

Big Bias Hunting in Amazonia: Large-scale Computation and Exploitation of RC4 Biases

Kenny Paterson

Information Security Group

@kennyog ; www.isg.rhul.ac.uk/~kp



ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON

Overview

- RC₄
- Attacking RC₄ in TLS
- Big bias hunting: Attacking RC₄ in WPA/TKIP
- Concluding remarks



RC₄



RC4

- Designed by Ron Rivest in late 1980s, became public in 1994.
 - A byte-oriented stream cipher.
 - Variable-length key.
 - Elegant design, fast in software, very compact description, easy to implement.
- Widely adopted in secure communications protocols:
 - TLS
 - WEP
 - WPA/TKIP
 - Kerberos
 - MPPE
 - ...

RC4

RC4 State

Byte permutation \mathcal{S} and indices i and j

RC4 Key scheduling

```
begin
  for  $i = 0$  to 255 do
    |  $\mathcal{S}[i] \leftarrow i$ 
  end
   $j \leftarrow 0$ 
  for  $i = 0$  to 255 do
    |  $j \leftarrow j + \mathcal{S}[i] + K[i \bmod \text{keylen}] \bmod 256$ 
    | swap( $\mathcal{S}[i], \mathcal{S}[j]$ )
  end
   $i, j \leftarrow 0$ 
end
```

RC4 Keystream generation

```
begin
   $i \leftarrow i + 1 \bmod 256$ 
   $j \leftarrow j + \mathcal{S}[i] \bmod 256$ 
  swap( $\mathcal{S}[i], \mathcal{S}[j]$ )
   $Z \leftarrow \mathcal{S}[\mathcal{S}[i] + \mathcal{S}[j] \bmod 256]$ 
  return  $Z$ 
end
```

Cryptanalysis of RC₄ (Or: Isn't RC₄ Broken Already?)

- Given its wide-spread use, RC₄ has been subject to a lot of cryptanalysis.
 - Biases in keystreams
 - Key/state recovery attacks
 - Related key attacks
- It's usage in WEP was completely broken, starting with [FMSo1].
 - Full key-recovery attack now possible with 10k-20k packets [SVV11].
- Many short-term and long-term biases in its keystreams have been identified.
 - [FM00], [MSo1], [Mo2], [Mo5], [MPS11], [SMPS11],...
- Why then is it still so popular in applications?

Popularity of RC₄

- It's fast and easy to implement.
- It's hard to displace a widely-deployed algorithm without practical, demonstrated attacks.
- The WEP disaster can be argued as a special case, where the *use* of the algorithm was at fault, not the algorithm itself.
 - Composition of key from long-term key and public counter enabled special attacks.
- Lack of practical, demonstrated attacks on common applications.

Attacking RC₄ in TLS

Joint work with Nadhem AlFardan, Daniel J. Bernstein, Bertram Poettering and Jacob C.N. Schuldt

Broadcast Attack Setting

- Introduced by Mantin-Shamir in 2001.
- Imagine a fixed but unknown plaintext P is encrypted many times under RC_4 using different keys K_i .
- Attack recovers bytes of P by exploiting biases in RC_4 keystreams.
- Different from usual setting: recover K from many (P,C) pairs.

Broadcast Attack – Example

- **Example:**

- Mantin-Shamir bias: $\Pr[Z_2 = 0x00] \approx 1/128$
- But $C_r = P_r \oplus Z_r$.
- So $C_2 = P_2$ with probability $1/128$.
- Hence, with enough encryptions, can recover P_2 directly from C_2 .
- Just take the most common value of C_2 as estimate for P_2 !

- Does the attack extend to other bytes of plaintext?
- Is the attack really applicable to RC4 in TLS?
- How many ciphertexts are needed for reliable plaintext recovery?

Single-byte Biases in the RC₄ Keystream

Z_i = value of i -th keystream byte

[Mantin-Shamir 2001]:

$$\Pr[Z_2 = 0] \approx \frac{1}{128}$$

[Mironov 2002]:

Described distribution of Z_1 (bias away from 0, sine-like distribution)

[Maitra-Paul-Sen Gupta 2011]: for $3 \leq r \leq 255$

$$\Pr[Z_r = 0] = \frac{1}{256} + \frac{c_r}{256^2} \quad 0.242811 \leq c_r \leq 1.337057$$

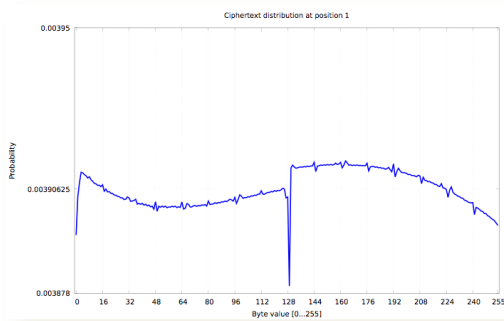
[Sen Gupta-Maitra-Paul-Sarkar 2011]:

$$\Pr[Z_l = 256 - l] \geq \frac{1}{256} + \frac{1}{256^2} \quad l = \text{keylength}$$

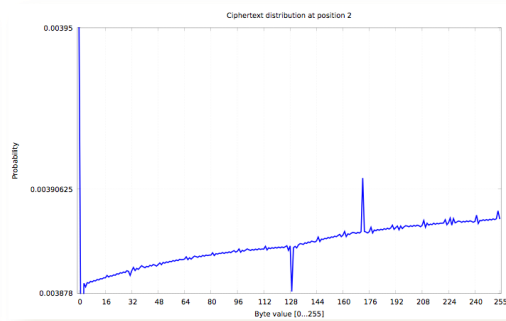
Complete Single-byte Keystream Distributions

Approach in [ABPPS13]:

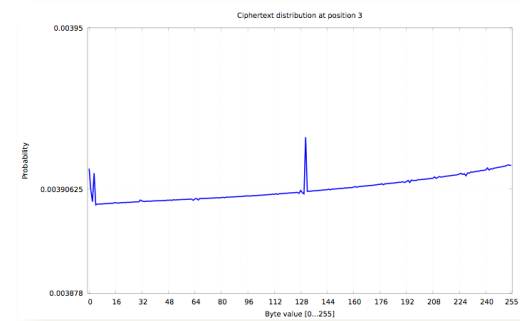
Based on the output from 2^{45} random independent 128-bit RC4 keys, estimate the keystream byte distributions for the first 256 bytes



Z_1



Z_2



Z_3

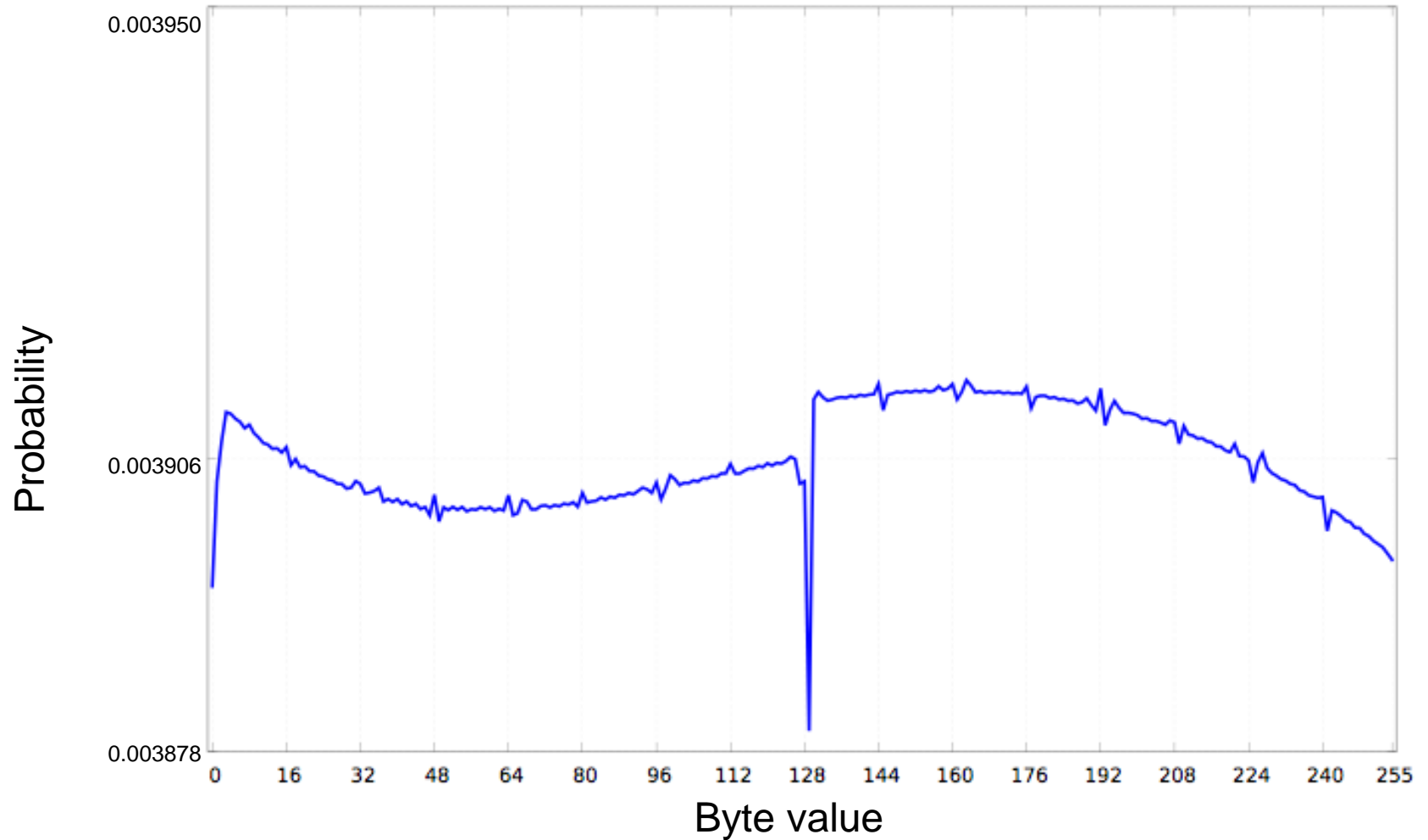
...

...

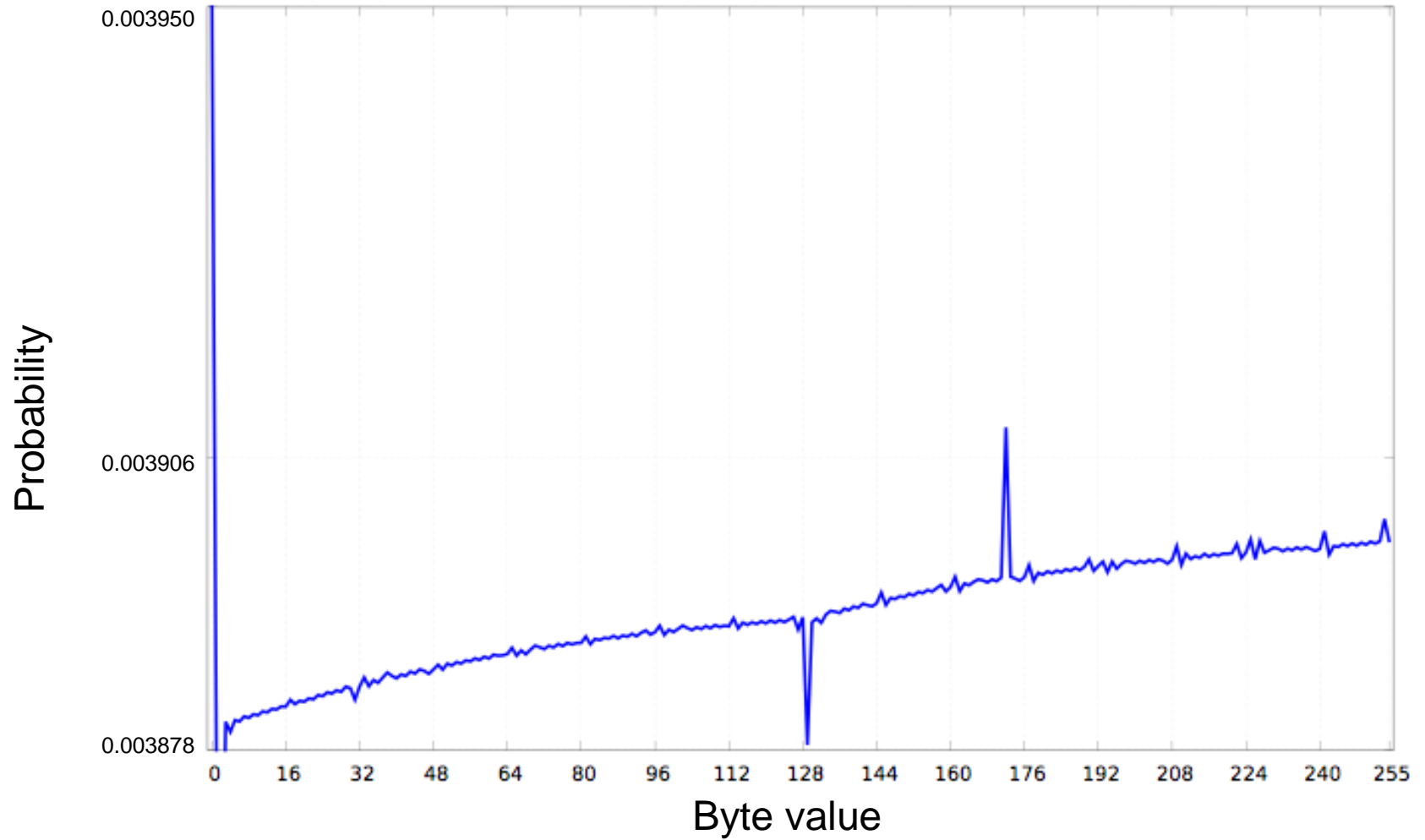
This computation revealed many new biases in the RC4 keystream.

(Some of these were independently discovered and exploited in [IOWM13].)

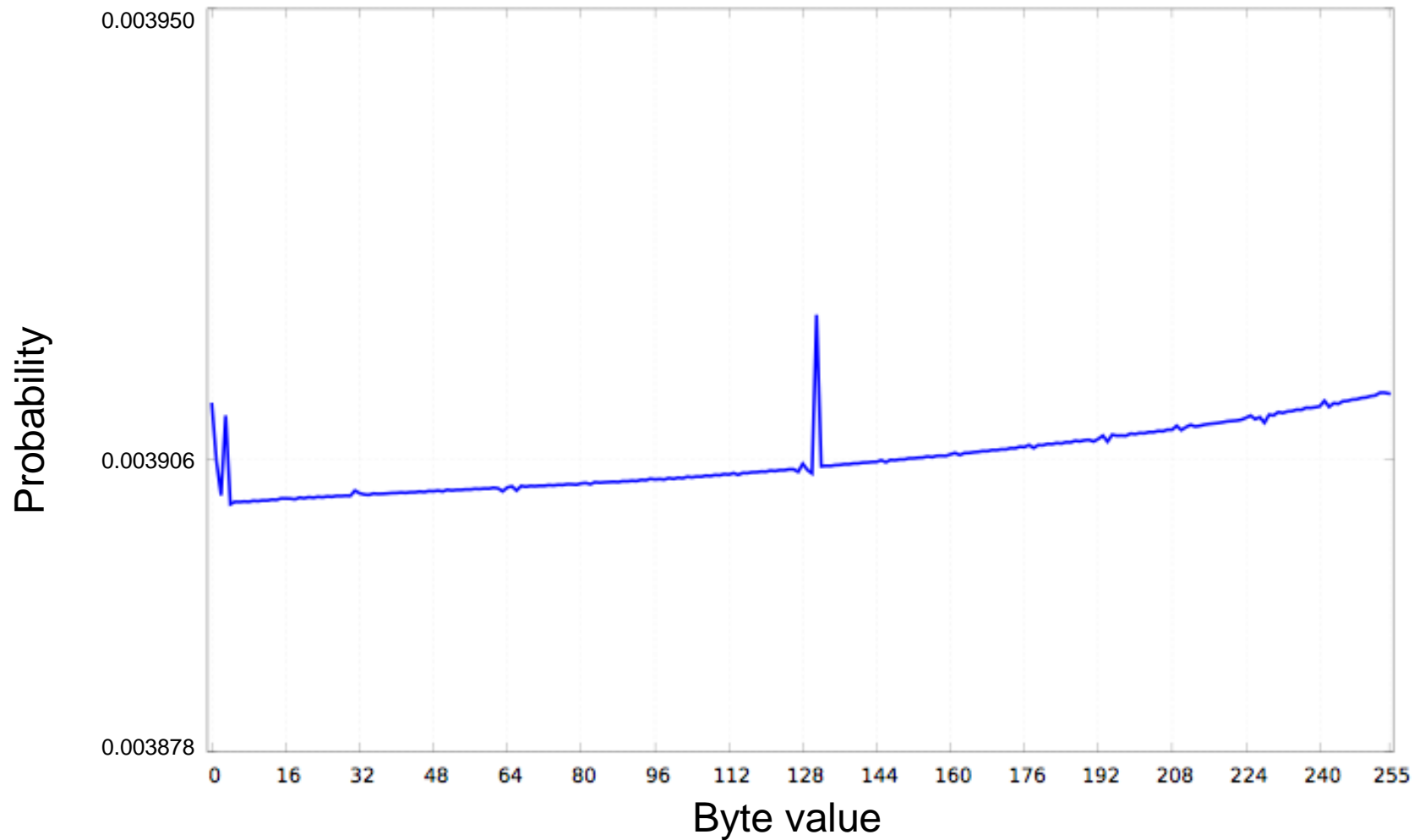
Keystream Distribution at Position 1



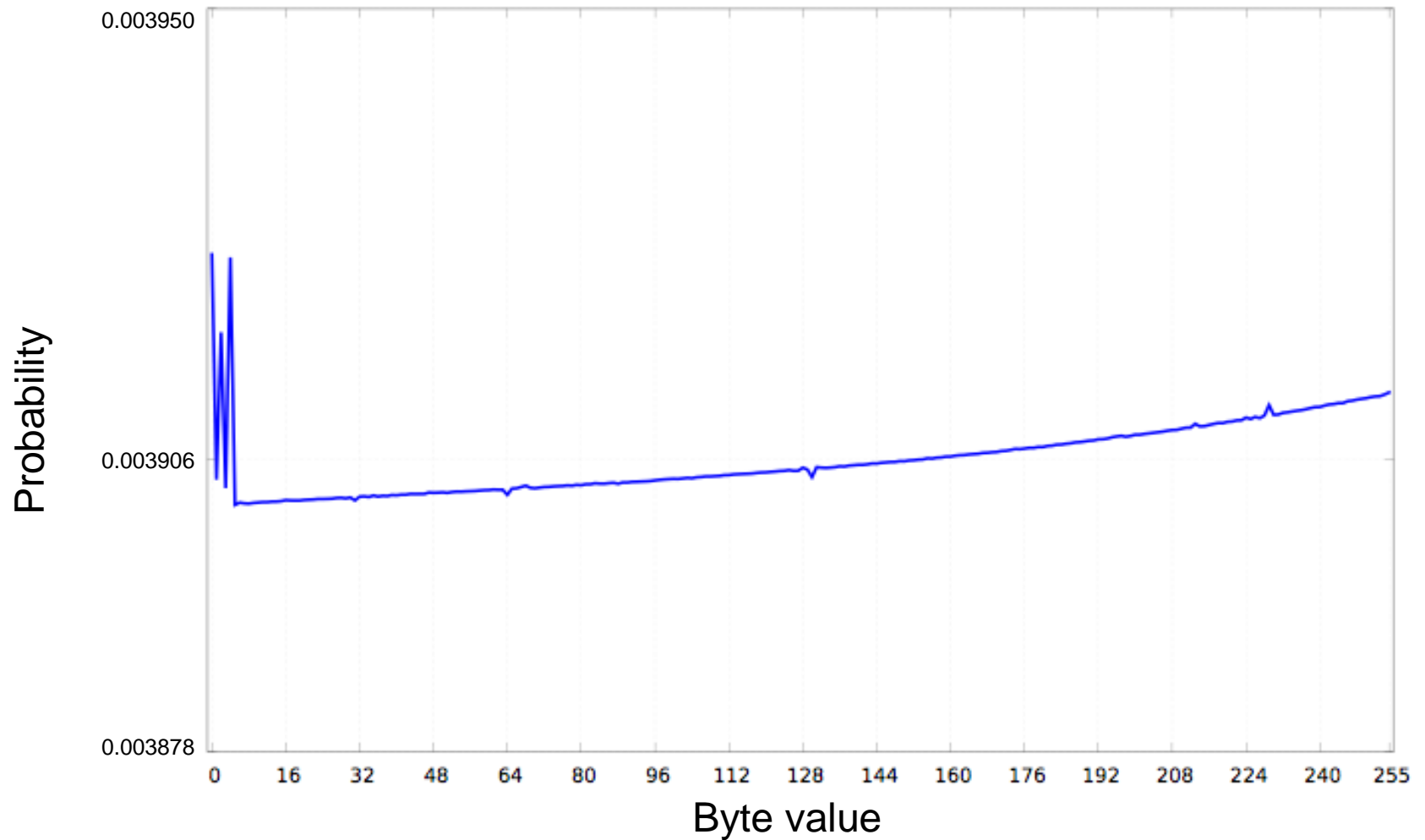
Keystream Distribution at Position 2



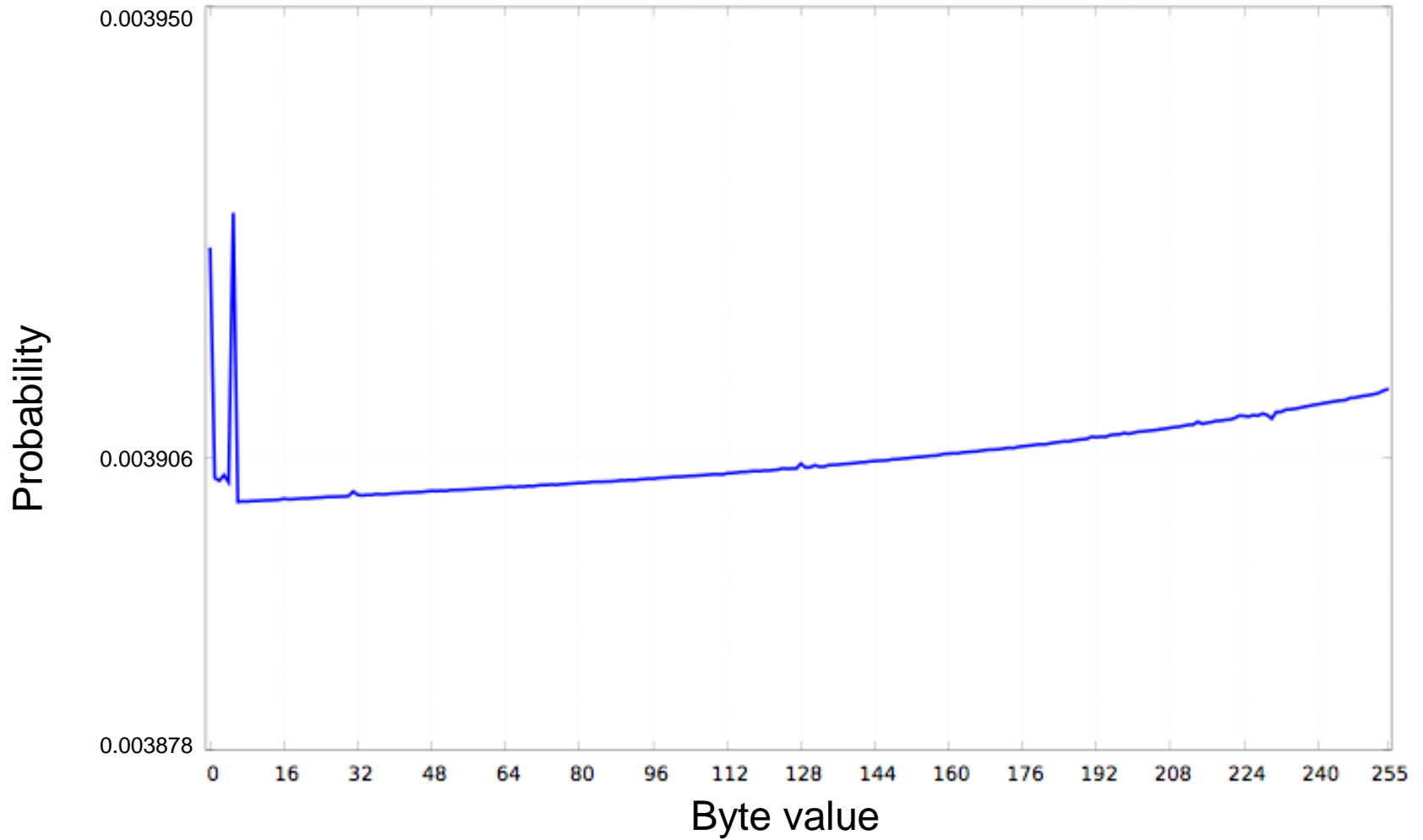
Keystream Distribution at Position 3



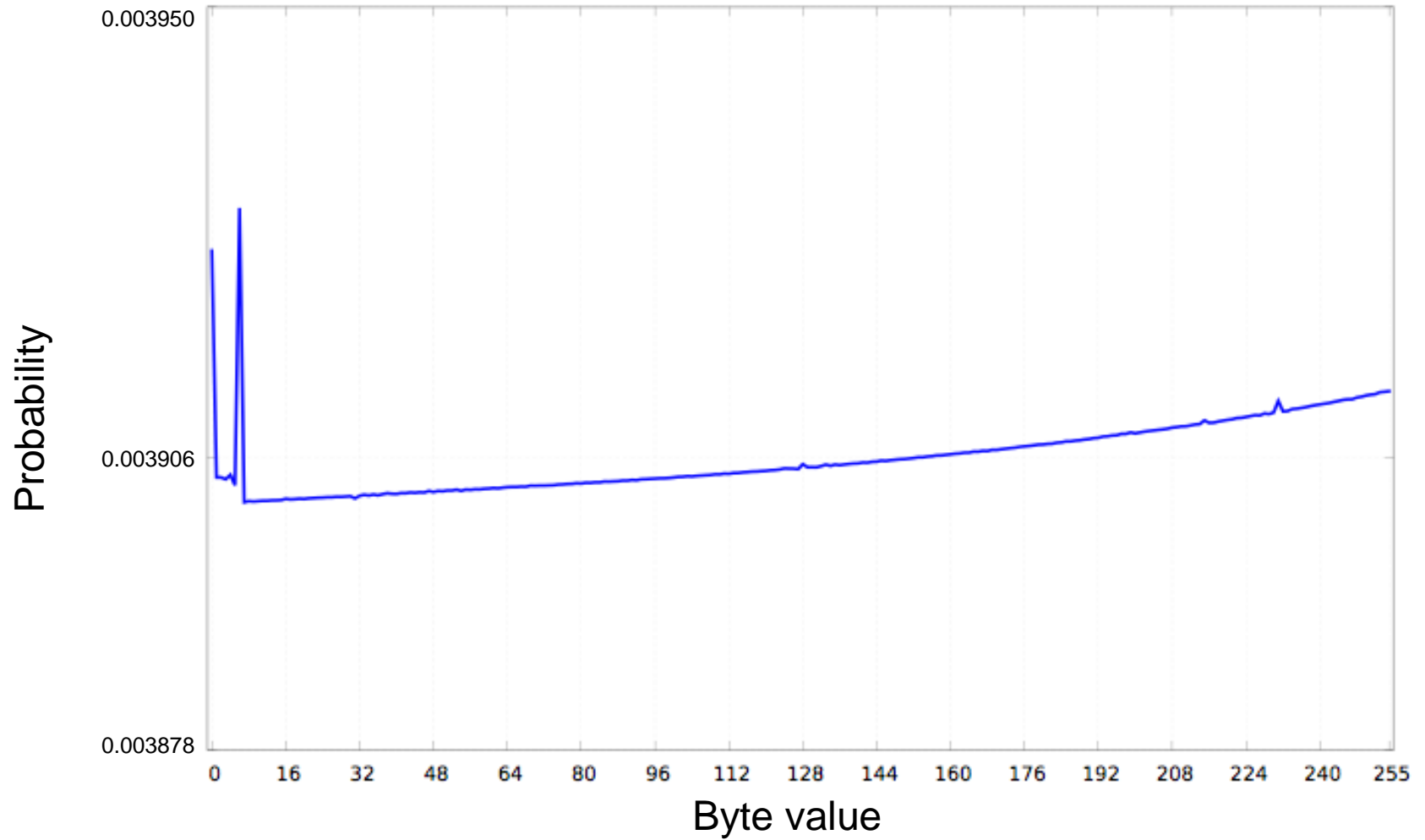
Keystream Distribution at Position 4



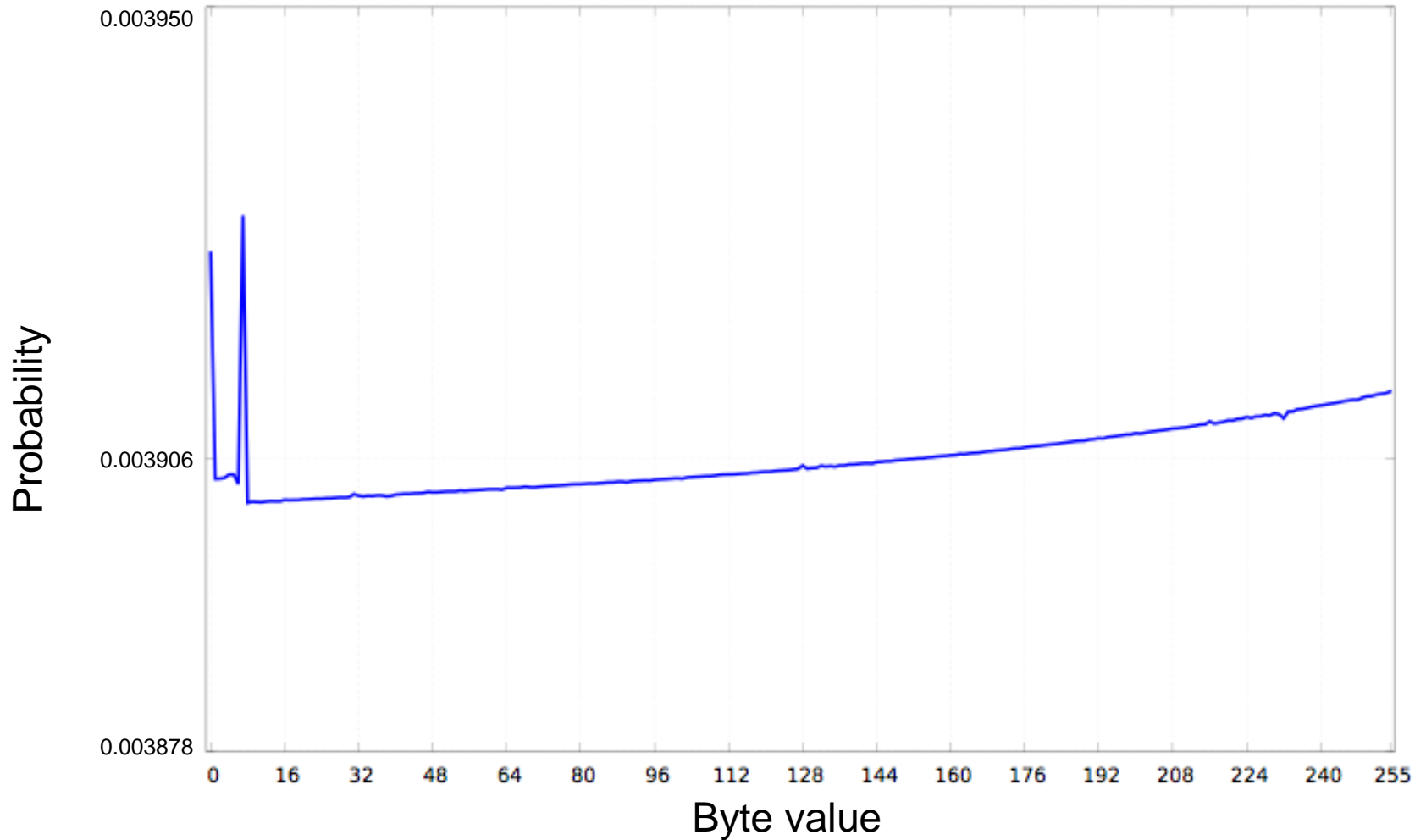
Keystream Distribution at Position 5



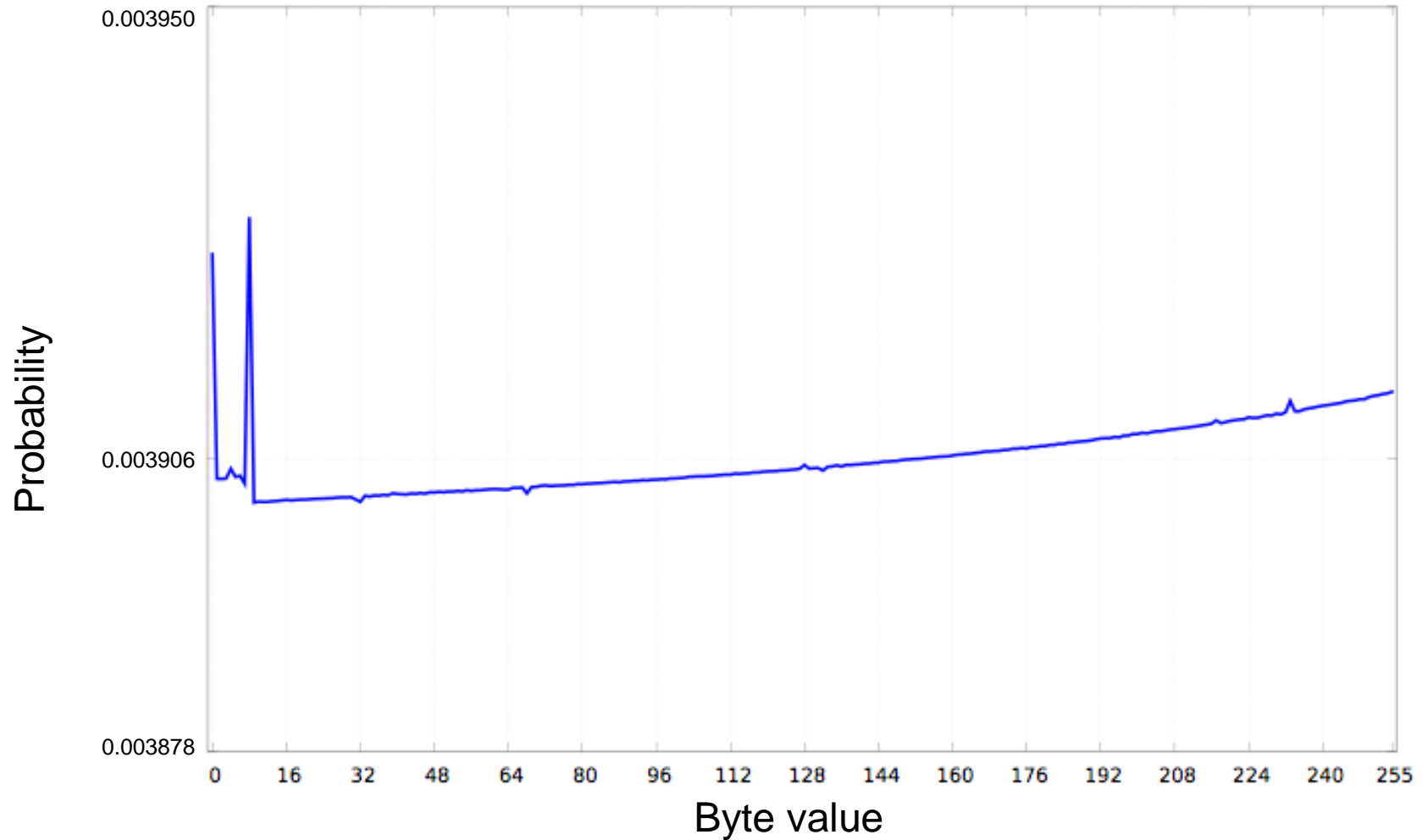
Keystream Distribution at Position 6



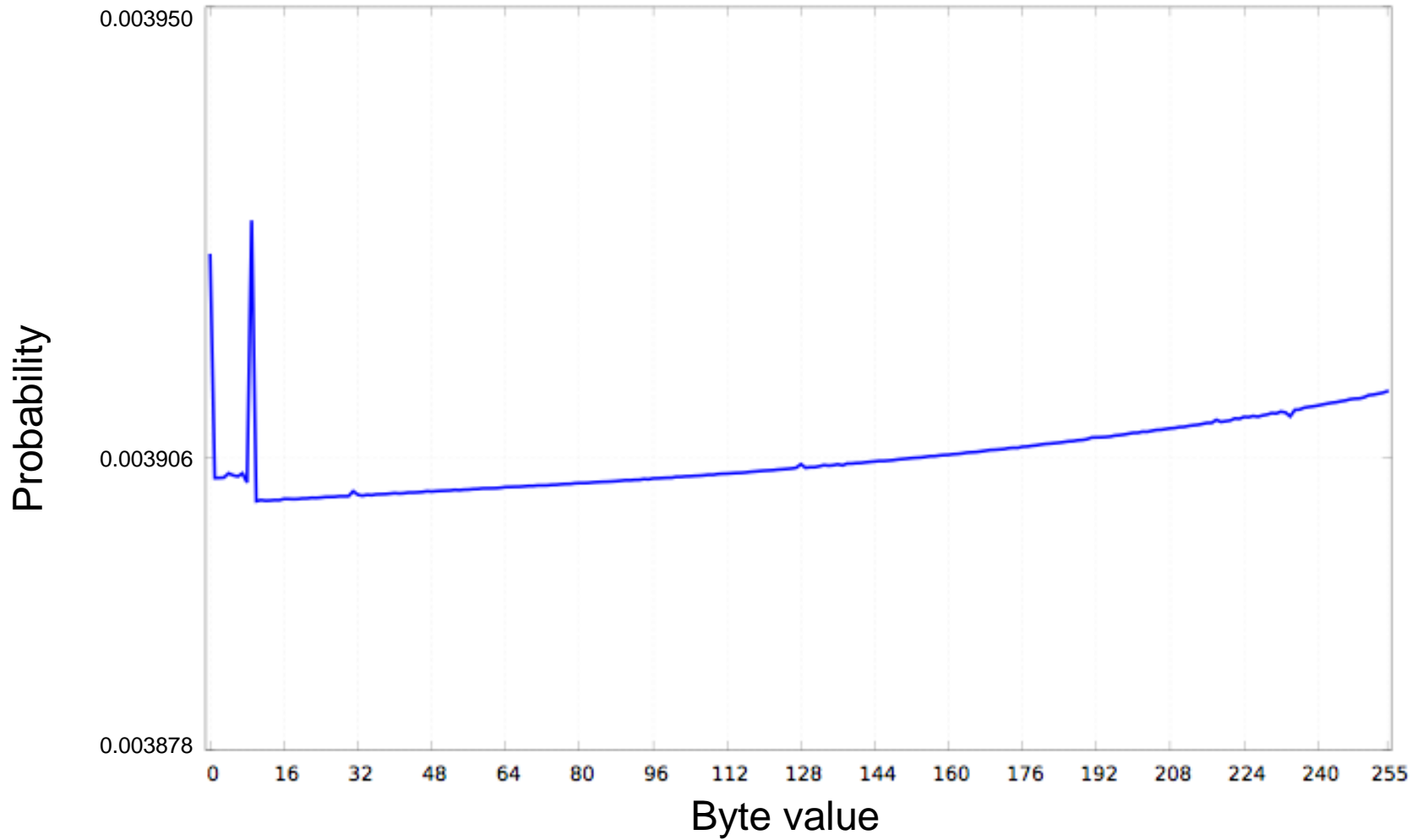
Keystream Distribution at Position 7



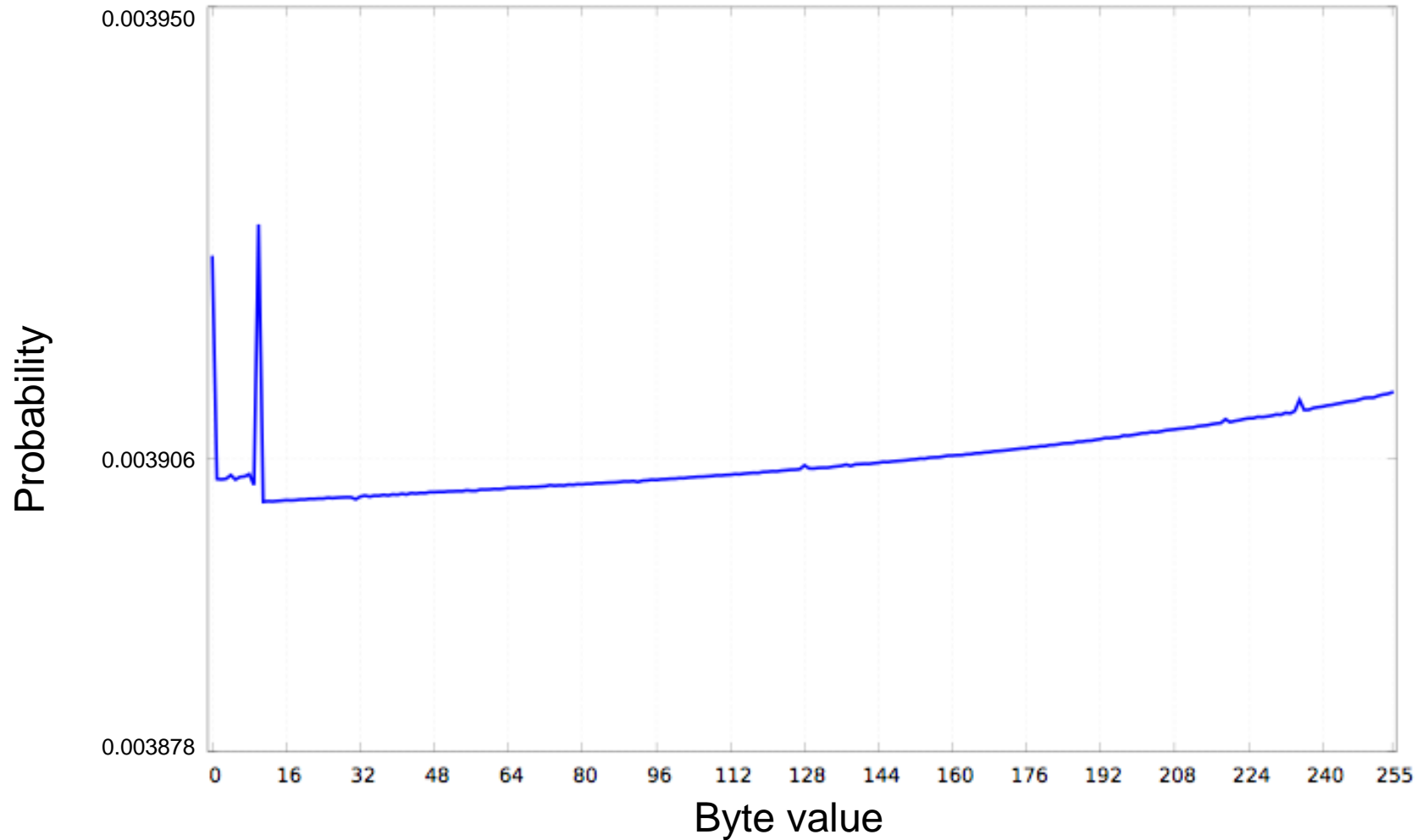
Keystream Distribution at Position 8



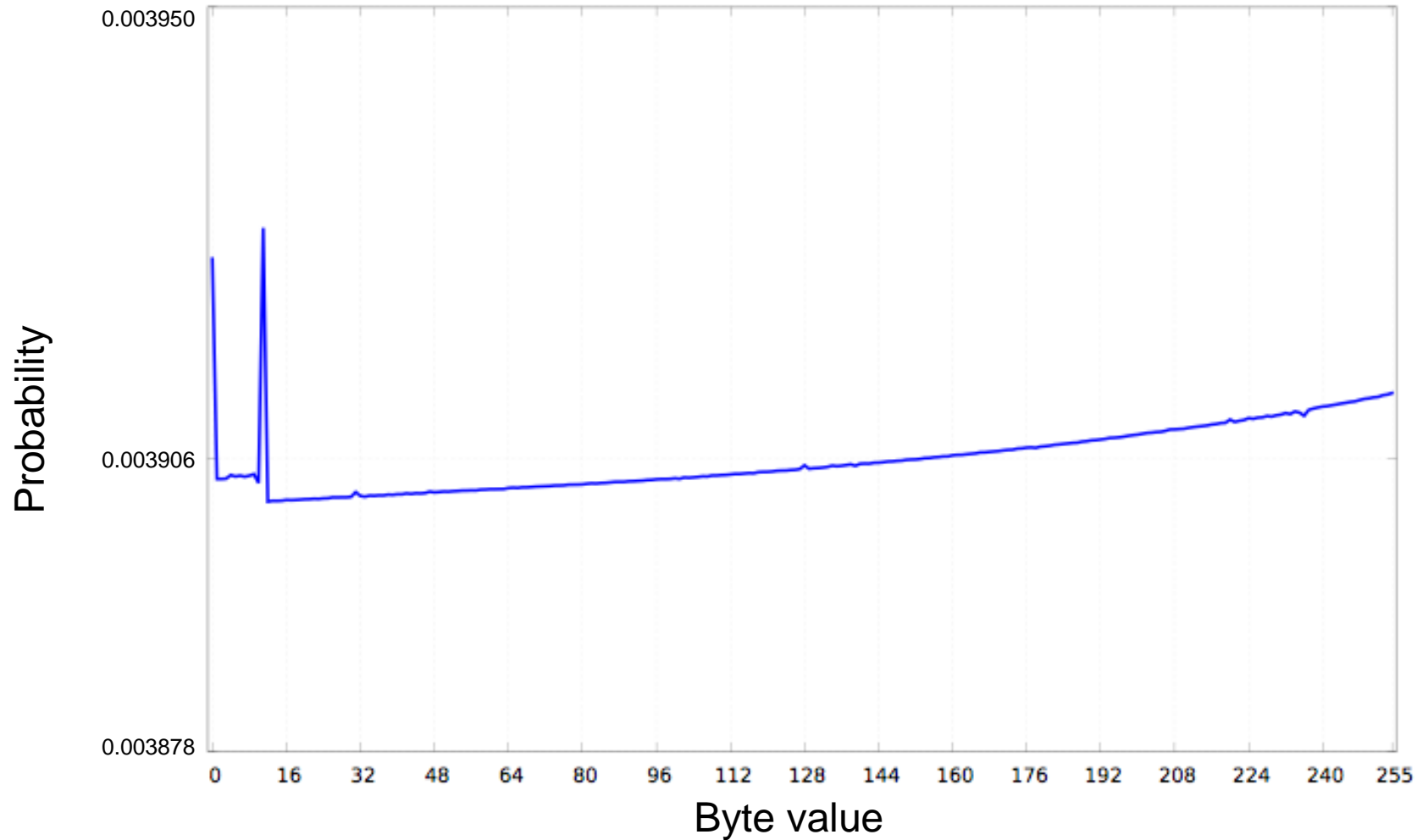
Keystream Distribution at Position 9



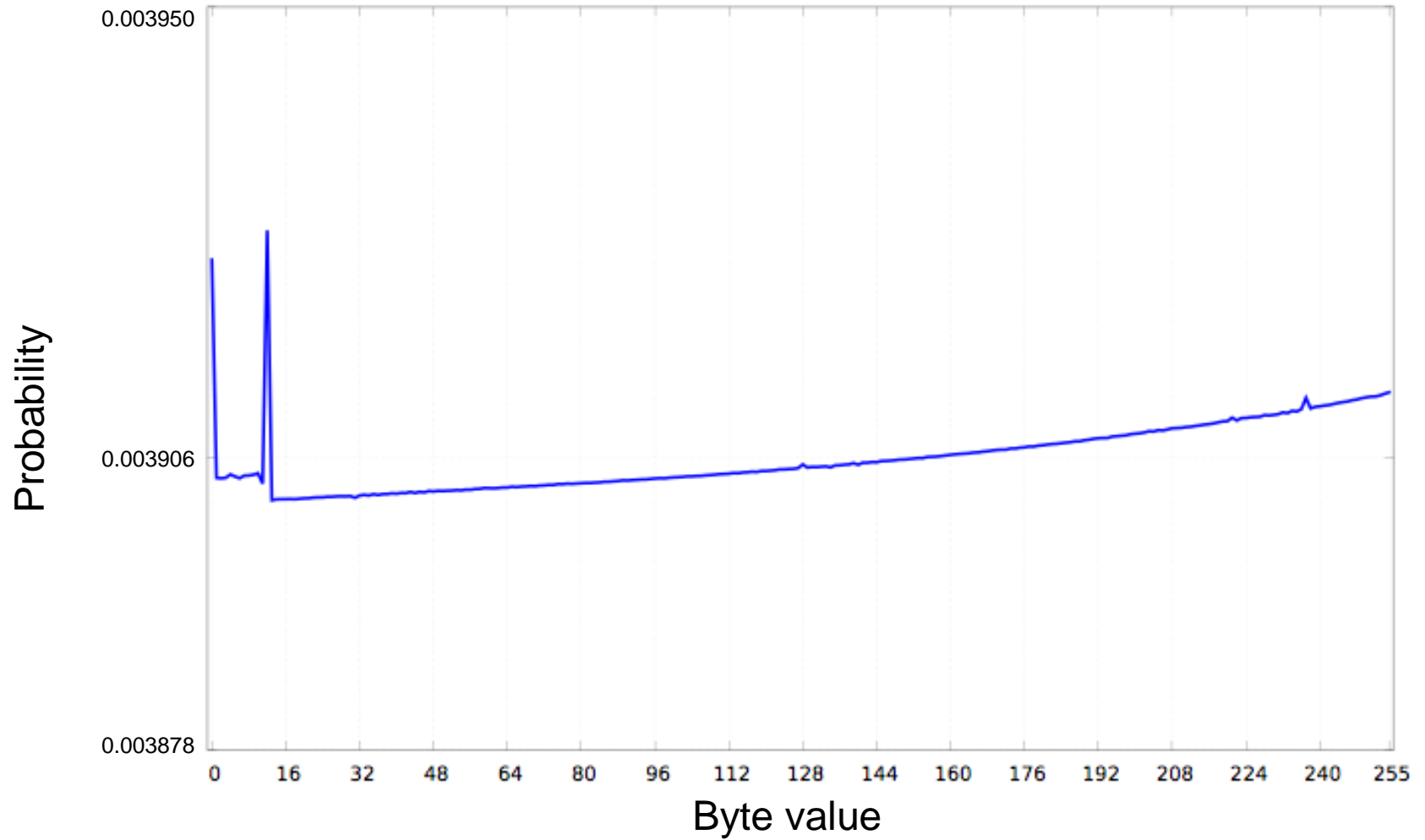
Keystream Distribution at Position 10



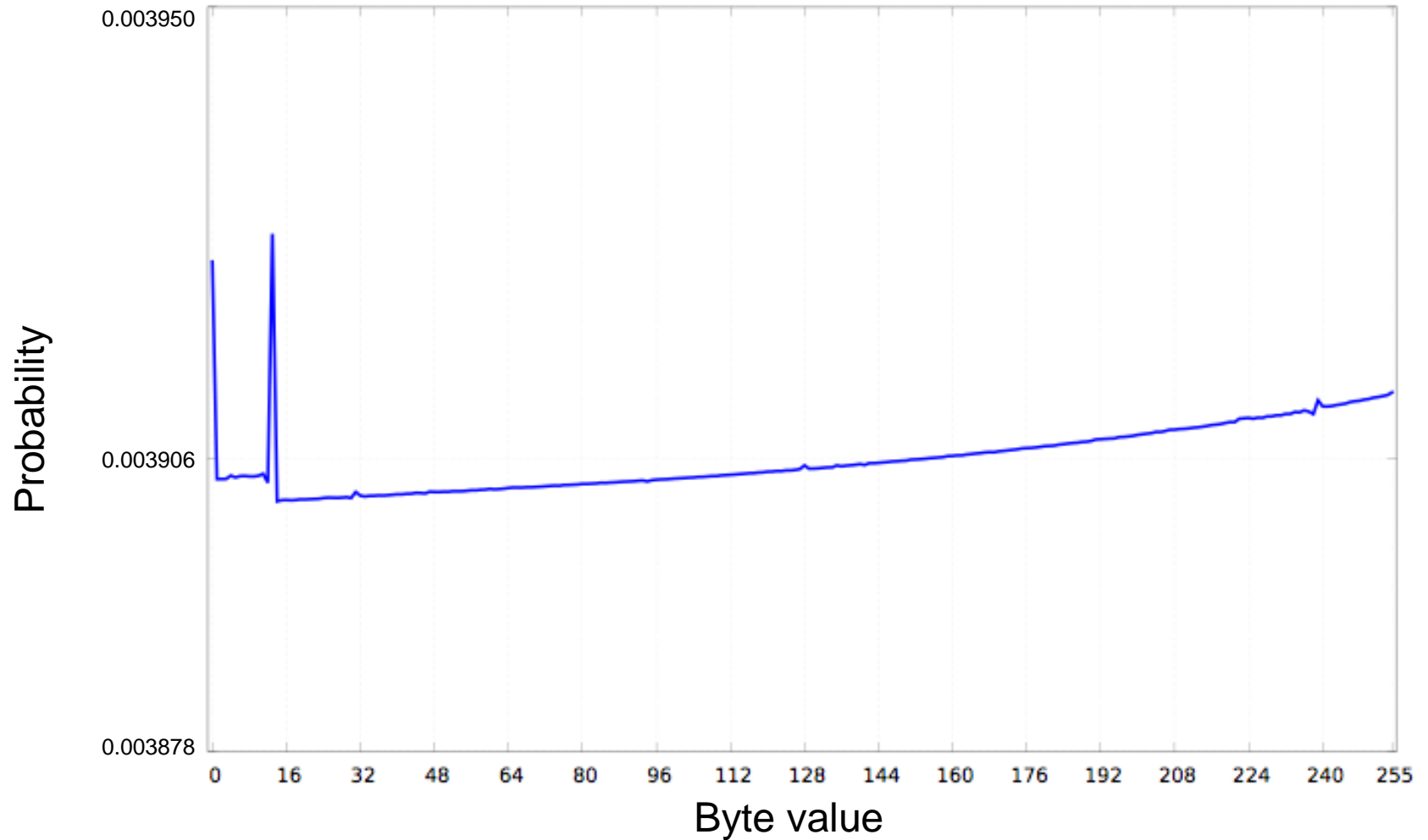
Keystream Distribution at Position 11



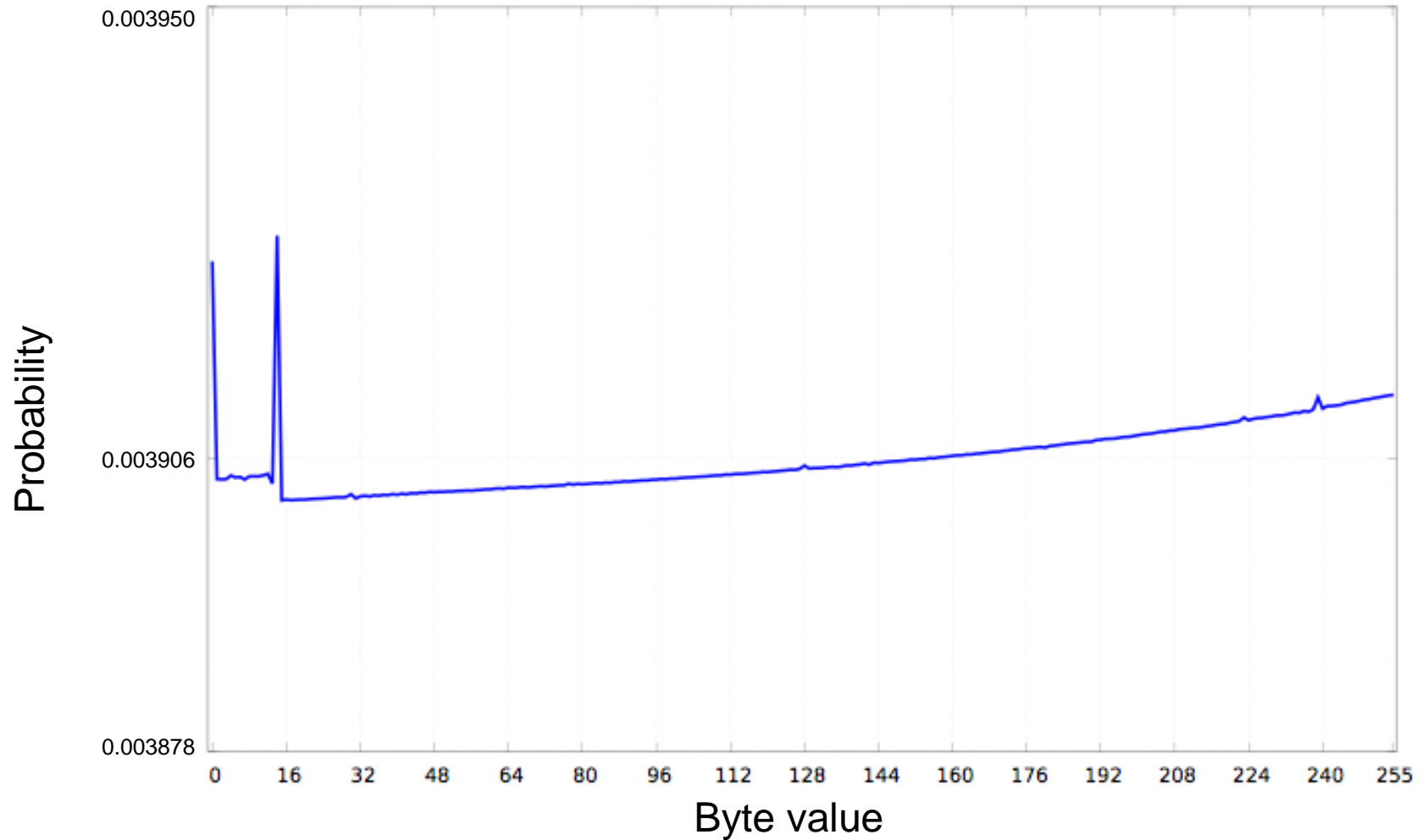
Keystream Distribution at Position 12



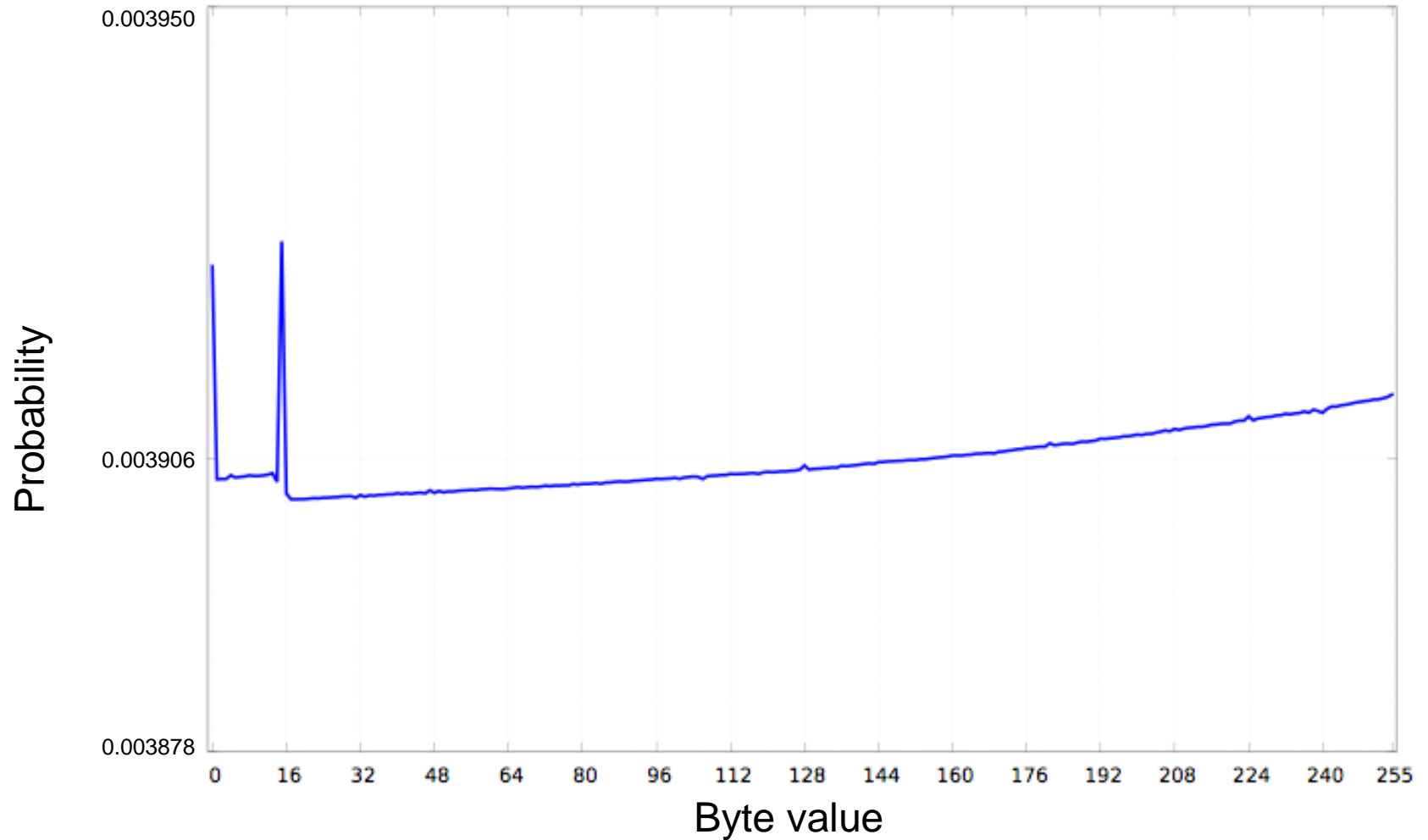
Keystream Distribution at Position 13



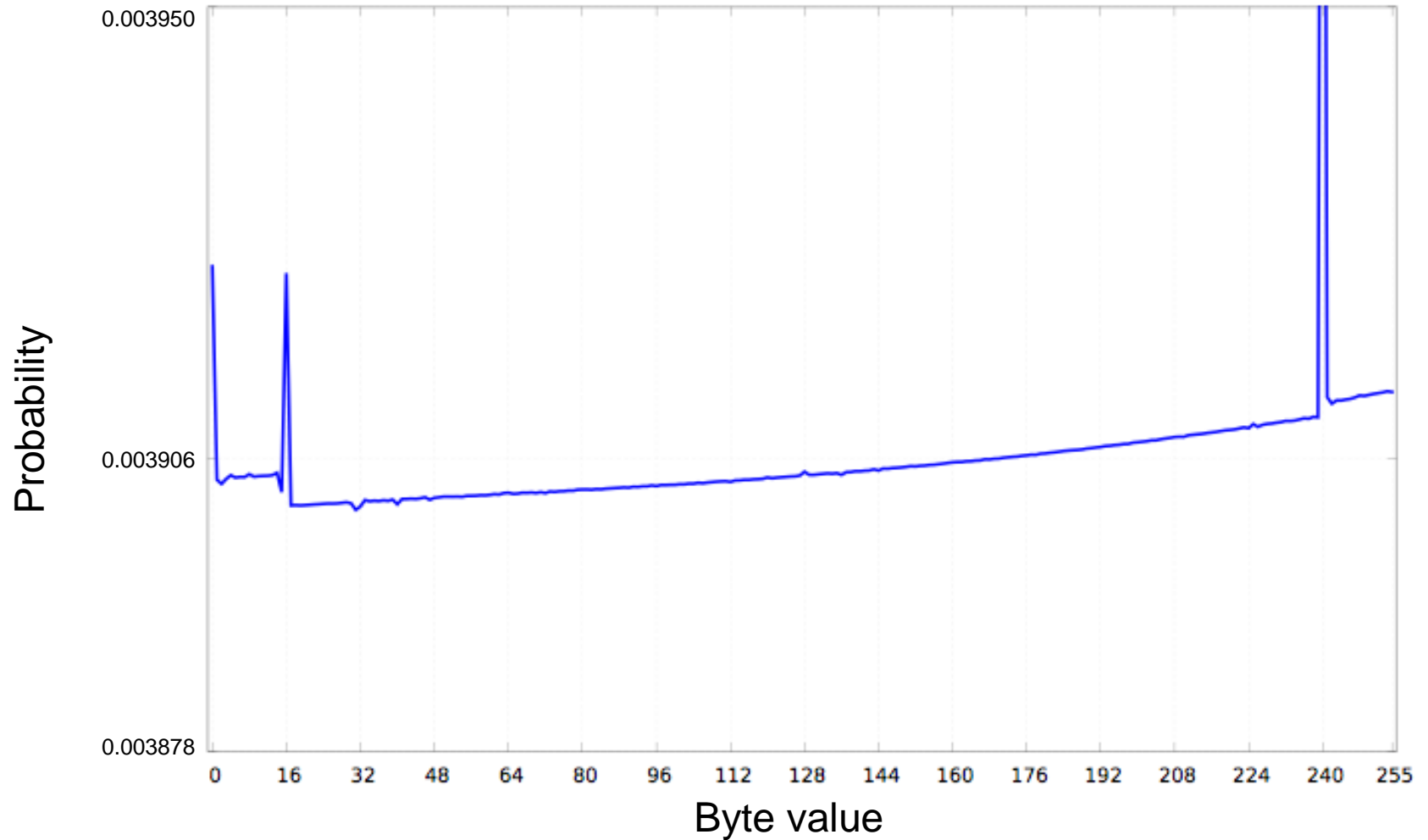
Keystream Distribution at Position 14



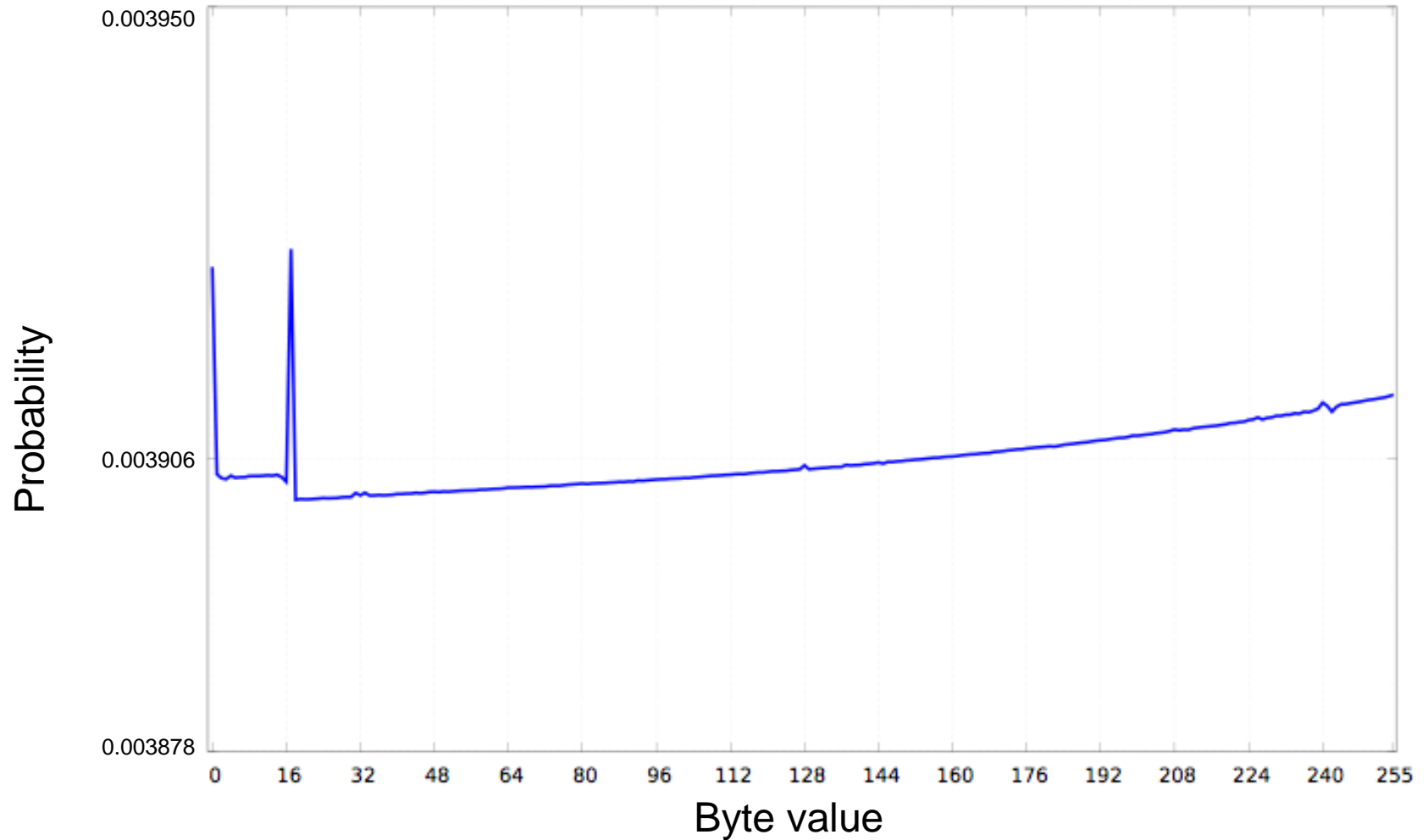
Keystream Distribution at Position 15



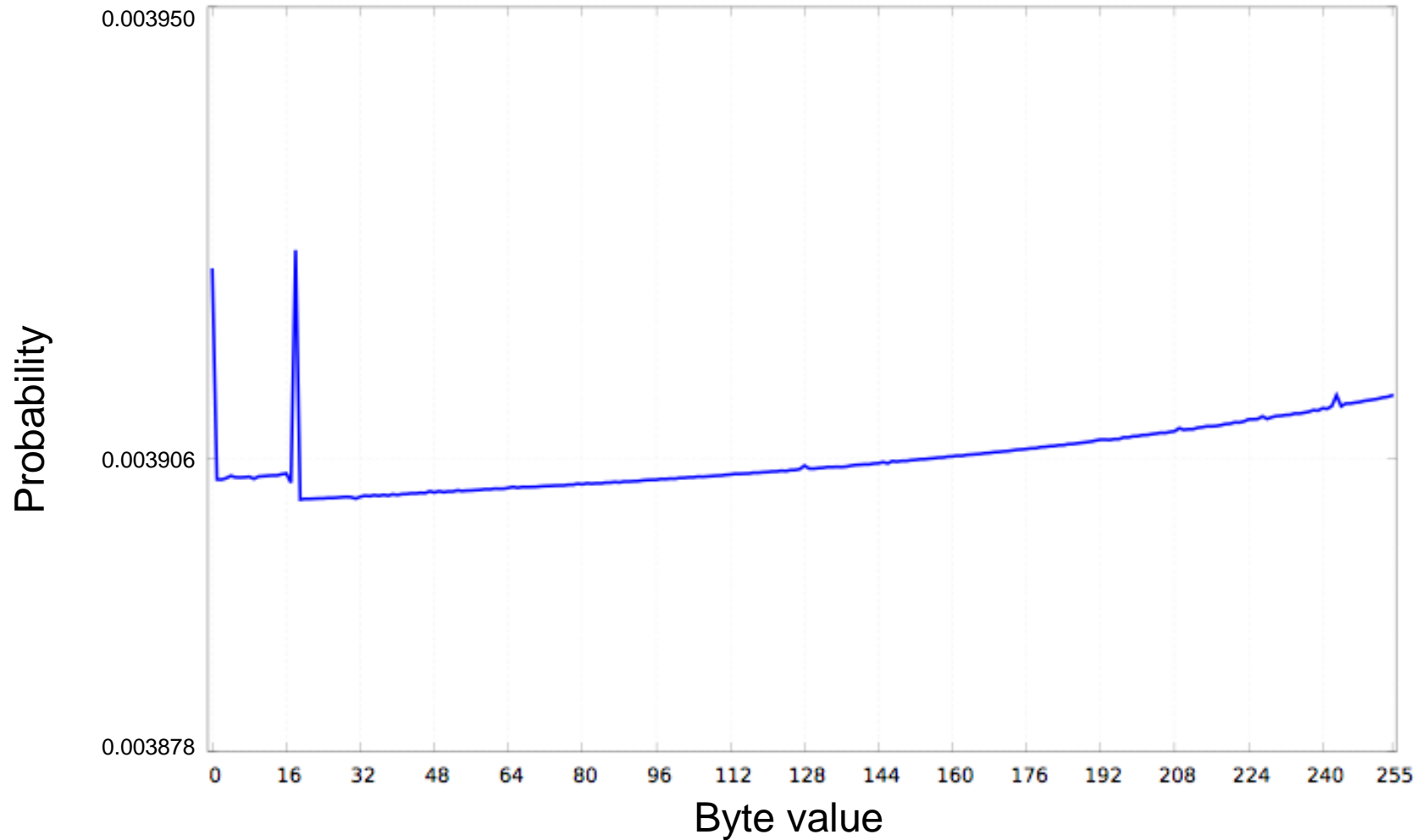
Keystream Distribution at Position 16



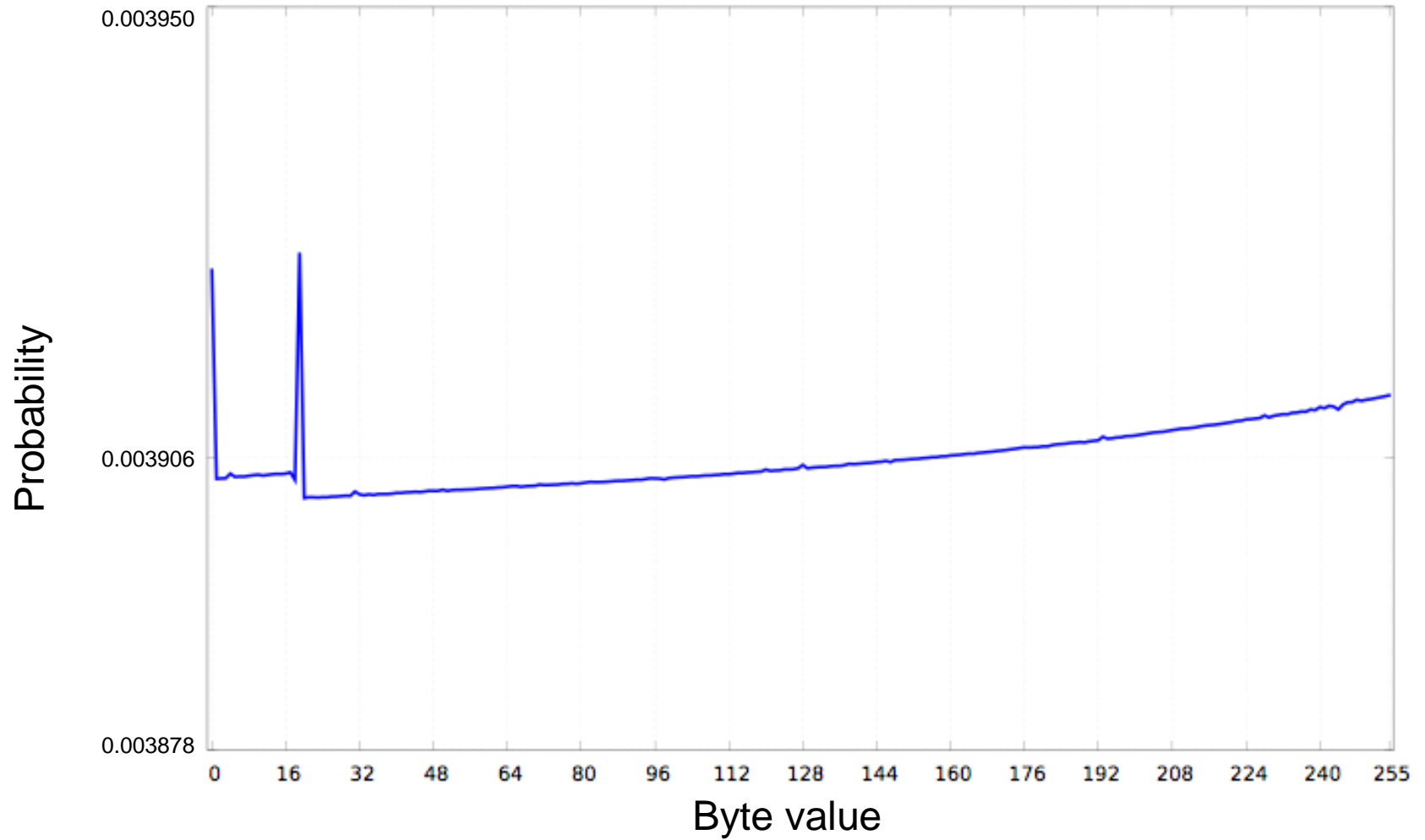
Keystream Distribution at Position 17



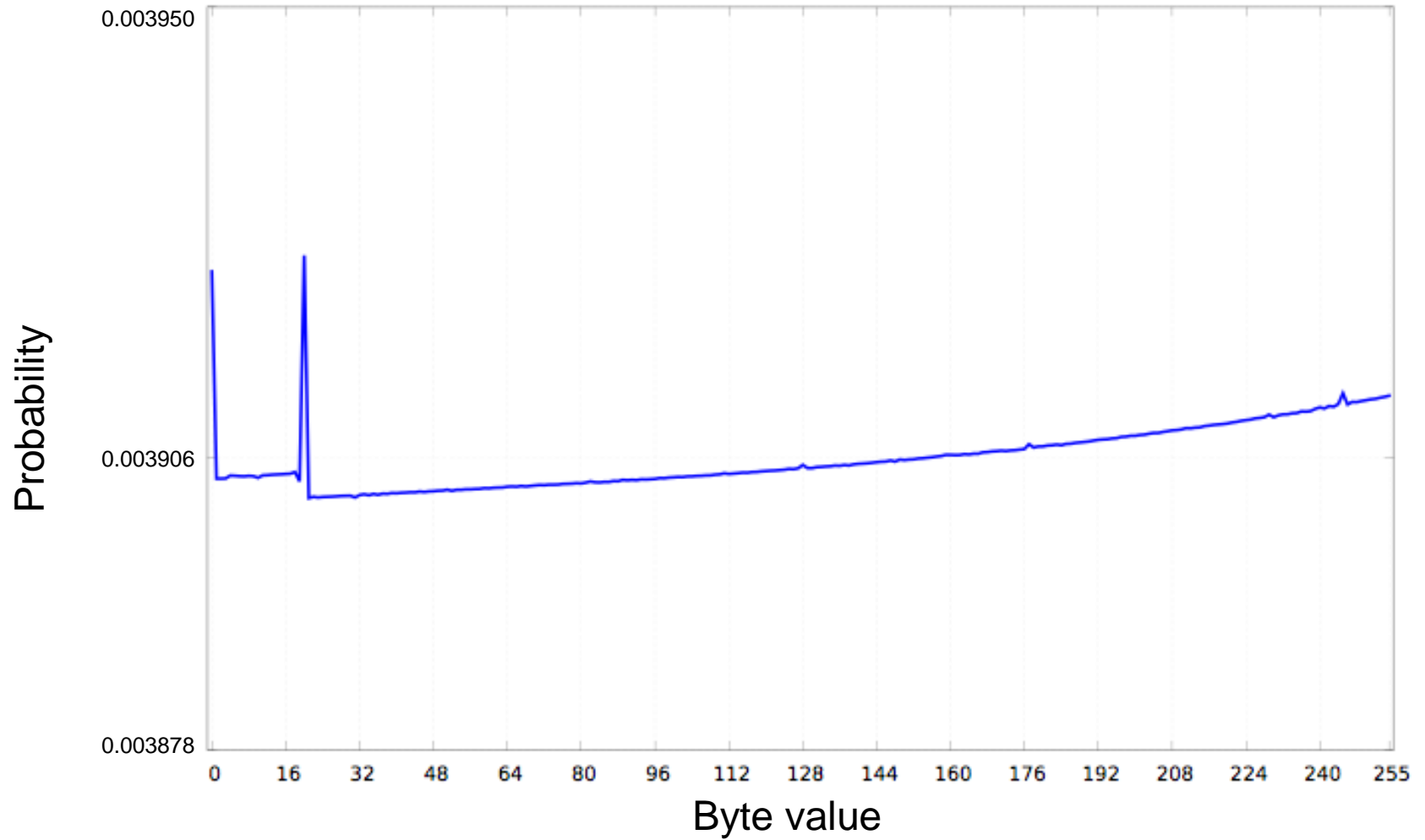
Keystream Distribution at Position 18



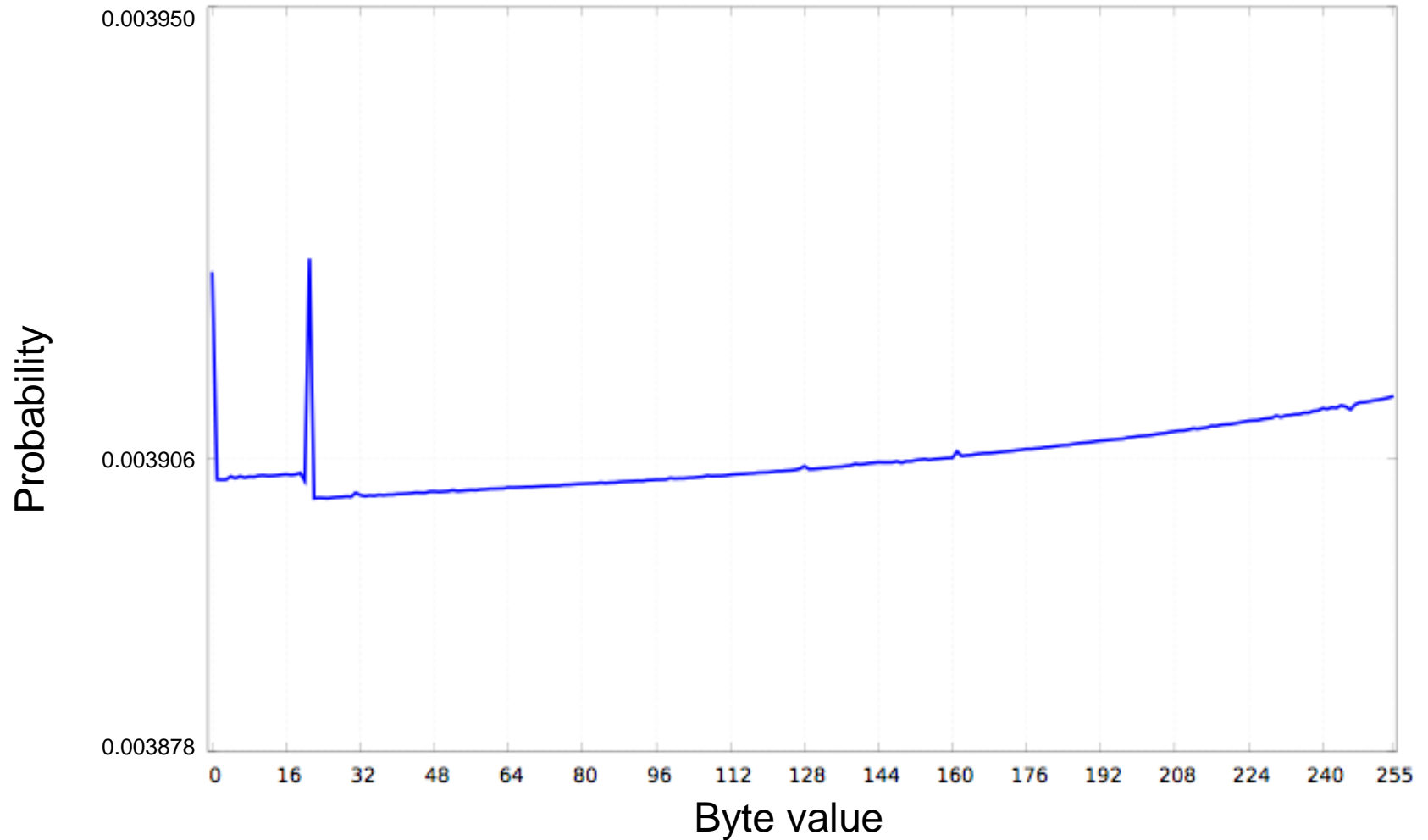
Keystream Distribution at Position 19



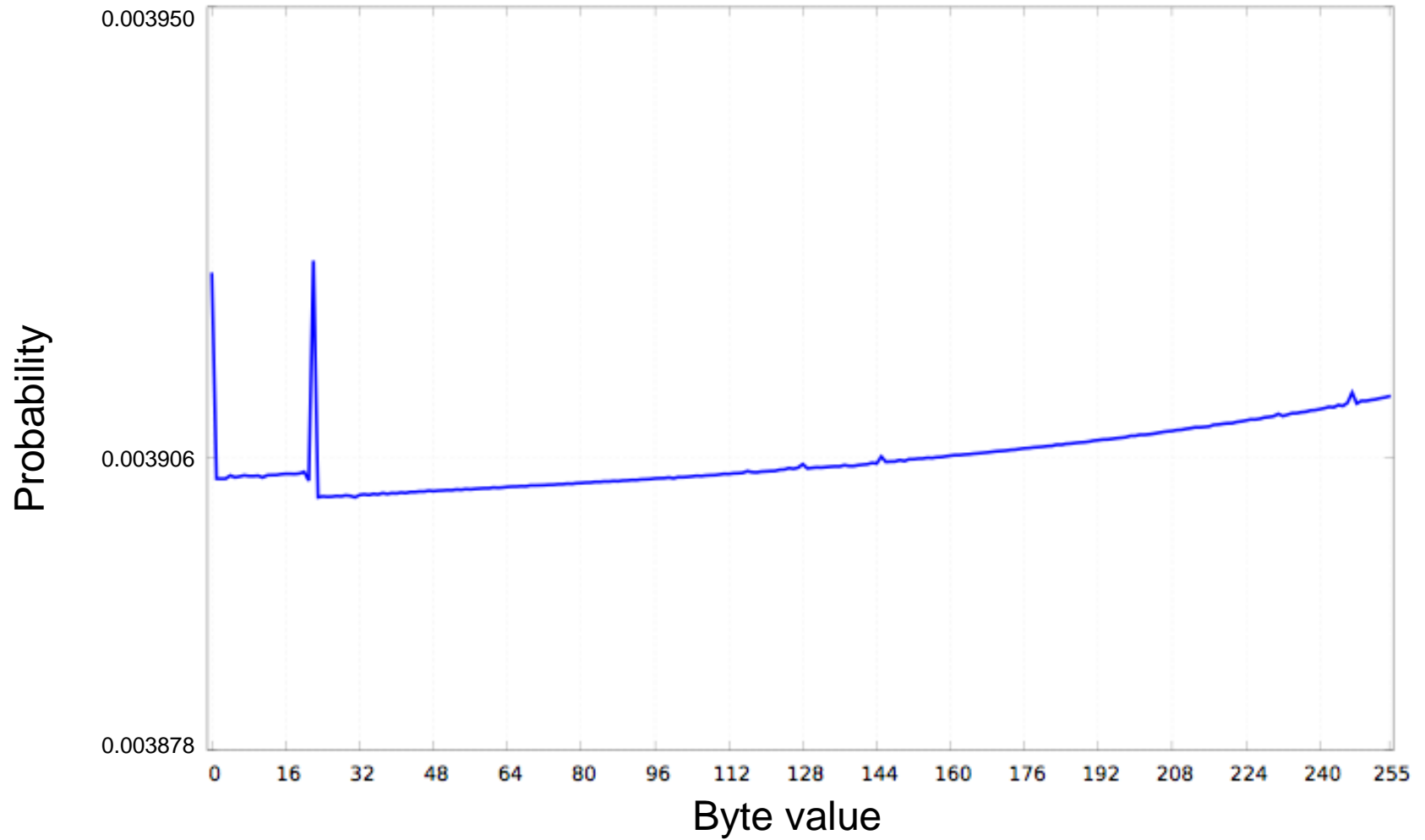
Keystream Distribution at Position 20



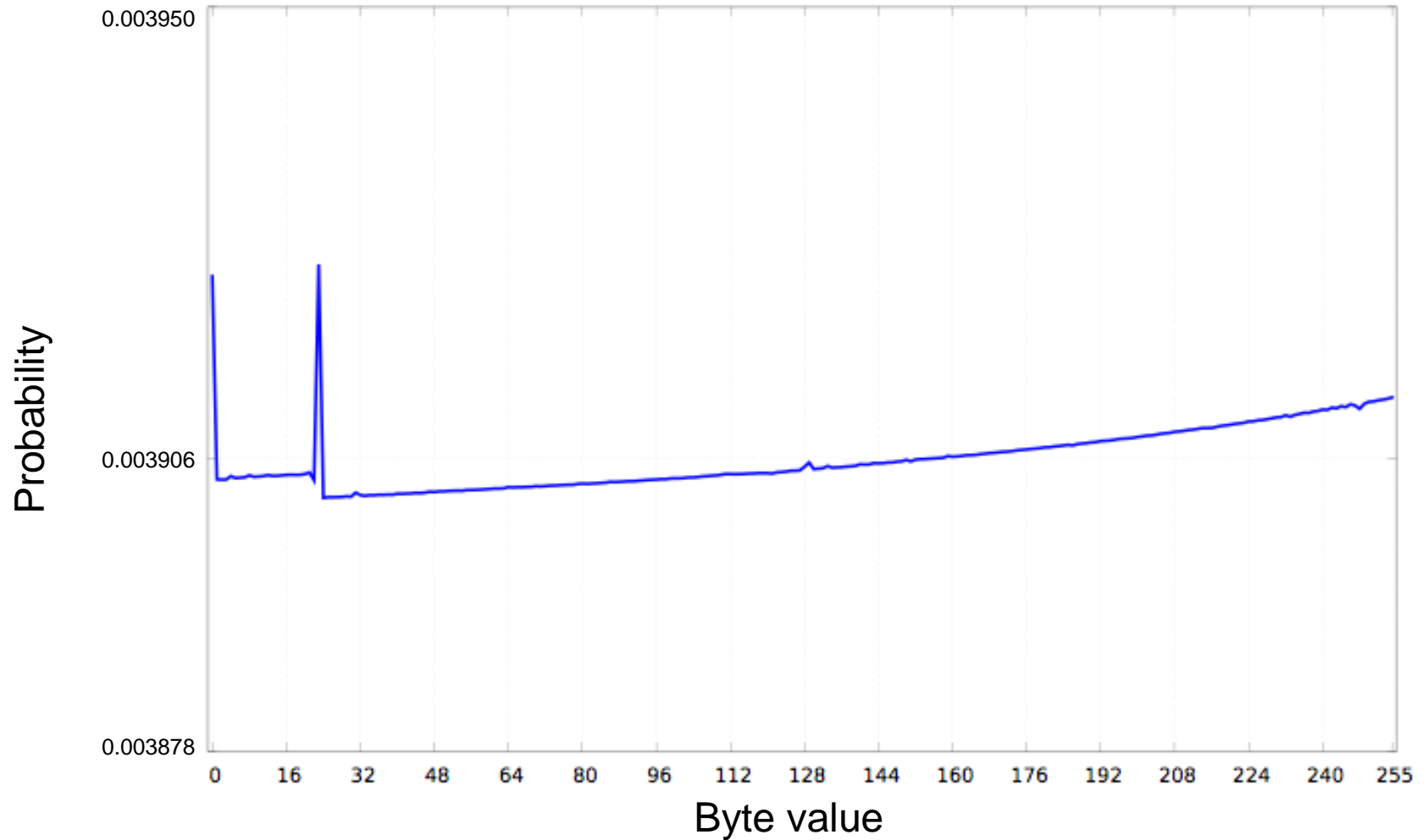
Keystream Distribution at Position 21



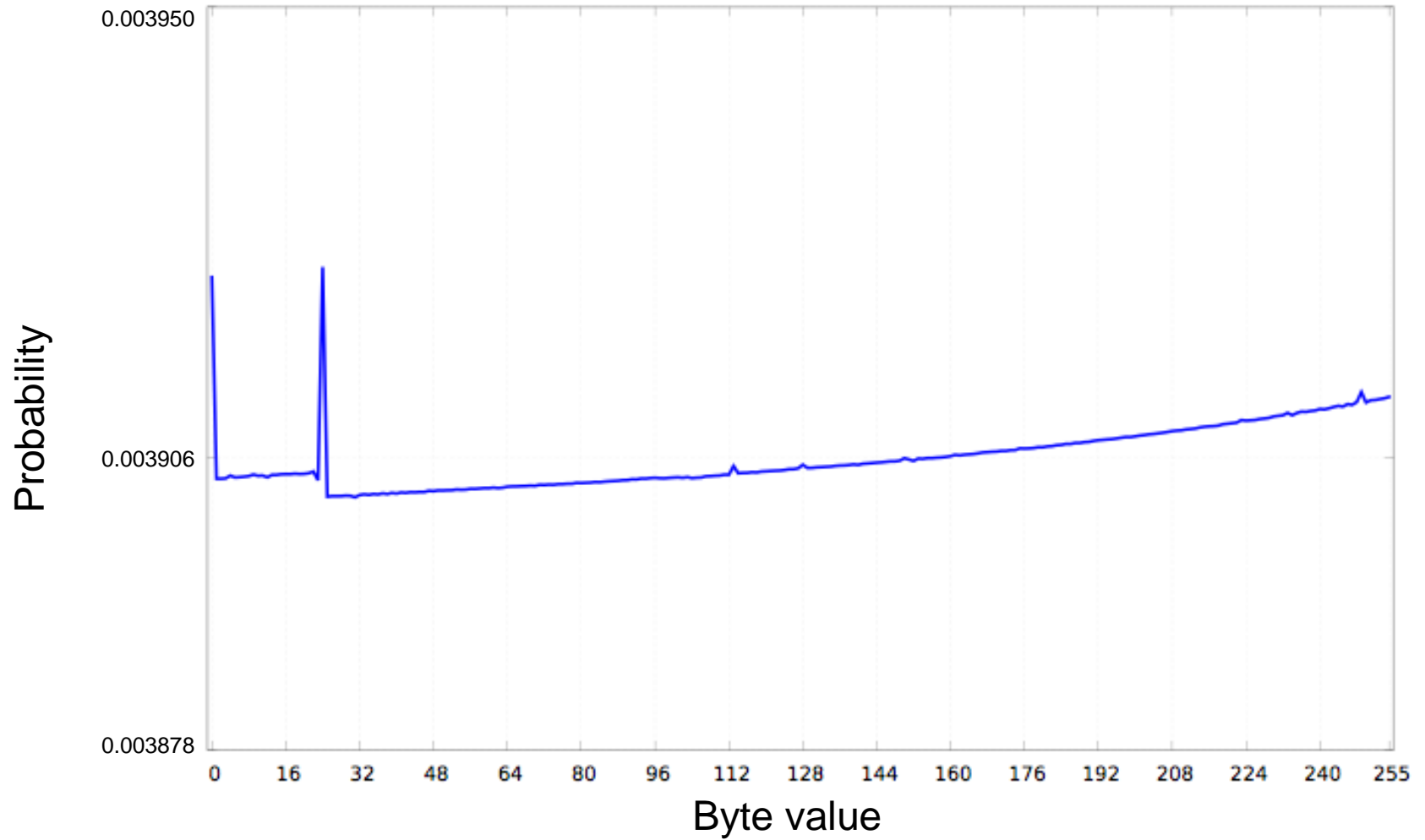
Keystream Distribution at Position 22



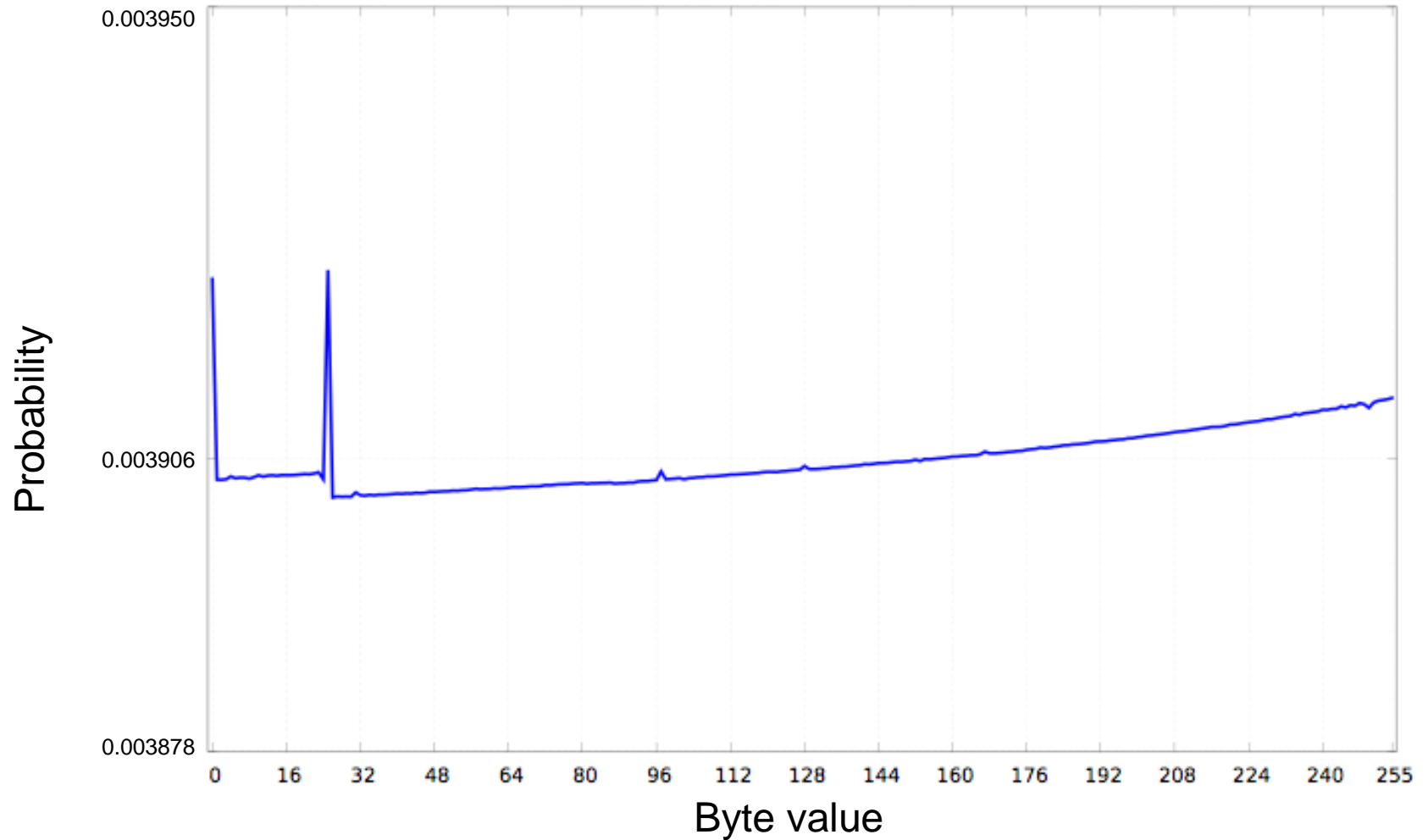
Keystream Distribution at Position 23



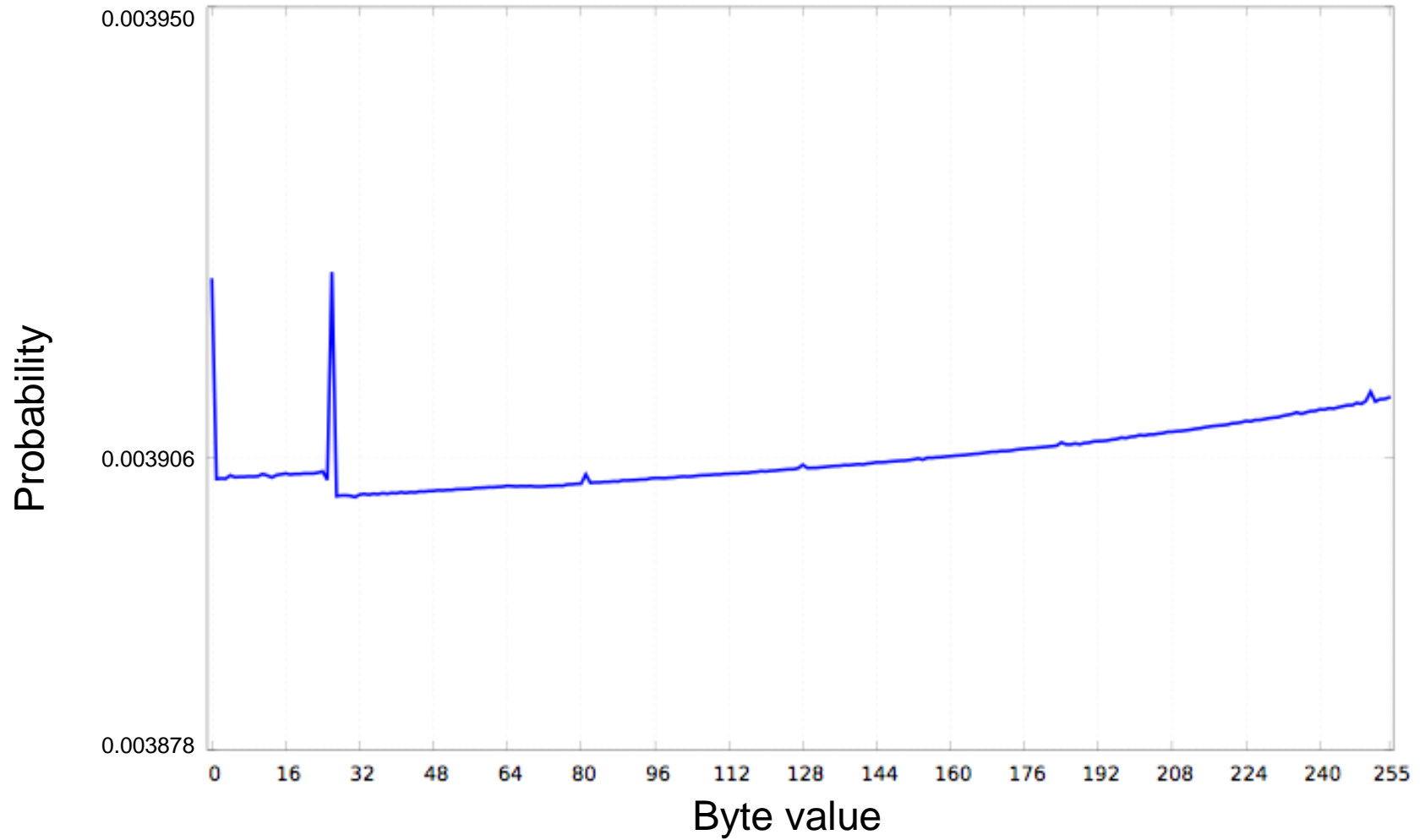
Keystream Distribution at Position 24



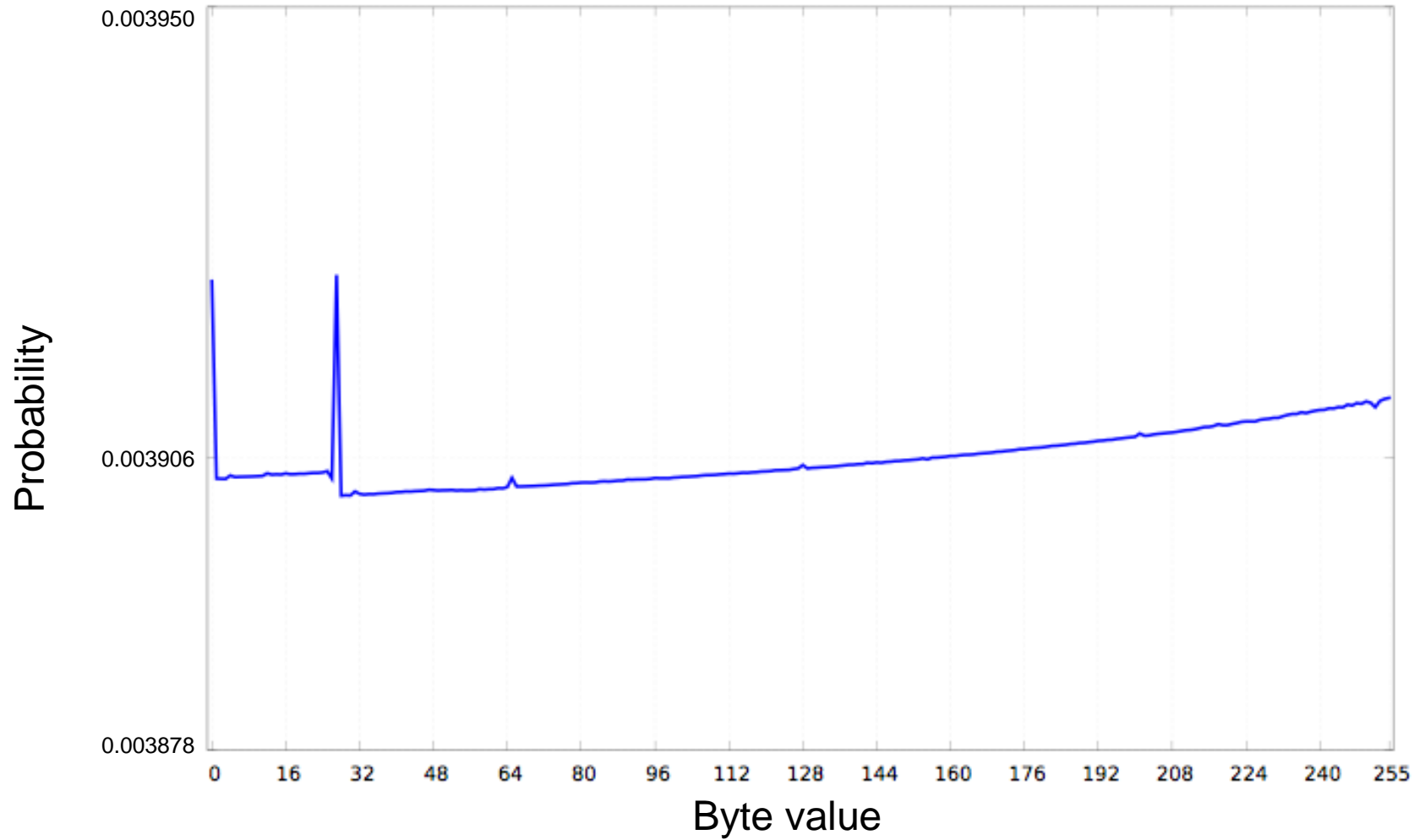
Keystream Distribution at Position 25



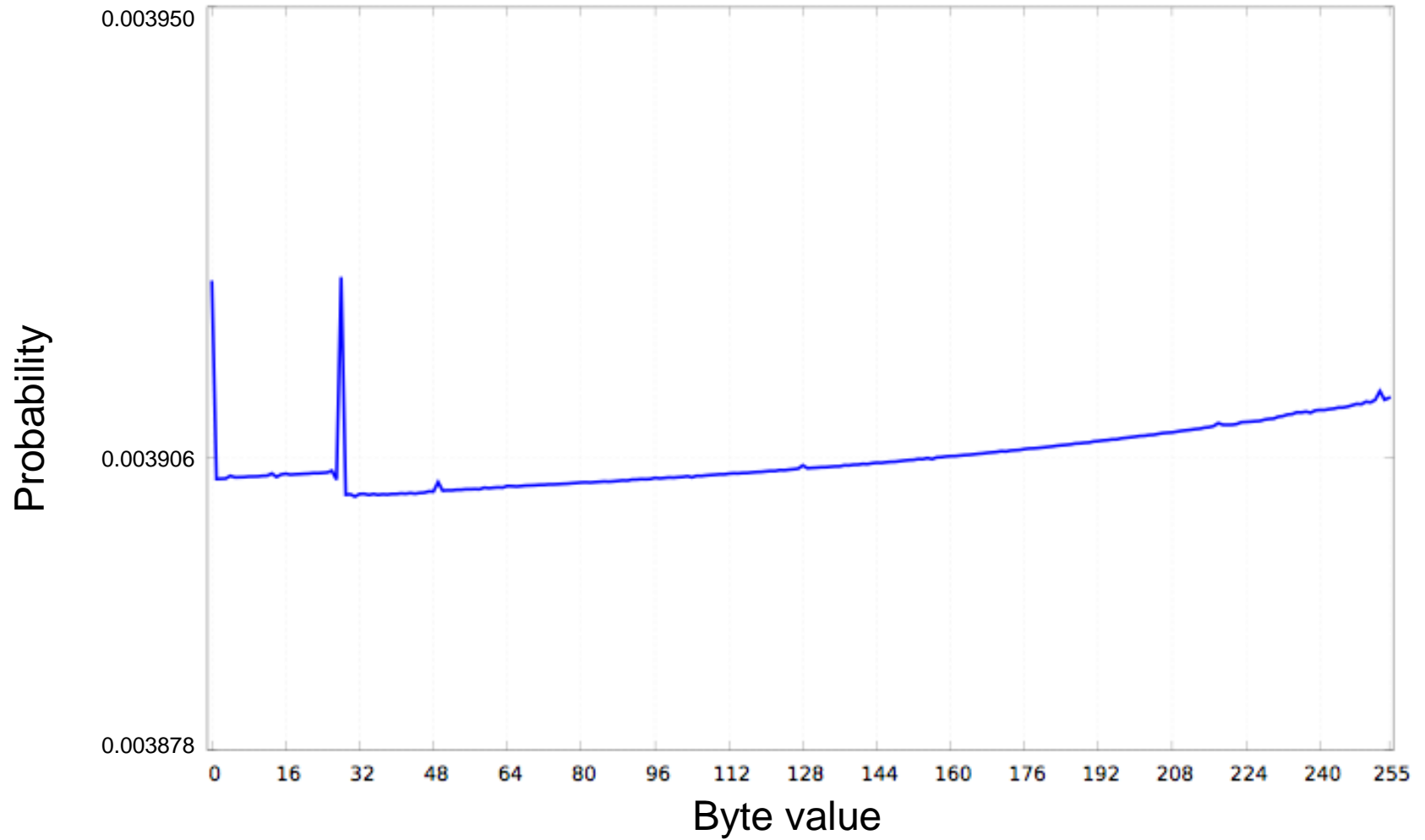
Keystream Distribution at Position 26



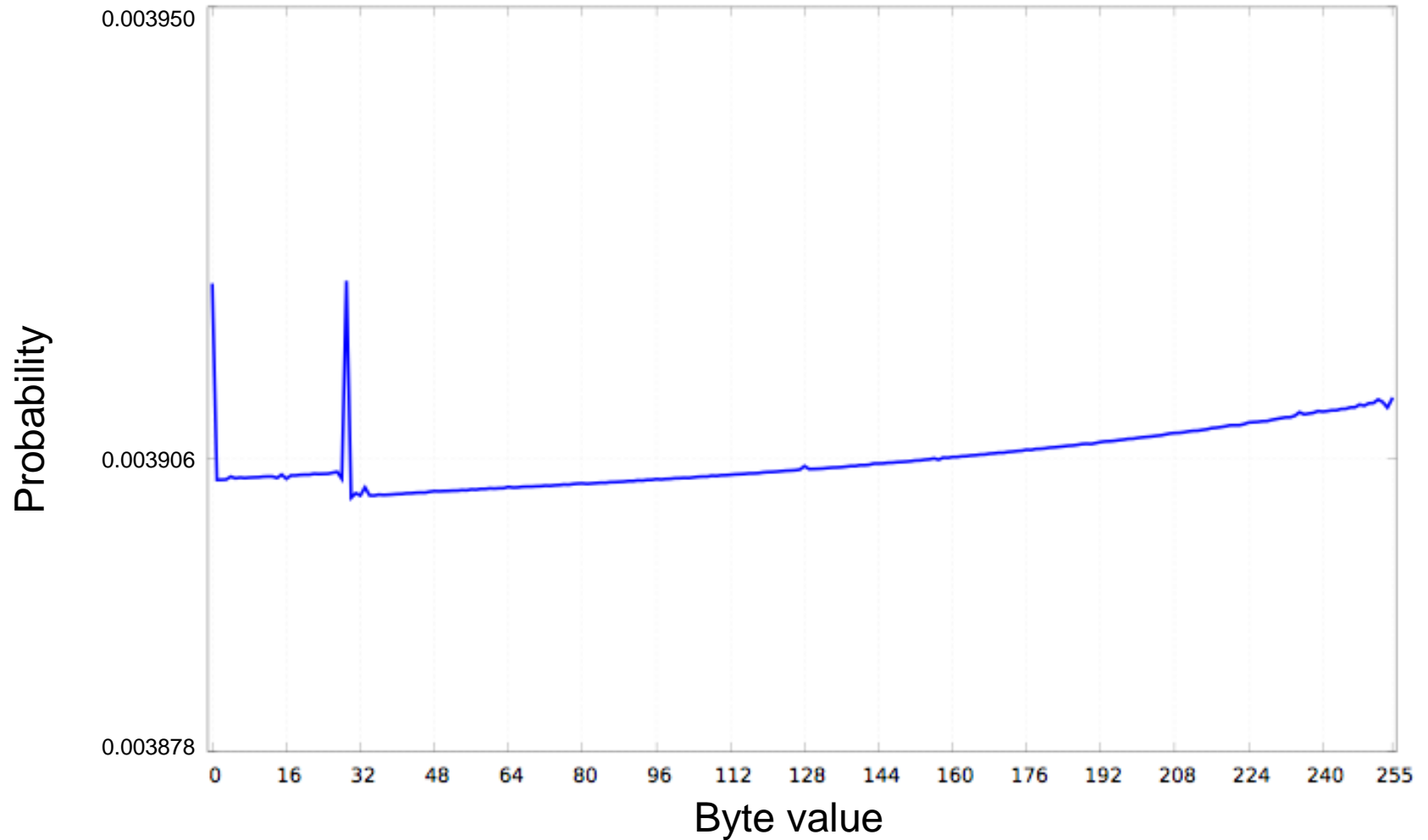
Keystream Distribution at Position 27



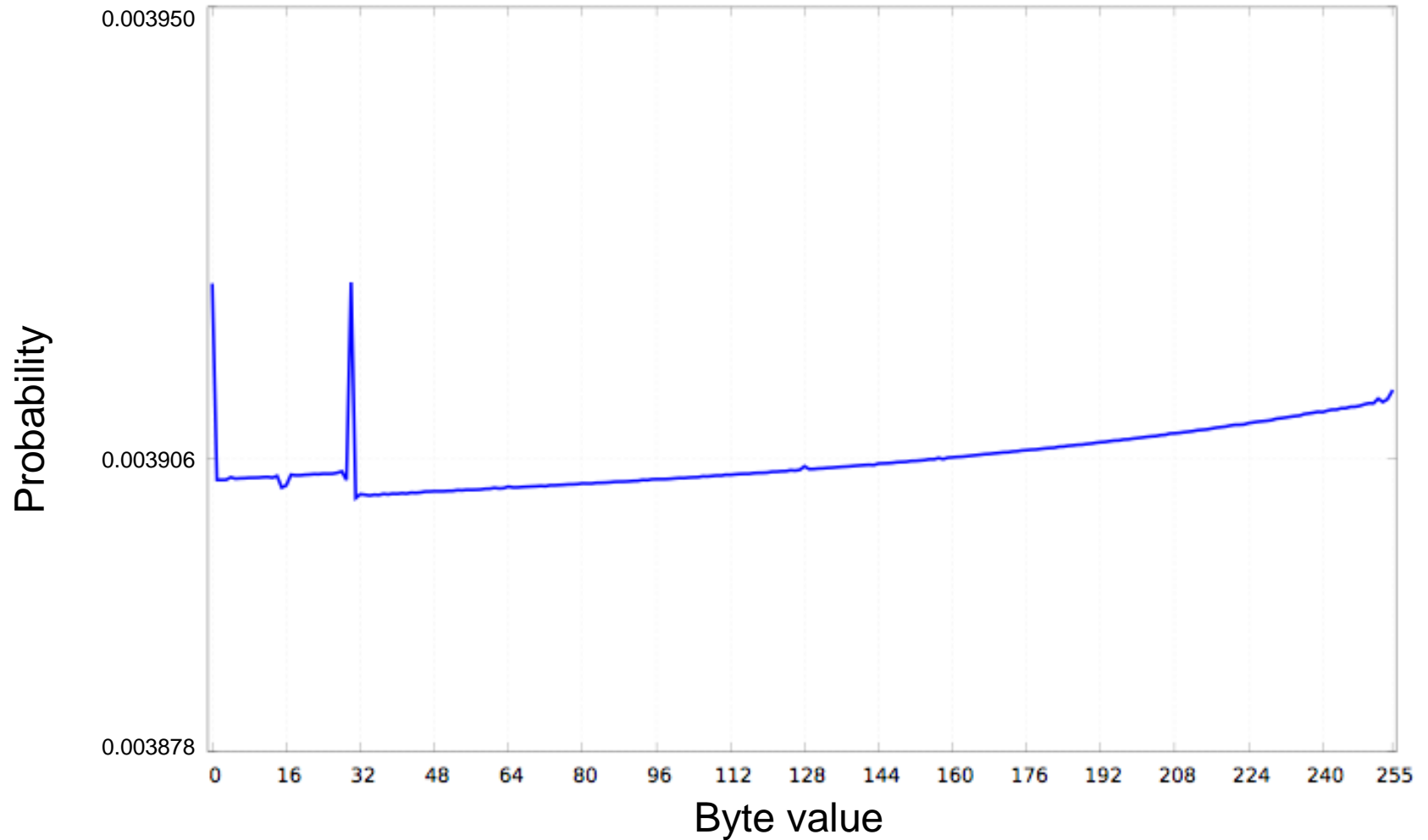
Keystream Distribution at Position 28



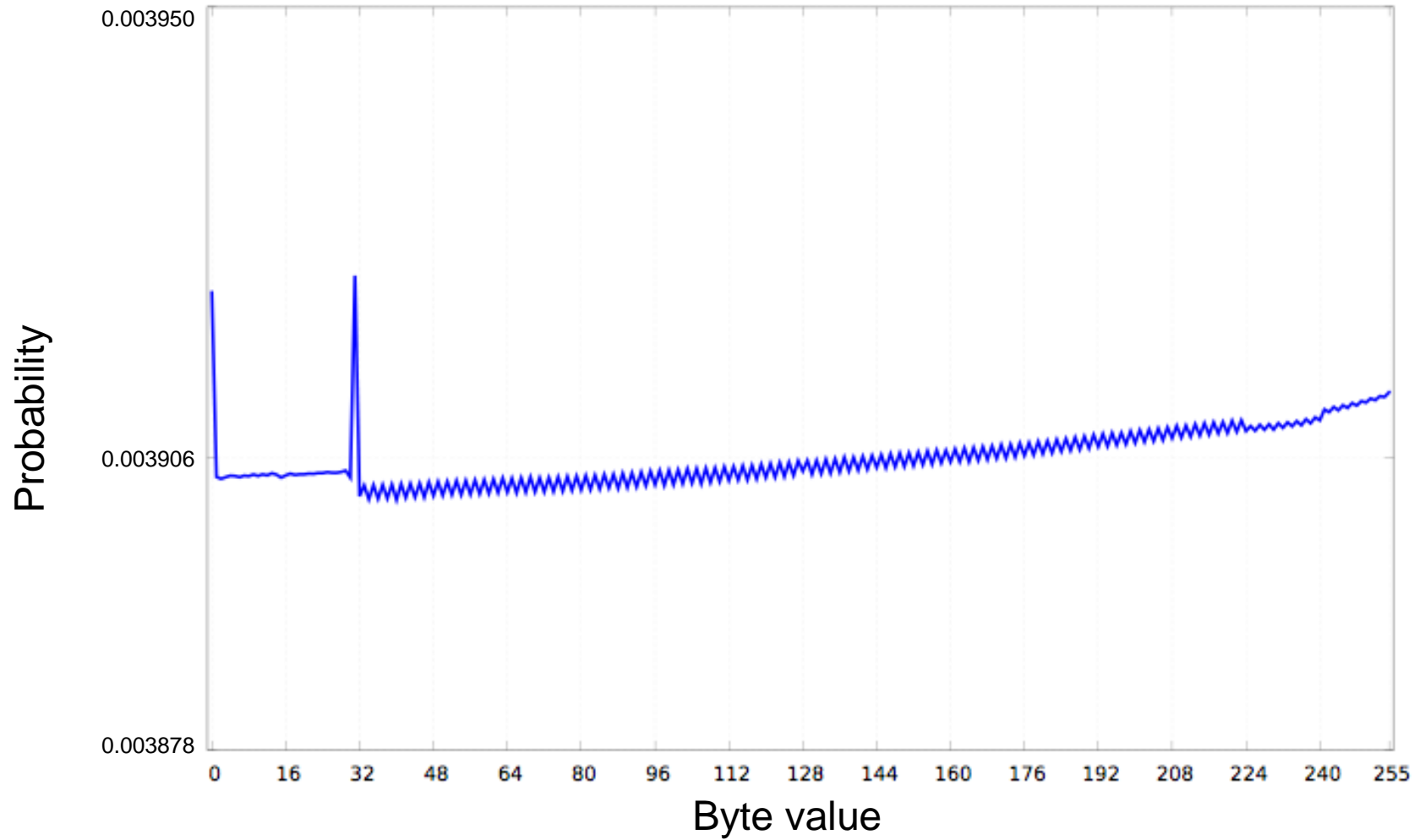
Keystream Distribution at Position 29



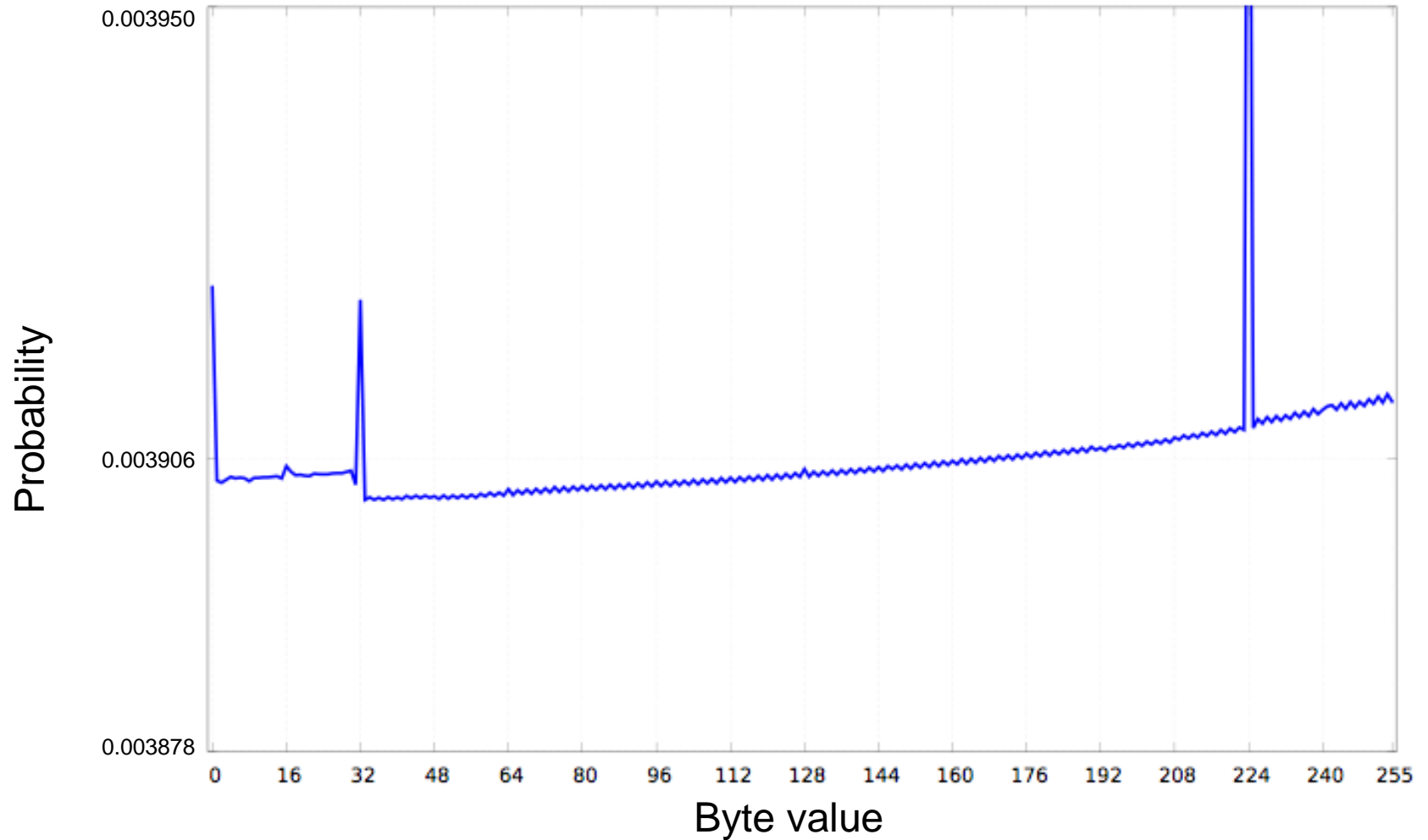
Keystream Distribution at Position 30



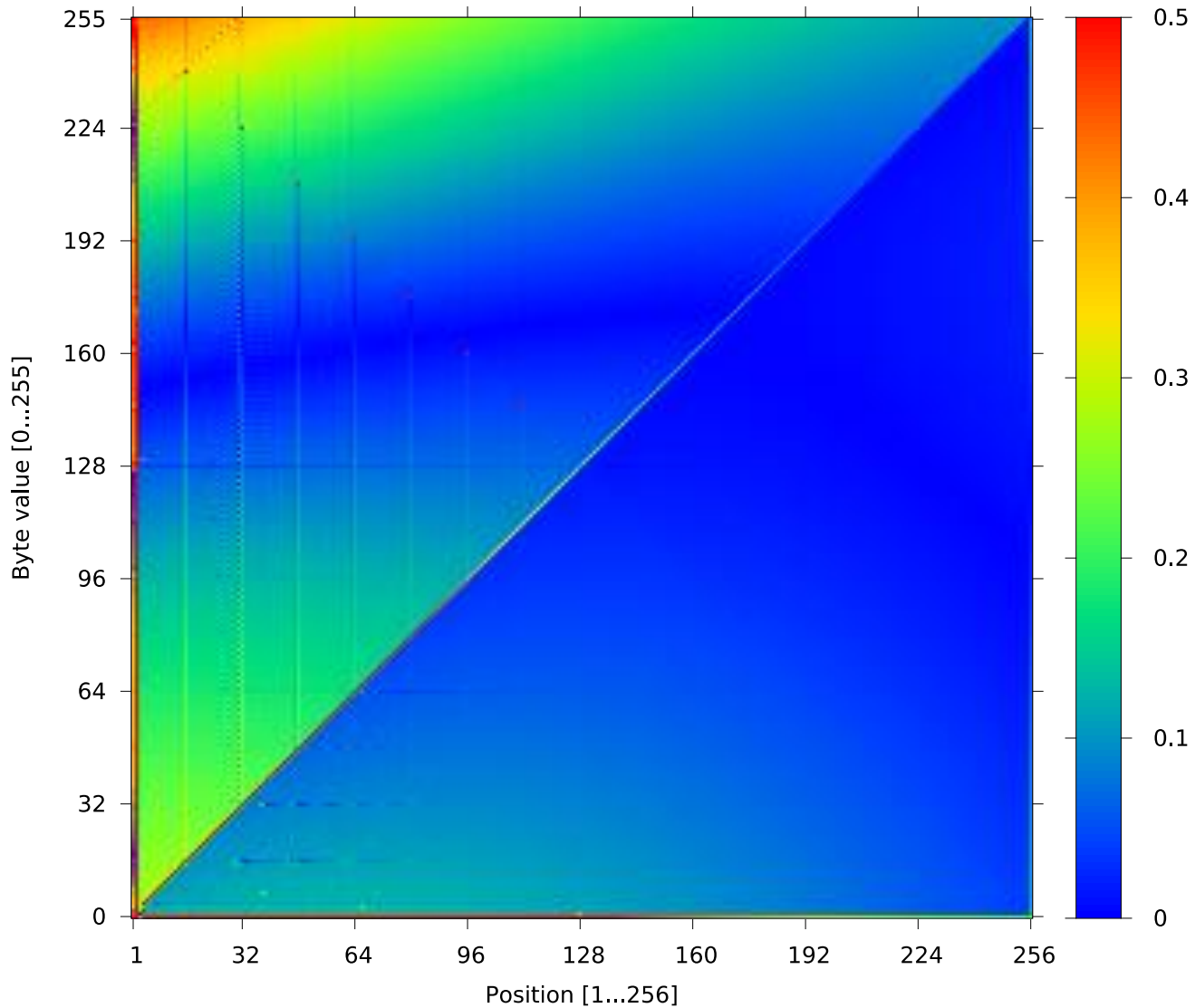
Keystream Distribution at Position 31



Keystream Distribution at Position 32



All the Biases



Broadcast Attack for RC₄ in TLS

Is the attack really applicable to RC₄ in TLS?

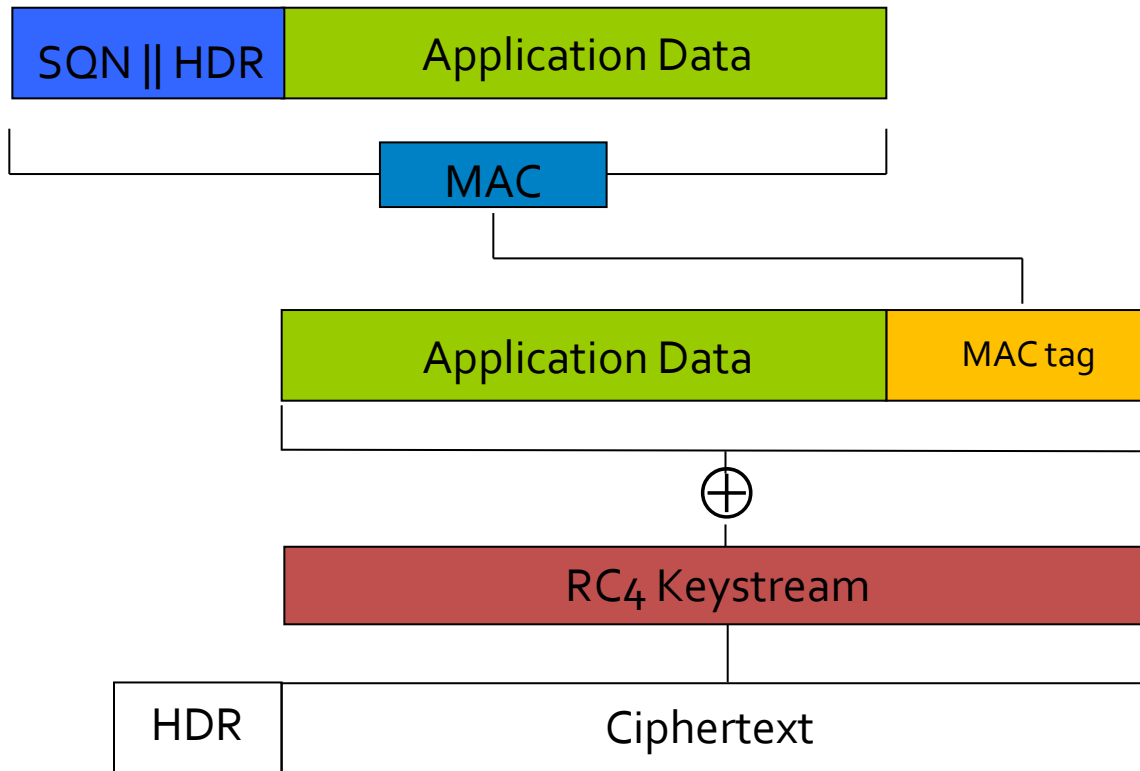
- How is the RC₄ algorithm actually used in TLS?
- How much TLS traffic is actually encrypted using RC₄?
- How can we ensure that the same plaintext is repeatedly encrypted under different keys?
- What is a good target for the repeated plaintext?
- Can we ensure the target plaintext aligns with the positions where the keystream biases are present?
- How can we deal with the fact that there are multiple biases in each keystream position Z_r ?

Broadcast Attack for RC₄ in TLS

Is the attack really applicable to RC₄ in TLS?

- How is the RC₄ algorithm actually used in TLS?
- How much TLS traffic is actually encrypted using RC₄?
- How can we ensure that the same plaintext is repeatedly encrypted under different keys?
- What is a good target for the repeated plaintext?
- How can we deal with the fact that there are multiple biases in each keystream position Z_r ?

Use of RC₄ in TLS



MAC

HMAC-MD5, HMAC-SHA1, HMAC-SHA256

Encrypt

CBC-AES128, CBC-AES256, CBC-3DES, RC4-128

Use of RC₄ in TLS

- Fresh 128-bit key for RC₄ for each TLS *connection*.
 - The key is derived from the TLS *master secret* and *nonces* exchanged in the TLS Handshake Protocol run between TLS Client and Server.
 - Different key in each direction on secure channel.
 - Think of it as a random 128-bit value.
- All bytes of RC₄ keystream are used.
- But the first 36 bytes (roughly) are used to encrypt unpredictable messages.
 - The TLS Handshake *Finished* messages.
 - So the Mantin-Shamir bias is not exploitable in this application ☹️


Rate of RC₄ Usage in TLS

- In the face of the BEAST and Lucky 13 attacks on CBC-based ciphersuites in TLS, switching to RC₄ was a recommended mitigation.



- Use of RC₄ in the wild:

ICSI Certificate Notary



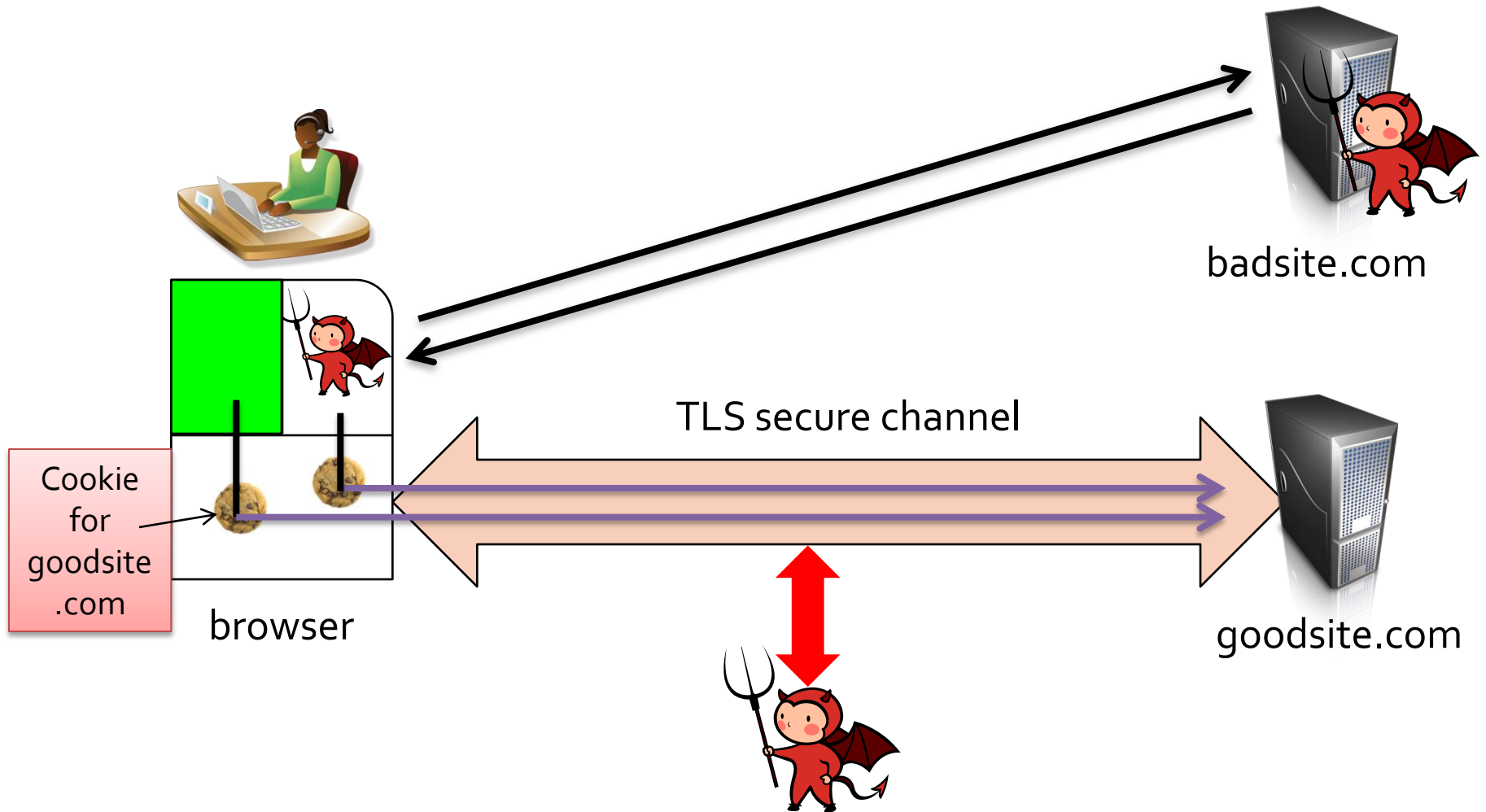
Jan. 2013 survey of 16 billion TLS connections:
Approx. **50%** protected via RC4 ciphersuites

Broadcast Attack for RC₄ in TLS

Is the attack really applicable to RC₄ in TLS?

- How is the RC₄ algorithm actually used in TLS?
- How much TLS traffic is actually encrypted using RC₄?
- How can we ensure that the same plaintext is repeatedly encrypted under different keys?
- What is a good target for the repeated plaintext?
- How can we deal with the fact that there are multiple biases in each keystream position Z_r ?

How the Web Works



How the Web Works

- Target plaintext is an HTTP secure cookie for goodsite.com.
- Browser Same Origin Policy prevents direct access to cookie.
- JavaScript running in the browser from badsite.com gives attacker the repeated plaintext capability.
- The cookie is added automatically to every HTTP request sent from the browser.
- JavaScript can pad requests in various ways to control exact position of the cookie.
- Attacker needs to force a new TLS connection for each HTTP request.
 - Can do this by having active MITM component closing TCP connection via TCP RST or sequence of TCP FIN/ACK messages.

Broadcast Attack for RC₄ in TLS

Is the attack really applicable to RC₄ in TLS?

- How is the RC₄ algorithm actually used in TLS?
- How much TLS traffic is actually encrypted using RC₄?
- How can we ensure that the same plaintext is repeatedly encrypted under different keys?
- What is a good target for the repeated plaintext?
- How can we deal with the fact that there are multiple biases in each keystream position Z_r ?

Plaintext Recovery for TLS-RC₄

We use an optimal statistical procedure based on Bayes' theorem.

- This automatically deals with the presence of multiple biases in the keystream bytes.
- [IOWM14] used a sub-optimal procedure relying only on the largest bias in each position (and did not consider in detail the applicability to TLS).

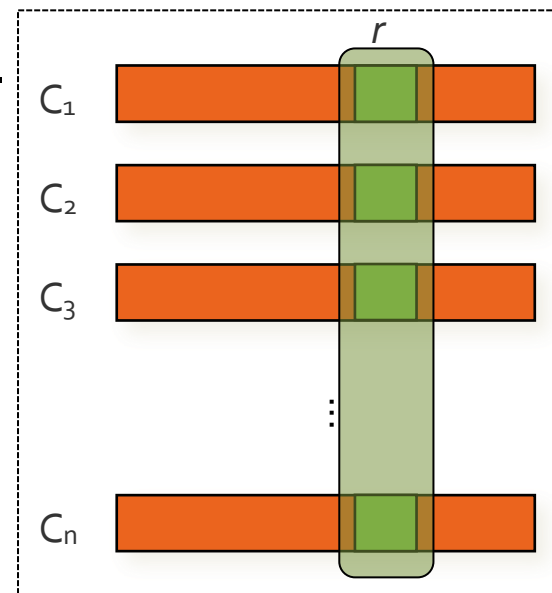
Details of Statistical Analysis

Let \mathbf{c} denote the n -vector of ciphertext bytes in position r .

We wish to maximise $\Pr[P=p|\mathbf{C}=\mathbf{c}]$.

Bayes theorem:

$$\begin{aligned}\Pr[P=p|\mathbf{C}=\mathbf{c}] &= \Pr[\mathbf{C}=\mathbf{c} | P=p].\Pr[P=p]/\Pr[\mathbf{C}=\mathbf{c}] \\ &= \Pr[\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)].\Pr[P=p]/\Pr[\mathbf{C}=\mathbf{c}]\end{aligned}$$



$\Pr[\mathbf{C}=\mathbf{c}]$ is independent of the choice of p .

For simplicity, assume $\Pr[P=p]$ is constant.

Then to maximise $\Pr[P=p | \mathbf{C}=\mathbf{c}]$ over all choices of p , we simply need to maximise the expression:

$$\Pr[\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)].$$

Details of Statistical Analysis

To maximise $\Pr[P=p \mid \mathbf{C}=\mathbf{c}]$ over all choices of p , we simply need to maximise the expression:

$$\Pr[\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)].$$

Formally, this is the *likelihood* of the keystream bytes $\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)$.

Let $\mathbf{q} = (q_{00}, q_{01}, \dots, q_{FF})$ denote the vector of keystream byte probabilities in position r .

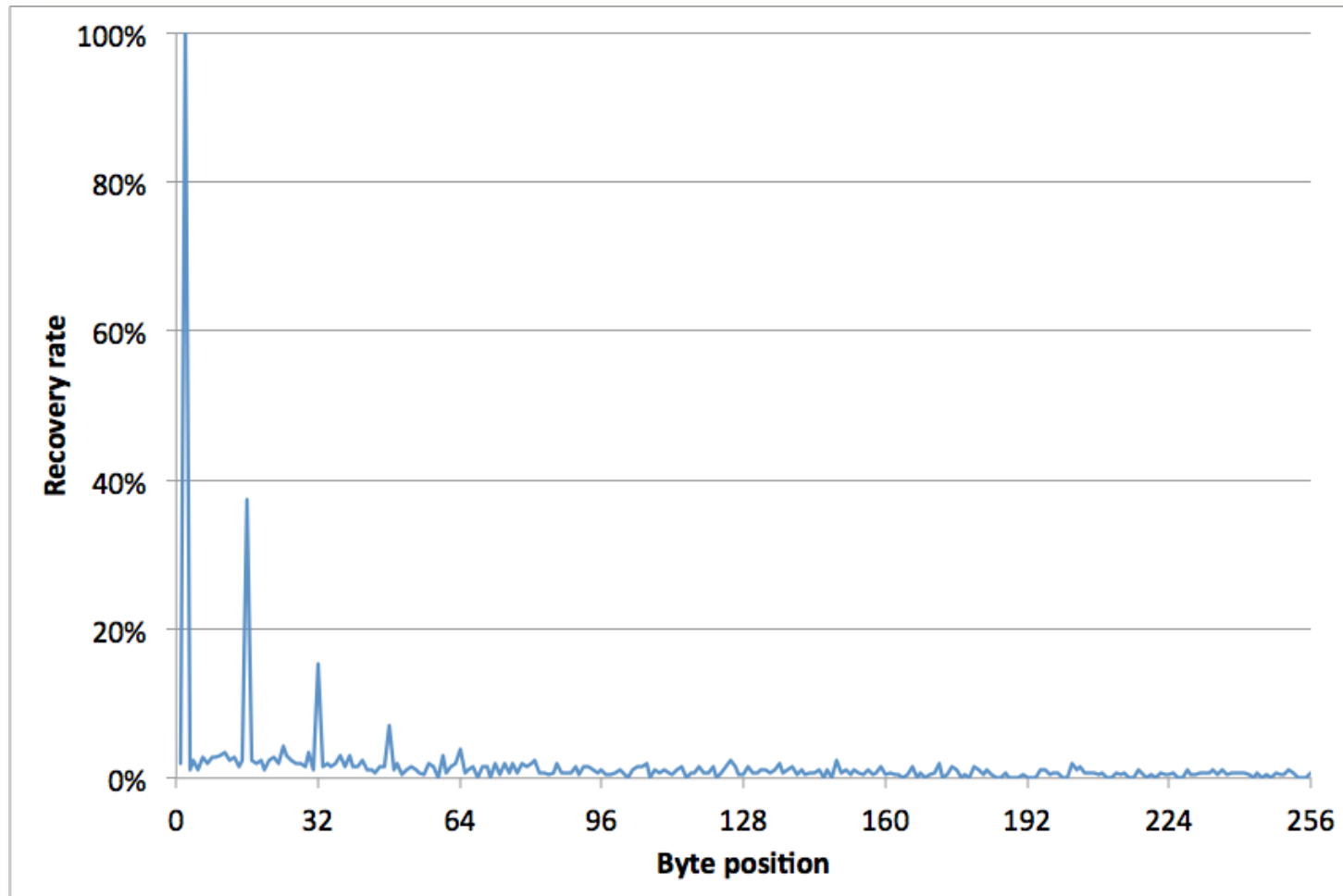
Let n_x be the number of occurrences of byte value x in $\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)$.

Then:

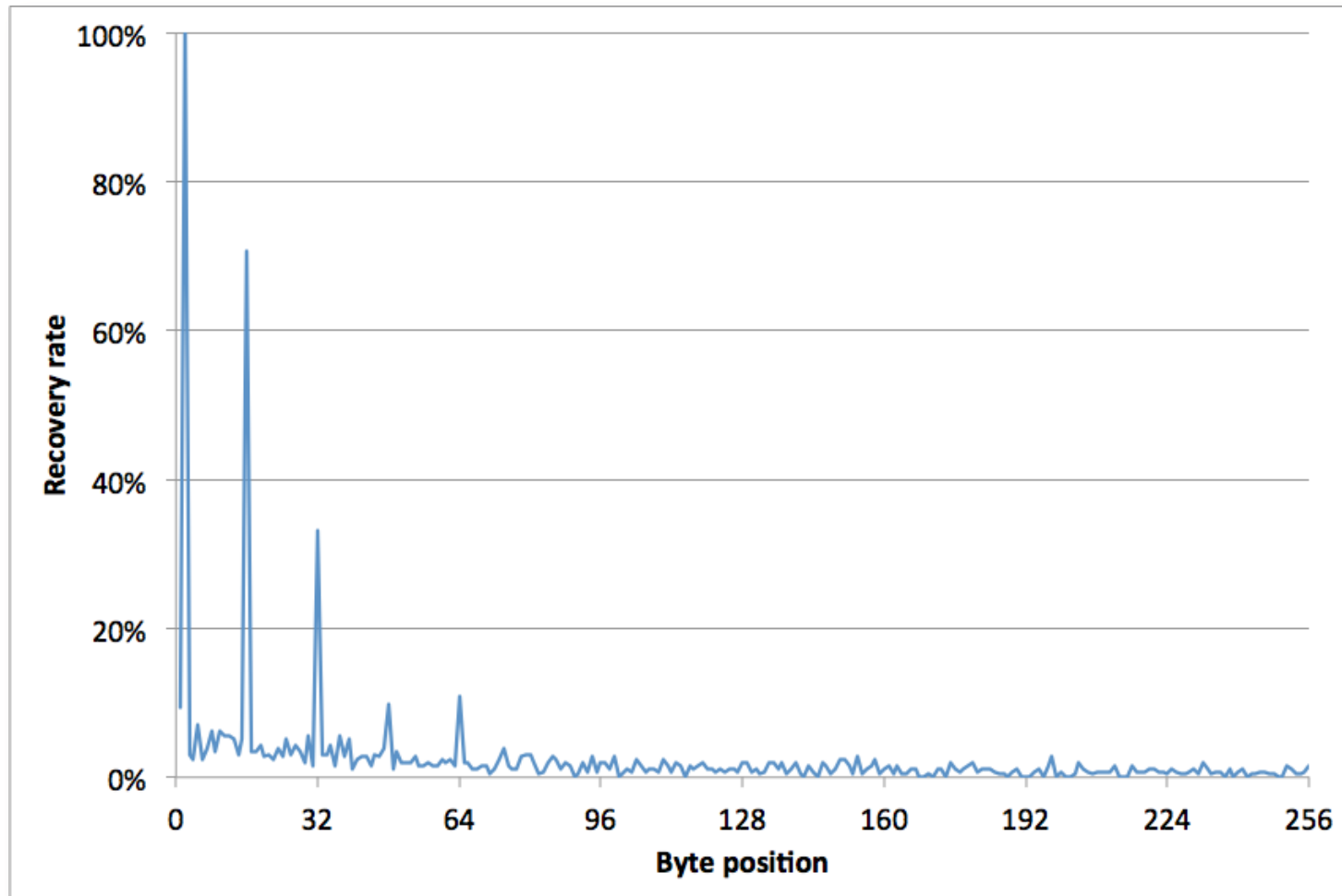
$$\Pr[\mathbf{Z}=\mathbf{c} \oplus (p,p,\dots,p)] = q_{00}^{n_{00}} q_{01}^{n_{01}} \dots q_{FF}^{n_{FF}}$$

Attack: compute this expression for each candidate p and output the value of p giving the maximum value.

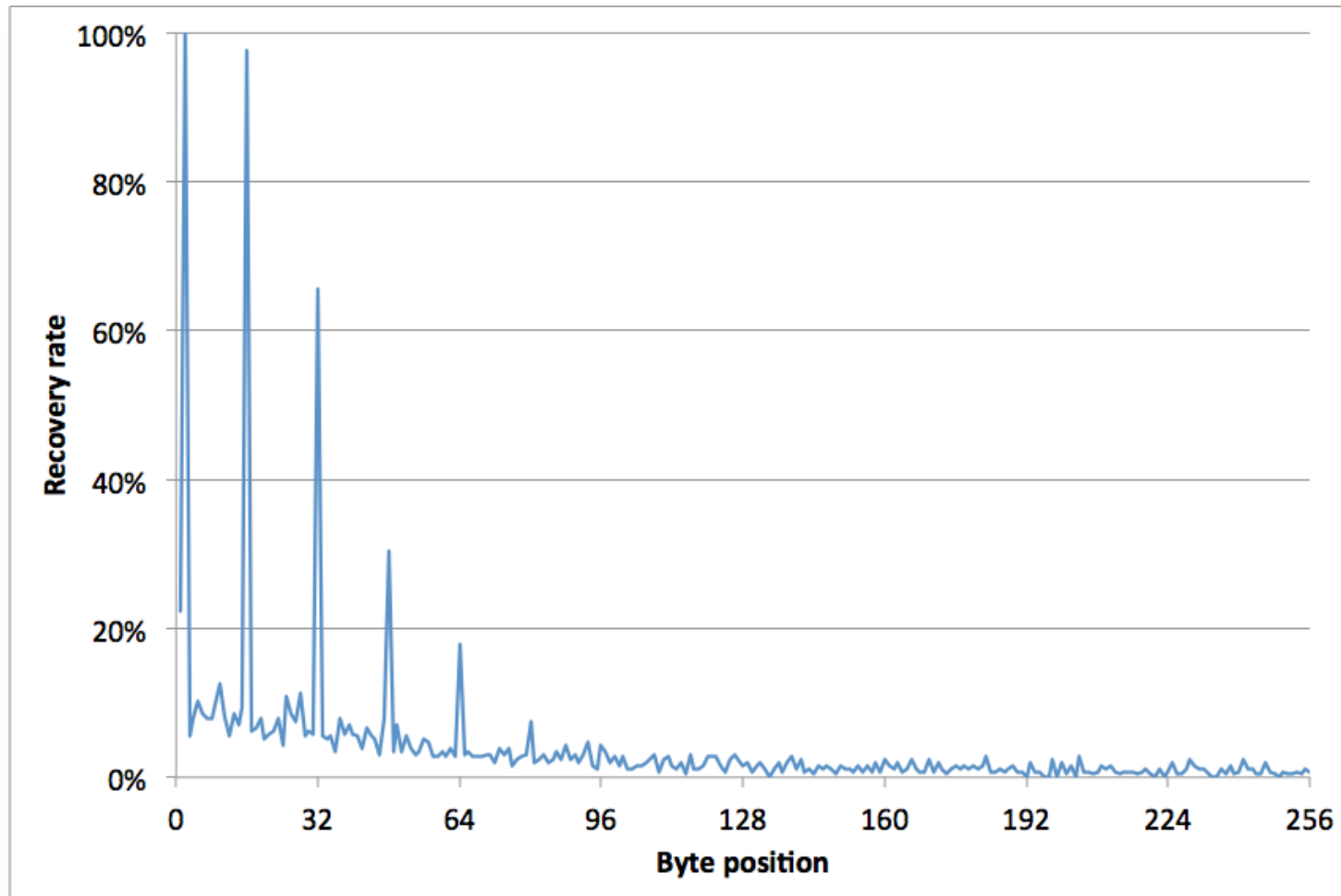
Success Probability 2^{20} Connections



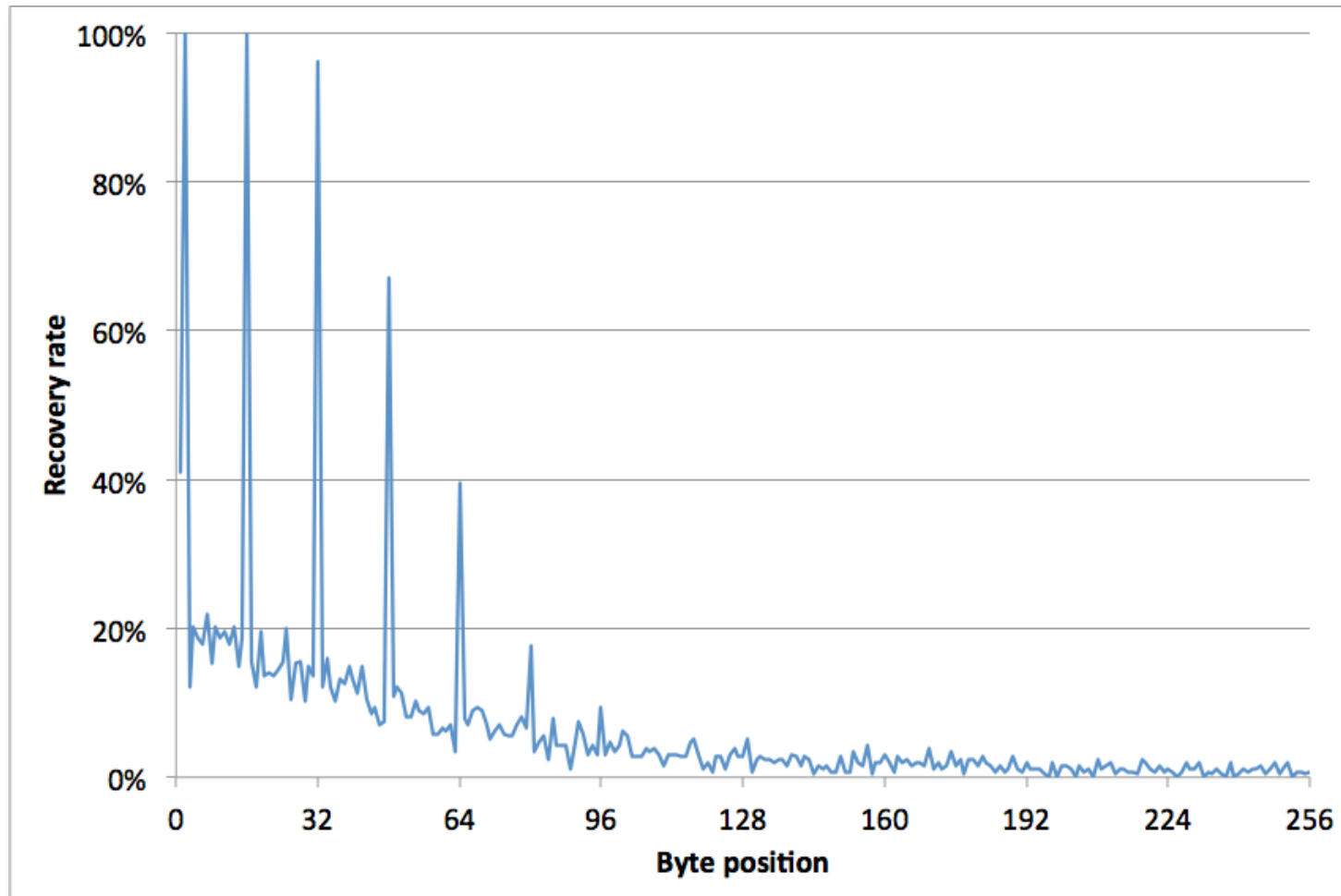
Success Probability 2^{21} Connections



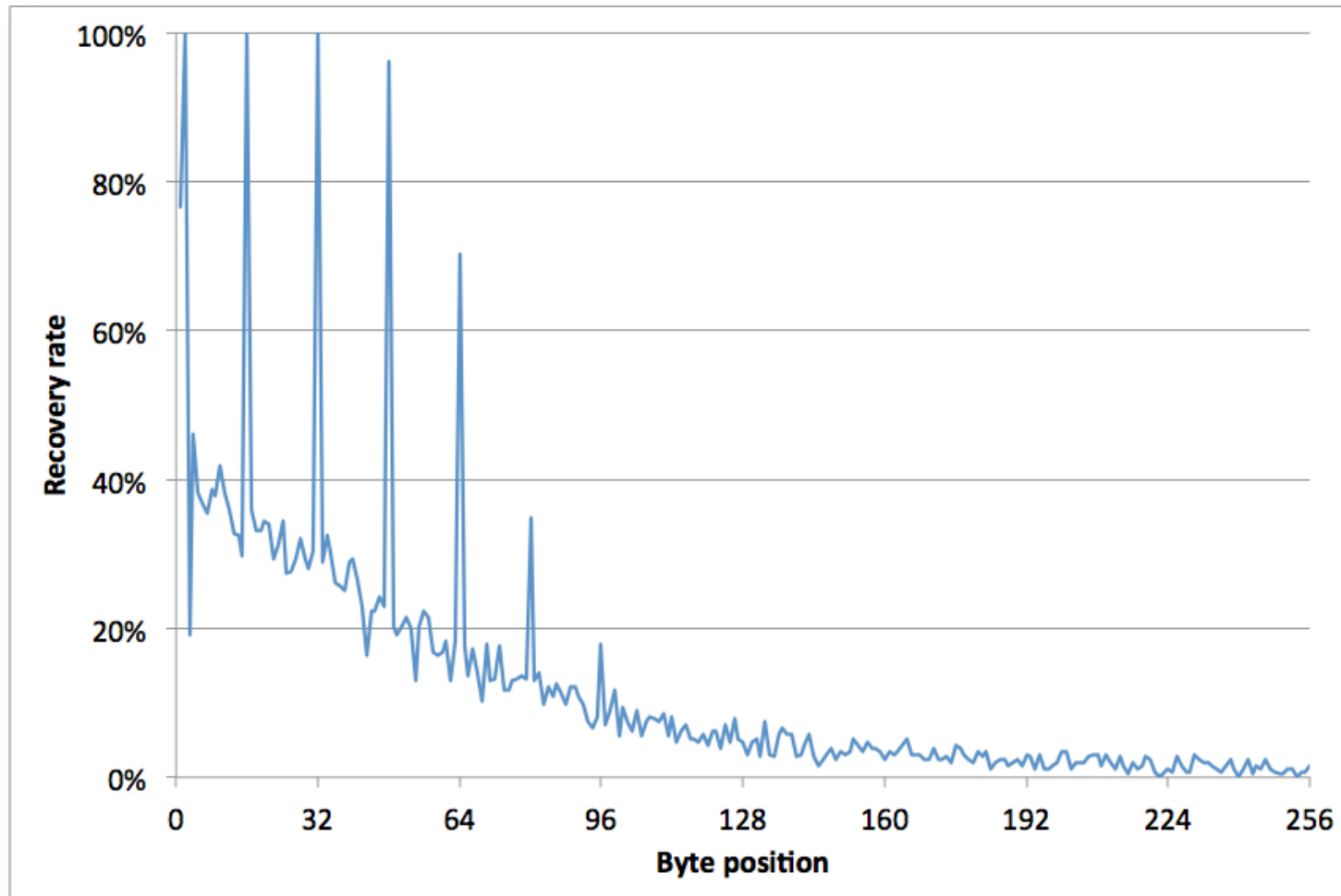
Success Probability 2^{22} Connections



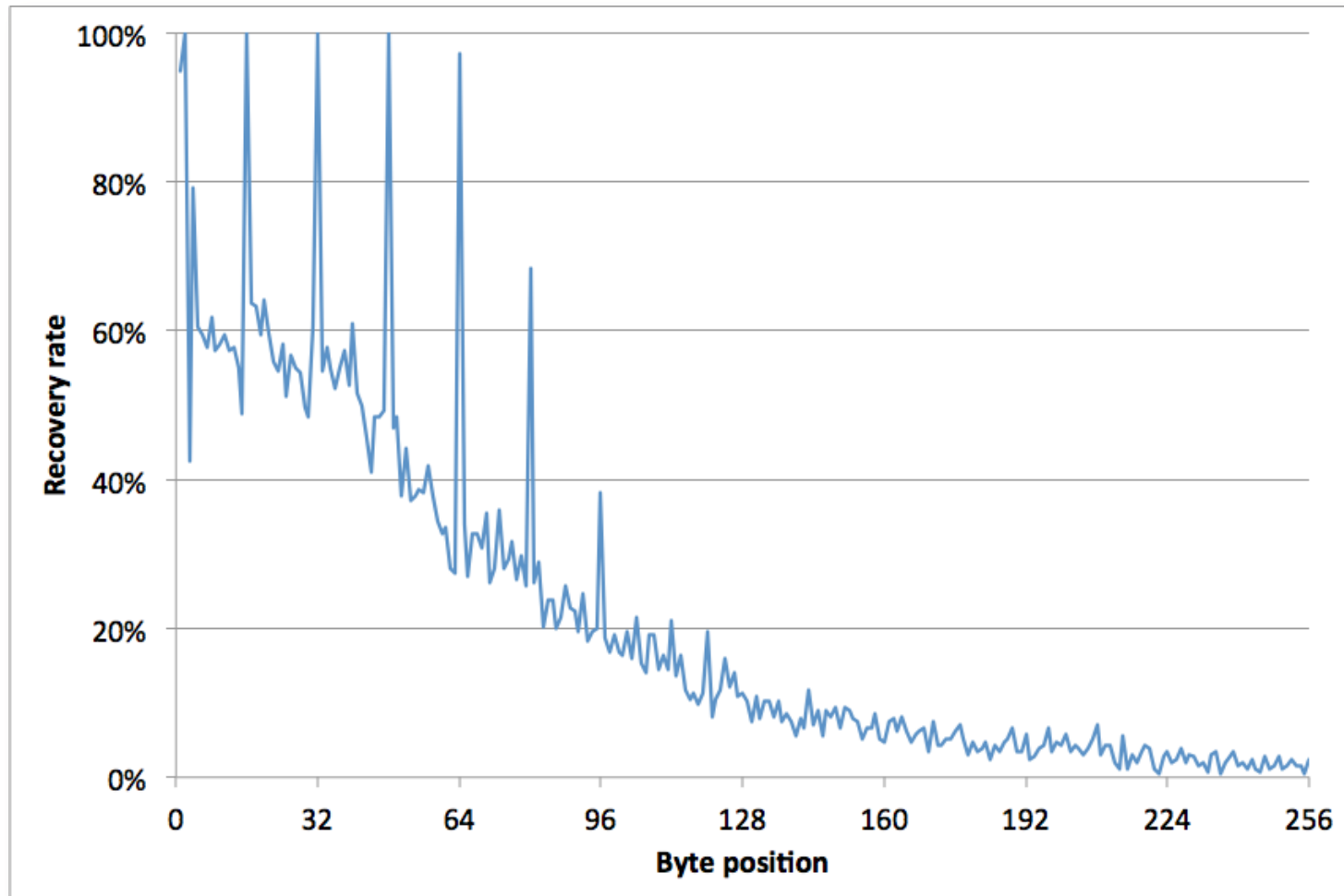
Success Probability 2^{23} Connections



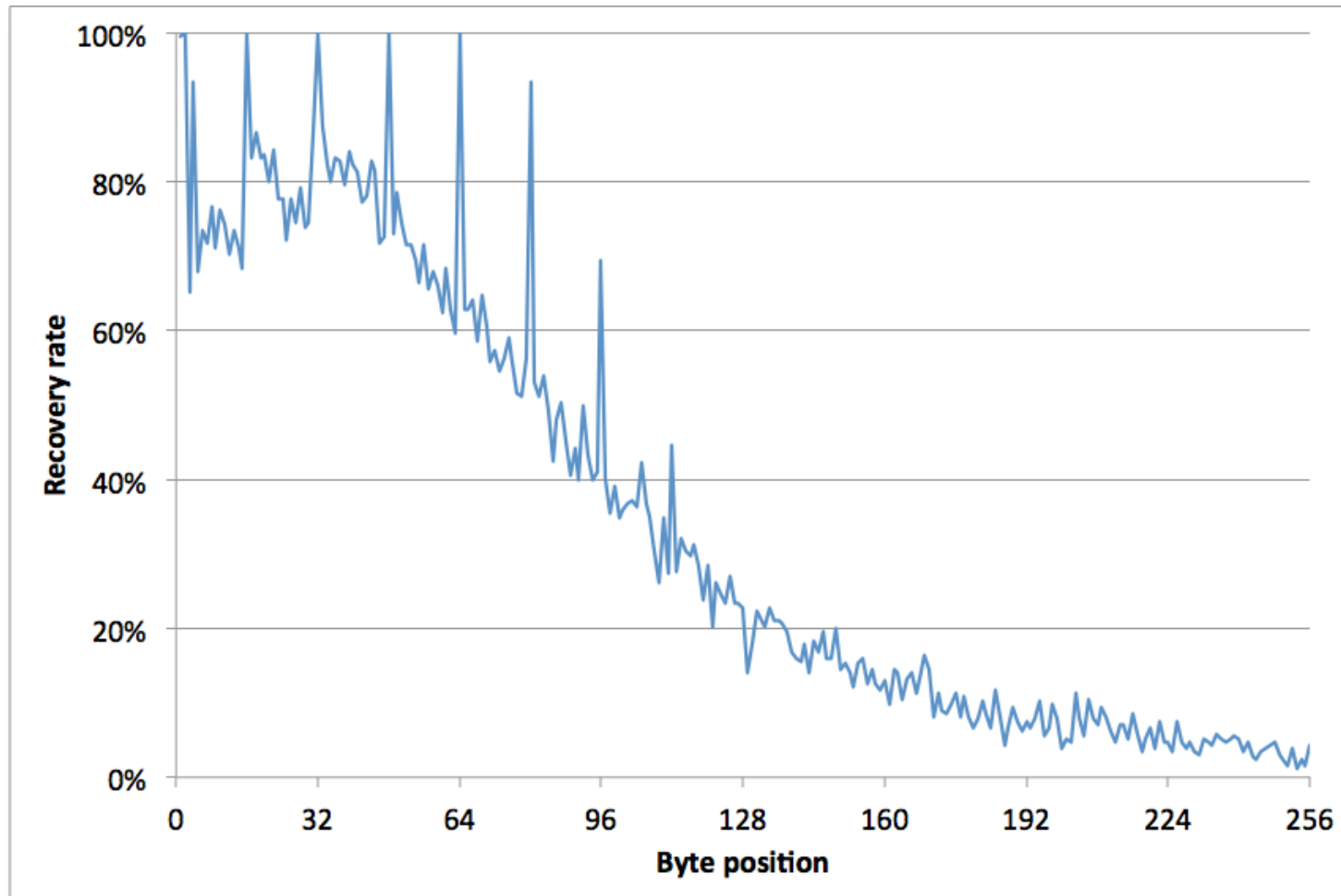
Success Probability 2^{24} Connections



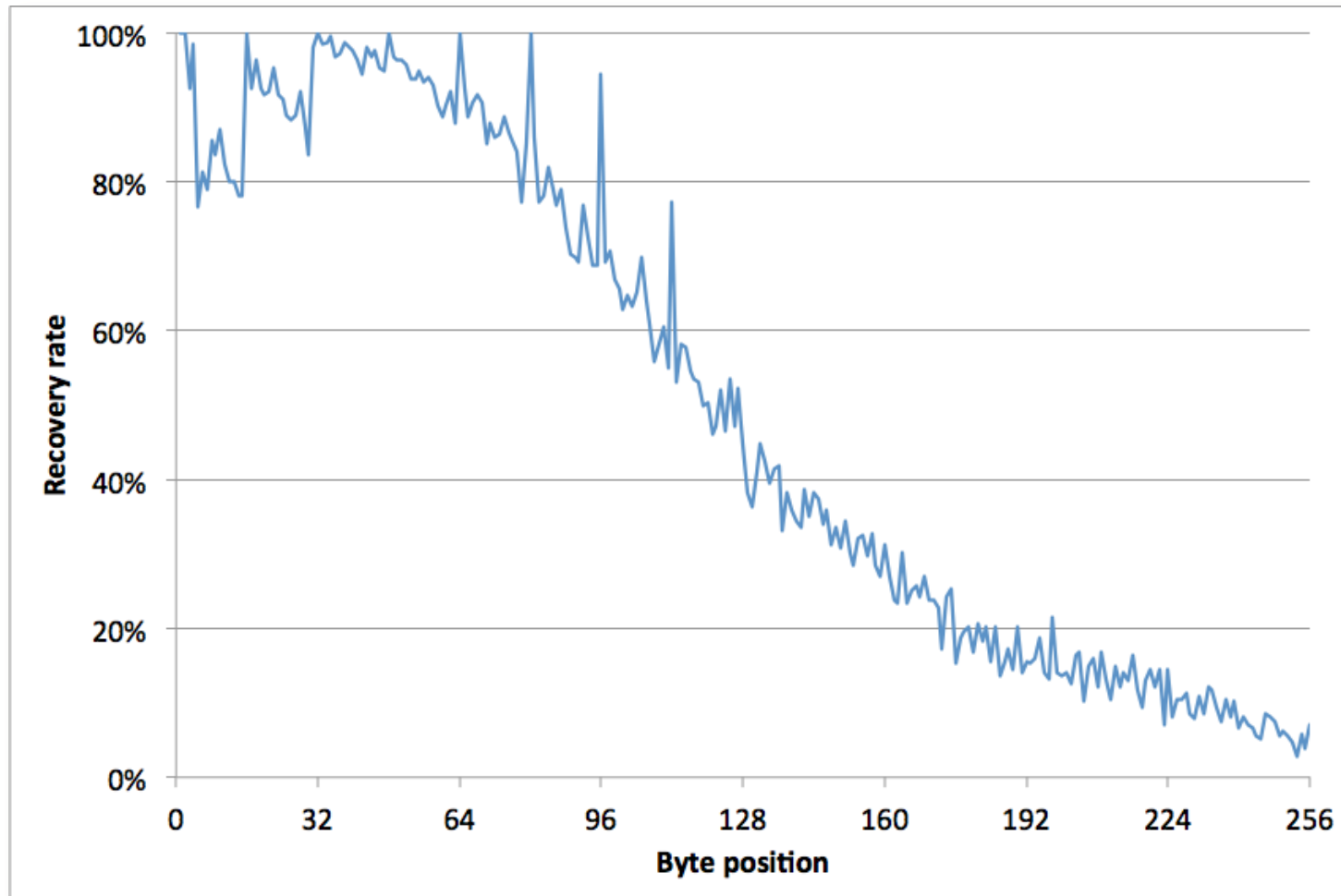
Success Probability 2^{25} Connections



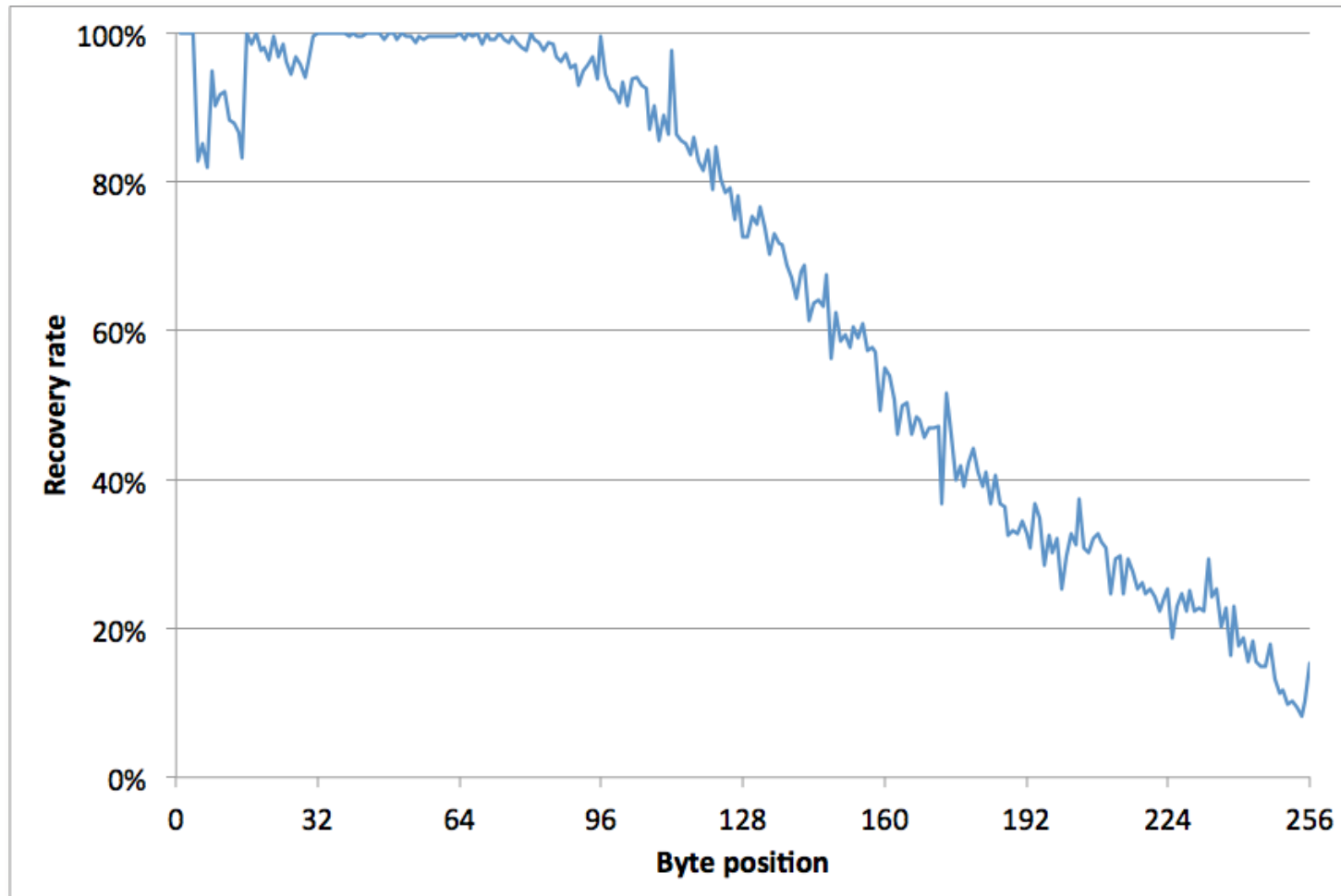
Success Probability 2^{26} Connections



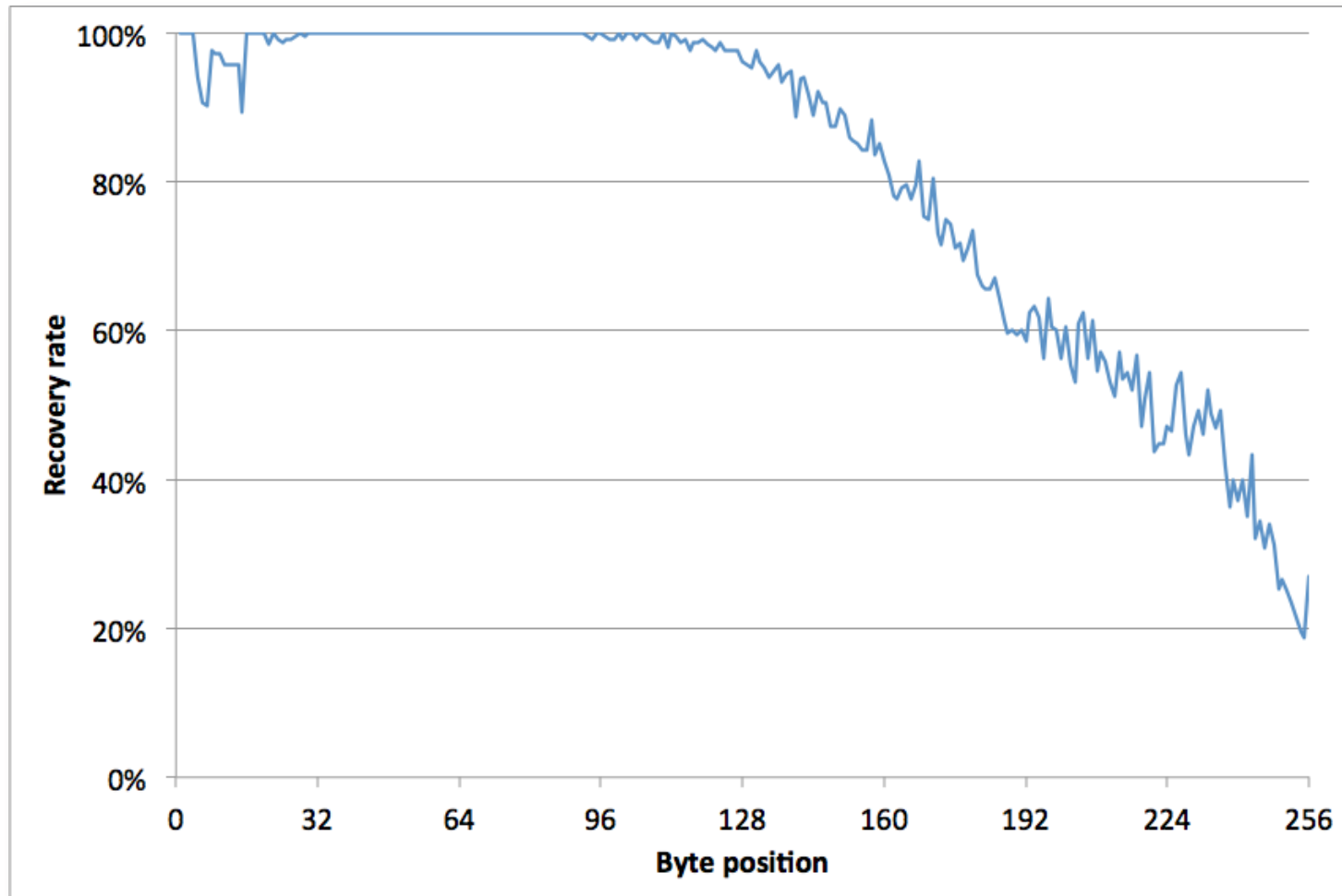
Success Probability 2^{27} Connections



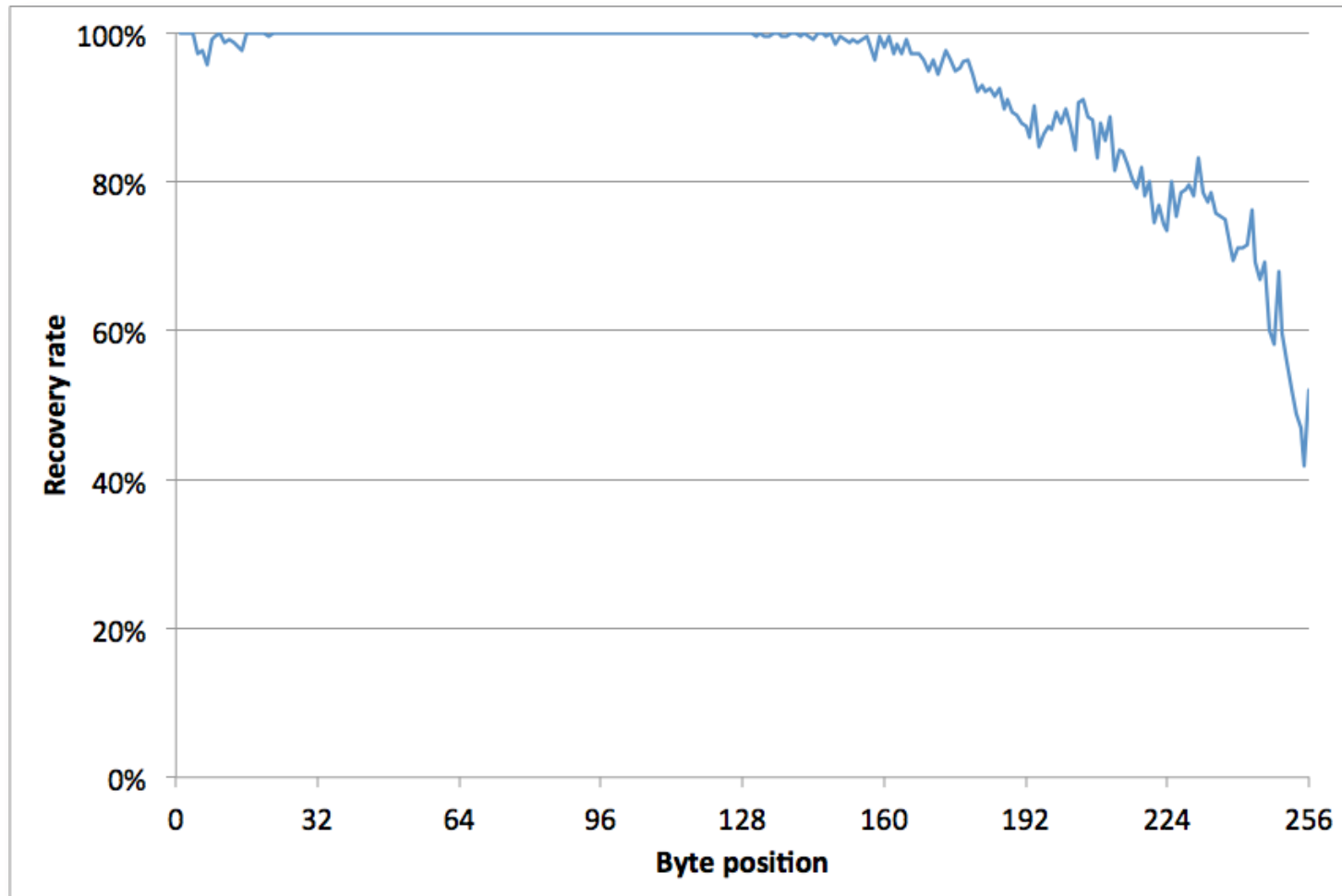
Success Probability 2^{28} Connections



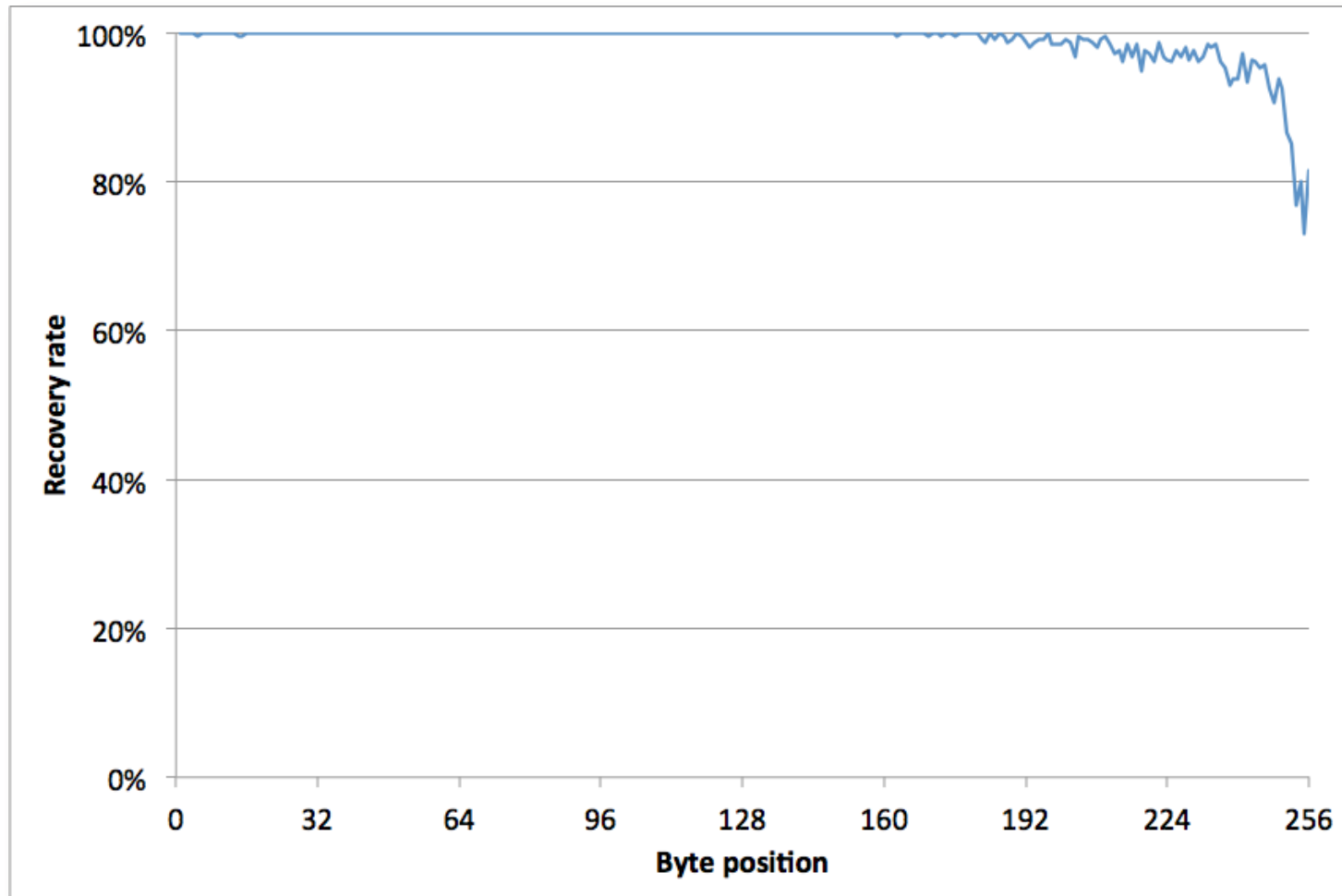
Success Probability 2^{29} Connections



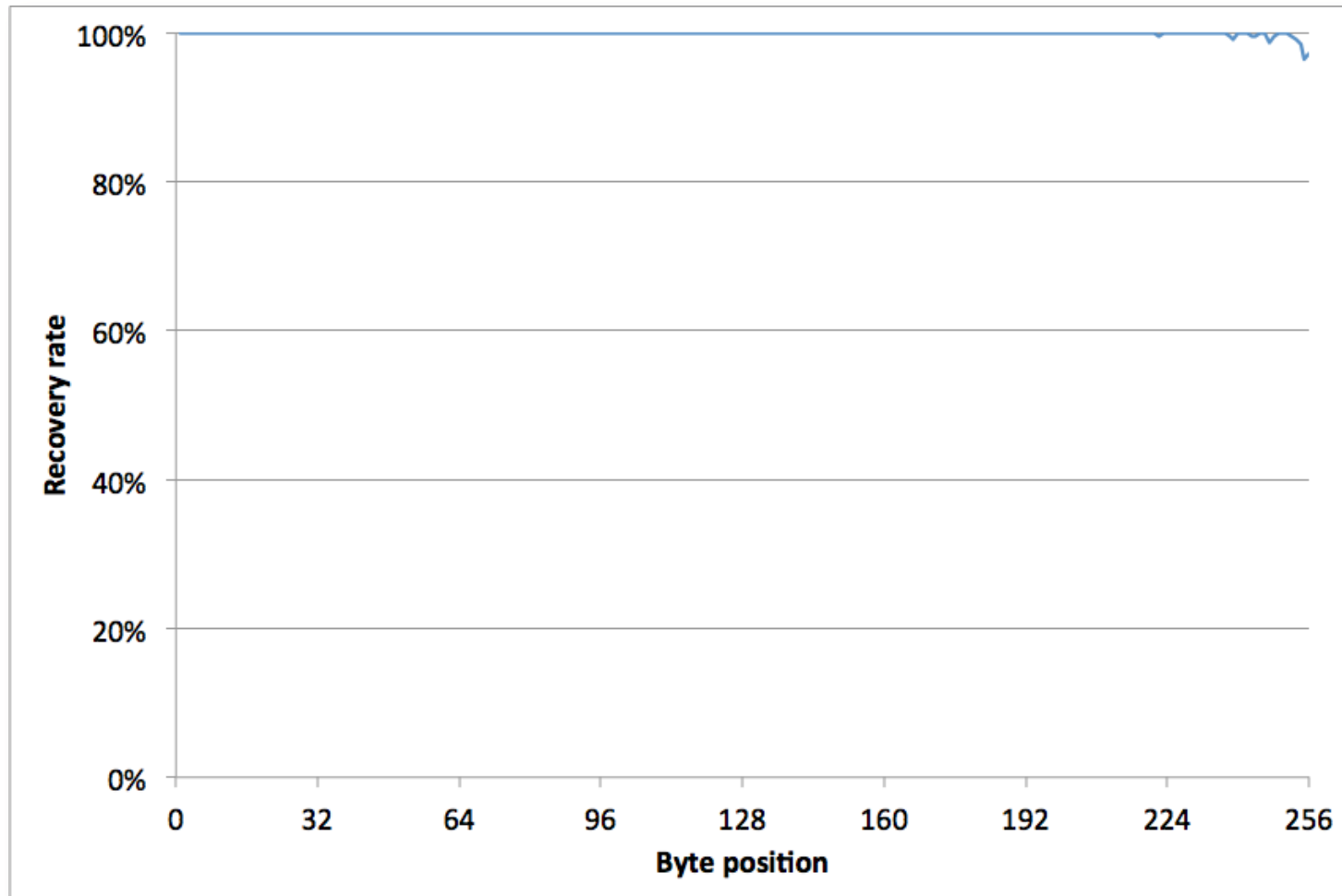
Success Probability 2^{30} Connections



Success Probability 2^{31} Connections

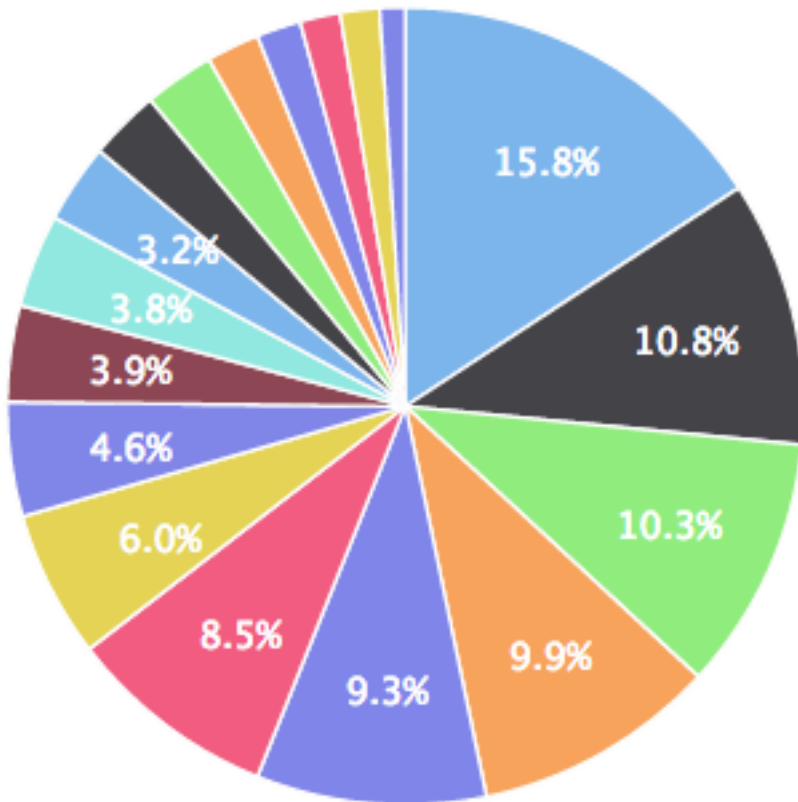


Success Probability 2^{32} Connections



Current Status of RC4 in TLS

Snapshot from ICSI Certificate Notary Project:



- RSA_RC4_128_SHA ←
- ECDHE_RSA_AES_128_CBC_SHA
- RSA_AES_256_CBC_SHA
- ECDHE_ECDSA_AES_128_GCM_SHA256
- ECDHE_RSA_AES_128_GCM_SHA256
- RSA_RC4_128_MD5 ←
- ECDHE_RSA_RC4_128_SHA ←
- RSA_AES_128_CBC_SHA
- other
- ECDHE_RSA_AES_128_CBC_SHA256
- ECDHE_ECDSA_AES_128_CBC_SHA
- ECDHE_RSA_AES_256_CBC_SHA384
- ECDHE_RSA_AES_256_CBC_SHA
- ECDHE_ECDSA_RC4_128_SHA ←
- RSA_NULL_SHA
- DHE_RSA_AES_256_CBC_SHA
- ECDHE_ECDSA_CHACHA20_POLY1305_SHA
- RSA_AES_128_CBC_SHA256

Comments

- Amount of TLS-RC₄ traffic is declining, but not as quickly as we might hope.
 - ICSI: from 50% to ~33%.
 - SSL Pulse: 81% of 150k sites surveyed still support RC₄.
 - Security Pitfalls: 1% of 400k sites support **only** RC₄!
- But attacks only get better with time...
 - Double-byte bias attack in [ABPPS13] – more ciphertexts, but single connection, so faster overall.
 - Exploitation of known plaintext distributions.
 - Ranking of plaintext candidates – e.g. for password recovery, only need password to be in top T candidates.
- **IT'S TIME TO STOP USING RC₄ IN TLS!**



Big bias hunting in Amazonia:

Attacking RC₄ in WPA/TKIP

Joint work with Bertram Poettering
and Jacob C.N. Schuldt

Introduction to WPA/TKIP

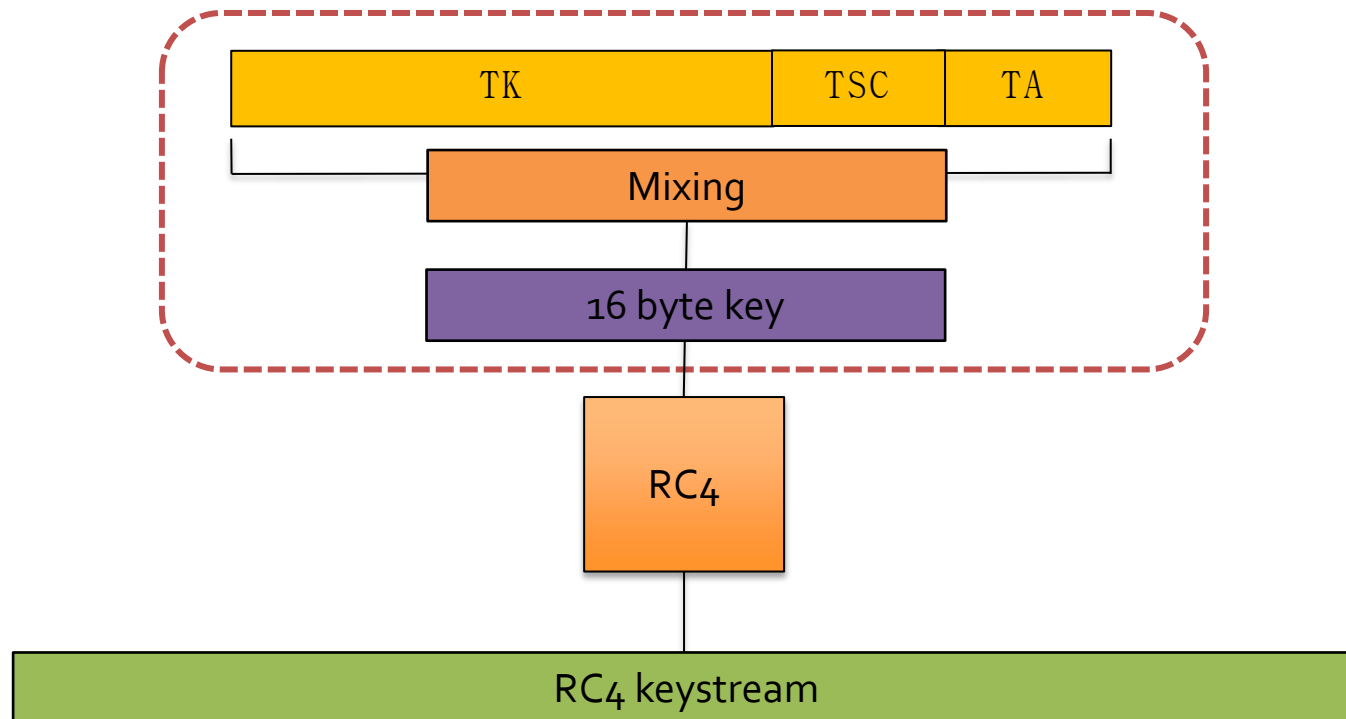
- WEP, WPA, WPA2 are all IEEE standards for wireless LAN encryption under the 802.11 family.
- WEP (1999) is considered to be badly broken
 - Beginning with [FMS01], now roughly 10k-20k packets needed for key recovery.
 - Other attacks on integrity, authentication.
- WPA/TKIP was proposed by IEEE in 2003 as an intermediate solution.
 - Allow reuse of same hardware, firmware-only upgrade.
 - Hence only limited changes to WEP design were possible.
 - Introduction of supposedly stronger per-frame keys (TKIP: Temporal Key Integrity Protocol).

Introduction to WPA/TKIP

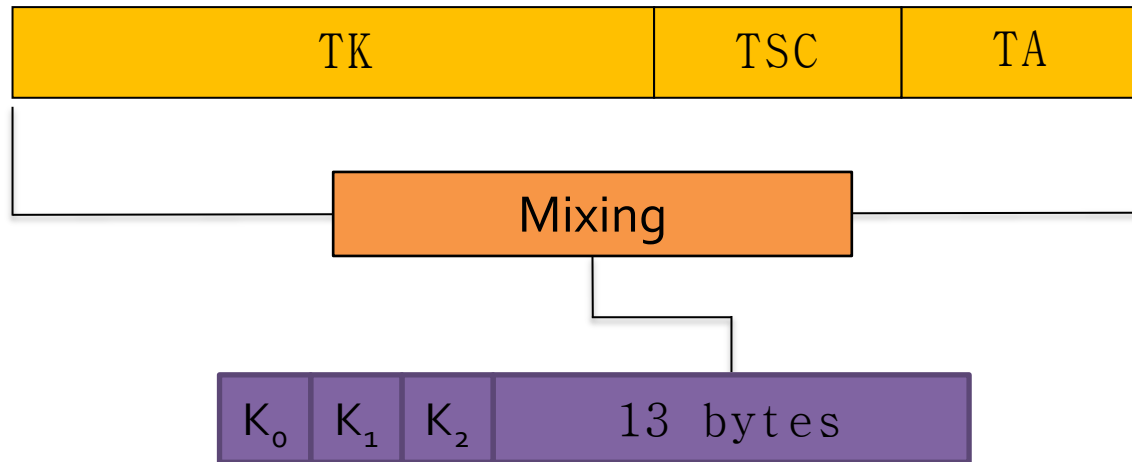
- WPA was only intended as a temporary fix.
- WPA2 (2004) introduces a strong cryptographic solution based on AES-CCM.
- But WPA is still in widespread use today.
- **Vanhoef-Piessens (2013):**
 - 71% of 6803 networks surveyed still permit WPA/TKIP; 19% allowed *only* WPA/TKIP.
- Significant previous analysis of WPA in [TB09], [SVV11].

Overview of WPA/TKIP Encryption

- TK (Temporal Key): 128 bits, used to protect many consecutive frames.
- TSC (TKIP Sequence Counter) : 48 bits, incremented for each frame sent.
- TA (Transmitter Address): 48 bits, MAC address of sender.



WPA/TKIP Key Mixing Function



$$K_0 = TSC_1$$

$$K_1 = (TSC_1 \text{ OR } 0x20) \text{ AND } 0x7f$$

$$K_2 = TSC_0$$

(TSC_0 and TSC_1 are the two least significant bytes of TSC)

Exploiting TSC Information

- We can immediately apply previous statistical attacks to WPA/TKIP, with quite some success.
 - Using keystream distributions for random keys having WPA/TKIP structure.
 - See full version of [ABPPS13] for details.

- But recall that WPA/TKIP keys have additional structure:

$$K_0 = TSC_1$$

$$K_1 = (TSC_1 \text{ OR } 0x20) \text{ AND } 0x7f$$

$$K_2 = TSC_0$$

- Recall also that the TSC value is transmitted in clear as part of the WPA/TKIP frame.

Exploiting TSC Information

- **Idea:**

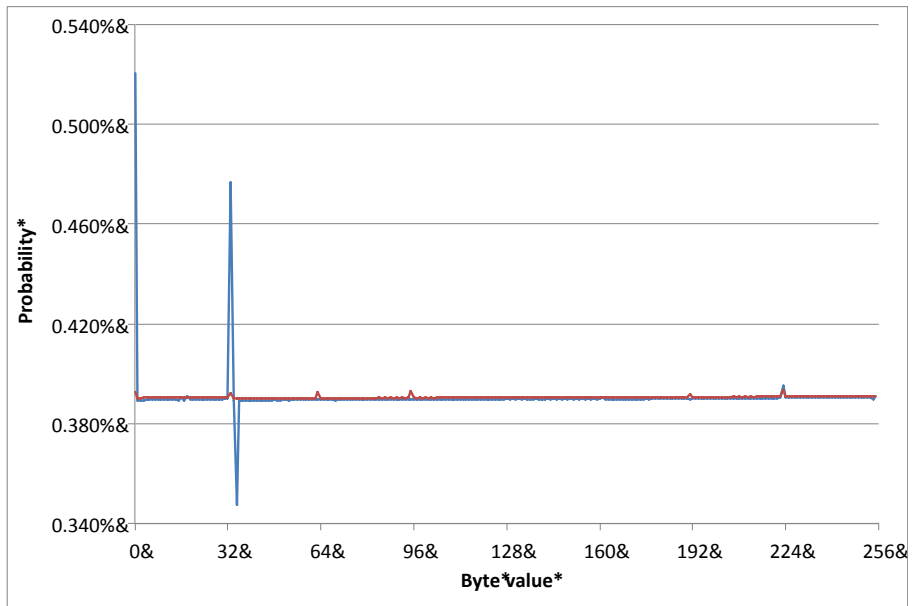
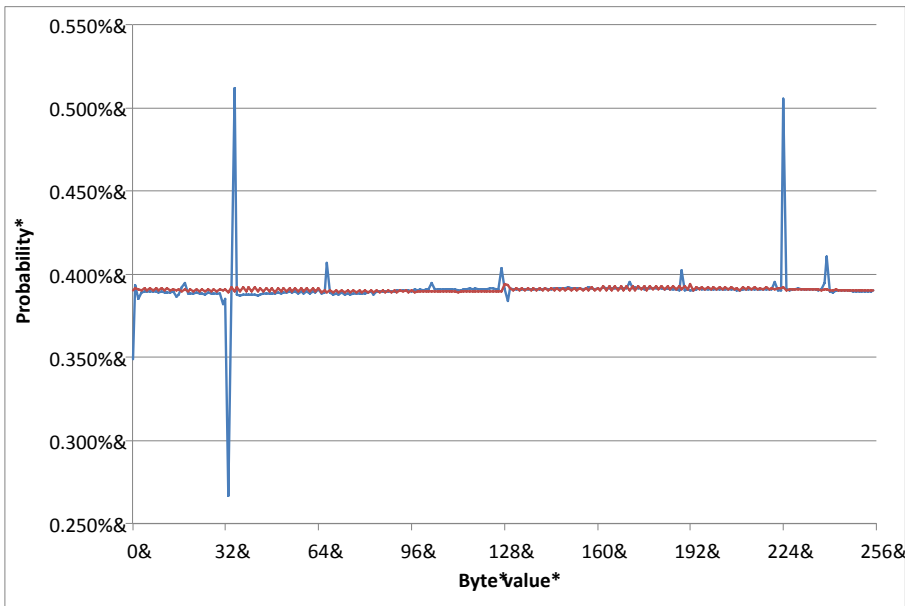
There may be even larger keystream biases that arise for *specific* (TSC_0, TSC_1) values; these could disappear when aggregating over *all* (TSC_0, TSC_1) values.

- Exploitation in plaintext recovery attack:

- Bin available ciphertexts into 2^{16} bins according to (TSC_0, TSC_1) value.
- Carry out likelihood analysis in each bin using bin-specific keystream distribution.
- *Multiply* likelihoods across bins to compute plaintext likelihoods.

- Similar (but different) ideas were developed in [SMMPS14].

Confirming Existence of Large (TSC_0, TSC_1) –specific Biases



Output byte 1, $(TSC_0, TSC_1) = (0x00, 0x00)$

Output byte 33, $(TSC_0, TSC_1) = (0x00, 0x00)$

Blue: TSC-specific biases
Red: fully aggregated WPA/TKIP biases

Exploiting TSC Information

- **Problem:**

This approach requires a large number of keystreams to get accurate estimates for each of the 2^{16} different (TSC_0, TSC_1) -specific keystream distributions.

- At a minimum, we would like to use *at least* 2^{32} keystreams for each (TSC_0, TSC_1) value, hence 2^{48} in total.
- With our local computing setup, computing 2^{24} keystreams for each of the 2^{16} (TSC_0, TSC_1) values required 2^6 core days of computation.
 - This computation did indicate the presence of many new biases.
- Desired computation would then need 2^{14} core days.

TSC₀ Aggregation

- TSC₁ is used in computing two key bytes; TSC₀ in only one:

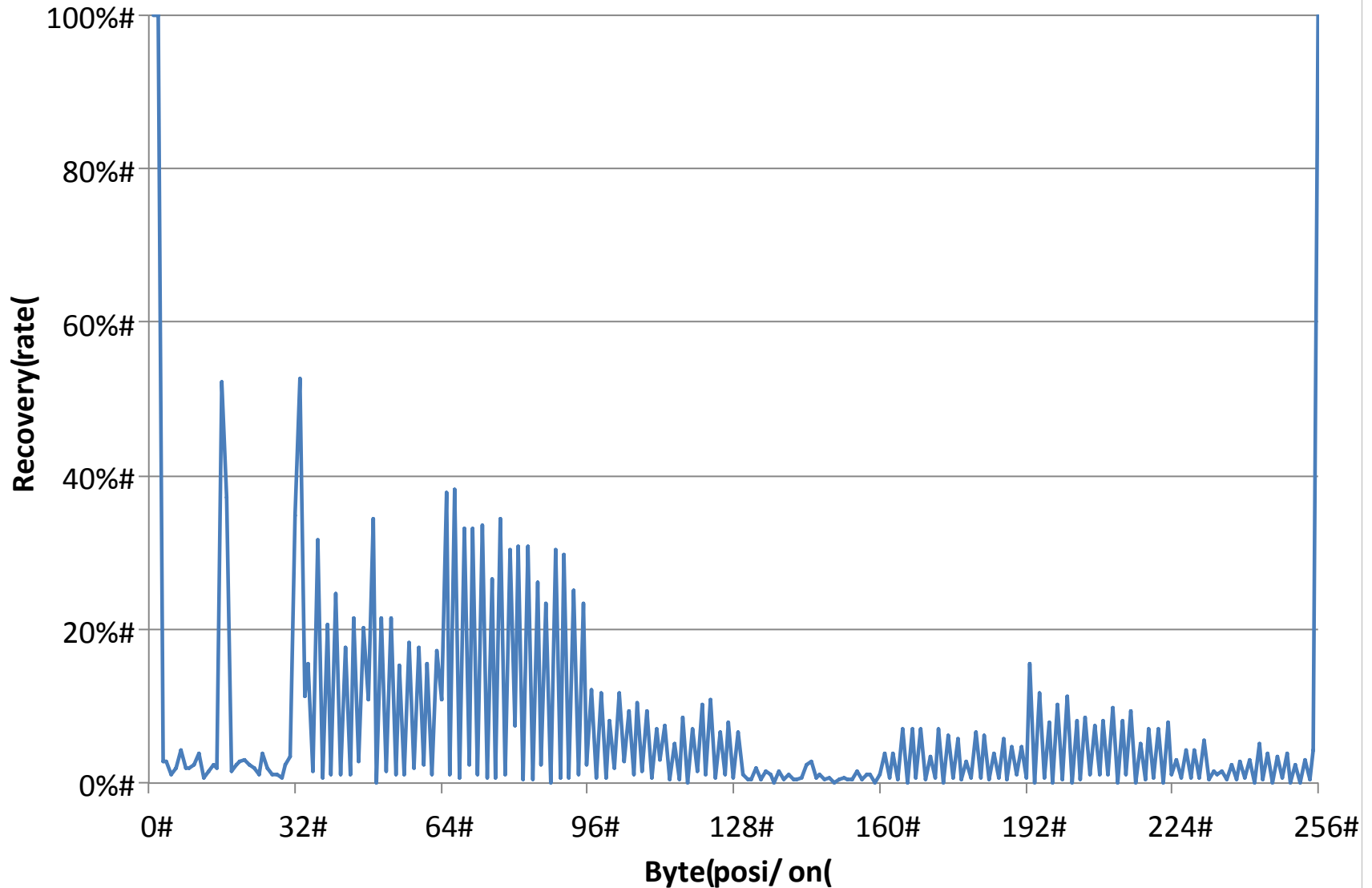
$$K_0 = \text{TSC}_1$$

$$K_1 = (\text{TSC}_1 \text{ OR } 0x20) \text{ AND } 0x7f$$

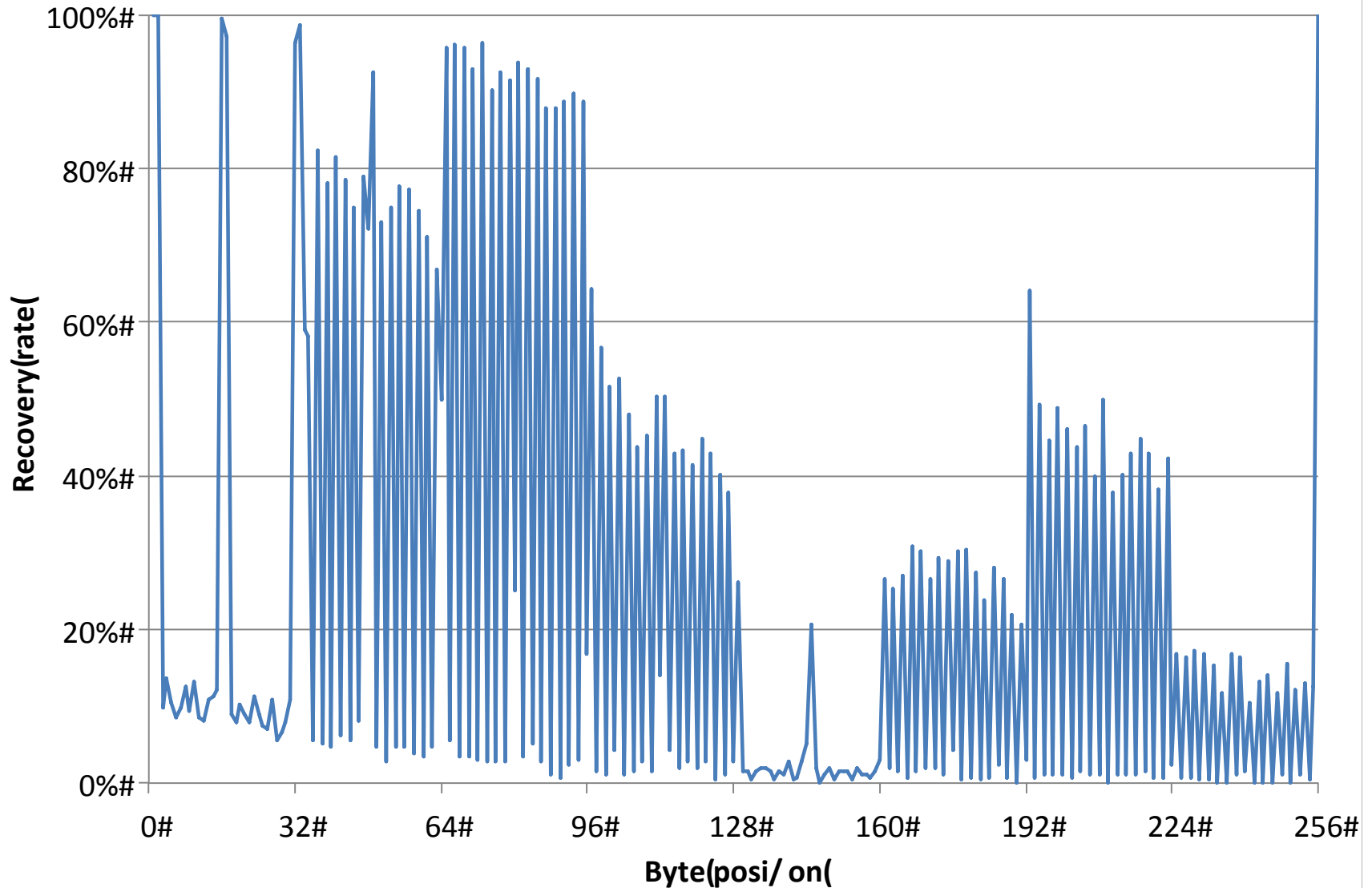
$$K_2 = \text{TSC}_0$$

- Hence we may expect biases to depend more strongly on TSC₁ than on TSC₀.
- So we could ignore TSC₀ and look only at how biases depend on TSC₁.
- Effectively, we would then be *aggregating* biases over TSC₀.
 - We call this TSC₀ aggregation
- In the plaintext recovery attack, we would then use only 2⁸ bins instead of 2¹⁶.
- And we'd need 2⁸ times fewer keystreams for estimating distributions.
- Our first attack applied to WPA/TKIP can then be seen as the variant where we aggregate over *both* TSC₀ and TSC₁, using just 1 bin.
 - We call this *full aggregation*.

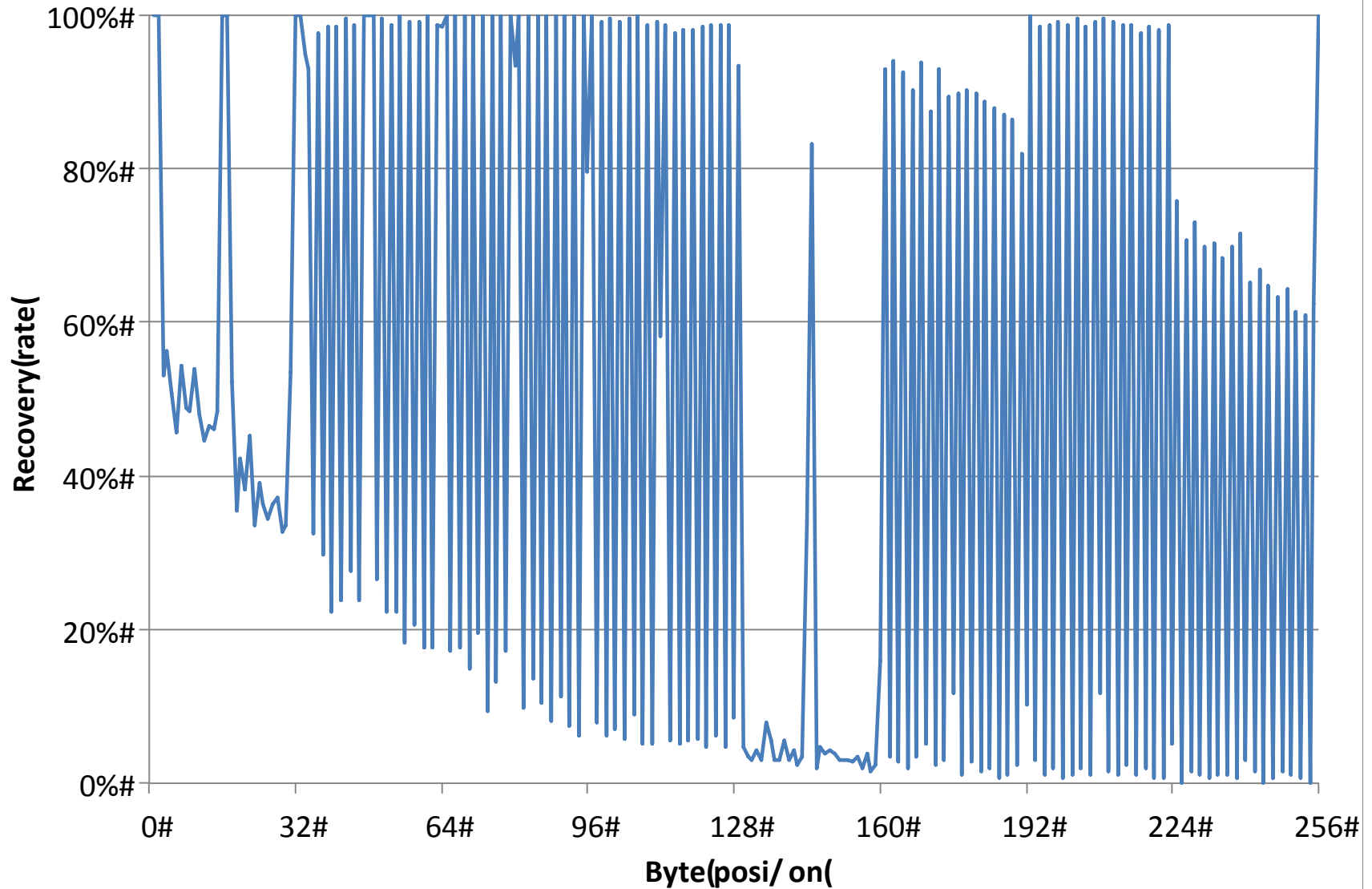
Plaintext recovery based on TSC_0 aggregation: 2^{20} frames



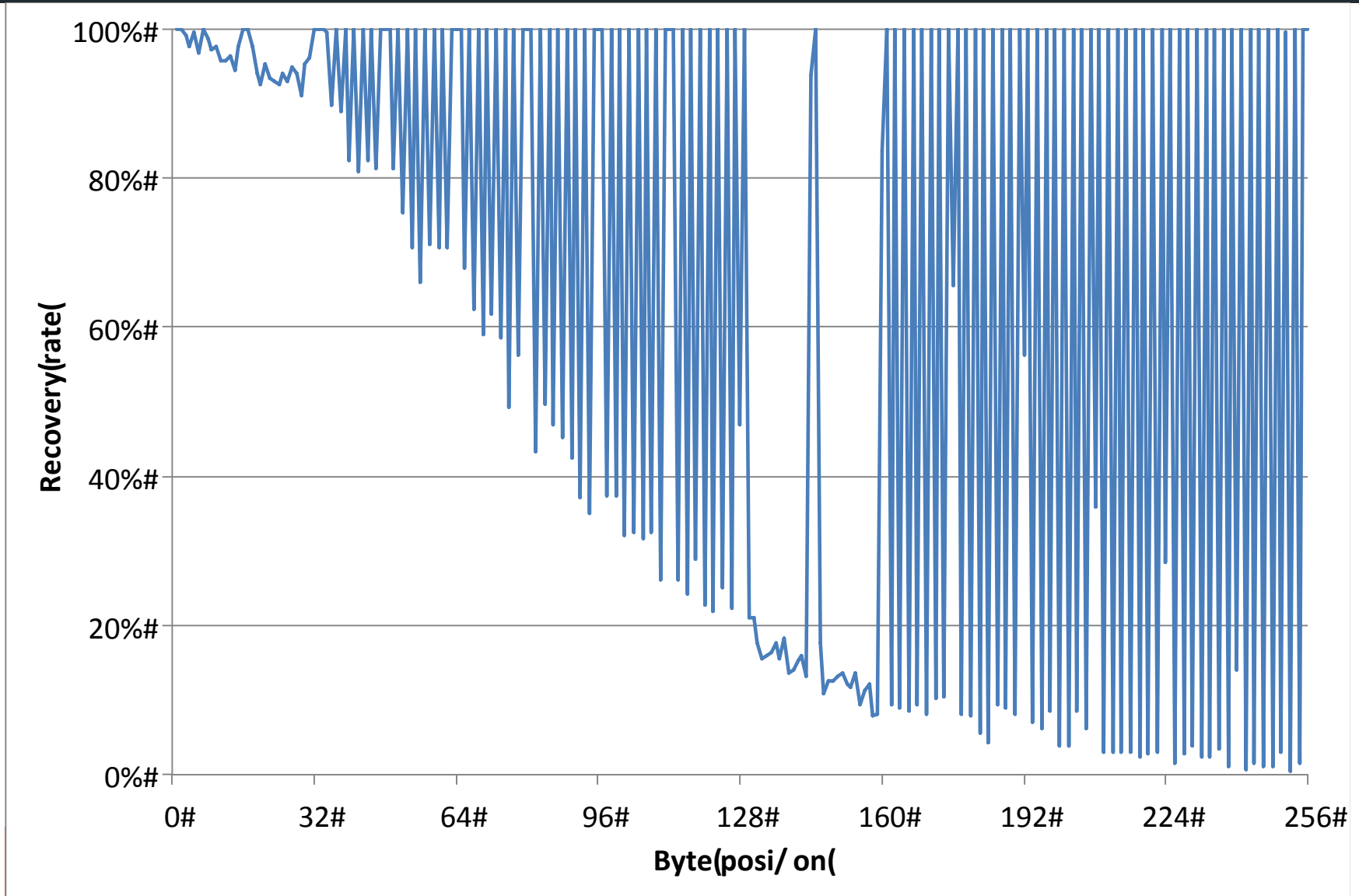
Plaintext recovery based on TSC_0 aggregation: 2^{22} frames



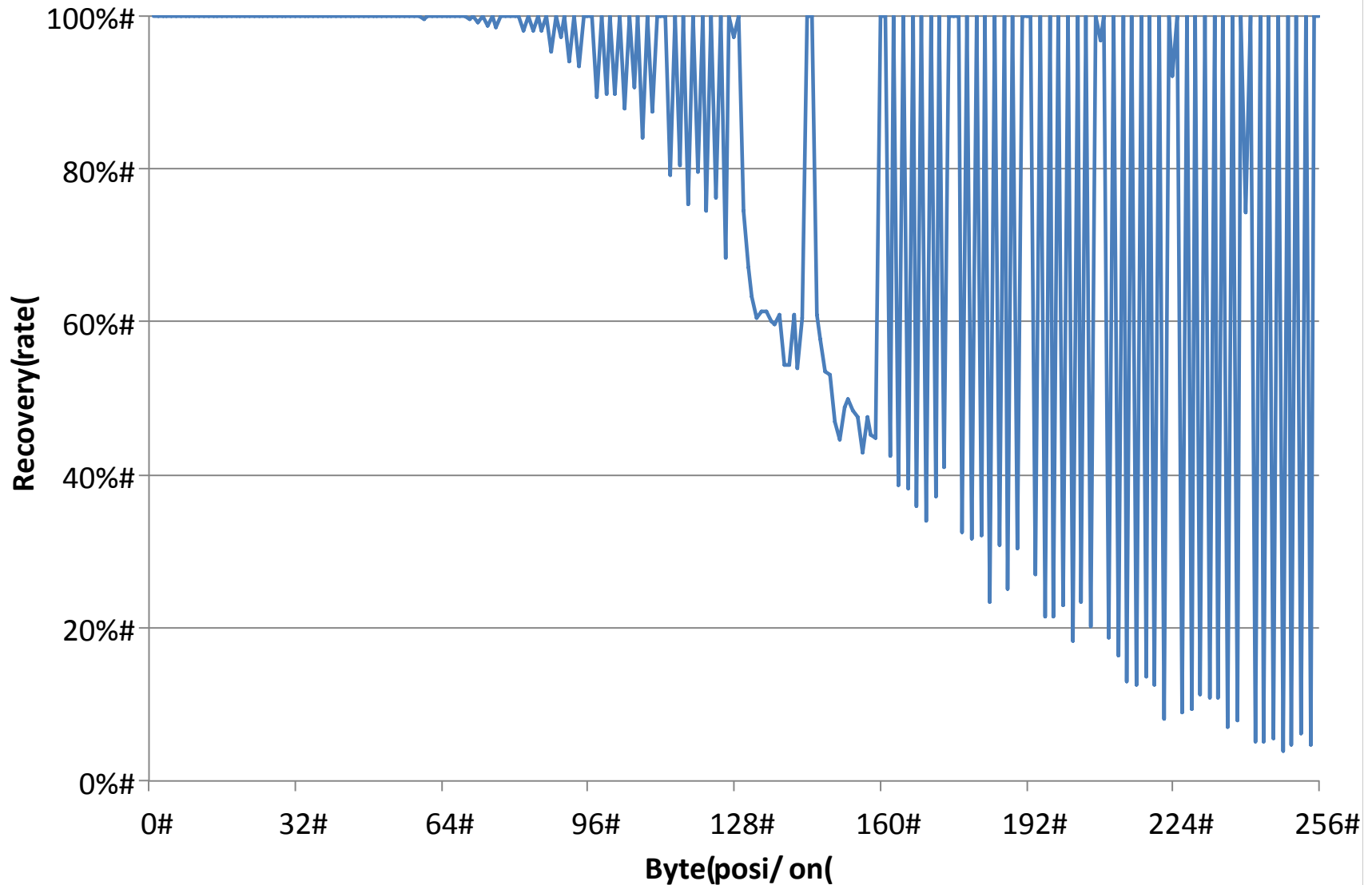
Plaintext recovery based on TSC_0 aggregation: 2^{24} frames



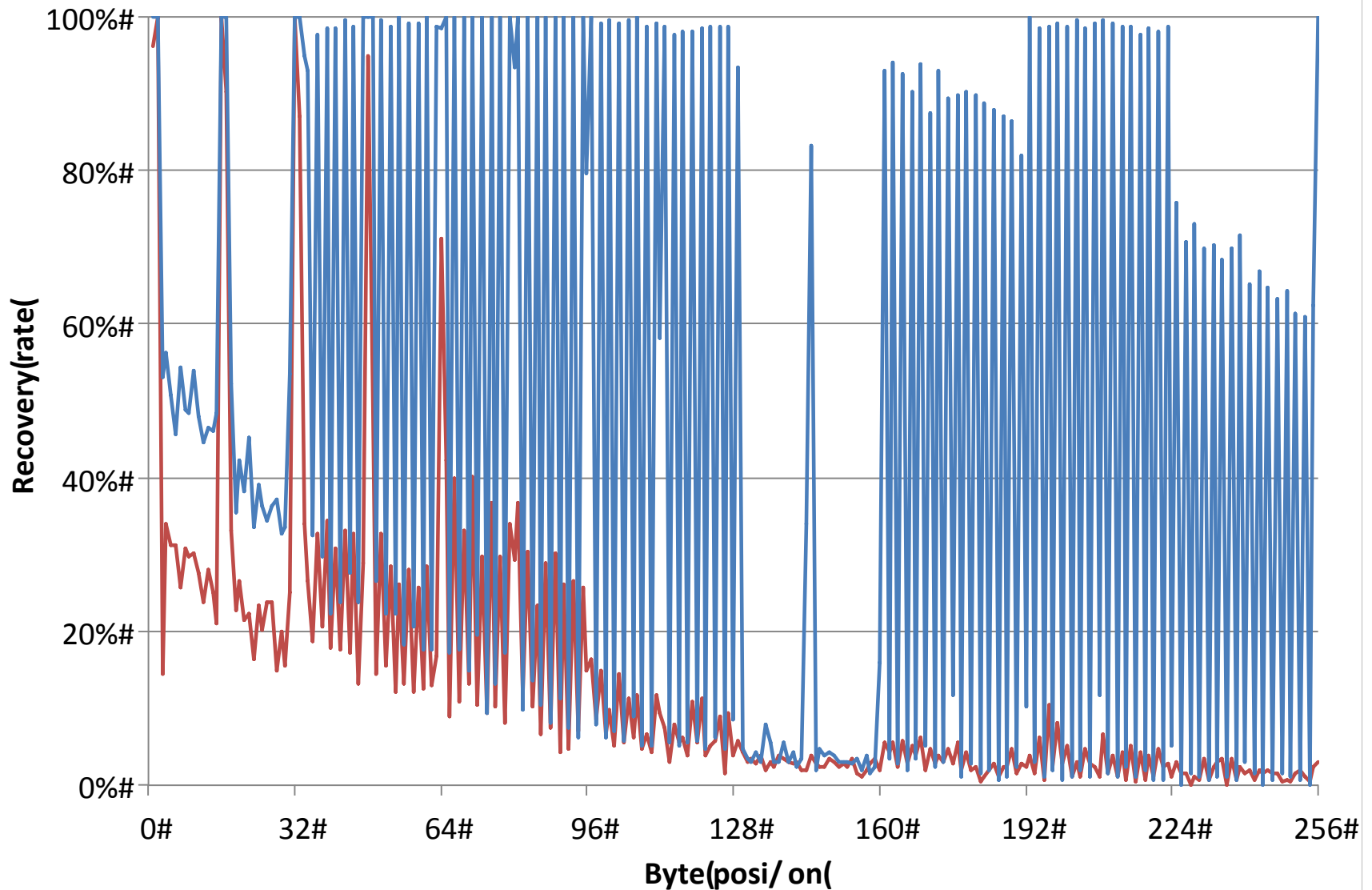
Plaintext recovery based on TSC_0 aggregation: 2^{26} frames



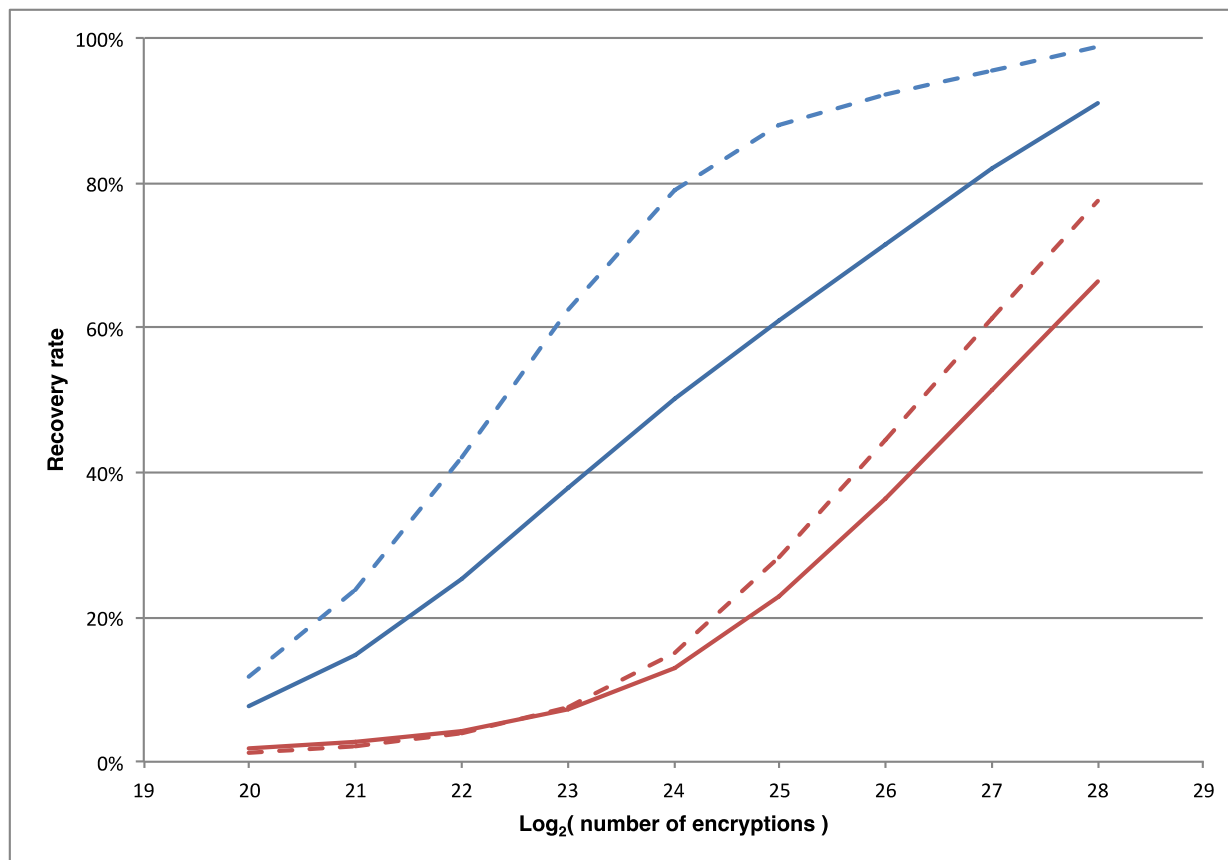
Plaintext recovery based on TSC_0 aggregation: 2^{28} frames



Plaintext recovery with TSC_0 aggregation (blue) compared to full aggregation (red): 2^{24} frames



Performance of Single-byte Plaintext Recovery Attacks



Red: basic attack (full aggregation); blue: TSC₀ aggregation.

Solid line: average over all 256 posns; dotted line: average over odd posns

Big-bias Hunting

- We then obtained a grant from the UK government enabling us to compute accurate per (TSC_0, TSC_1) keystream distributions.
- We used Amazon EC2 to carry out large-scale single-byte and double-byte keystream distribution computations.
 - For first 512 positions of keystreams in each case.
 - Single-byte: 2^{32} keystreams per (TSC_0, TSC_1) value, 2^{48} in total.
 - Double-byte: 2^{30} keystreams per (TSC_0, TSC_1) value, 2^{46} in total.
 - Total computation was about 63 virtual core years.
 - Approximately 5% of computation involved in RSA-768 sieving step.
 - (Or just 1% of the latest EPFL computations!)

Big-bias Hunting

- We exploited the inherent parallelism in the problem.
- We used, in both computations, 256 'c3.x8large' instances in parallel.
 - 8192 virtual cores, mapping onto 4096 Intel Xeon 2.8GHz processors.
 - Essentially, an entire Amazon EC2 data centre.
 - Boto (Python) + Ubuntu 13.10 + OpenSSL + careful cache optimisations.
 - Running cost: \$614 per hour (+ 20% tax).
 - Make very sure your code is correct before executing it!
 - Make sure to terminate instances as soon as computation is done!

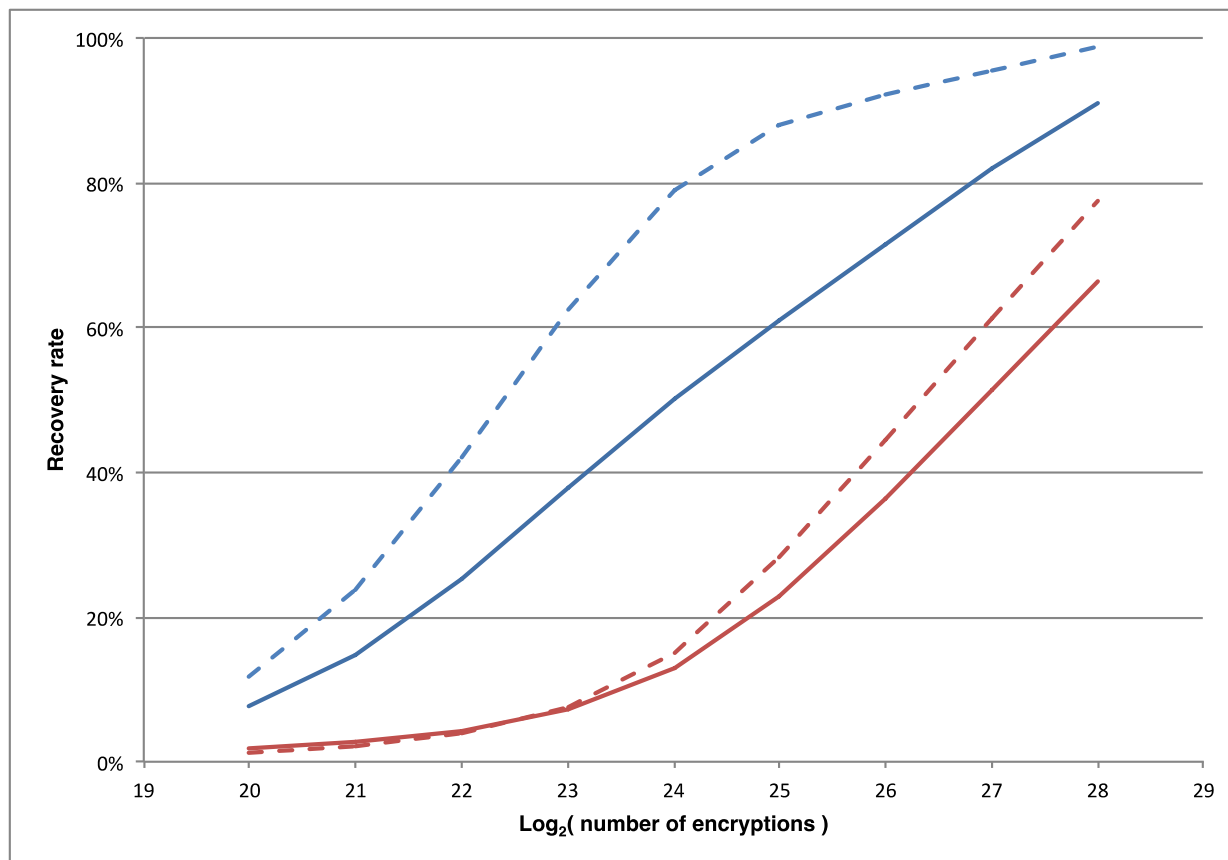
Big-bias Hunting: Single-Byte Computation

- Single-byte computation ran for 32 hours, or 30 virtual core years, and cost approximately \$20k.
- Produced dataset consisting of $2^{16} \times 2^9 \times 2^8$ 32-bit integers.
 - One counter per (TSC_0, TSC_1) , per position, and per keystream byte value.
 - 32GB of distribution data in total.

Big-bias Hunting: Double-Byte Computation

- Double-byte computation ran for 35 hours, or 33 virtual core years, and cost approximately \$23k.
- Produced dataset consisting of $2^{16} \times 2^9 \times 2^{16}$ 32-bit integers.
 - One counter per (TSC_0, TSC_1) , per position, and per *pair* of keystream byte values.
 - 8TB in total, storage cost of \$410 per month on EC2.
 - Data transfer to our local RAID was charged at \$0.12 per GB, \$983 in total.
 - We used bbcp tool developed by experimental physicists; 48 hours at 50MB per second.

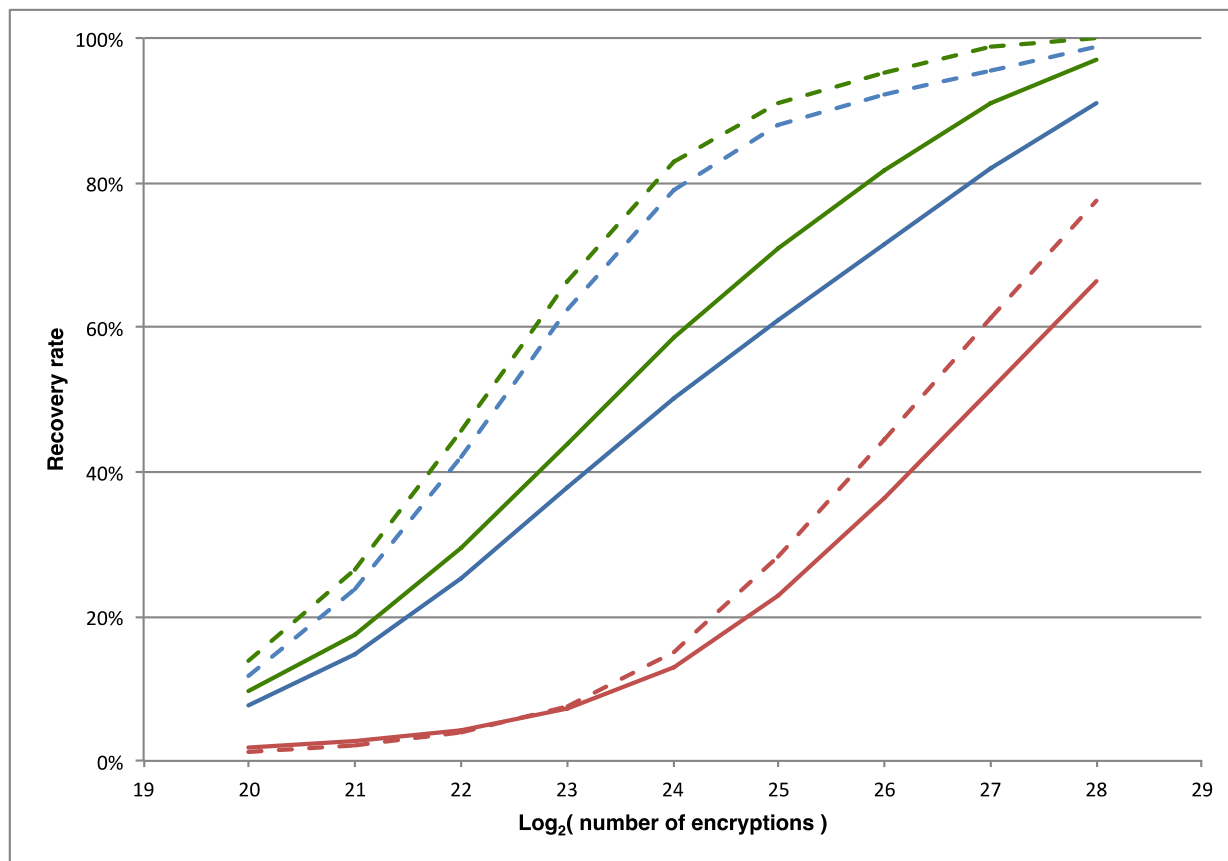
Performance of Single-byte Plaintext Recovery Attacks



Red: basic attack (full aggregation); blue: TSC₀ aggregation.

Solid line: average over all 256 posns; dotted line: average over odd posns

Performance of Single-byte Plaintext Recovery Attacks



Red: basic attack (full aggregation); blue: TSC_0 aggregation; green: no aggregation.
Solid line: average over all 256 posns; dotted line: average over odd posns

WPA/TKIP Closing Remarks

- Plaintext recovery for WPA/TKIP is possible for the first 256 bytes of frames, provided sufficiently many independent encryptions of the same plaintext are available.
- Security is far below the level implied by the 128-bit key TK.
- Suitable targets for attack might include fixed but unknown fields in encapsulated protocol headers.
- Targeting HTTP traffic via client-side Javascript also possible, as in TLS attacks.



Concluding Remarks

Concluding Remarks

- RC₄ is still widely used for performance and legacy reasons.
- Don't underestimate the power of inertia.

- Broadcast attacks can be made practical.
- Interesting statistical questions arise.

- As a community, we should work more on making our attacks practical and maximising their real-world impact.
- This requires understanding of and interaction with that real world!

“Thank You”

My thanks to:

- The Asiacrypt 2014 program chairs and general chairs for the invitation and for their hospitality.
- My co-authors Bertram Poettering and Jacob Schuldt.
- My additional co-authors on [ABPPS13] – Nadhem AlFardan and Dan Bernstein.
- Martin Albrecht, Jon Hart and Adrian Thomas at RHUL for their assistance with sourcing, building and maintaining our local computing infrastructure and for help in managing AWS.
- EPSRC and the UK government for funding our adventures in Amazonia.