All Your Biases Belong To Us: Breaking RC4 in WPA-TKIP and TLS

Mathy Vanhoef and Frank Piessens, KU Leuven
USENIX Security 2015 (best student paper)

Presentation for OWASP
Intriguingly simple stream cipher

~ 10 lines in Python

WEP
WPA-TKIP
SSL / TLS
PPP/MPPE
And others ...
RC4

Intriguingly simple stream cipher

~ 10 lines in Python

Key

`RC4` → `Keystream` ⊕ `Plaintext` = `Ciphertext`
High level description

Shuffles permutation of [0..255]

| 77 | 37 | 102 | 233 | ... | 151 | 14 | 198 | 0 | 56 |

Secret index $j$
- pseudo-randomly updated value

Public index $i$
- $= \text{keystream position mod 256}$

→ Output byte selected based on index $j$ and $i$
Why study RC4?

Immune to several attacks on SSL/TLS:

- 2003: Padding oracle
- 2011: BEAST
- 2013: Lucky 13
- 2014: POODLE

Solution: use stream cipher or up-to-date TLS library

Only widely supported option was RC4

Target CBC mode encryption (block ciphers)
RC4 was heavily used!

ICSI Notary: #TLS connections using RC4

<table>
<thead>
<tr>
<th>Month</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 2013</td>
<td>50%</td>
</tr>
<tr>
<td>Februari 2015</td>
<td>30%</td>
</tr>
<tr>
<td>July 2015</td>
<td>13%</td>
</tr>
<tr>
<td>April 2016</td>
<td>3%</td>
</tr>
</tbody>
</table>
Browser support today (April 2016)

Chrome: dropped support in v48 (20 Jan. 2016)

Firefox: dropped support in v44 (26 Jan. 2016)

IE11: supports RC4

Edge: supports RC4

“will be disabled in forthcoming update”

Has fallback to RC4
Fallback to RC4

Client

- ClientHello: without RC4

Server

- Browser first tries without RC4
- ServerHello: use AES
Client

ClientHello: without RC4

Alert: Handshake Failed

Server

Browser first tries without RC4

If that fails …
ClientHello: without RC4

Alert: Handshake Failed

ClientHello: with RC4

ServerHello: use RC4

Server

Browser first tries without RC4

If that fails …

… fallback to RC4

Alert: Handshake Failed
ClientHello:

- ClientHello: with RC4
- Alert: Handshake Failed
- ClientHello: without RC4

ServerHello:

- ServerHello: use RC4

Browser first tries without RC4

- ... fallback to RC4

**Fallback provides no security**

- Forgeable by attacker!

**But useful to determine how many servers require RC4**
Contributions: how did we kill RC4?

New Biases

Plaintext Recovery

Break WPA-TKIP

Attack HTTPS

\[ \lambda_{\hat{\mu}} = (1 - \alpha(g))|C| \cdot |\hat{\nu}| \cdot \alpha(g)|\hat{\mu}| \]
Contributions: how did we kill RC4?

New Biases

\[ \lambda_{\hat{\mu}} = (1 - \alpha(g))|c| - |\hat{\mu}| \cdot \alpha(g)|\hat{\mu}| \]

Plaintext Recovery

Break WPA-TKIP

Attack HTTPS
First: Existing Biases

Distribution keystream byte 2

\[ \Pr[Z_2 = 0] = \frac{2}{256} \quad [\text{MS01}] \]
First: Existing Biases

Short-term biases

Distribution keystream byte 1 (to 256)

AlFardan et al. ‘13: first 256 bytes biased

We want a straight line..
Long-Term Biases

Fluhrer-McGrew (2000):
- Some consecutive values are biased
  Examples: (0, 0) and (0, 1)

Mantin’s ABSAB Bias (2005):
- A byte pair \((A, B)\) likely reappears
Search for new biases

Traditional empirical approach:

- Generate large amount of keystreams
- Manually inspect data or graph

Fluhrer-McGrew biases: only 8 of 65,536 pairs are biased

How to automate the search?
Search for new biases

Traditional empirical approach:
- Generate large amount of keystreams
- Manually inspect data or graph

Hypothesis tests!
- Uniformly distributed: Chi-squared test.
- Correlated: M-test (detect outliers = biases)

→ Allows a large-scale search, revealing many new biases
Example: keystream byte 258
Example: keystream byte 320
Example: keystream byte 352

Biases quickly become quite weak
New Long-term Bias

\[(Z_{256 \cdot w}, Z_{256 \cdot w+2}) = (0, 128)\]

with probability \(2^{-16}(1 + 2^{-8})\)

Every block of 256 bytes
Additional Biases

See paper!
Contributions: how did we kill RC4?

New Biases

Plaintext Recovery

Break WPA-TKIP

Attack HTTPS
Existing Methods [AlFardan et al. ‘13]

Plaintext encrypted under several keystreams

Verify guess: how close to real keystream distribution?

Induced keystream distribution
Example: Decrypt byte 1

Ciphertext Distribution
Example: Decrypt byte 1

RC4 & Ciphertext distribution
Example: Decrypt byte 1

If plaintext byte $\mu = 0x28$: **RC4 & Induced**

$\mu = 0x28$ has **low likelihood**
Example: Decrypt byte 1

If plaintext byte $\mu = 0x5C$: **RC4 & Induced**

$\mu = 0x5C$ has **higher likelihood**
Example: Decrypt byte 1

If plaintext byte $\mu = 0x5A$: **RC4 & Induced**

$\mu = 0x5A$ has **highest likelihood!**
Types of likelihood estimates

Previous works: pick value with highest likelihood.

Better idea: list of candidates in decreasing likelihood:
- Most likely one may not be correct!
- Prune bad candidates (e.g. bad CRC)
- Brute force cookies or passwords

How to calculate list of candidates?
1st idea: Generate List of Candidates

Gist of the Algorithm: Incremental approach

Calculate candidates of length 1, length 2, ...
2nd idea: abusing the ABSAB bias

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>S</td>
<td>A’</td>
<td>B’</td>
</tr>
</tbody>
</table>

Known Plaintext

Unknown Plaintext

Assume there’s surrounding known plaintext!

- Derive values of \((A, B)\)
- Combine with ABSAB bias to (probabilistically) predict \((A’, B’)\)
  - Ordinary likelihood calculation over only \((A’, B’)\)

Likelihood estimate:

\[
\lambda_{\hat{\mu}} = (1 - \alpha(g))^{\left|c - |\hat{\mu}|\right|} \cdot \alpha(g)^{|\hat{\mu}|}
\]
Contributions: how did we kill RC4?

New Biases

\[ \lambda_{\hat{\mu}} = (1 - \alpha(g))^{\lvert C \rvert - \lvert \hat{\mu} \rvert} \cdot \alpha(g)^{\lvert \hat{\mu} \rvert} \]

Plaintext Recovery

Break WPA-TKIP

Attack HTTPS
TKIP Background

How are packets sent/received?

IV | Data | MIC | CRC

Encrypted
TKIP Background

How are packets sent/received?

1. Add Message Integrity Check (MIC)
TKIP Background

How are packets sent/received?

1. Add Message Integrity Check (MIC)
2. Add CRC (leftover from WEP)
## TKIP Background

### How are packets sent/received?

<table>
<thead>
<tr>
<th>IV</th>
<th>Data</th>
<th>MIC</th>
<th>CRC</th>
</tr>
</thead>
</table>

1. Add Message Integrity Check (MIC)
2. Add CRC (leftover from WEP)
3. Add IV (increments every frame)
TKIP Background

How are packets sent/received?

- 1. Add Message Integrity Check (MIC)
- 2. Add CRC (leftover from WEP)
- 3. Add IV (increments every frame)
- 4. Encrypt using RC4 (per-packet key)
Flaw #1: TKIP Per-packet Key

(Key, IV)

per-packet key
Flaw #1: TKIP Per-packet Key

\[(IV_0, IV_1) \rightarrow \text{Anti-FMS} \rightarrow (\text{Key, IV}) \rightarrow \text{per-packet key} \rightarrow IV\text{-dependent biases in keystream} \rightarrow \text{[Gupta/Paterson et al.]} \]

(To avoid weak keys which broke WEP)
Flaw #2: MIC is invertible

If decrypted, reveals MIC key

→ With the MIC key, an attacker can inject and decrypt some packets [AsiaCCS '13]
Goal: decrypt data and MIC

<table>
<thead>
<tr>
<th>IV</th>
<th>Data</th>
<th>MIC</th>
<th>CRC</th>
</tr>
</thead>
</table>

If decrypted, reveals MIC key

Generate identical packets (otherwise MIC changes):
- Assume victim connects to server of attacker
- Retransmit identical TCP packet

List of plaintext candidates (unknown MIC and CRC)
- Prune bad candidates based on CRC
Evaluation

Simulations with $2^{30}$ candidates:

- Need $\approx 2^{24}$ captures to decrypt with high success rates

Empirical tests:

- Server can inject 2 500 packets per second
- Roughly one hour to capture sufficient traffic
- Successfully decrypted packet & found MIC key!
Contributions: how did we kill RC4?

New Biases

Plaintext Recovery

Break WPA-TKIP

Attack HTTPS
TLS Background

Client → Server

Negotiate keys

Handshake protocol

Record protocol

Encrypt data

→ Focus on record protocol with RC4 as cipher
Targeting HTTPS Cookies

Previous attacks only used Fluhrer-McGrew (FM) biases

We combine FM biases and ABSAB biases

To use ABSAB biases we first surround cookie with known data

1. Remove unknown plaintext around cookie
2. Inject known plaintext around cookie
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: a.site.com
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: auth=????????????????; P=aaaaaaaaaaaaaaa

Surrounded by known plaintext at both sides

Headers are predictable
Preparation: manipulating cookies

a.site.com  HTTPS  fake.site.com

Client

Remove & inject secure cookies!
Performing the attack!

JavaScript: Cross-Origin requests in WebWorkers

STEP 1
Attacker injects code to generate requests
Performing the attack!

**STEP 1**
Attacker injects code to generate requests

**STEP 2**
Attacker captures the encrypted requests

Keep-Alive connection to generate them fast
Performing the attack!

**STEP 1**
Attacker injects code to generate requests

**STEP 2**
Attacker captures the encrypted requests

https://site.com

**STEP 3**
Attacker computes likely cookies and tries each one

Combine Fluhrer-McGrew and ABSAB biases
Decrypting 16-character cookie

- 2^{23} candidates
- 1 candidate

\(~ one billion encryptions of cookie\)

Ciphertext copies times 2^{27}
Decrypting 16-character cookie

Takes 75 hours with 4450 requests / second

~ one billion encryptions of cookie

Ciphertext copies times $2^{27}$
Decrypting 16-character cookie

DEMO!

rc4nomore.com
Questions?

May the bias be ever in your favor
Questions?

*May the bias be ever in your favor*
Distribution keystream byte 513

P-value $\approx 10^{-300}$

$2^{47}$ keystreams
Additional Biases

Short-Term:
- $Z_1$ and $Z_2$ influence initial 256 bytes
- Consecutive bytes likely (in)equal

Long-term Biases:
- Byte value “likely” reappears

See paper!
Keystream bytes $Z_1$ and $Z_2$

$Z_1$ and $Z_2$ influence all initial 256 bytes

- $Z_1 = 257 - i \Rightarrow Z_i = 0$
Keystream bytes $Z_1$ and $Z_2$

$Z_1$ and $Z_2$ influence all initial 256 bytes

- $Z_1 = 257 - i \Rightarrow Z_i = 0$
- $Z_1 = 257 - i \Rightarrow Z_i = i$
Keystream bytes $Z_1$ and $Z_2$

$Z_1$ and $Z_2$ influence all initial 256 bytes

- $Z_1 = 257 - i \Rightarrow Z_i = 0$
- $Z_1 = 257 - i \Rightarrow Z_i = i$
- $Z_2 = 0 \Rightarrow Z_i \neq i$
Keystream bytes $Z_1$ and $Z_2$

$Z_1$ and $Z_2$ influence all initial 256 bytes

- $Z_1 = 257 - i \Rightarrow Z_i = 0$
- $Z_1 = 257 - i \Rightarrow Z_i = i$
- $Z_2 = 0 \Rightarrow Z_i \neq i$
- And others

| 1 | 2 | ... | 50 | 51 | ...
|---|---|-----|----|----|---
Example: Decrypt byte 1

If plaintext byte $\mu = 0x28$: **RC4 & Induced**

Likelihood of $\mu = \text{probability of witnessing induced, given the RC4 distribution}
Example: Decrypt byte 1

If plaintext byte $\mu = 0x5C$: **RC4 & Induced**

Likelihood of $\mu$ = probability of witnessing *induced*, given the **RC4 distribution**
Example: Decrypt byte 1

If plaintext byte $\mu = 0x5A$: **RC4 & Induced**

Likelihood of $\mu = \text{probability of witnessing induced, given the RC4 distribution}$