Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS

Christina Garman[†] *Kenny Paterson*[‡] **Thyla van der Merwe**[‡]

[†] *Johns Hopkins University* [‡] *Royal Holloway, University of London*

12 August 2015

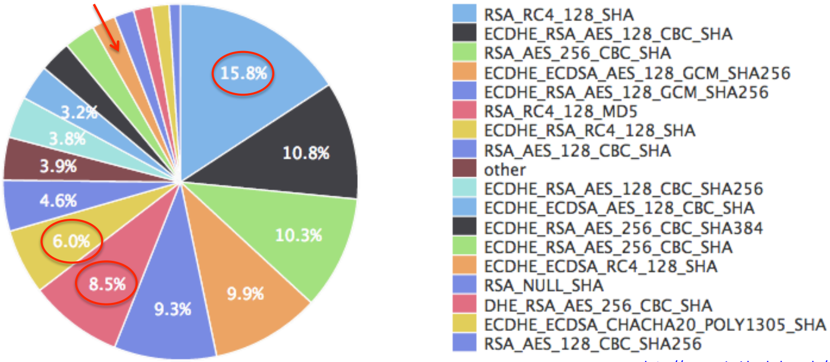
EPSRC
Engineering and Physical Sciences Research Council
Google mozilla whiteops

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Motivation

- Despite AlFardan-Bernstein-Paterson-Poettering-Schuldt (USENIX 2013), RC4 usage stood at **35%** of TLS connections

ICSI Notary Statistics [Dec., 2014]

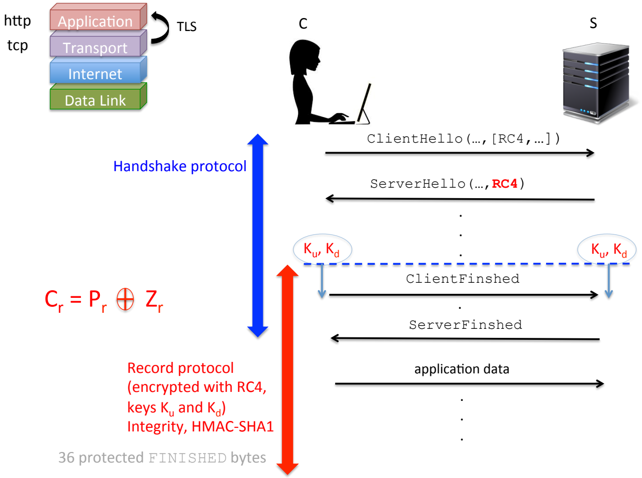


http://notary.icsi.berkeley.edu/

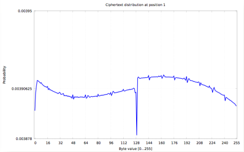
Motivation

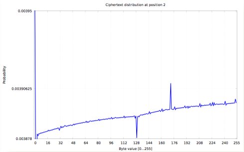
- Despite AlFardan-Bernstein-Paterson-Poettering-Schuldt (USENIX 2013), RC4 usage stood at **35%** of TLS connections
- Can we strengthen these attacks?
- Passwords are widely used for authentication and the fact that they are not uniformly distributed may give us a boost
- Get RC4 closer to the point where it needs to be abandoned!

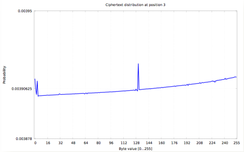
RC4 in TLS

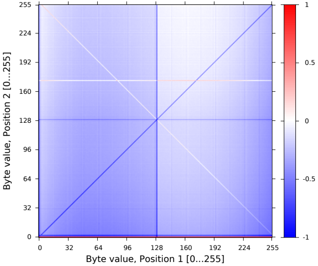

http
tcp
Application
Transport
Internet
Data Link
TLS
C
S
ClientHello(…,[RC4,…])
ServerHello(…,RC4)
Handshake protocol
K_u, K_d
K_u, K_d
ClientFinshed
ServerFinshed
$C_r = P_r \oplus Z_r$
application data
Record protocol
(encrypted with RC4,
keys K_u and K_d)
Integrity, HMAC-SHA1
36 protected FINISHED bytes

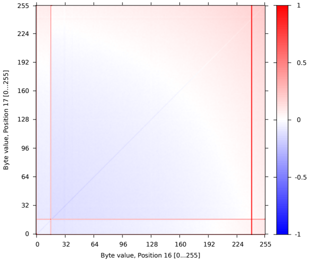
RC4 Biases


Ciphertext distribution at position 1
Probability
0.00395
0.00390625
0.003878
0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 255
Byte value [0...255]


Ciphertext distribution at position 2
Probability
0.00395
0.00390625
0.003878
0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 255
Byte value [0...255]

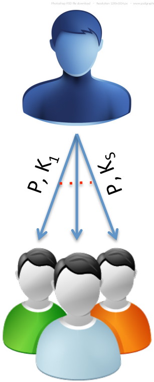

Ciphertext distribution at position 3
Probability
0.00395
0.00390625
0.003878
0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 255
Byte value [0...255]

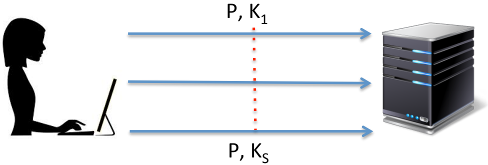

Byte value, Position 2 [0...255]
0 32 64 96 128 160 192 224 255
Byte value, Position 1 [0...255]
1 0.5 0 -0.5 -1


Byte value, Position 17 [0...255]
0 32 64 96 128 160 192 224 255
Byte value, Position 16 [0...255]
1 0.5 0 -0.5 -1

Attack Setting

- First described by Mantin and Shamir in 2001
- A fixed plaintext, P, is encrypted multiple times under independent RC4 keys, K_i





Plaintext Recovery via Bayesian Analysis

We want to maximize (for a position in the plaintext stream r):

$$\Pr(X = x \mid C = c)$$

X is the random variable corresponding to a plaintext byte, x

C is the random variable corresponding to a **vector** of ciphertext bytes

Plaintext Recovery via Bayesian Analysis

Using Bayes' Theorem:

$$\Pr(X = x \mid C = c) = \frac{\Pr(C = c \mid X = x) \cdot \Pr(X = x)}{\Pr(C = c)}$$
$$= \frac{\Pr(C = c \mid X = x) \cdot \Pr(X = x)}{\sum_{x' \in \mathcal{X}} \Pr(C = c \mid X = x') \cdot \Pr(X = x')}$$

Plaintext Recovery via Bayesian Analysis

So we actually want to maximize this:

$$\Pr(C = c \mid X = x) \cdot \Pr(X = x)$$

However,

$$\Pr(C = c \mid X = x) = \Pr(Z = z)$$

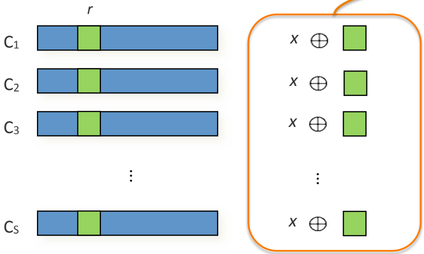
and it suffices to maximize:

$$\Pr(X = x) \cdot \Pr(Z = z)$$

Plaintext Recovery via Bayesian Analysis

encryptions of fixed byte under different keys

byte candidate x



yields induced distribution on keystream bytes Z_r

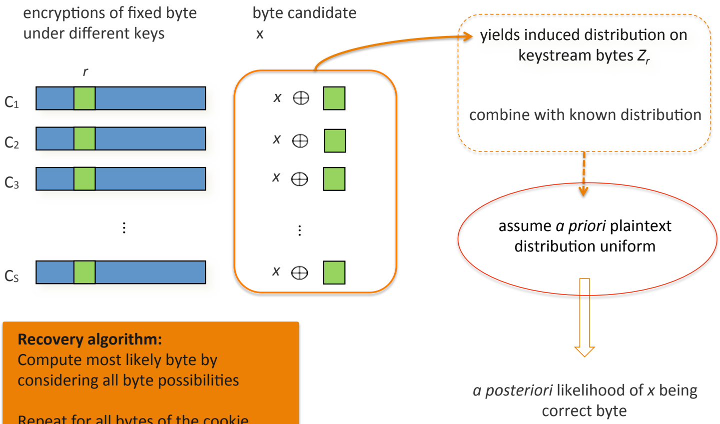
combine with known distribution

Combine with *a priori* plaintext distribution

a posteriori likelihood of *x* being correct byte

Recovery algorithm:
Compute most likely byte by considering all byte possibilities

Attacking Cookies [ABPPS13]



Recovery algorithm:
Compute most likely byte by considering all byte possibilities

Repeat for all bytes of the cookie

✗ 256 positions, 2^{34} encryptions, 2000 hrs!

Attacking Passwords

- Widely used for authentication on the web, **NOT** uniformly distributed
- `RockYou` leak of 32 million passwords in 2009, about 14 million unique, 123456 most popular
- Have *a priori* information from leaked datasets
- Multiple bytes, not just one...

Attacking Passwords

For n bytes we want to maximize

$$\Pr(X = x) \cdot \Pr(Z = z)$$

where X is the random variable corresponding to a **vector** of plaintext bytes, $x = (x_0, x_1, \ldots, x_{n-1})$

Z is the random variable corresponding to the **matrix** of keystream bytes

$$??\ \Pr(Z = z)??$$

Attacking Passwords

For n bytes we want to maximize

$$\Pr(X = x) \cdot \Pr(Z = z)$$

where X is the random variable corresponding to a **vector** of plaintext bytes, $x = (x_0, x_1, \ldots, x_{n-1})$

Z is the random variable corresponding to the **matrix** of keystream bytes

$$??\Pr(Z = z)??$$

Approximations

Pr(Z = z)

Attack 1:
Assume keystream bytes behave independently – use single-byte probabilities (product distribution)

Attack 2:
Assume keystream byte is influenced only by byte directly adjacent to it – use double- and single-byte probabilities

(Picture of the double-byte biases, 2^{44} keystreams, 4800 core-days)

Approximations

Pr(Z = z)

Attack 1:
Assume keystream bytes behave independently – use single-byte probabilities (product distribution)

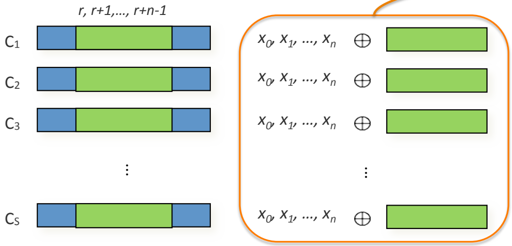
Attack 2:
Assume keystream byte is influenced only by byte directly adjacent to it – use double- and single-byte probabilities

(Picture of the double-byte biases, 2^{44} keystreams, 4800 core-days)

Approximations

encryptions of fixed password under different keys

password candidate
$x = x_0, x_1, \ldots, x_n$



yields induced distribution on keystream bytes $Z_r, Z_{r+1}, \ldots, Z_{r+n-1}$

combine with known distribution

approximate using known distribution

combine with *a priori* password distribution

⇩

a posteriori likelihood of *x* being correct password

Recovery algorithm:
Compute most likely password from dictionary of *N* passwords

What's different?

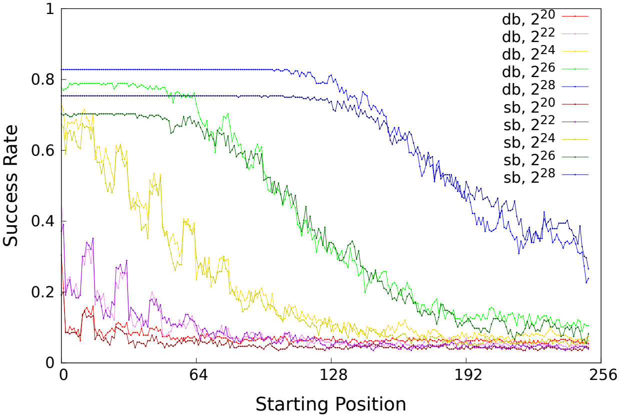
- n bytes instead of one
- T attempts before lockout
- dictionary of size N
- single-byte vs double-byte estimator
- Base64 or ASCII
- r starting position
- S ciphertexts
- **guessing attacks**

Simulation Results

- Use a dictionary built from `RockYou` leak dataset to attack `Singles.org` dataset
- More realistic but limits our success rate
- Default parameters, $n = 6,\ T = 5,\ S = 2^{20}, 2^{22}, \ldots, 2^{28}$
- Success rate based on 256 experiments

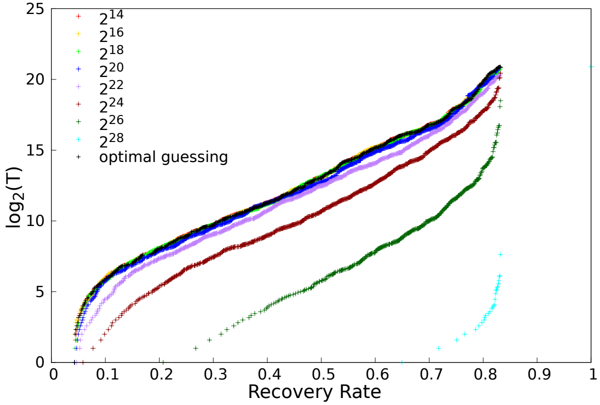
Simulation Results

Single-byte vs double-byte, $n = 6$, $T = 5$



Simulation Results

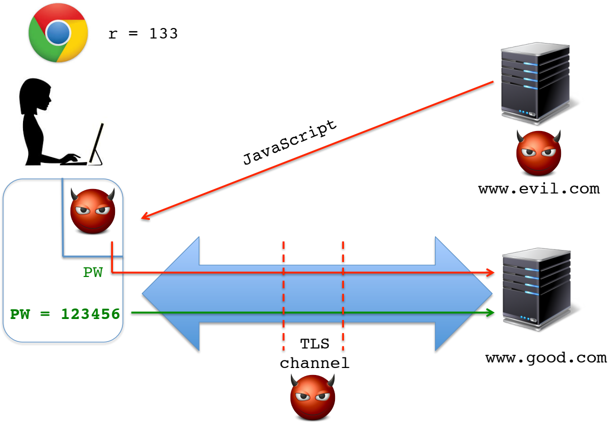
T vs success rate, $n = 6, r = 133$ - double-byte and guessing



Practical Validation

- Applicable to BasicAuth and IMAP
- We need multiple, independent encryptions of the password
- We need the password to be encrypted at a favourable position

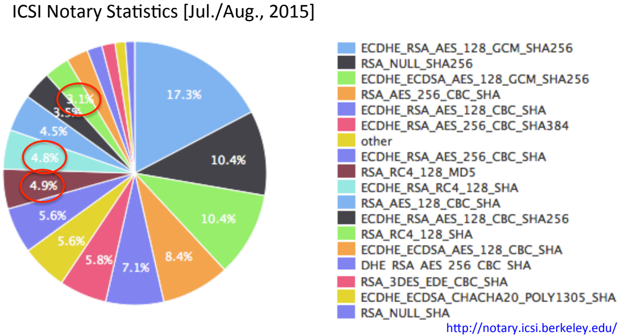
Practical Validation



- Resumption latency of 250ms, 2^{26}, 6 parallel connections, 776 hours (at 100ms, 312 hours)

Closing Remarks

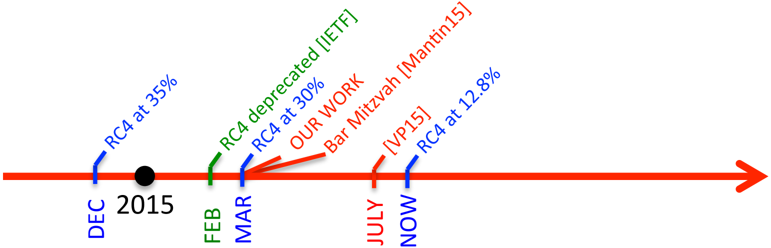
- Made use of a **generally applicable** Bayesian inference technique
- Strengthened the results of AlFardan et al., good recovery rates at 2^{26} vs. 2^{34} ciphertexts and an attack time of 312 vs. 2000 hours



12.8% of TLS connections make use of RC4

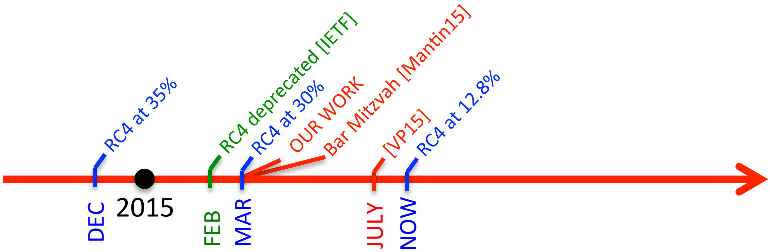
Closing Remarks

- Made use of a **generally applicable** Bayesian inference technique
- Strengthened the results of AlFardan et al., good recovery rates at 2^{26} vs. 2^{34} ciphertexts and an attack time of 312 vs. 2000 hours



Closing Remarks

- Made use of a **generally applicable** Bayesian inference technique
- Strengthened the results of AlFardan et al., good recovery rates at 2^{26} vs. 2^{34} ciphertexts and an attack time of 312 vs. 2000 hours



We need to stop using RC4!