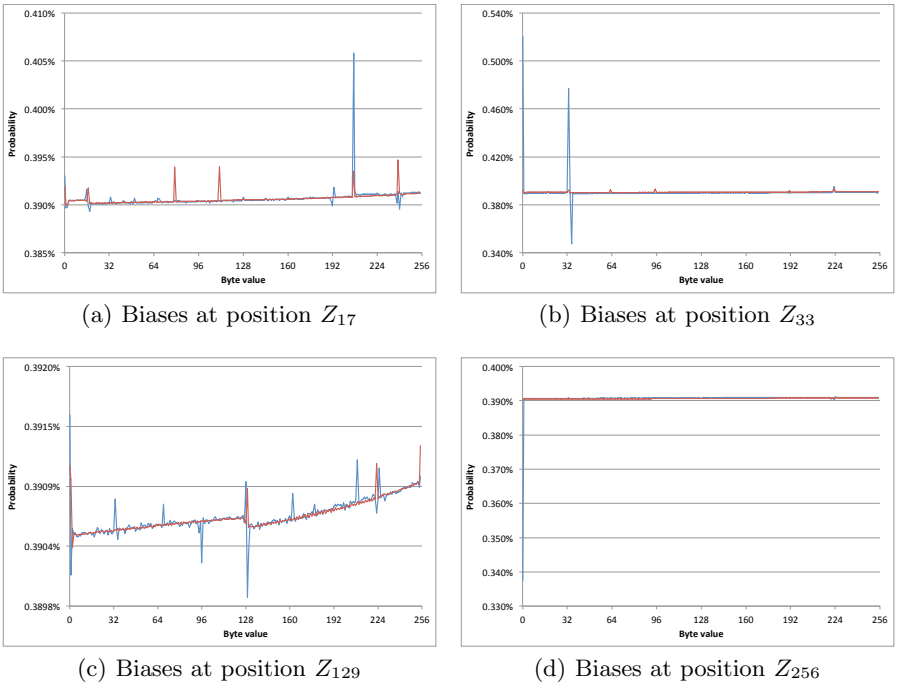
(c) Biases for $(\texttt{TSC}_0, \texttt{TSC}_1) = (\texttt{0x8F}, \texttt{0x34})$

(d) Biases for $(\texttt{TSC}_0, \texttt{TSC}_1) = (\texttt{0xFF}, \texttt{0xFF})$

Fig. 5. Measured distribution of the TKIP RC4 keystream at position Z_1 for $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs $(\texttt{0x00}, \texttt{0x00})$, $(\texttt{0x00}, \texttt{0x20})$, $(\texttt{0x8F}, \texttt{0x34})$, $(\texttt{0xFF}, \texttt{0xFF})$ (blue). These estimates were obtained by considering more than 2^{36} keys per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair. For reference, we overlay the corresponding fully aggregated TKIP keystream biases (red).

We recall from the introduction our hunch that the TKIP keystream biases would be particularly dependent on the single byte $\texttt{TSC}_1$. To test this, we used our bias data for all $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs to compute $\texttt{TSC}_0$-*aggregated biases*, that is, we aggregated our previous data over $\texttt{TSC}_0$ values to obtain 2^8 different keystream distributions for positions 1–256, one distribution for each value of $\texttt{TSC}_1$. Effectively, this gives us distribution estimates based on 2^{32} keystreams for each value of $\texttt{TSC}_1$. We then compared the original distributions to the $\texttt{TSC}_0$-aggregated data.

An indication towards the correctness of our hunch is provided by Figure 7(b) that reports strengths of biases similarly to Figure 7(a) – however aggregating over $\texttt{TSC}_0$ values as described. Indeed, both the obvious similarity of the graphs and the applied scaling factor of 256 along the y-axis are exactly as expected when assuming that each strong bias in the aggregated counts appears 256 times in the plain (unaggregated) counts. This suggests that our hunch concerning the relative importance of $\texttt{TSC}_1$ is correct, and gives weight to the idea of considering



(a) Biases at position Z_{17}

(b) Biases at position Z_{33}

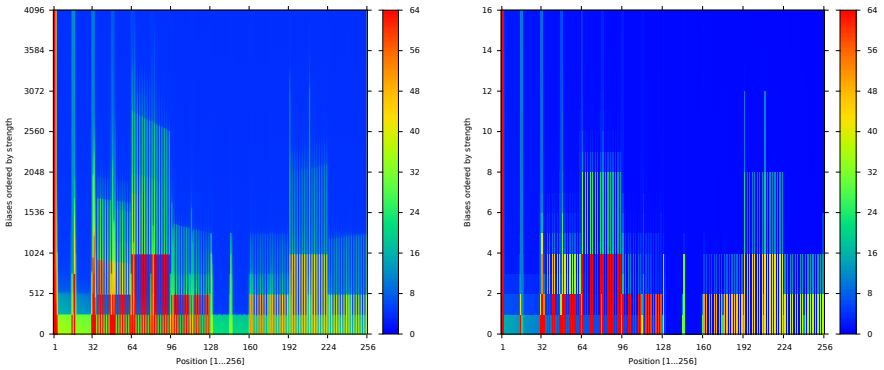
(c) Biases at position Z_{129}

(d) Biases at position Z_{256}

Fig. 6. Measured distribution of the TKIP keystream at positions Z_{17}, Z_{33}, Z_{129}, Z_{256} for $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair $(\texttt{0x00}, \texttt{0x00})$ (blue; see Figure 5(a) for distribution of Z_1). These estimates were obtained by considering more than 2^{36} keys per TSC pair. For reference, we overlay the corresponding fully aggregated TKIP keystream biases (red).

$\texttt{TSC}_0$-aggregated biases in our plaintext recovery attack (the first of our two methods designed to compensate for the problem of not having very accurate keystream bias estimates as mentioned in the introduction).

Given a fixed position in the keystream, our attack on TKIP works best if there are strong biases for many $(\texttt{TSC}_0, \texttt{TSC}_1)$ combinations (or $\texttt{TSC}_1$ values) for that position. In practice, only TKIP frames with such TSC values will contribute noticeably to plaintext recovery. While Figures 7(a) and 7(b) tell us that quite large biases do exist almost everywhere in the first 256 positions of the RC4 keystream *for some* TSC values, it is yet unclear for which TSC values they occur. We provide Figures 8(a) and 8(b) to shed more light on the distribution of 'bias-friendly' TSC values. We see that for positions 1–3, 16–17, 32–33, 48–49, 64–65, 80–81, and 96–97 strong biases exist for more than 50% of all $\texttt{TSC}_1$ values. Orthogonally to that, a $\texttt{TSC}_1$ value of 127 guarantees strong biases in the first 128 keystream positions. Further, it is quite interesting to trace the origin of the alternating behaviour in Figures 7(a) and 7(b) at position ranges 32–128 and 160–256. Finally, note the strong tendency at positions 1–128 of the TKIP cipher to produce byte values 128 and 129, and also value 65 at positions 1–64.



(a) Largest 4096 biases of the RC4 keystream, taken over all 2^{16} $(\texttt{TSC}_0, \texttt{TSC}_1)$ combinations

(b) Largest 16 biases of the RC4 keystream, taken over all 2^8 aggregated counts for $\texttt{TSC}_1$

Fig. 7. Pictorial representation of the number of large biases in TKIP keystream distributions across different positions in the keystream. For each position we show the strengths of the largest biases, sorted in descending order (largest bias on the bottom line). The colouring scheme encodes the absolute difference between the occurring probabilities and the (expected) probability 1/256, scaled up by a factor of 2^{16}, capped to a maximum of 64.

Again, we note that finding the underlying reasons for the observed bias behaviours in TKIP keystreams is an interesting theoretical problem.

4 Plaintext Recovery Attacks

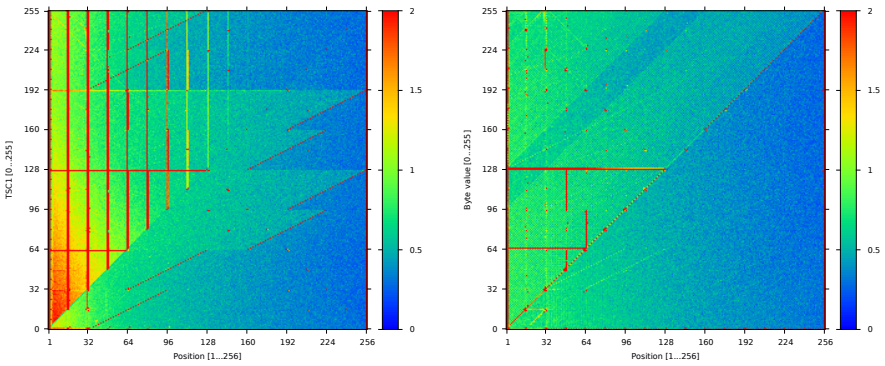
4.1 The Attack of AlFardan *et al.*

The idea behind the single-byte bias attack of AlFardan *et al.* [3] is to first obtain a detailed picture of the distributions of RC4 keystream bytes Z_r, for all positions r of interest, by gathering statistics from keystreams generated using a large number of independent keys. That is, for all r, we (empirically) estimate

$$p_{r,k} := \Pr(Z_r = k), \quad k = \texttt{0x00}, \ldots, \texttt{0xFF},$$

where the probability is taken over a random choice of the RC4 encryption key. In [3], these keys were taken to be random 128-bit values, reflecting how session keys are set in TLS; for TKIP, these keys should be generated according to the procedure described in Section 3.1.

The second step in the approach of [3] is to use the $p_{r,k}$ estimates to recover plaintext using a maximum-likelihood approach, as follows. Suppose we have S ciphertexts $C_1, \ldots, C_S$ available for our attack (for the r-th byte of ciphertext C_j we write $C_{j,r}$). For any fixed position r and any candidate plaintext byte μ



(a) Strength of largest available bias for each combination of position and $\texttt{TSC}_1$ value

(b) Strength of largest available bias for each combination of position and byte value

Fig. 8. Pictorial representation of the correlation between position in TKIP keystream, $\texttt{TSC}_1$ value (respectively, byte value), and the corresponding strength of the largest occurring bias. The colouring scheme encodes the absolute difference between the occurring probabilities and the (expected) probability $1/256$, scaled up by a factor of 2^{16}, capped to a maximum of 2.

for that position, vector $(N^{(\mu)}_{\texttt{0x00}}, \ldots, N^{(\mu)}_{\texttt{0xFF}})$ with

$$N^{(\mu)}_k = |\{j \mid C_{j,r} = k \oplus \mu\}_{1 \le j \le S}| \qquad (\texttt{0x00} \le k \le \texttt{0xFF})$$

represents the distribution on Z_r required to obtain the observed ciphertext bytes $\{C_{j,r}\}_{1 \le j \le S}$ by encrypting μ. We compare these *induced* distributions (one for each possible μ) with the accurate distribution $p_{r,\texttt{0x00}}, \ldots, p_{r,\texttt{0xFF}}$ and interpret a close match as an indication for the corresponding plaintext candidate μ being the correct one, i.e., $P_r = \mu$. More formally, we observe that the probability λ_μ that plaintext byte μ is encrypted to ciphertext bytes $\{C_{j,r}\}_{1 \le j \le S}$ follows a multinomial distribution:

$$\lambda_\mu = \frac{S!}{N^{(\mu)}_{\texttt{0x00}}! \cdots N^{(\mu)}_{\texttt{0xFF}}!} \prod_{k \in \{\texttt{0x00},\ldots,\texttt{0xFF}\}} p_{r,k}^{N^{(\mu)}_k} \quad . \tag{2}$$

The approach of [3] then determines the (optimal) maximum-likelihood plaintext byte value μ by computing λ_μ for all $\texttt{0x00} \le \mu \le \texttt{0xFF}$ and identifying μ such that λ_μ is largest. Algorithm 3 more formally specifies the described attack, incorporating some optimizations discussed in [3] (in particular, as the fraction in equation (2) is independent of μ, we compute the λ_μ values only up to that constant; in fact, we actually compute and compare $\log \lambda_\mu$, rather than λ_μ).

4.2 Attack Based on $(\texttt{TSC}_0, \texttt{TSC}_1)$ Pair Binning

We next discuss our extension of the attack in Algorithm 3 that uses the single-byte RC4 biases, along with their strengths, on a per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair basis.

Algorithm 3: Single-byte bias attack from [3]

input : $\{C_j\}_{1 \le j \le S}$ – S independent encryptions of fixed plaintext P
r – byte position
$(p_{r,k})_{\mathtt{0x00} \le k \le \mathtt{0xFF}}$ – keystream distribution at position r
output: P_r^* – estimate for plaintext byte P_r
begin
  $N_{\mathtt{0x00}} \leftarrow 0, \ldots, N_{\mathtt{0xFF}} \leftarrow 0$
  for $j = 1$ **to** S **do**
    $N_{C_{j,r}} \leftarrow N_{C_{j,r}} + 1$
  for $\mu = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
    for $k = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
      $N_k^{(\mu)} \leftarrow N_{k \oplus \mu}$
    $\lambda_\mu \leftarrow \sum_{k=\mathtt{0x00}}^{\mathtt{0xFF}} N_k^{(\mu)} \log p_{r,k}$
  $P_r^* \leftarrow \arg\max_{\mu \in \{\mathtt{0x00}, \ldots, \mathtt{0xFF}\}} \lambda_\mu$
  return P_r^*

For ease of notation, we let $\overline{\mathtt{TSC}}$ denote the pair $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ in mathematical expressions.

The idea is to first obtain a detailed picture of the distributions of RC4 keystream bytes Z_r, for all positions r in some range, on a per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair basis, by gathering statistics from keystreams generated using a large number of keys (2^{24} per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair in our case). That is, for all r in our selected range, we now estimate

$$p_{\overline{\mathtt{TSC}},r,k} := \Pr(Z_r = k),\ \overline{\mathtt{TSC}} = (\mathtt{0x00}, \mathtt{0x00}), \ldots, (\mathtt{0xFF}, \mathtt{0xFF}),\ k = \mathtt{0x00}, \ldots, \mathtt{0xFF}$$

where the probability is taken over the random choice of the RC4 encryption key $\mathtt{K}$, subject to the structure on $\mathtt{K}_0$, $\mathtt{K}_1$, $\mathtt{K}_2$ induced by $\overline{\mathtt{TSC}} = (\mathtt{TSC}_0, \mathtt{TSC}_1)$.

Using these biases $p_{\overline{\mathtt{TSC}},r,k}$, in a second step, plaintext can be recovered using a variation of the preceding maximum-likelihood approach, as follows.

Suppose we have S ciphertexts $C_1, \ldots, C_S$ available for our attack. We partition these into 2^{16} groups according to the value of the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair; for convenience, we assume the resulting bins of ciphertexts are all of equal size $T = S/2^{16}$, but this need not be the case. Let the bin of ciphertexts associated with a particular $\overline{\mathtt{TSC}} = (\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair be denoted $\mathcal{S}_{\overline{\mathtt{TSC}}}$ and have members $C_{\overline{\mathtt{TSC}},j}$ for $j = 1, \ldots, T$; we denote the byte at position r of $C_{\overline{\mathtt{TSC}},j}$ by $C_{\overline{\mathtt{TSC}},j,r}$. For any fixed position r and any candidate plaintext byte μ for that position, vector $(N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0x00}}, \ldots, N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0xFF}})$ with

$$N^{(\mu)}_{\overline{\mathtt{TSC}},k} = |\{j \mid C_{\overline{\mathtt{TSC}},j,r} = k \oplus \mu\}_{1 \le j \le T}| \qquad (\mathtt{0x00} \le k \le \mathtt{0xFF})$$

represents the distribution on Z_r required to obtain the observed ciphertext bytes $\{C_{\overline{\mathtt{TSC}},j,r}\}_{1 \le j \le T}$ for bin $\mathcal{S}_{\overline{\mathtt{TSC}}}$ by encrypting μ. We compare these *induced* distributions (one for each possible μ and for each possible $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair) with

the accurate distribution $p_{\overline{\mathtt{TSC}},r,\mathtt{0x00}}, \ldots, p_{\overline{\mathtt{TSC}},r,\mathtt{0xFF}}$ and interpret a close match as being an indication for the corresponding plaintext candidate μ being the correct one, i.e., $P_r = \mu$, in bin $\mathcal{S}_{\overline{\mathtt{TSC}}}$. The probability $\lambda_{\overline{\mathtt{TSC}},\mu}$ that plaintext byte μ is encrypted to ciphertext bytes $\{C_{\overline{\mathtt{TSC}},j,r}\}_{1 \le j \le T}$ in bin $\mathcal{S}_{\overline{\mathtt{TSC}}}$ now follows a multinomial distribution:

$$\lambda_{\overline{\mathtt{TSC}},\mu} = \frac{T!}{N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0x00}}! \cdots N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0xFF}}!} \prod_{k \in \{\mathtt{0x00},\ldots,\mathtt{0xFF}\}} p_{\overline{\mathtt{TSC}},r,k}^{N^{(\mu)}_{\overline{\mathtt{TSC}},k}} \ . \tag{3}$$

The probability that plaintext byte μ is encrypted to ciphertext bytes $\{C_{\overline{\mathtt{TSC}},j,r}\}_{1 \le j \le T}$ across all bins $\mathcal{S}_{\overline{\mathtt{TSC}}}$ can then be precisely calculated as

$$\lambda_\mu = \prod_{(\mathtt{0x00},\mathtt{0x00}) \le \overline{\mathtt{TSC}} \le (\mathtt{0xFF},\mathtt{0xFF})} \lambda_{\overline{\mathtt{TSC}},\mu} \ .$$

By computing λ_μ for all $\mathtt{0x00} \le \mu \le \mathtt{0xFF}$, and identifying μ such that λ_μ is largest, we determine the (optimal) maximum-likelihood plaintext byte value. This informal description, together with some optimisations that we describe next, is specified in algorithmic form in Algorithm 4.

Observe that, for each fixed position r and set of ciphertexts $\{C_{\overline{\mathtt{TSC}},j,r}\}_{1 \le j \le T}$, values $N^{(\mu)}_{\overline{\mathtt{TSC}},k}$ can be computed from values $N^{(\mu')}_{\overline{\mathtt{TSC}},k}$ by equation $N^{(\mu)}_{\overline{\mathtt{TSC}},k} = N^{(\mu')}_{\overline{\mathtt{TSC}},k \oplus \mu' \oplus \mu}$, for all k. In other words, for a fixed $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair, vectors $(N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0x00}}, \ldots, N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0xFF}})$ and $(N^{(\mu')}_{\overline{\mathtt{TSC}},\mathtt{0x00}}, \ldots, N^{(\mu')}_{\overline{\mathtt{TSC}},\mathtt{0xFF}})$ are permutations of each other; by consequence, the term $T!/(N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0x00}}! \cdots N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0xFF}}!)$ in equation (3) is a constant for each choice of μ (but not necessarily constant across different values for the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair). If T is fixed (as we assume it to be), then the $T!$ terms can all be omitted from all calculations. Furthermore, computing and comparing $\log(\lambda_{\overline{\mathtt{TSC}},\mu})$ and $\log(\lambda_\mu)$ instead of $\lambda_{\overline{\mathtt{TSC}},\mu}$ and λ_μ makes the computation more efficient and accuracy easier to maintain.

Comparing with Algorithm 3, we see that our new Algorithm 4, at its core, runs Algorithm 3 once for each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair, and then combines the resulting likelihood estimates $\lambda_{\overline{\mathtt{TSC}},\mu}$ to obtain the final estimate λ_μ for plaintext candidate μ. Some care is needed, however, to use the correct scaling factors ($T!$ and $N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0x00}}! \cdots N^{(\mu)}_{\overline{\mathtt{TSC}},\mathtt{0xFF}}!$) for each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair.

4.3 Attack Based on Aggregation over $\mathtt{TSC}_0$ Values

As mentioned in the introduction, one method of coping with noisy estimates for the probabilities $p_{\overline{\mathtt{TSC}},r,k}$ is to consider aggregation of biases over $\mathtt{TSC}_0$. This is supported by the experiments reported in Section 3.2, where we saw that there is broad agreement between the $\mathtt{TSC}_0$-aggregated data and the data for individual $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs.

It is not difficult to see how to modify Algorithm 4 to work with 2^8 bins, one for each value of $\mathtt{TSC}_1$, instead of 2^{16} bins. The execution of the modified

Algorithm 4: Plaintext recovery attack using $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning

input : $\{C_{\overline{\mathtt{TSC}},j}\}_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF}),1\leq j\leq T}$ – $S = 2^{16}\cdot T$ independent encryptions of fixed plaintext P
r – byte position
$(p_{\overline{\mathtt{TSC}},r,k})_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF}),\mathtt{0x00}\leq k\leq\mathtt{0xFF}}$ – keystream distributions for all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs at position r
output: P_r^* – estimate for plaintext byte P_r
begin
  $N_{(\mathtt{0x00},\mathtt{0x00}),\mathtt{0x00}} \leftarrow 0, \ldots, N_{(\mathtt{0xFF},\mathtt{0xFF}),\mathtt{0xFF}} \leftarrow 0$
  for $\overline{\mathtt{TSC}} = (\mathtt{0x00}, \mathtt{0x00})$ **to** $(\mathtt{0xFF}, \mathtt{0xFF})$ **do**
    for $j = 1$ **to** T **do**
      $k \leftarrow C_{\overline{\mathtt{TSC}},j,r}$
      $N_{\overline{\mathtt{TSC}},k} \leftarrow N_{\overline{\mathtt{TSC}},k} + 1$
  for $\overline{\mathtt{TSC}} = (\mathtt{0x00}, \mathtt{0x00})$ **to** $(\mathtt{0xFF}, \mathtt{0xFF})$ **do**
    $F_{\overline{\mathtt{TSC}}} \leftarrow \sum_{\mathtt{0x00}\leq j\leq\mathtt{0xFF}} \log((N_{\overline{\mathtt{TSC}},j})!)$
    for $\mu = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
      for $k = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
        $N^{(\mu)}_{\overline{\mathtt{TSC}},k} \leftarrow N_{\overline{\mathtt{TSC}},k\oplus\mu}$
      $\lambda_{\overline{\mathtt{TSC}},\mu} \leftarrow -F_{\overline{\mathtt{TSC}}} + \sum_{k=\mathtt{0x00}}^{\mathtt{0xFF}} N^{(\mu)}_{\overline{\mathtt{TSC}},k} \log(p_{\overline{\mathtt{TSC}},r,k})$
  for $\mu = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
    $\lambda_\mu \leftarrow \sum_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF})} \lambda_{\overline{\mathtt{TSC}},\mu}$
  $P_r^* \leftarrow \arg\max_{\mu\in\{\mathtt{0x00},\ldots,\mathtt{0xFF}\}} \lambda_\mu$
  return P_r^*

algorithm becomes in practice faster, since each estimate for a plaintext byte μ now only involves calculation of $\lambda_{\overline{\mathtt{TSC}},\mu}$ over 2^8 $\mathtt{TSC}_1$ values instead of 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair values.

4.4 Further Optimizations

In specific settings where the attacker has *a priori* information about the encrypted plaintext the performance of Algorithms 3 and 4 can be further improved. Here, the considerations are similar to those in [3] and so we omit further discussion.

5 Experimental Results

In this section, we report on the results obtained by simulating the plaintext recovery attacks described in Section 4. To be exact, we did not mount the attacks against real TKIP traffic, but instead generated TKIP ciphertexts corresponding to a plaintext consisting of `0x00` bytes and then tested our attacks' abilities to recover this plaintext.

5.1 Attack Using Fully Aggregated Biases

We first ran 256 times the attack in Algorithm 3 for each of $S = 2^{24}, 2^{26}, 2^{28}, 2^{30}$ simulated frames to estimate the attack's success rate. We used the fully aggregated biases described in Section 3.1 in the attack. The results are shown in Figures 9(a)–9(d), which display the success rate of recovering the correct plaintext byte versus the byte position r in the keystream. Some notable features of these figures are:

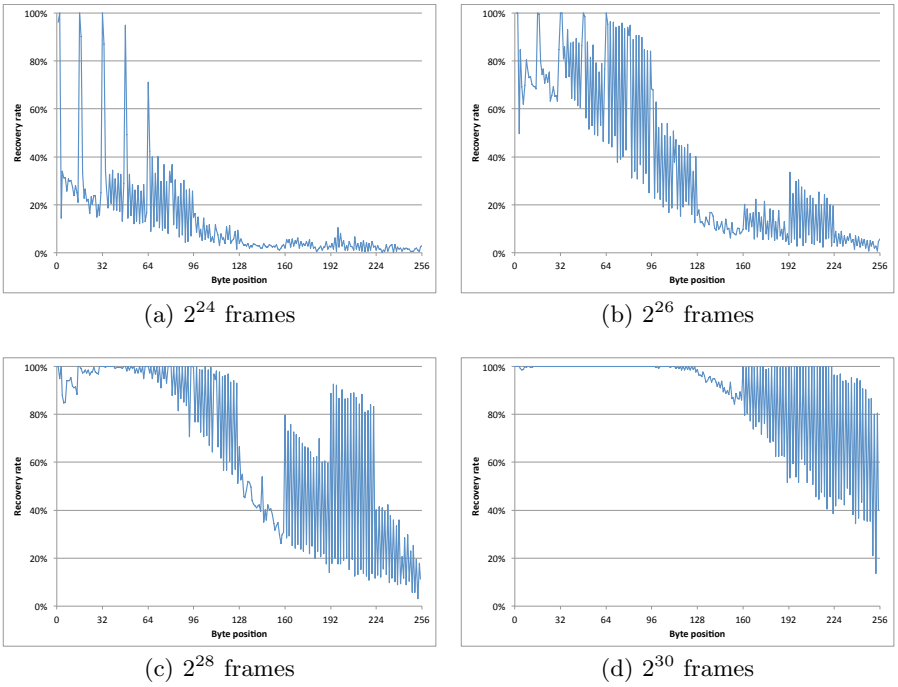
- With $S = 2^{26}$ frames, the first 55 plaintext bytes are recovered with rate at least 50% per byte. Comparing Figure 9(b) with the corresponding Figure 5(a) from [3] created for random 128-bit keys reveals that many plaintext bytes are recovered with a significantly higher rate in the TKIP case, leading to a higher average recovery rate.
- With $S = 2^{30}$ frames, the first 130 plaintext bytes are recovered with rate close to 100%; the first 211 bytes are recovered with rate at least 50%. Note again that, while according to Figure 5(c) in [3] for random 128-bit keys the first 251 bytes are recovered with at least 50% probability, in the TKIP case many plaintext bytes are recovered with significantly higher probability.
- Independently of the number S of considered frames, the recovery rate is highly correlated with the strength of the bias towards $\mathtt{0x00}$ at the same position: to see this, compare Figures 9(a)–9(d) with Figure 3.

By comparing Figure 9 with the corresponding figures in [3], we observe that the structure on keys enforced by (1), which was aiming to 'preclude the use of known RC4 weak keys' [2] (to prevent WEP key recovery), effectively allows easier recovery of plaintext bytes than with uniform keys, at least in some positions.

5.2 Attacks Using TSC Binning

Secondly, we simulated the attack based on $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair binning described in Algorithm 4, as well as the variant of the attack described in Section 4.3 which aggregates the biases over all $\mathtt{TSC}_0$ values. For both attacks, we used per-output-byte probabilities $\{p_{\overline{\mathtt{TSC}},r,k}\}_{1 \leq r \leq 256, \mathtt{0x00} \leq k \leq \mathtt{0xFF}}$ derived from the keystream distribution estimate described in Section 3.2. Recall that this estimate was generated based on 2^{24} RC4 keystreams for each of the 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs. To judge the effect of noise in the keystream distribution estimate, we furthermore simulated the attacks using an idealised estimate. Specifically, we used a modified set of per-output-byte probabilities $\{p'_{\overline{\mathtt{TSC}},r,k}\}_{1 \leq r \leq 256, \mathtt{0x00} \leq k \leq \mathtt{0xFF}}$ for which all probabilities corresponding to a bias below a threshold of four times the standard deviation for a normal distribution were replaced by the average value of these probabilities. All of the simulations were done for 2^{24} frames and each attack was run 256 times. The resulting recovery rates are shown in Figures 10(a)–10(d). We make the following observations:

- The recovery rate for all byte positions improves significantly for the attack based on full $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning when using an idealised keystream distribution estimate. This indicates that the level of noise in our keystream



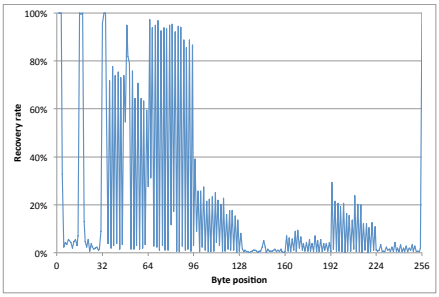
(a) 2^{24} frames (b) 2^{26} frames

(c) 2^{28} frames (d) 2^{30} frames

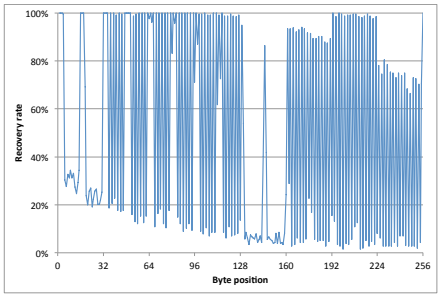
Fig. 9. Success rate for 256 runs of attack based on fully aggregated TKIP biases using 2^{24}, 2^{26}, 2^{28} and 2^{30} simulated frames (4, 16, 64, and 256 keys generating 2^{22} frames each).

distribution estimate based on 2^{24} RC4 keystreams for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs has an adverse effect on the recovery rate and is too significant for the binning attack to work optimally.

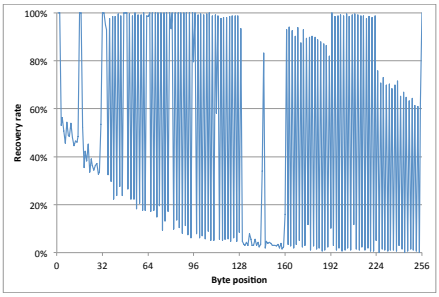
- The recovery rate for the attack based on aggregation over $\texttt{TSC}_0$ values is very similar when using idealised and non-idealised keystream distribution estimates. The recovery rate in the latter case is in fact slightly higher than in the former. This indicates that the idealisation, using a threshold of four times the standard deviation of the normal distribution, removes structure from the keystream distribution estimate that would otherwise improve the recovery rate, and that the level of noise does not have a significant effect on the recovery rate. Note that when aggregating over all $\texttt{TSC}_0$ values, the keystream distribution estimate for each $\texttt{TSC}_1$ value is based on 2^{32} RC4 keystreams.
- The recovery rate for the attack based on aggregation over all $\texttt{TSC}_0$ values is noticeably higher than the recovery rate for the attack based on full $(\texttt{TSC}_0, \texttt{TSC}_1)$ binning, even if using an idealised keystream distribution estimate in the latter case.



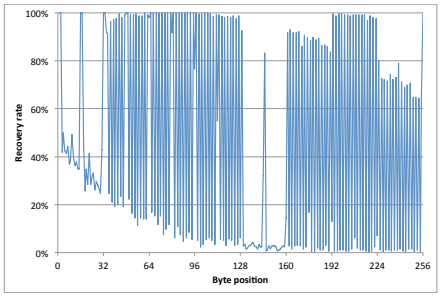
(a) Attack based on $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair binning using non-idealised keystream distribution estimate.



(b) Attack based on $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair binning using idealised keystream distribution estimate.



(c) Attack based on aggregation of $\mathtt{TSC}_0$ values using non-idealised keystream distribution estimate.

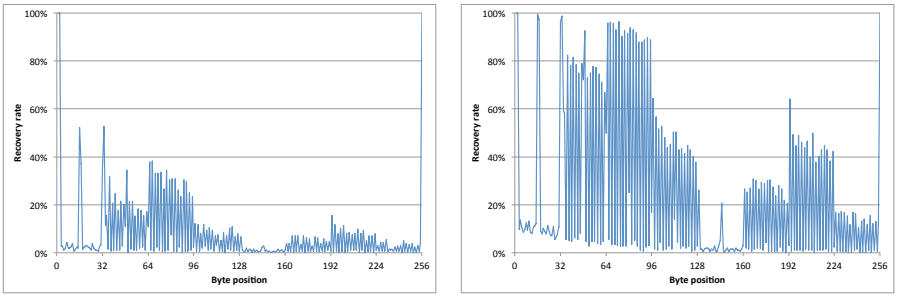


(d) Attack based on aggregation of $\mathtt{TSC}_0$ values using idealised keystream distribution estimate.

Fig. 10. Success rate for attacks on TKIP based on $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair binning and aggregation of $\mathtt{TSC}_0$ values, for both idealised and non-idealised keystream distribution estimates. All success rates are based on 256 runs of each attack using 2^{24} simulated frames.

Based on the above observations, we decided to study in more detail the attack based on aggregation over all $\mathtt{TSC}_0$ values using a non-idealised keystream estimate. Specifically, we ran the simulation of this attack 256 times for $S = 2^{20}$, 2^{22}, 2^{24}, 2^{26}, 2^{28} simulated frames. The resulting recovery rates can be seen in Figures 10(c) and 11(a)–11(d). We observe the following:

- Even with as few as $S = 2^{20}$ frames, a few positions of the plaintext are correctly recovered with high probability. In particular, byte positions $1, 2$ and 256 are recovered with a rate of 100%, whereas positions that are low multiples of 16 are recovered with a rate higher than 50%.
- With $S = 2^{22}$ frames, 26 positions are recovered with a rate higher than 80%, and the average recovery rate is 24%. In comparison, for 2^{24} frames, the attack using fully aggregated biases recovers only 7 positions with a rate higher than 80% and has an average recovery rate of 13% (cf. Figure 9(a)).



(a) 2^{20} frames (b) 2^{22} frames

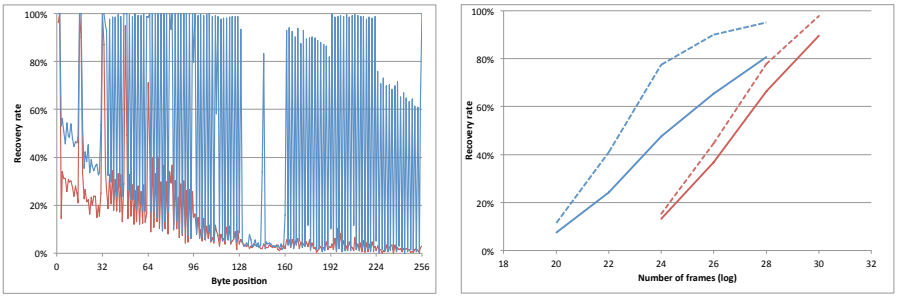
(c) 2^{26} frames (d) 2^{28} frames

Fig. 11. Success rate for attack on TKIP based on aggregation of $\texttt{TSC}_0$ values for 256 simulations, each using 2^{20}, 2^{22}, 2^{26}, and 2^{28} frames. See Figure 10(c) for the success rate for 2^{24} frames.

Furthermore, the recovery rate for even positions is comparable to that of the attack using fully aggregated biases for 2^{26} frames (cf. Figure 9(b)).

- With $S = 2^{26}$ frames, 146 positions are recovered with a rate higher than 80%, and the average recovery rate has increased to 65%. This is similar to the corresponding numbers for the attack using fully aggregated biases, but with 2^{28} frames. The recovery rate for the even positions has furthermore increased to 90%.
- For up to $S = 2^{26}$ frames, the number of positions recovered with a rate above 80% and the average recovery rate seems to exceed or match the corresponding numbers for the attack using fully aggregated biases with four times the number of frames. For even positions, the number of frames required in the attack using fully aggregated biases to get comparable recovery rates seems to be a factor of 16 larger.

To enable easy comparison of the different attacks, Figure 12(a) shows the recovery rate for 2^{24} frames for both the attack using fully aggregated biases and the attack based on aggregation of $\texttt{TSC}_0$ values, and Figure 12(b) shows the average recovery rates of the two attacks as the number of frames increases. As can be seen from these figures and the comparisons made above, the attack



(a) Recovery rate for 2^{24} frames. (b) Average recovery rate.

Fig. 12. Comparison of attack using fully aggregated biases (red) and attack based on aggregation over all $\mathtt{TSC}_0$ values (blue). Figure (a) shows the recovery rates of the two attacks for 256 runs each with 2^{24} simulated frames. Figure (b) shows the average recovery rates of the two attacks. The dashed lines correspond to the average recovery rates for even positions.

based on aggregation of $\mathtt{TSC}_0$ values is noticeably more successful in correctly recovering the correct plaintext bytes, in particular at even plaintext positions.

6 Practical Impact, Countermeasures and Open Problems

We have shown that plaintext recovery for RC4 in WPA is possible for the first 256 bytes of a frame, provided sufficiently many independent encryptions of the same plaintext are available. Certainly, the security level provided by RC4 is *far* below the strength implied by the 128-bit key in WPA. We are confident that the attacks could be improved further by using more accurate estimates of the TKIP keystream distributions in our full $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning attack; obtaining these estimates is simply a matter of computation.

6.1 Practical Impact

Concerning the practical impact of our attacks, we note that WPA frames are quite likely to contain fixed, but *known* bytes, as well as fixed but *unknown* bytes. Examples of the former were already used in keystream recovery attacks against TKIP, as a prelude to MIC key recovery and frame injection attacks — see, for example, [20,23]. The latter would be suitable targets for our attacks and would include fields in IP, UDP and TCP headers, such as source and destination IP addresses, the IP header protocol field, and UDP and TCP port numbers. Another target of potential interest would be passwords or cookies in HTTP traffic (that are not already protected with TLS), with Javascript running in a browser as in [3] providing one possible mechanism to generate the traffic needed in the attacks.

We do not claim that our work enables practical attacks against WPA, in the same way that the work of [20,23] does, for example. Rather our work unveils some fundamental weaknesses in the way in which RC4 is employed in WPA which make it easier to attack in the broadcast setting than should be the case. In this sense, our paper places limits on the security that can be achieved by WPA.

Our work does bear further comparison with previous attacks on WPA, however. In particular, we stress that our attacks are passive, ciphertext-only attacks, with modest ciphertext requirements. This contrasts with the active attacks of [20,23] and the known-plaintext attack of [19]. The active attacks are rate-limited, in that they cannot recover more than 1 byte of plaintext per minute (this is because of peculiarities of the way in which WPA reacts to MAC errors). The attack of [19] requires 2^{38} known plaintexts and has complexity 2^{96}. On the other hand, our attacks have a repeated (but unknown) plaintext requirement and can only target the initial bytes in a frame; furthermore, we only recover plaintext, in contrast to the key-recovery attack of [19].

6.2 Countermeasures

There are some countermeasures to the attacks. These include:

- Discarding the initial keystream bytes output by RC4, as recommended in [12] (but then double-byte-bias attacks like those developed in [3] may still be applicable).
- Changing the manner in which WPA's RC4 keys are computed (but then the analysis of RC4 in TLS from [3] might apply, so security might be increased, but not all known attacks eliminated).
- Abandoning TKIP and switching to CCMP, a confidentiality mode that is based on the CCM authenticated encryption scheme.

Of these, the third is the only one that can be recommended, since the first two would still leave vulnerabilities and would require further changes to the WPA/TKIP specification.

6.3 Open Problems

Open problems suggested by this paper include:

- Carrying out a larger-scale computation to obtain more accurate estimates of the per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair keystream distributions, and investigating the effect of using these better estimates in our $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning attack.
- Explaining the genesis of the RC4 keystream biases when TKIP keys are used. This has already been completed to some extent for the case of random 128-bit keys (as used in TLS) in [15], and it seems plausible that similar techniques could be deployed for the TKIP case. Indeed, recent progress on this problem has been made in [16]. A full theoretical description of the TKIP biases would obviate the need for extensive computations to establish the keystream distributions.

– Extending the 2-byte plaintext recovery attack of [3] to TKIP. This would, potentially, enable plaintext recovery attacks beyond the first 256 positions in each frame. It is also possible that there are strong dependencies between adjacent pairs of keystream bytes in the initial positions. A large computation would be needed to test this.
– Exploring whether it is possible to combine our attack methods with those of [19] to avoid the onerous known plaintext requirements of those previous attacks, and investigating whether it is possible to improve TKIP `TK` key recovery attacks further by using TSC-specific biases.
– Studying other applications of RC4 in which the RC4 key is changed frequently.

Acknowledgements

We thank Jon Hart of the ISG at RHUL for his assistance with computing infrastructure. The research of the authors was supported by an EPSRC Leadership Fellowship, EP/H005455/1.

References

1. Wireless LAN medium access control (MAC) and physical layer (PHY) specification (1997)
2. Wireless LAN medium access control (MAC) and physical layer (PHY) specification: Amendment 6: Medium access control (MAC) security enhancements (2004)
3. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.N.: On the security of RC4 in TLS. In: USENIX Security (2013), `https://www.usenix.org/conference/usenixsecurity13/security-rc4-tls`
4. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: The insecurity of 802.11. In: Rose, C. (ed.) MOBICOM. pp. 180–189. ACM (2001)
5. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2259, pp. 1–24. Springer (2001)
6. Fluhrer, S.R., McGrew, D.: Statistical analysis of the alleged RC4 keystream generator. In: Schneier, B. (ed.) FSE. Lecture Notes in Computer Science, vol. 1978, pp. 19–30. Springer (2000)
7. Halvorsen, F.M., Haugen, O., Eian, M., Mjølsnes, S.F.: An improved attack on TKIP. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec. Lecture Notes in Computer Science, vol. 5838, pp. 120–132. Springer (2009)
8. Jaganathan, K., Zhu, L., Brezak, J.: The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows. RFC 4757 (Informational) (Dec 2006), `http://www.ietf.org/rfc/rfc4757.txt`
9. Maitra, S., Paul, G., Sen Gupta, S.: Attack on broadcast RC4 revisited. In: Joux, A. (ed.) FSE. Lecture Notes in Computer Science, vol. 6733, pp. 199–217. Springer (2011)
10. Mantin, I.: Predicting and distinguishing attacks on RC4 keystream generator. In: Cramer, R. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 491–506. Springer (2005)

11. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE. Lecture Notes in Computer Science, vol. 2355, pp. 152–164. Springer (2001)
12. Mironov, I.: (Not so) random shuffles of RC4. In: Yung, M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2442, pp. 304–319. Springer (2002)
13. Moen, V., Raddum, H., Hole, K.J.: Weaknesses in the temporal key hash of WPA. Mobile Computing and Communications Review 8(2), 76–83 (2004)
14. Morii, M., Todo, Y.: Cryptanalysis for RC4 and breaking WEP/WPA-TKIP. IEICE Transactions 94-D(11), 2087–2094 (2011)
15. Sarkar, S., Sen Gupta, S., Paul, G., Maitra, S.: Proving TLS-attack related open biases of RC4. Cryptology ePrint Archive, Report 2013/502 (2013), `http://eprint.iacr.org/`
16. Sen Gupta, S., Maitra, S., Meier, W., Paul, G., Sarkar, S.: Some results on RC4 in WPA. Cryptology ePrint Archive, Report 2013/476 (2013), `http://eprint.iacr.org/`
17. Sen Gupta, S., Maitra, S., Paul, G., Sarkar, S.: (Non-) random sequences from (non-) random permutations – analysis of RC4 stream cipher. Journal of Cryptology 27(1), 67–108 (2014)
18. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Discovery and exploitation of new biases in RC4. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 74–91. Springer (2010)
19. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Statistical attack on RC4 – distinguishing WPA. In: Paterson, K.G. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 6632, pp. 343–363. Springer (2011)
20. Tews, E., Beck, M.: Practical attacks against WEP and WPA. In: Basin, D.A., Capkun, S., Lee, W. (eds.) WISEC. pp. 79–86. ACM (2009)
21. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 bit WEP in less than 60 seconds. In: Kim, S., Yung, M., Lee, H.W. (eds.) WISA. Lecture Notes in Computer Science, vol. 4867, pp. 188–202. Springer (2007)
22. Todo, Y., Ozawa, Y., Ohigashi, T., Morii, M.: Falsification attacks against WPA-TKIP in a realistic environment. IEICE Transactions 95-D(2), 588–595 (2012)
23. Vanhoef, M., Piessens, F.: Practical verification of WPA-TKIP vulnerabilities. In: Chen, K., Xie, Q., Qiu, W., Li, N., Tzeng, W.G. (eds.) ASIACCS. pp. 427–436. ACM (2013)
24. Vaudenay, S., Vuagnoux, M.: Passive-only key recovery attacks on RC4. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4876, pp. 344–359. Springer (2007)