Royal Holloway
University of London

On the Security of RC4 in TLS

Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering, Jacob Schuldt

Royal Holloway, University of London
University of Illinois at Chicago

http://www.isg.rhul.ac.uk/tls/

Information Security Group

UIC

Agenda

- Brief overview of TLS and use of RC4
- Analysis of RC4
- Two attacks against RC4 in TLS
  - Single-byte attack
  - Double-byte attack
- Conclusions

TLS

- TLS = Transport Layer Security
  - Security goal: provide confidential and authenticated channel between client and server

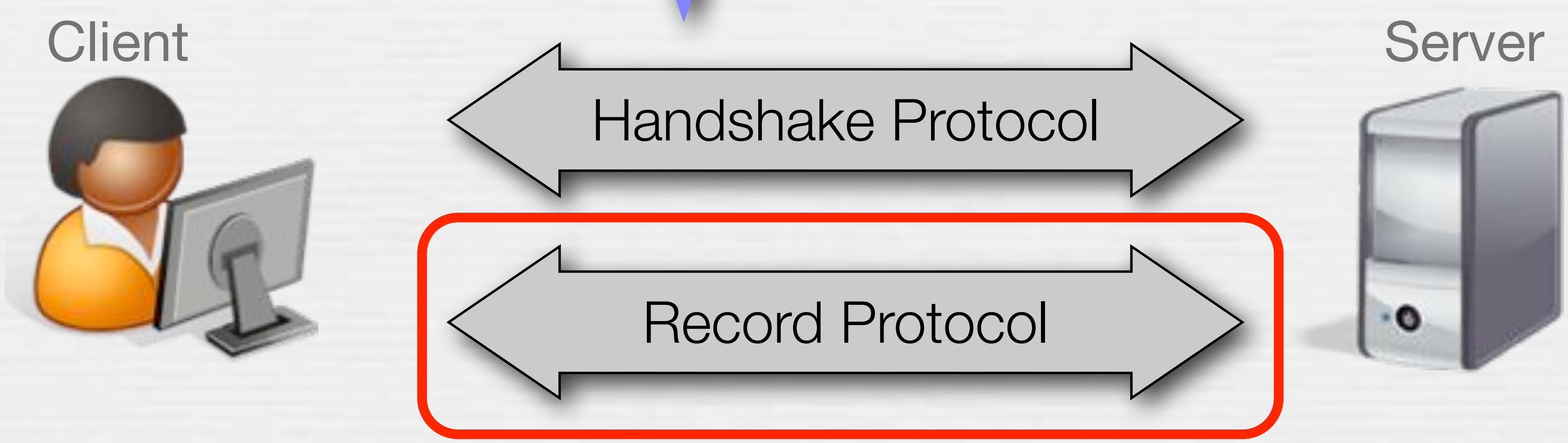
- Applications of TLS are ubiqutous
  - Secure websites (https://), secure e-mail (IMAP/TLS, POP/TLS, SMPT/TLS), mobile application, etc.

Brief History of TLS

- Started life as Secure Socket Layer (SSL) protocol
  - Developed at Netscape ~1994
  - SSL v3 (1996) still widely supported
- TLS = IETF standardization of SSL
  - TLS v1.0 in RFC 2246 (1999)
    - Based on SSL v3 but not compatible
  - TLS v1.1 in RFC 4346 (2006)
  - TLS v1.2 in RFC 5246 (2008)

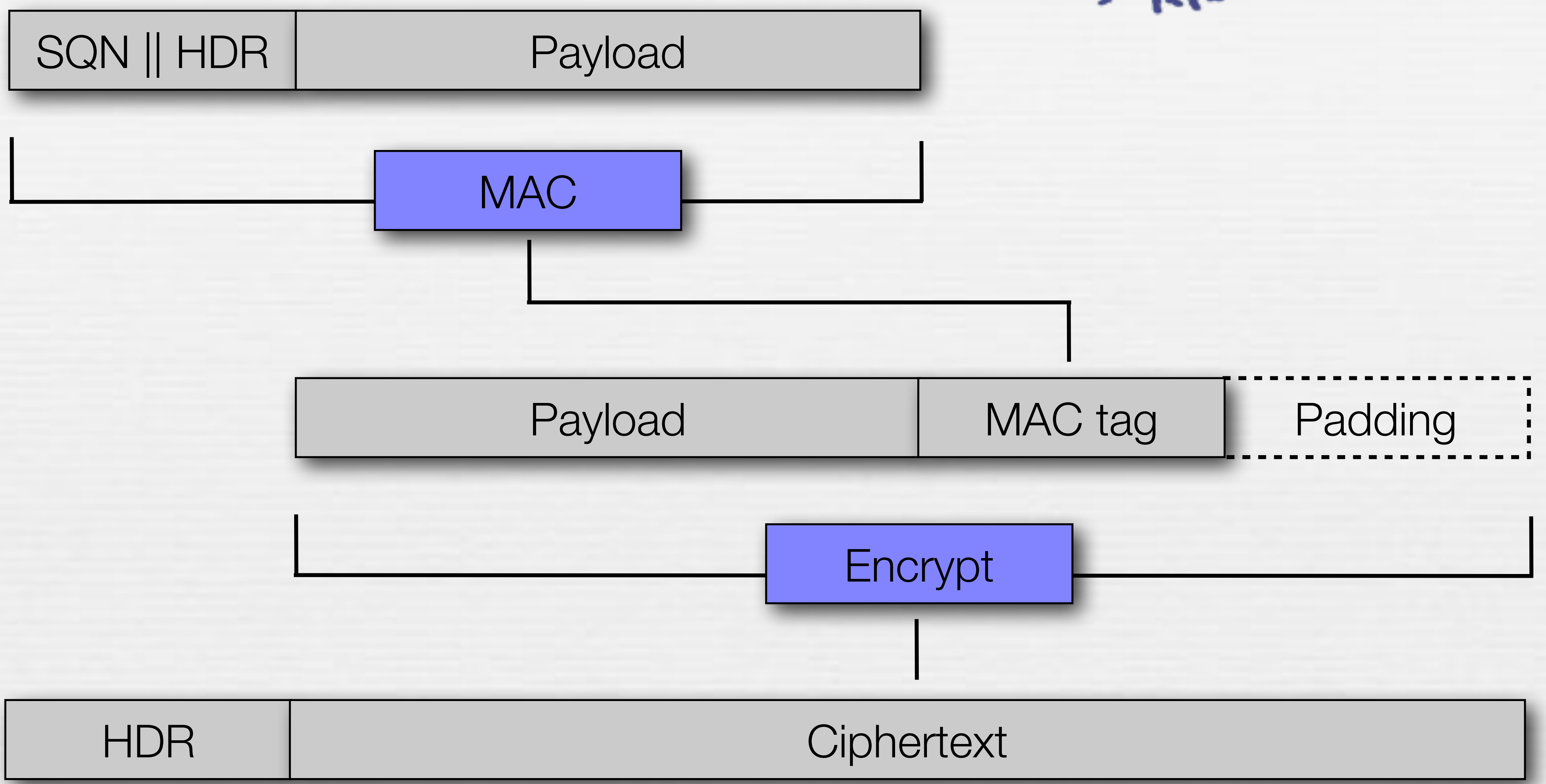
Simplified View of TLS

Used by client and server to
1. Negotiate ciphersuite
2. Authenticate
3. Establish keys used in the Record Protocol



Provides confidentiality and authenticity of application layer data using keys from Handshake Protocol

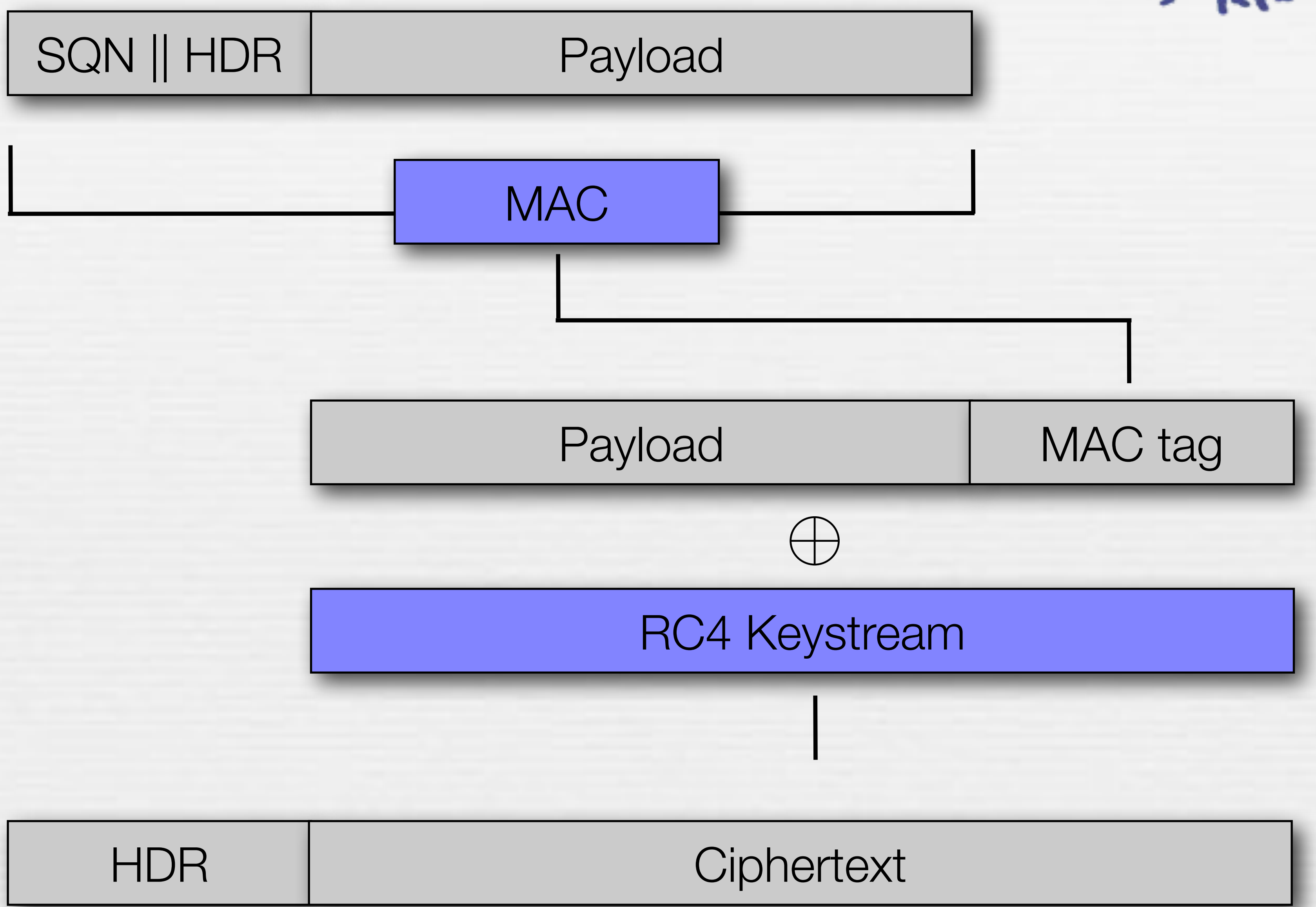
TLS Record Protocol: MAC-Encode-Encrypt



MAC — HMAC-MD5, HMAC-SHA1, HMAC-SHA256

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

TLS Record Protocol: RC4-128



TLS Record Protocol: RC4-128

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**
  - $j \leftarrow j + \mathcal{S}[i] + K[i \bmod keylen] \bmod 256$
  - $\mathrm{swap}(\mathcal{S}[i], \mathcal{S}[j])$
- **end**
- $i, j \leftarrow 0$

end

RC4 Keystream generation

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

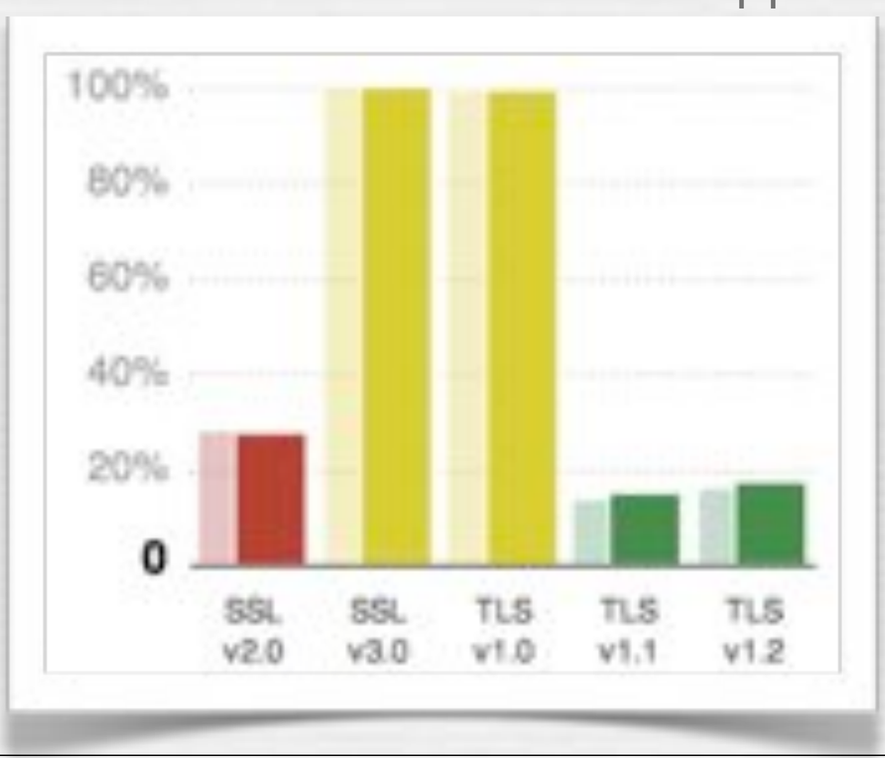
end

TLS Record Protocol: Authenticated Encryption

- TLS 1.2 additionally supports authenticated encryption
  - AES-GCM in RFC 5288
  - AES-CCM in RFC 6655

- However, TLS 1.2 is not widely supported

SSL Pulse: Webserver TLS support



Browser TLS support (out-of-the-box)

TLS v1.1

TLS v1.1

TLS v1.0

TLS v1.0

TLS v1.0

Use of RC4 in TLS

Information Security Group

- Recent attacks on CBC-based ciphersuites in TLS:
  - BEAST attack, Lucky 13
- In face of these, switching to RC4 has been a recommended mitigation approach (e.g. Qualys, F5)
- Use of RC4 in the wild:

ICSI Certificate Notary

Recent survey of 16 billion TLS connections:
Approx. 50% protected via RC4 ciphersuites

- Problem: RC4 is known to have statistical weaknesses

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 $3 \leq r \leq 255$

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

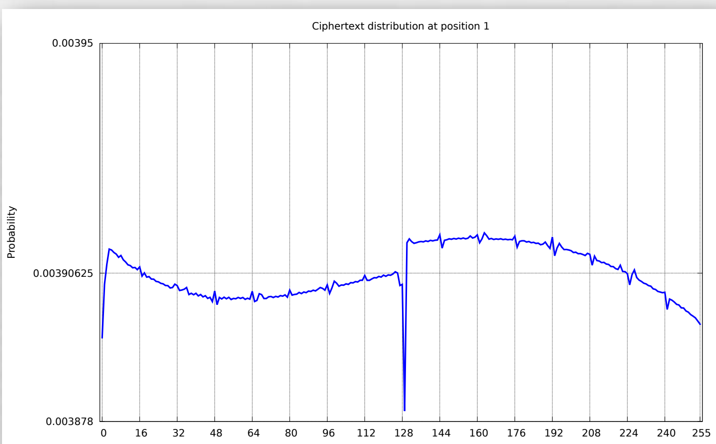
- [Sen Gupta-Maitra-Paul-Sakar 2011]:

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

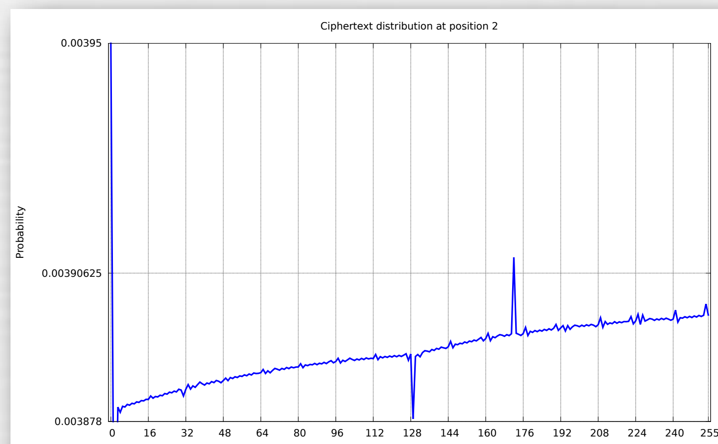
l = keylength

Complete Keystream Byte Distributions

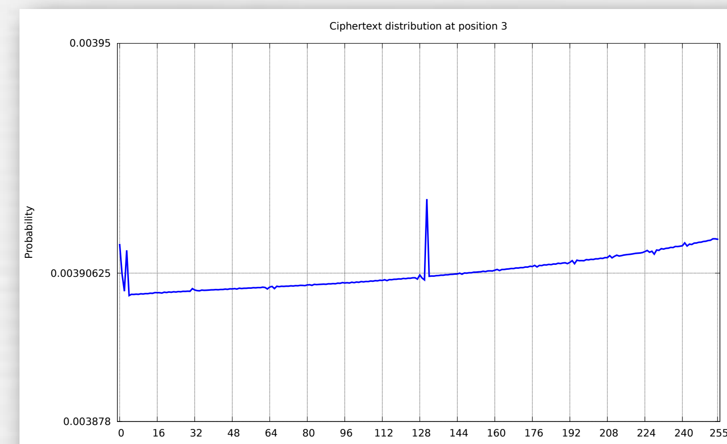
- Our approach
  - Based on the output from 2^{44} random independent 128 bit RC4 keys, estimate the keystream byte distribution of the first 256 bytes



Z_1



Z_2

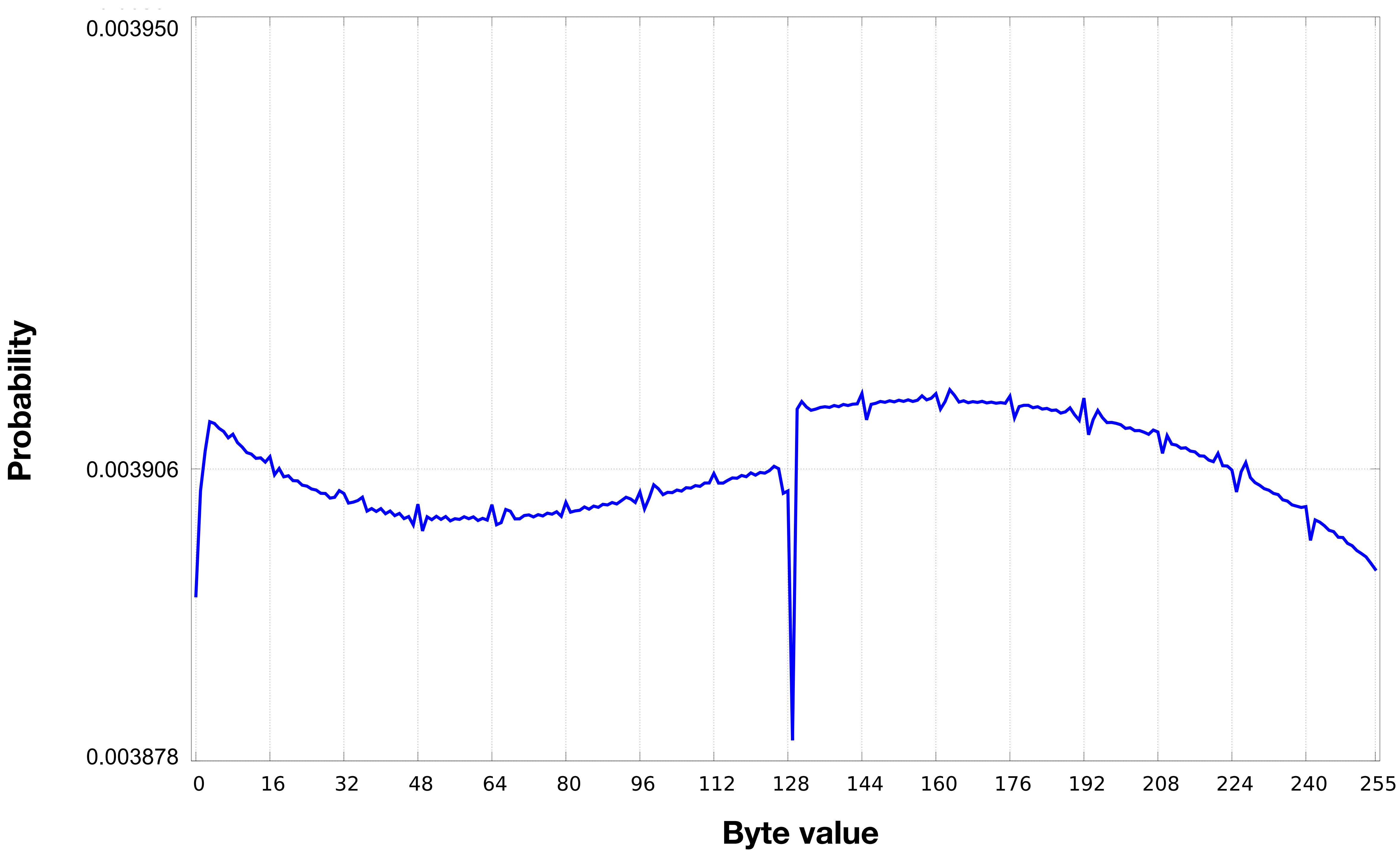


Z_3

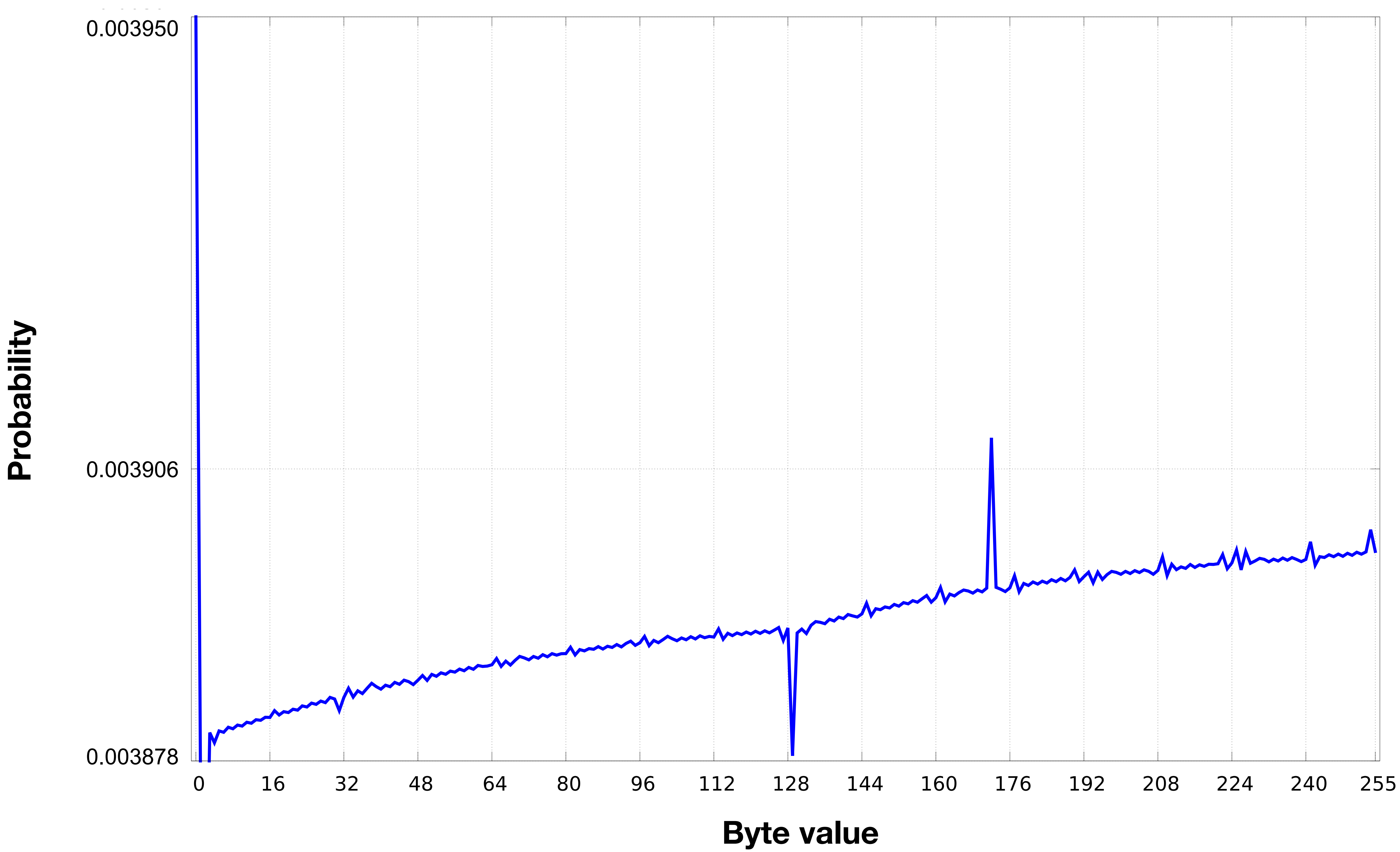
...

- Revealed many new biases in the RC4 keystream
  - (Some of these were independently discovered by [Isobe et al. 2013])

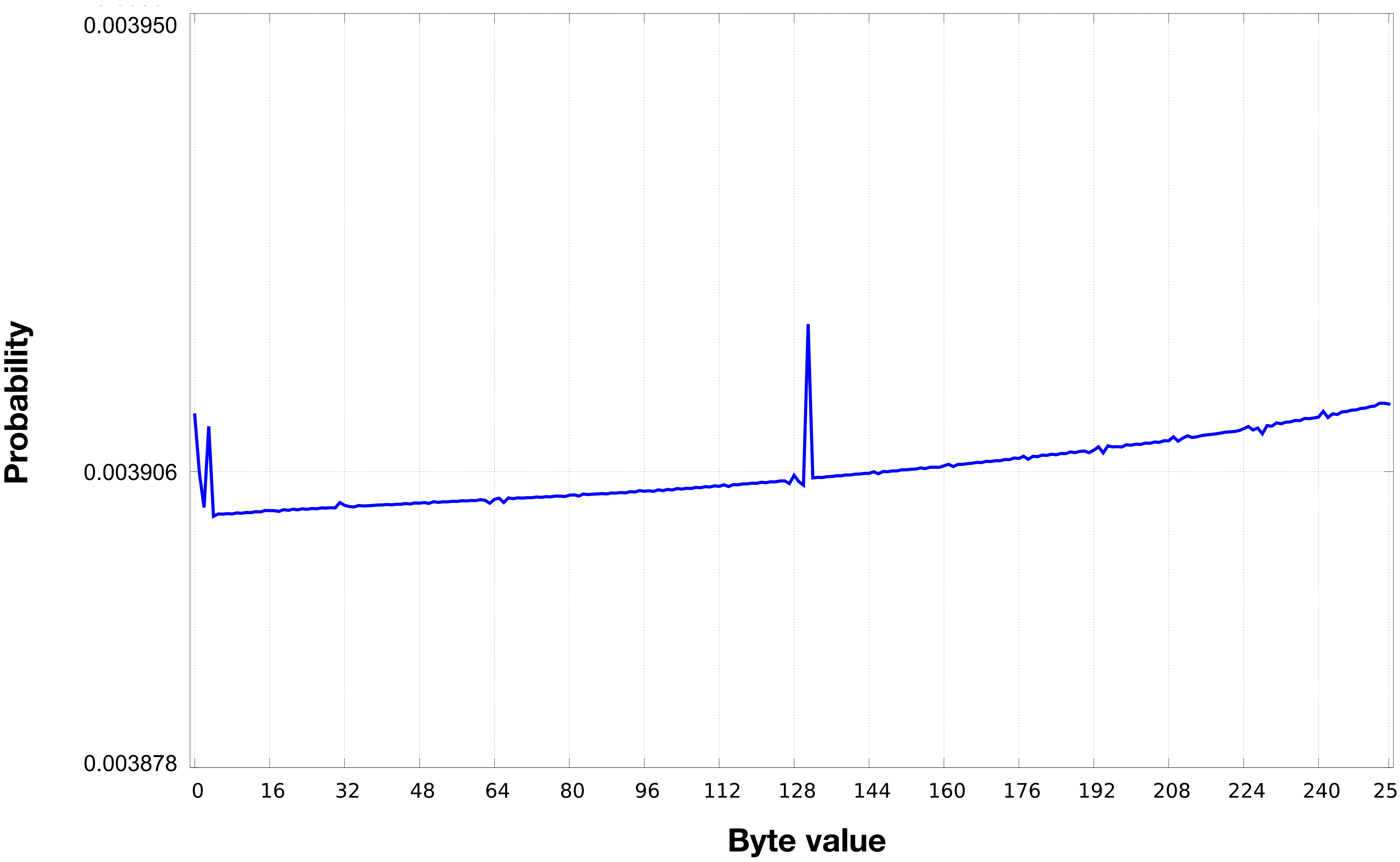
Keystream Distribution at Position 1



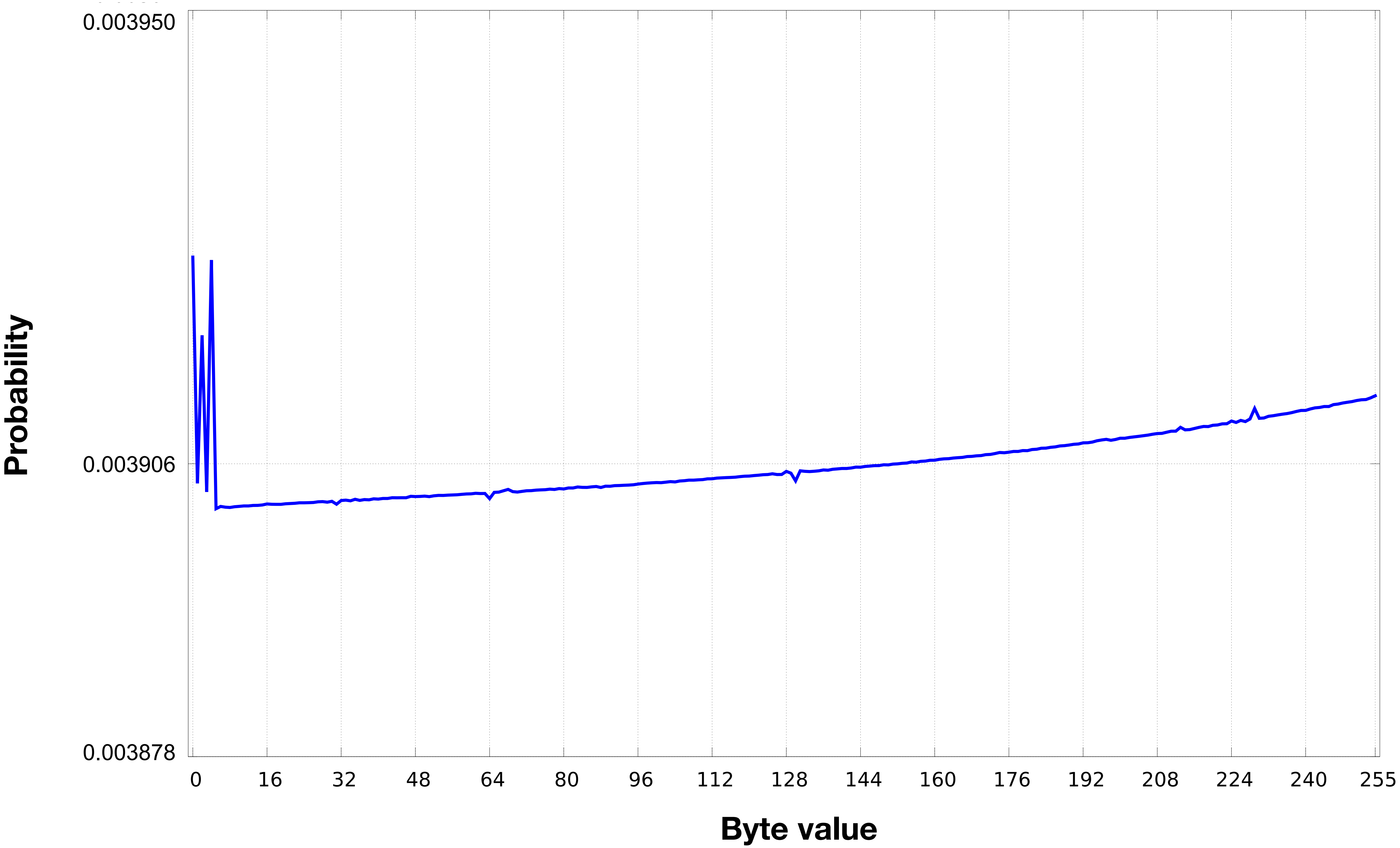
Keystream Distribution at Position 2



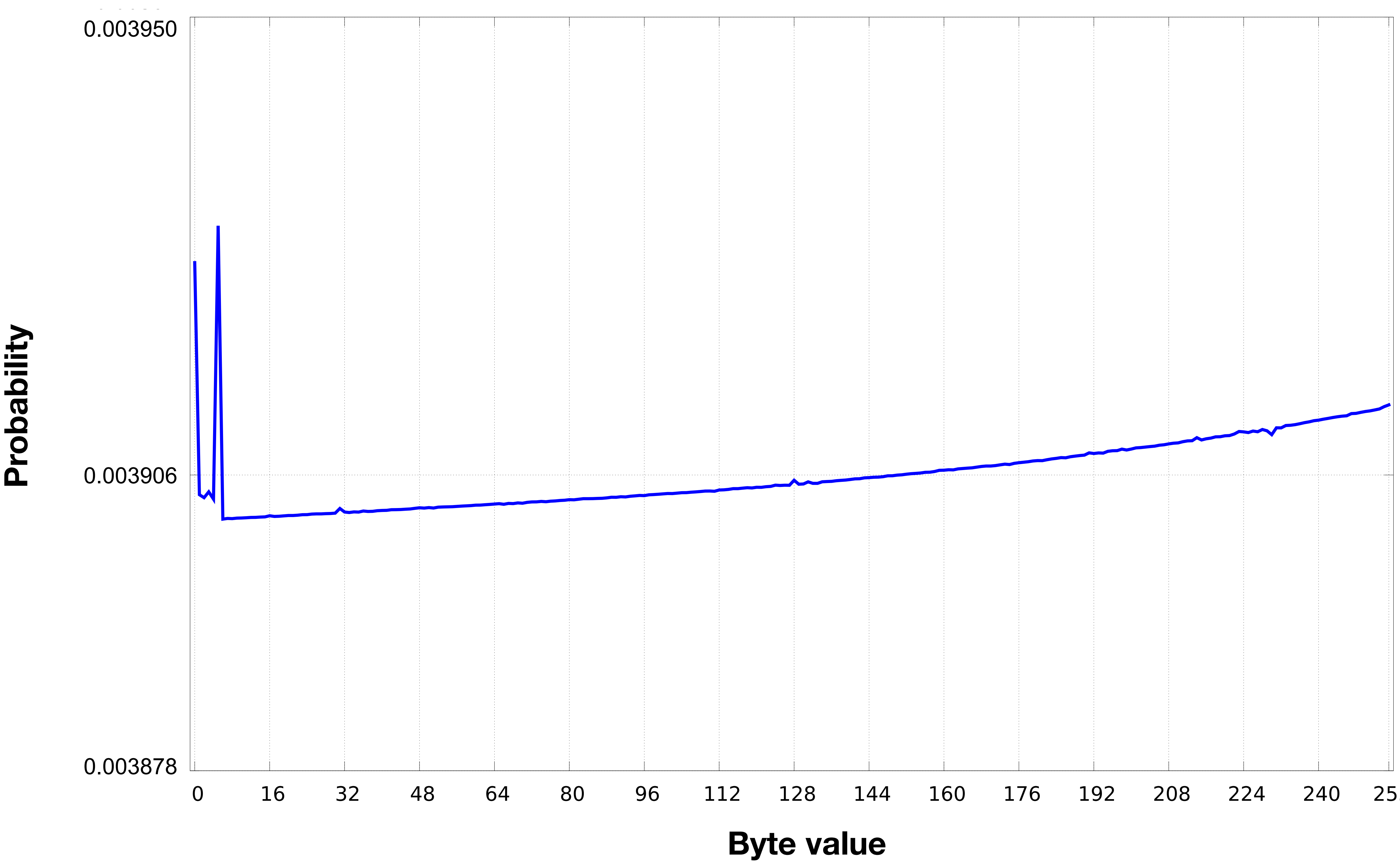
Keystream Distribution at Position 3



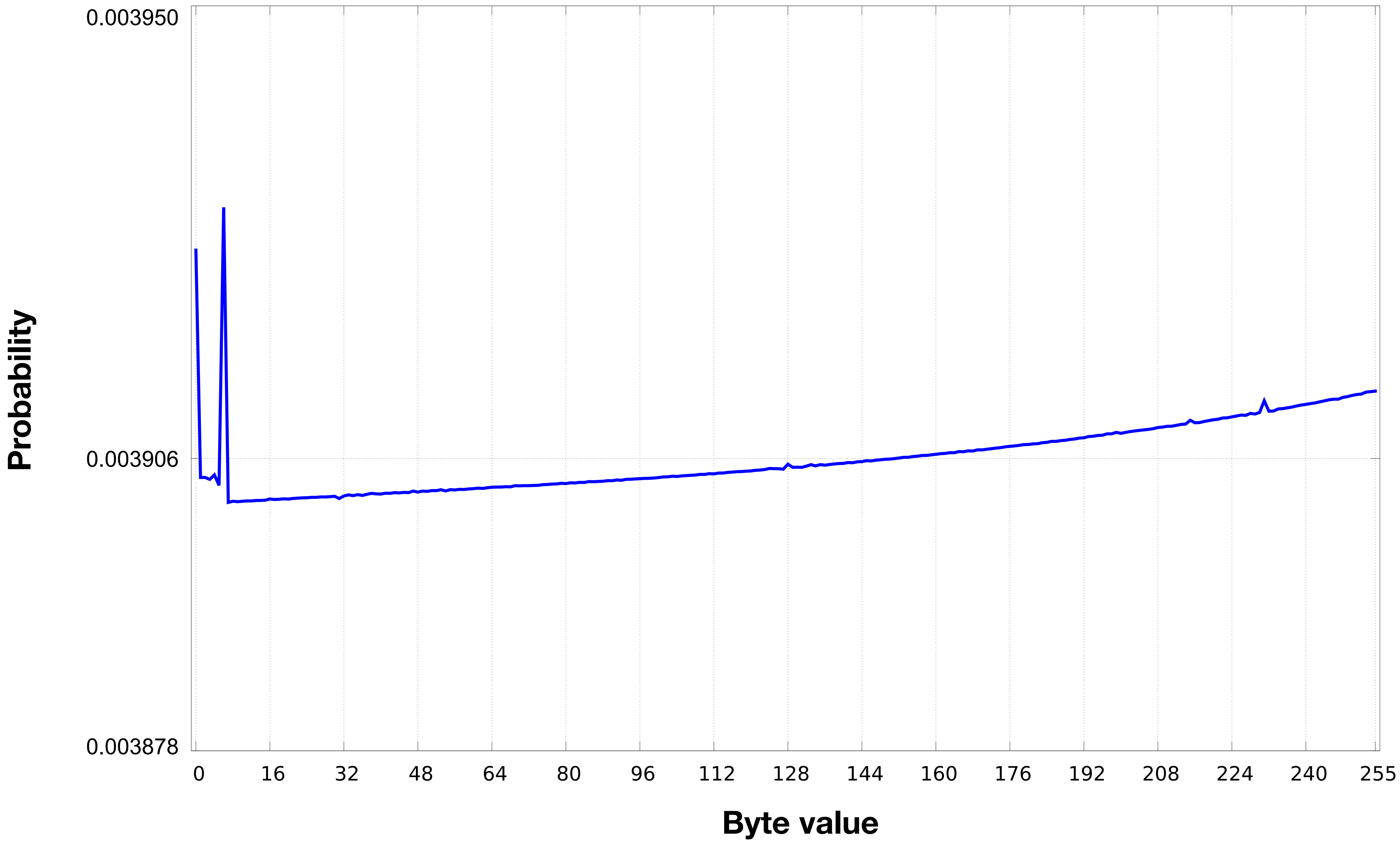
Keystream Distribution at Position 4



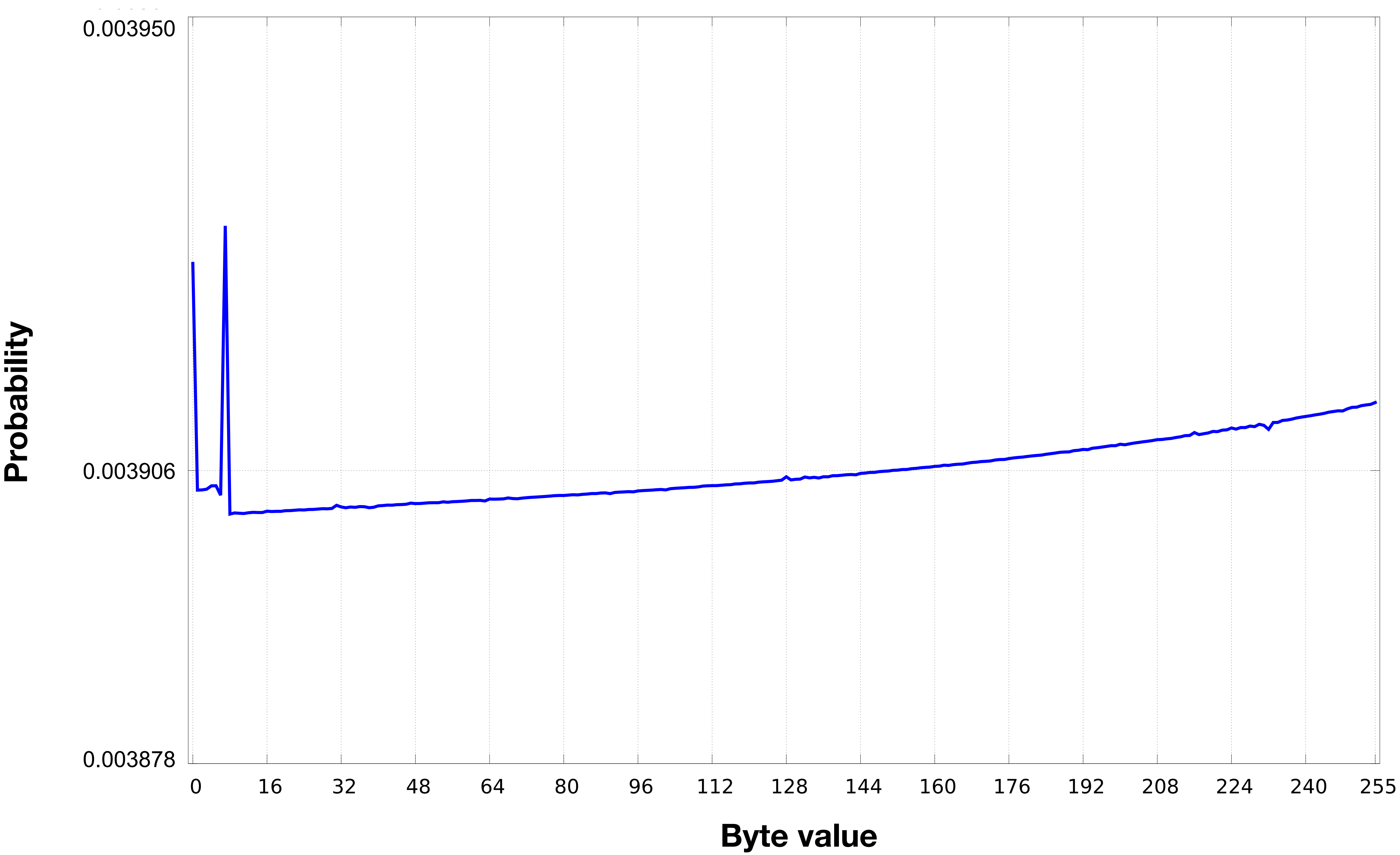
Keystream Distribution at Position 5



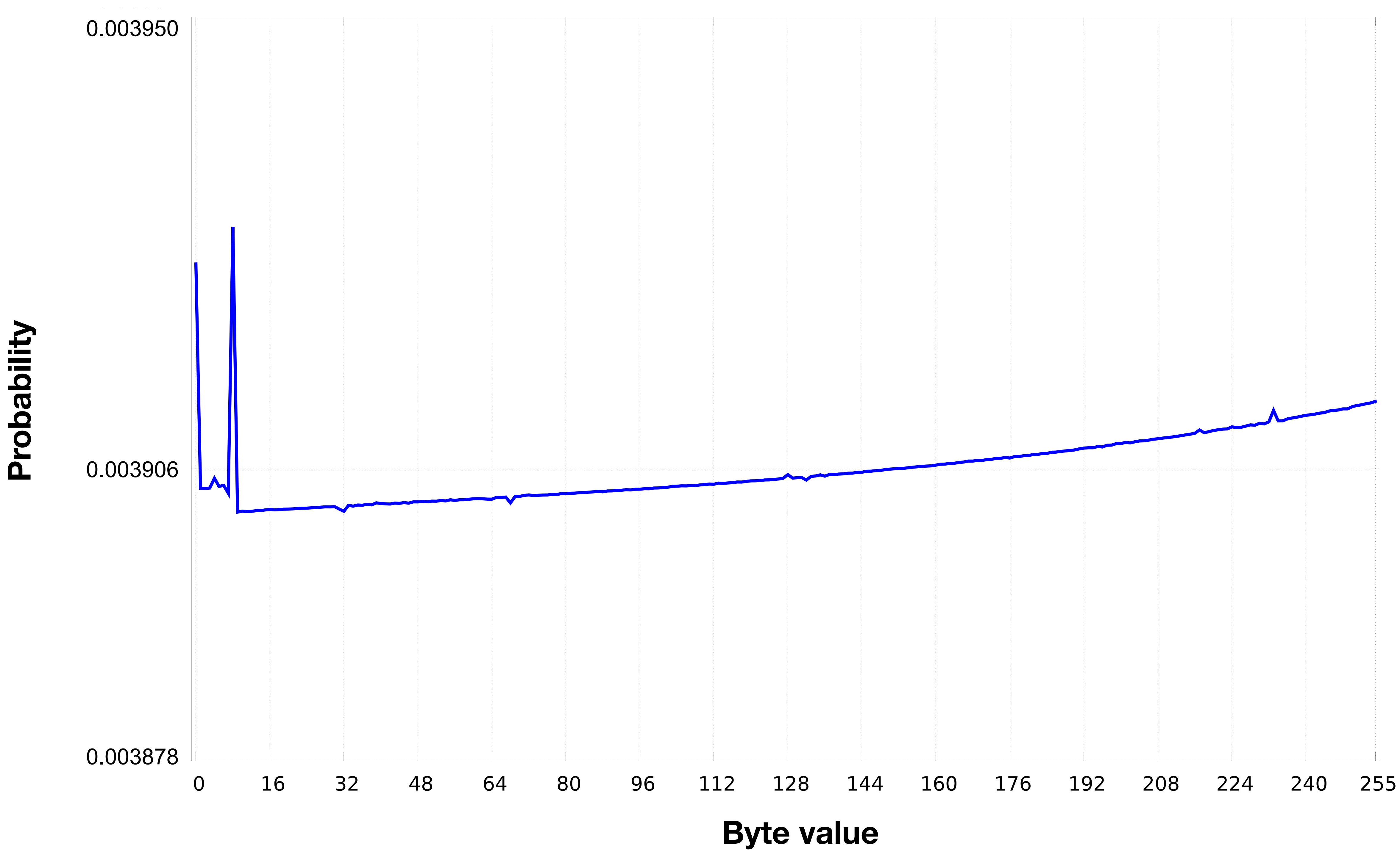
Keystream Distribution at Position 6



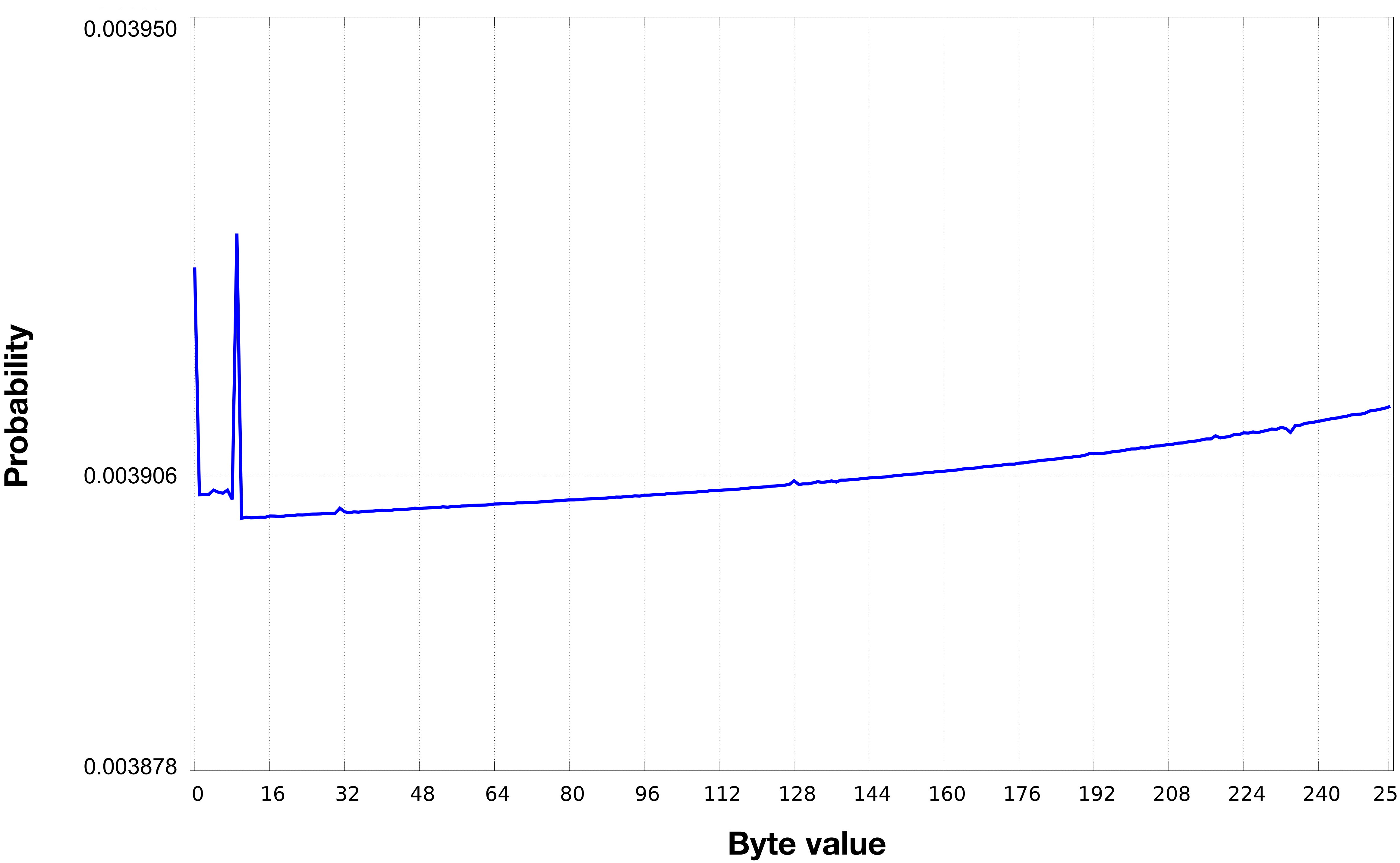
Keystream Distribution at Position 7



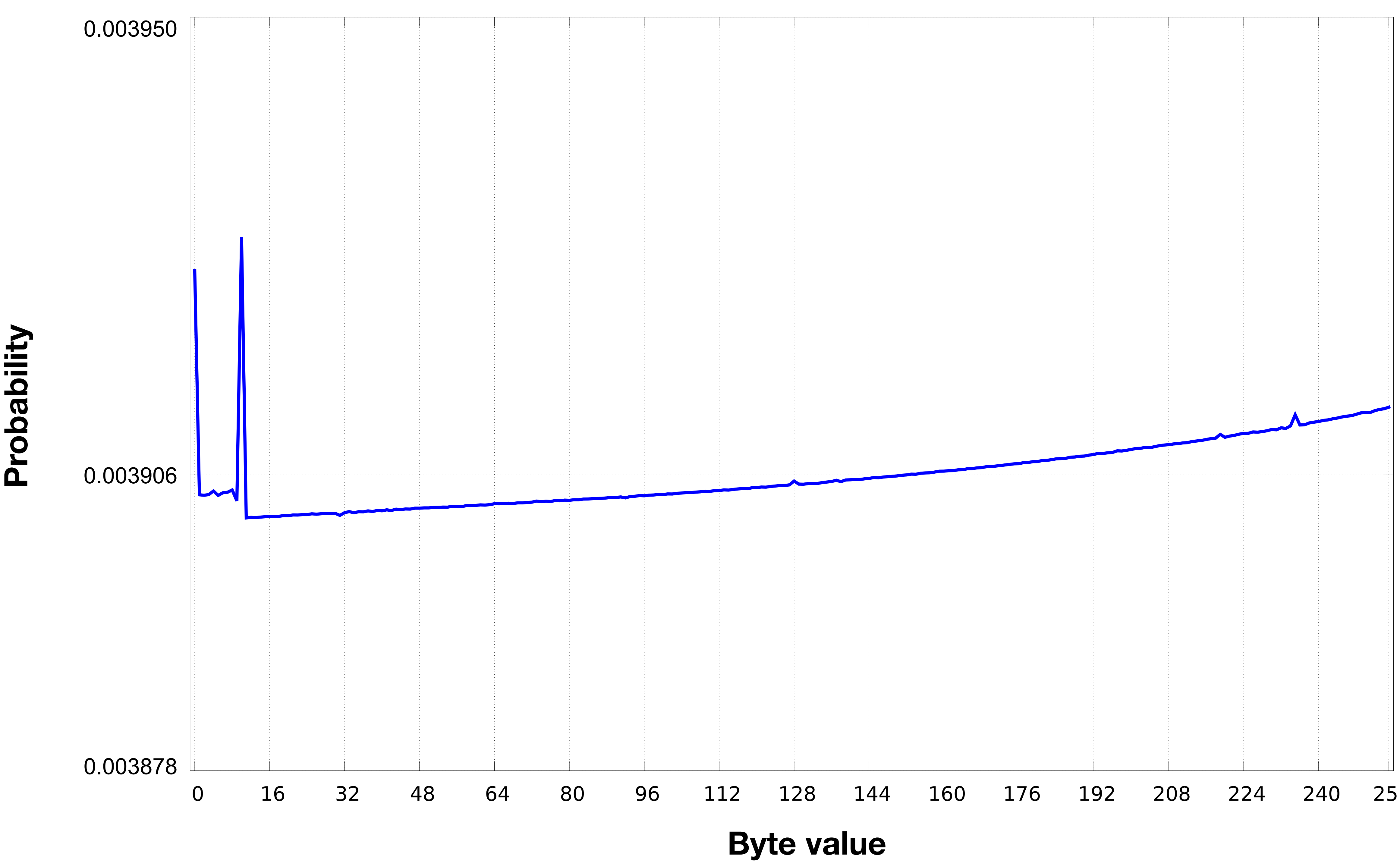
Keystream Distribution at Position 8



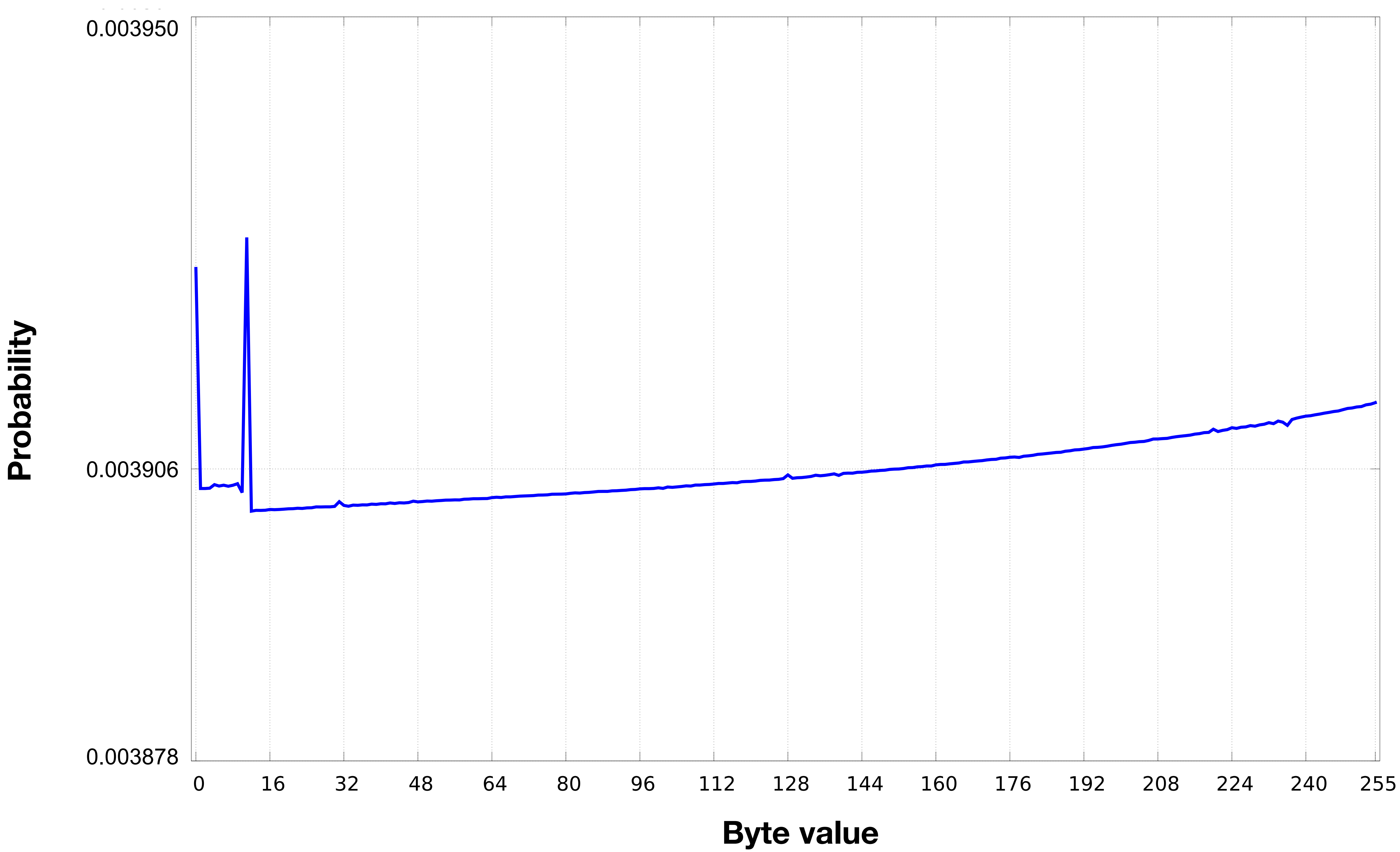
Keystream Distribution at Position 9



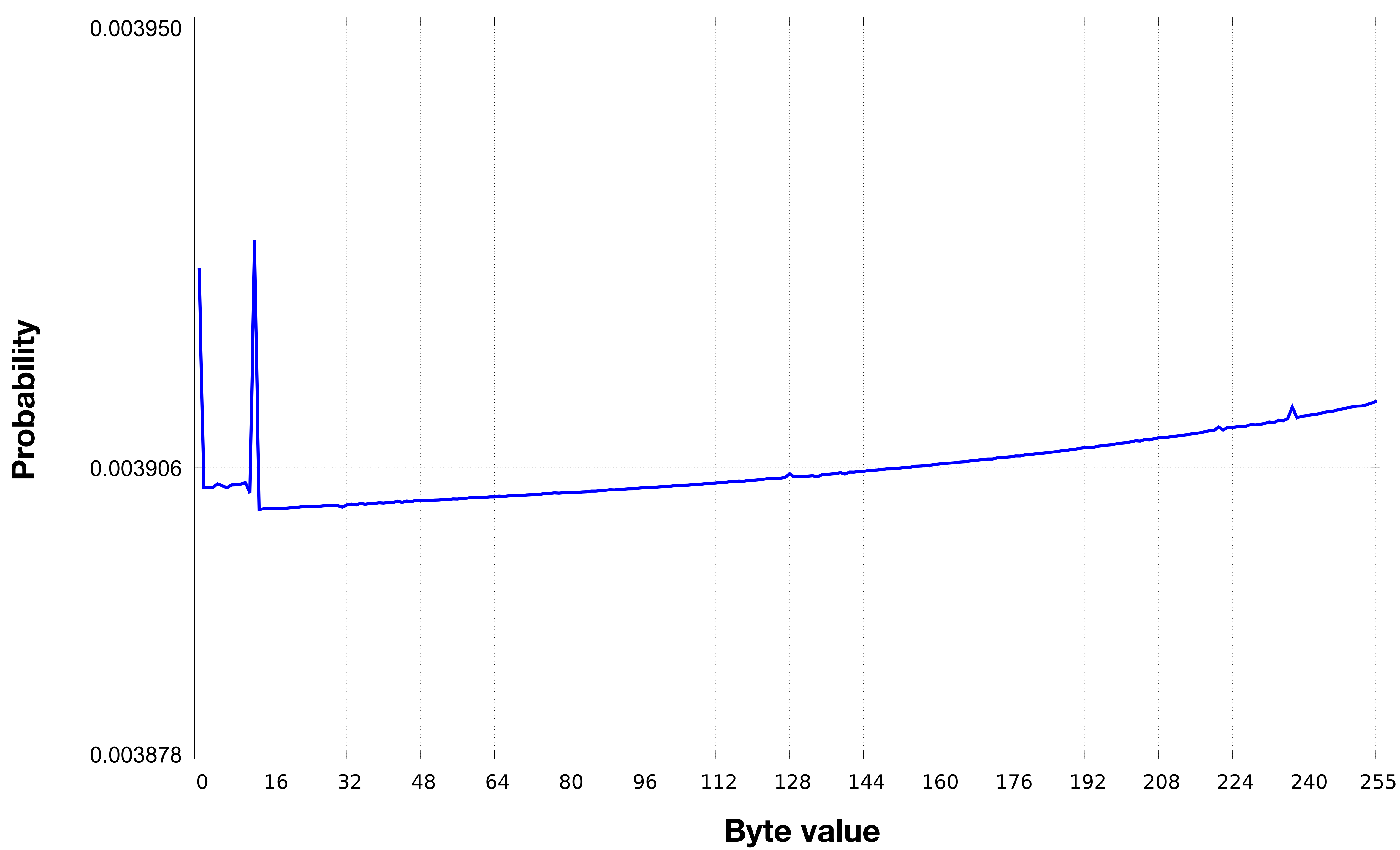
Keystream Distribution at Position 10



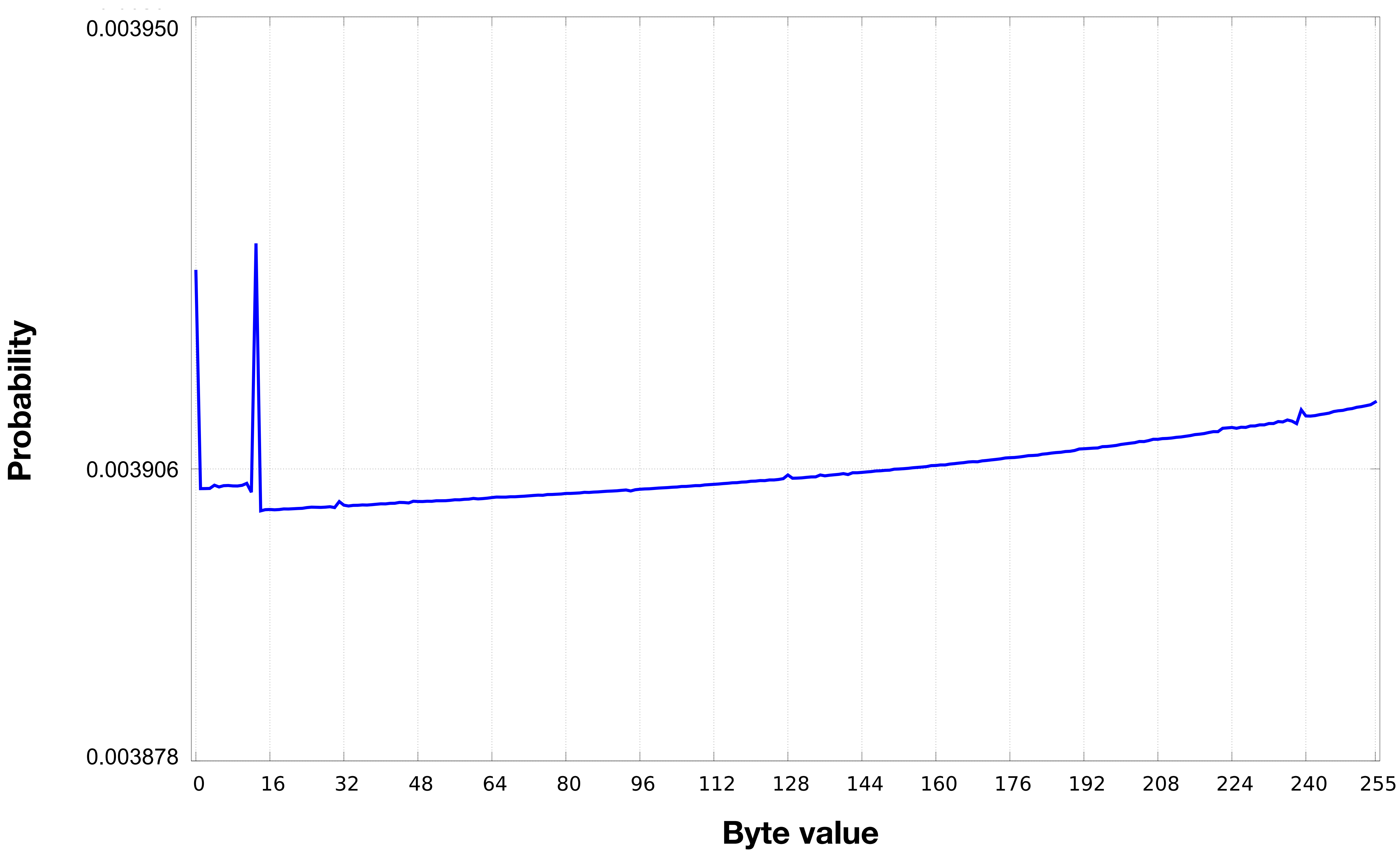
Keystream Distribution at Position 11



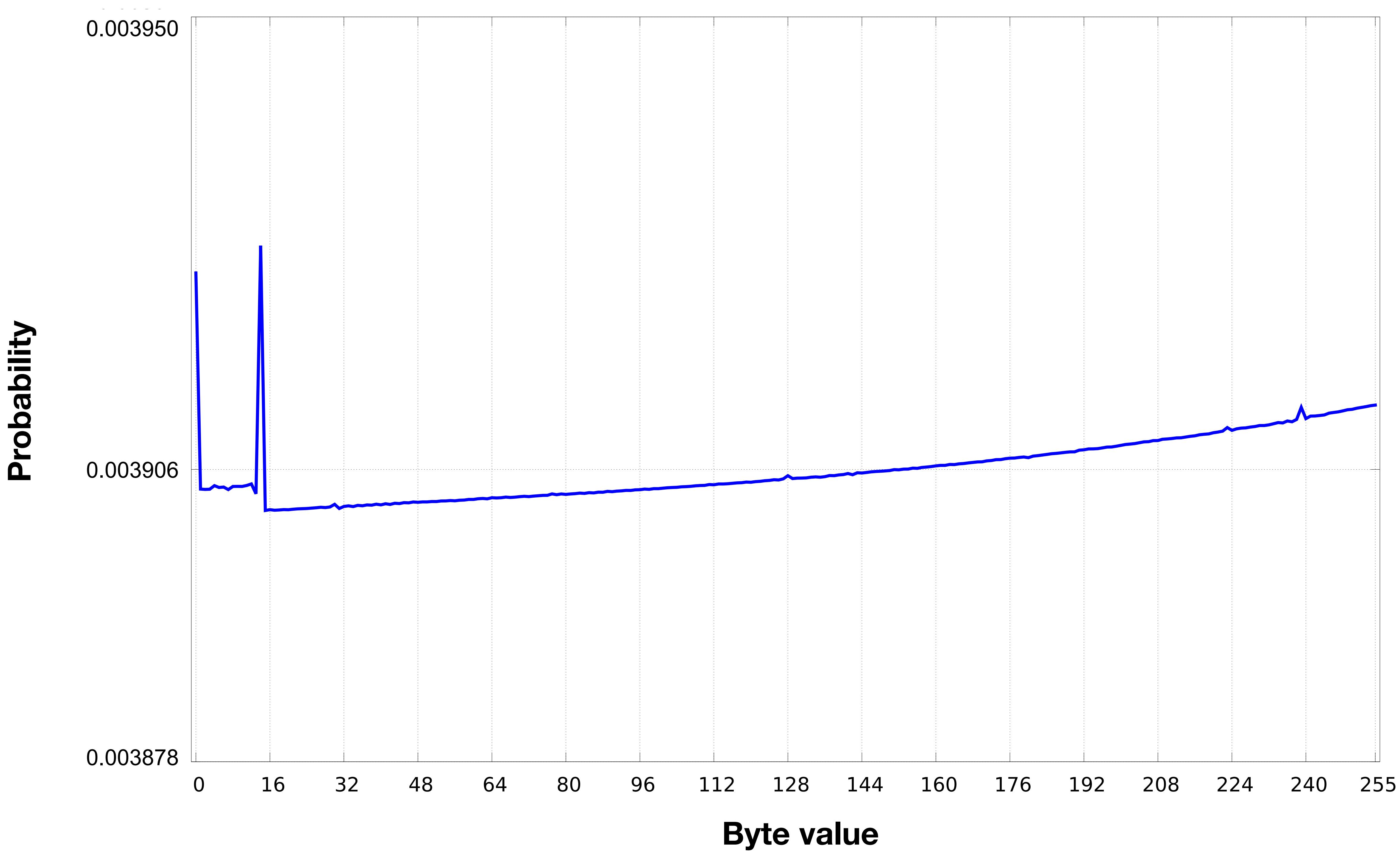
Keystream Distribution at Position 12



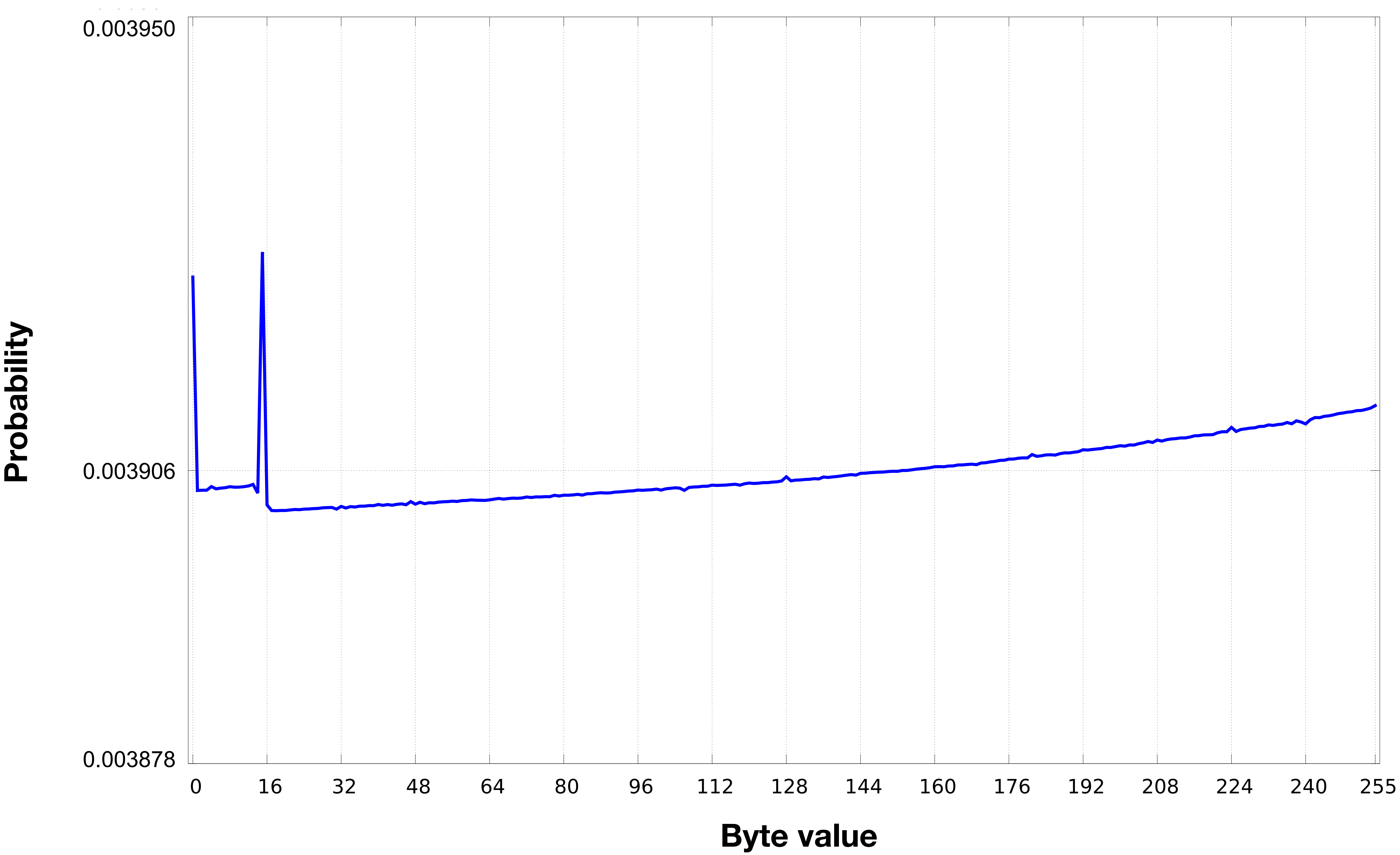
Keystream Distribution at Position 13



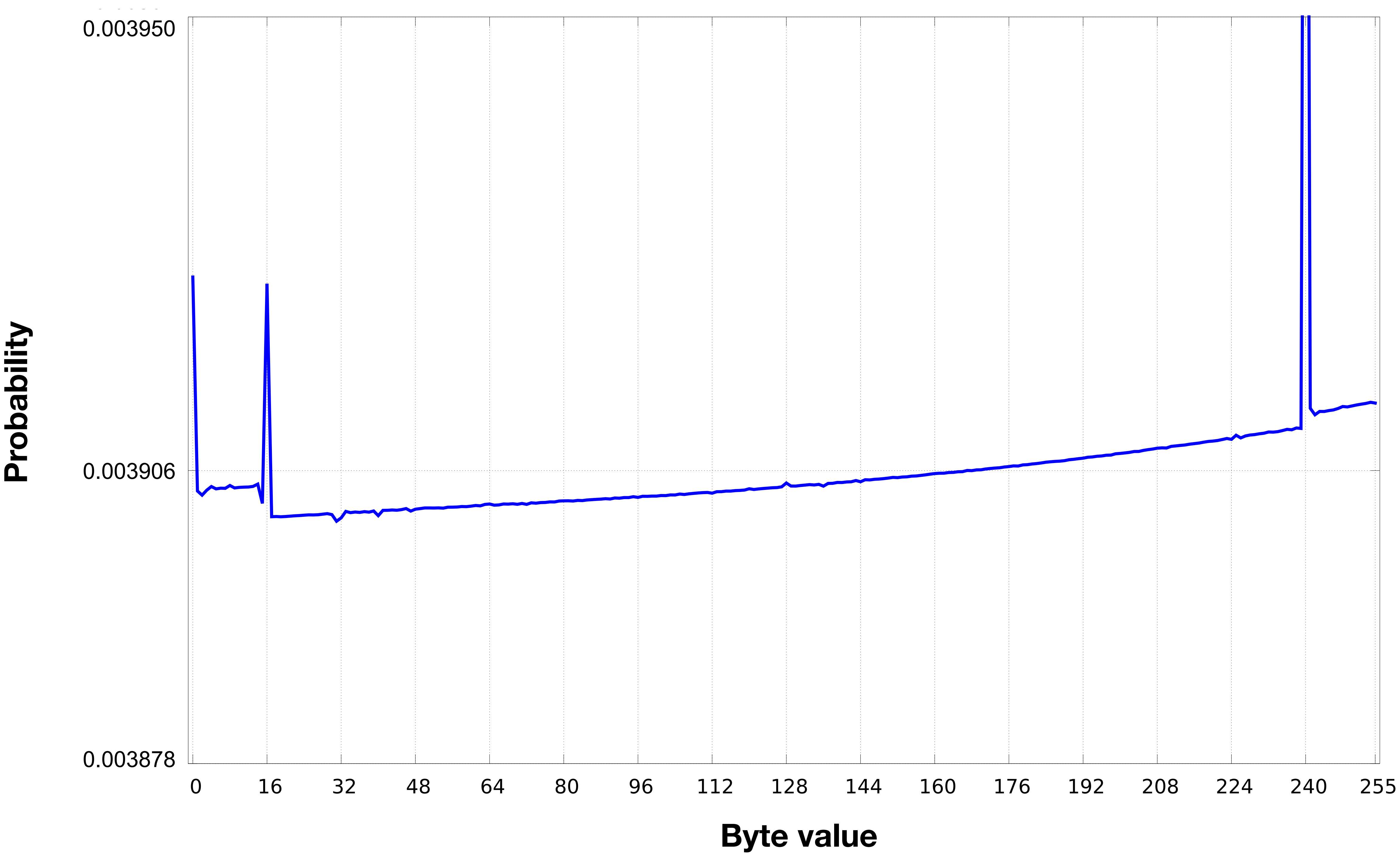
Keystream Distribution at Position 14



Keystream Distribution at Position 15



Keystream Distribution at Position 16

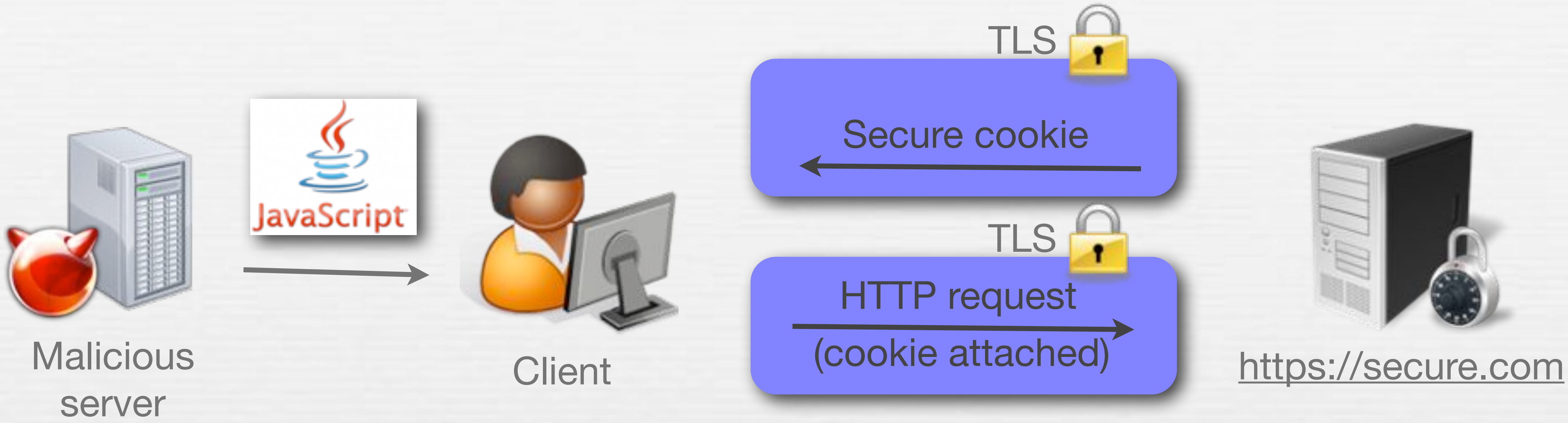


Plaintext Recovery

- Based on the keystream byte distribution, we can construct a plaintext recovery attack
  - Exploits all single-byte biases in the initial part of the RC4 keystream

- Attack requires the same plaintext to be encrypted under many different keys
  - Applicable when using TLS?

Targeting Secure HTTP Cookies



- Javascript
  - Uses XMLHttpRequest objects to generate POST requests
  - Request to secure site possible due to Cross-Origin Resource Sharing
  - Number of requests generated by script must be balanced to avoid browser overload

Plaintext Recovery

Information Security Group

Encryptions of plaintext under different keys

Plaintext candidate byte P_r

r

C_1

C_2

C_3

⋮

C_n

$P_r \oplus$

$P_r \oplus$

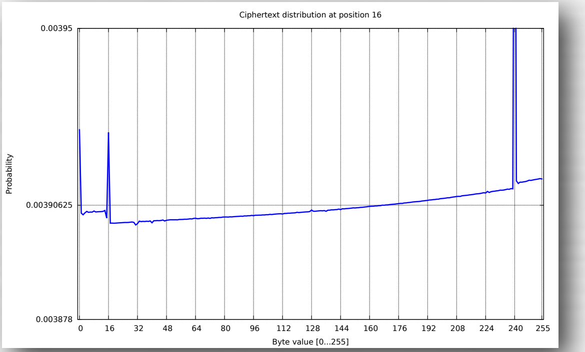
$P_r \oplus$

⋮

$P_r \oplus$

Induced distribution on Z_r

combine with

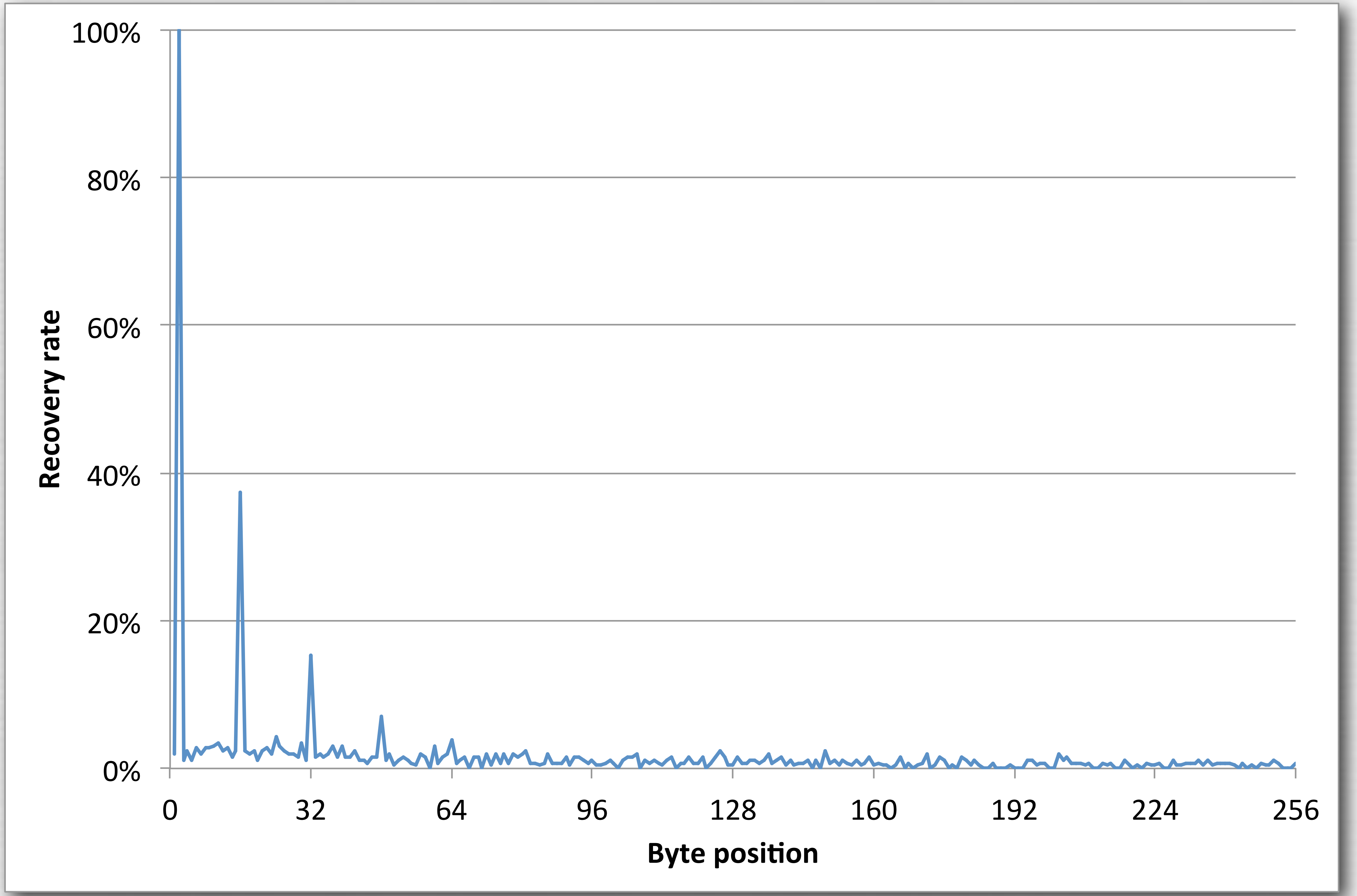


$\Downarrow$

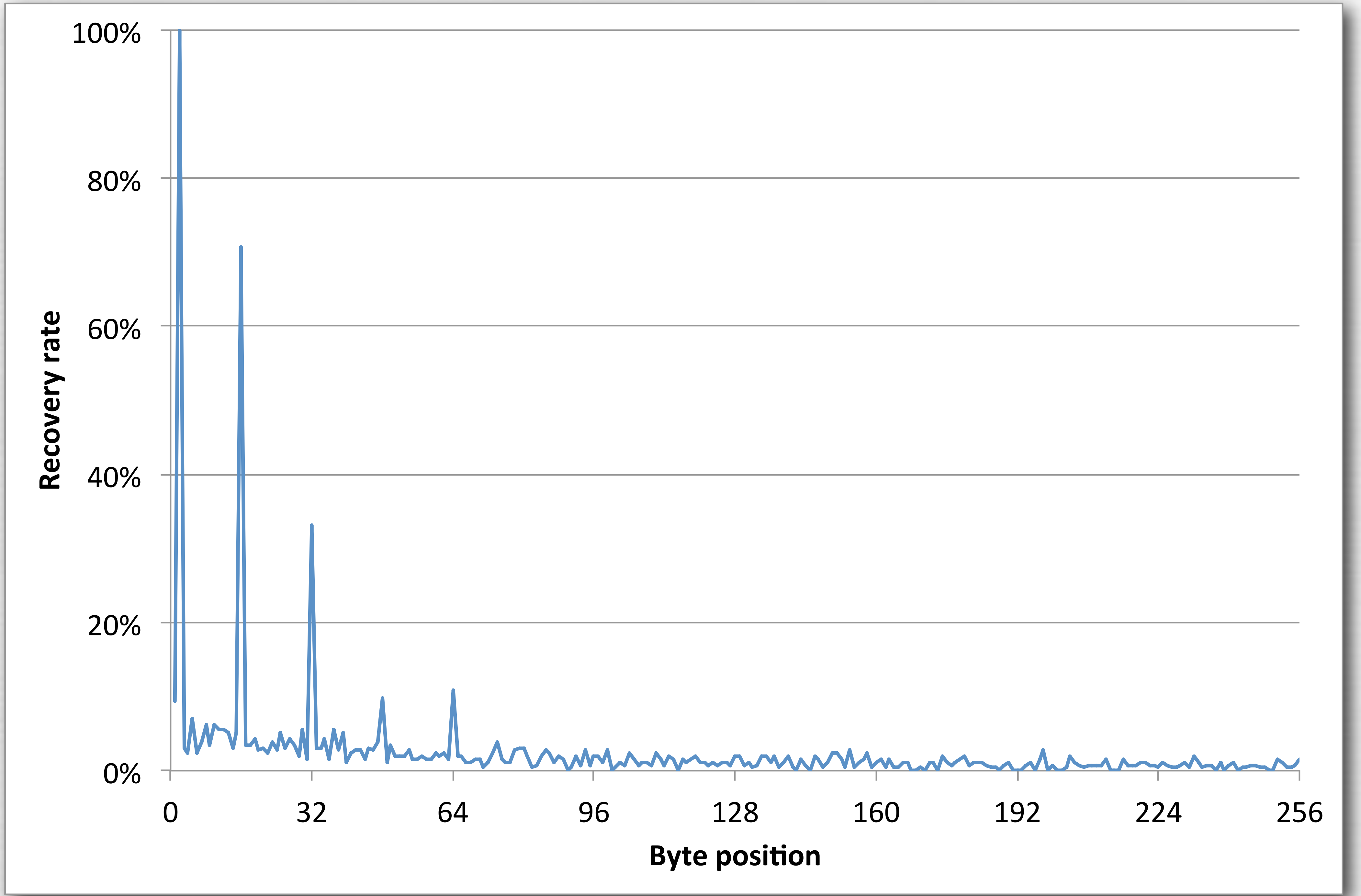
Likelihood of P_r being correct plaintext byte

Recovery algorithm:
Compute most likely plaintext byte

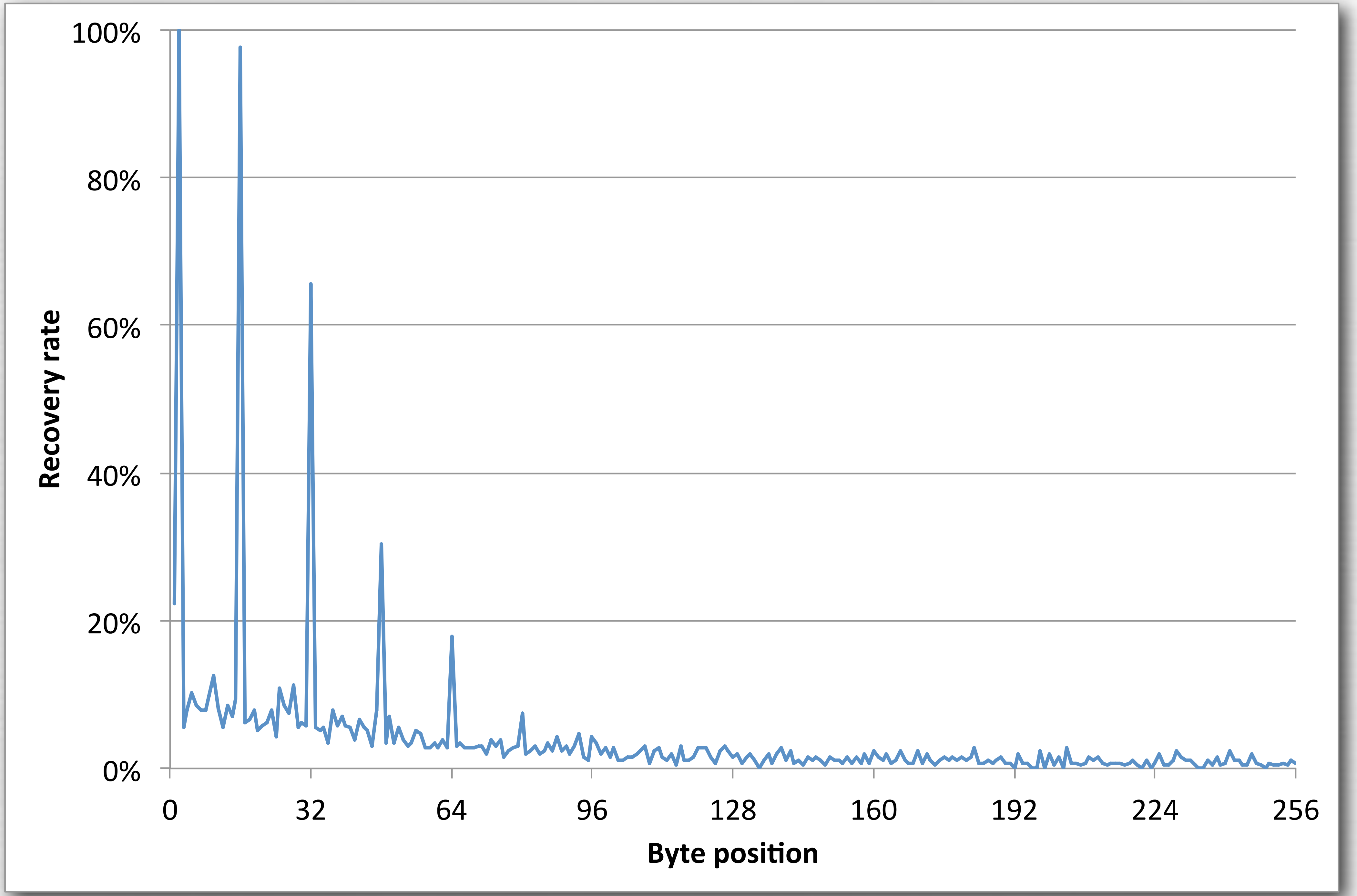
Success Probability
2^{20} Sessions



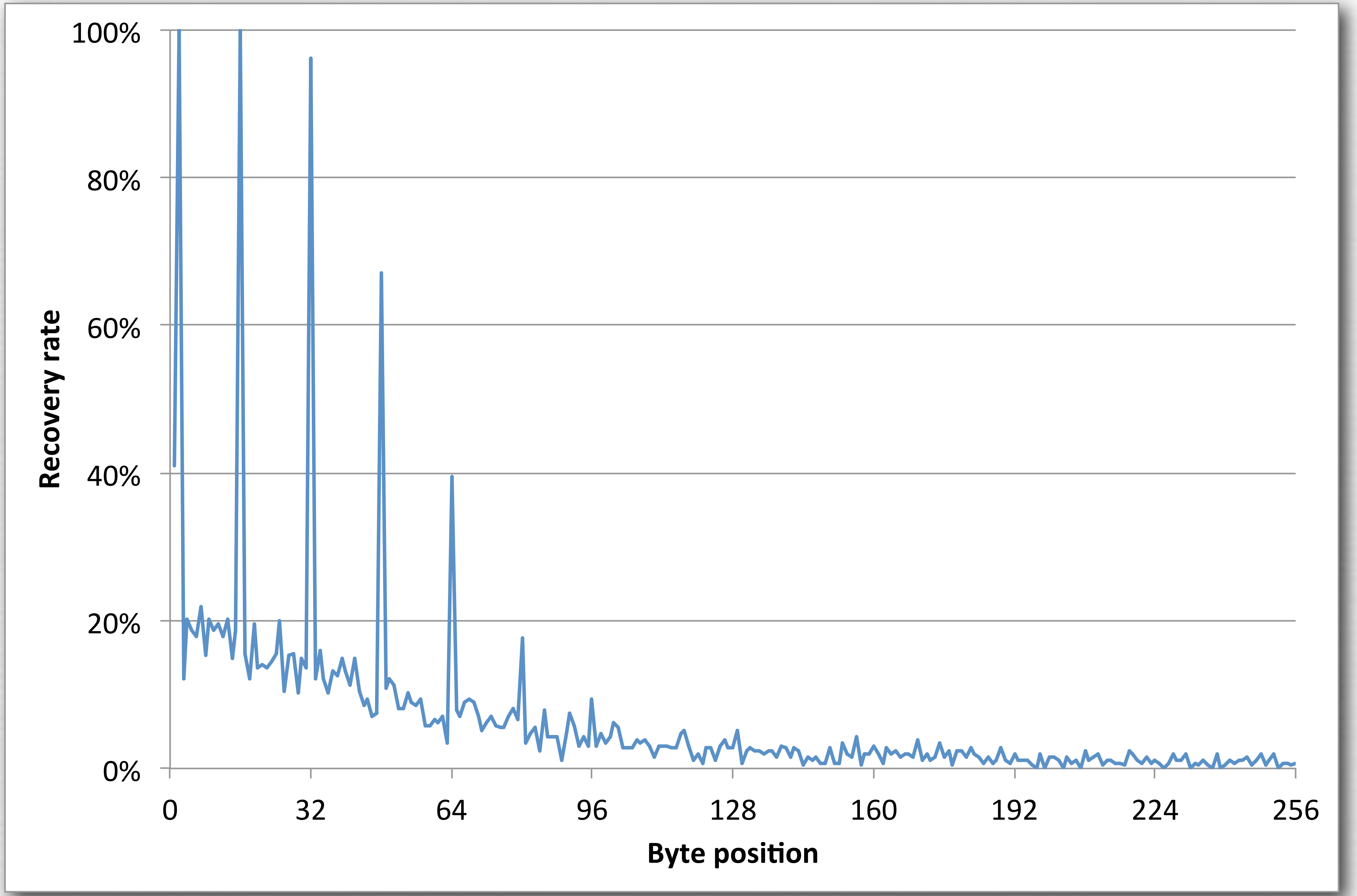
Success Probability
2^{21} Sessions



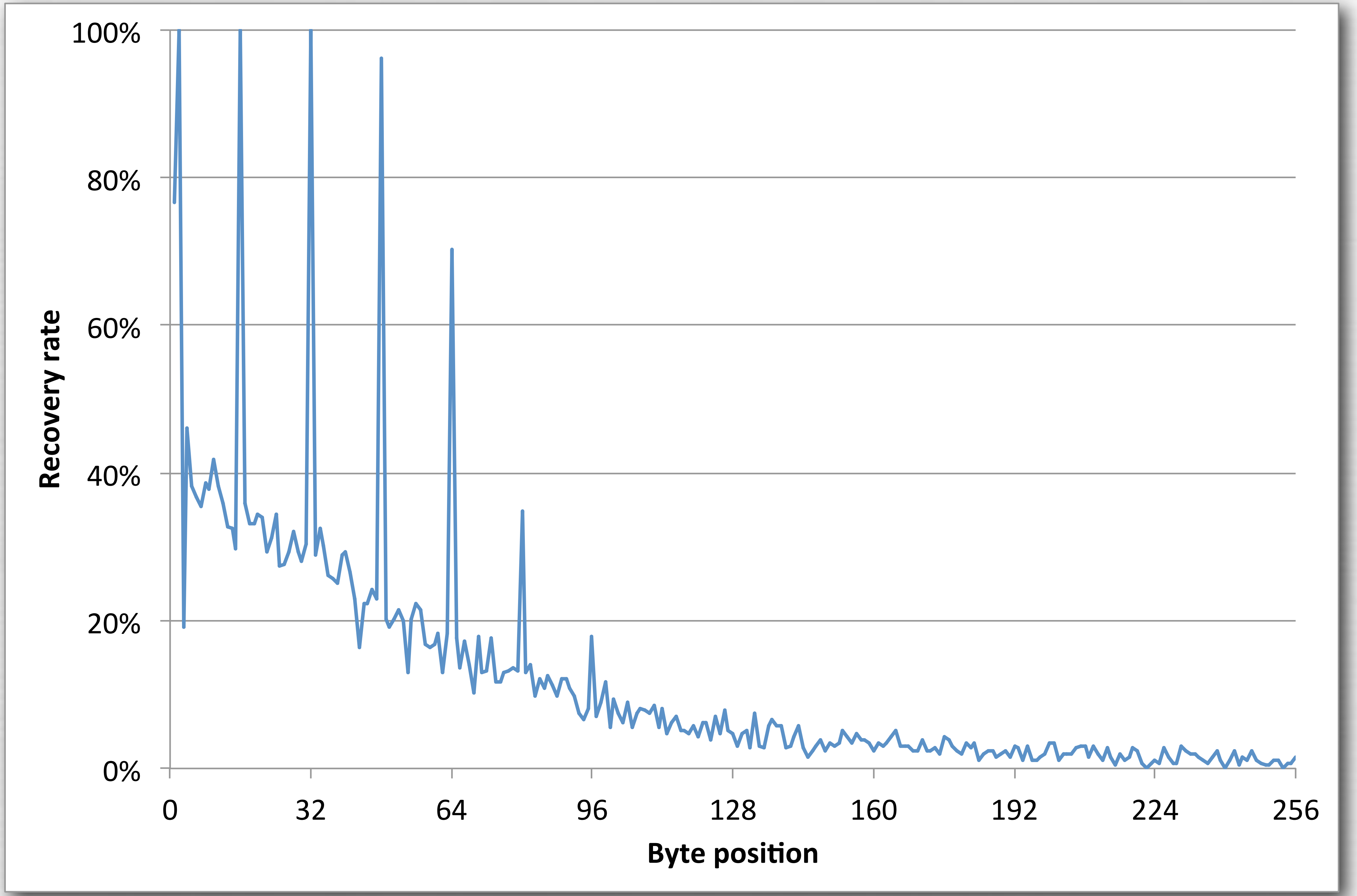
Success Probability
2^{22} Sessions



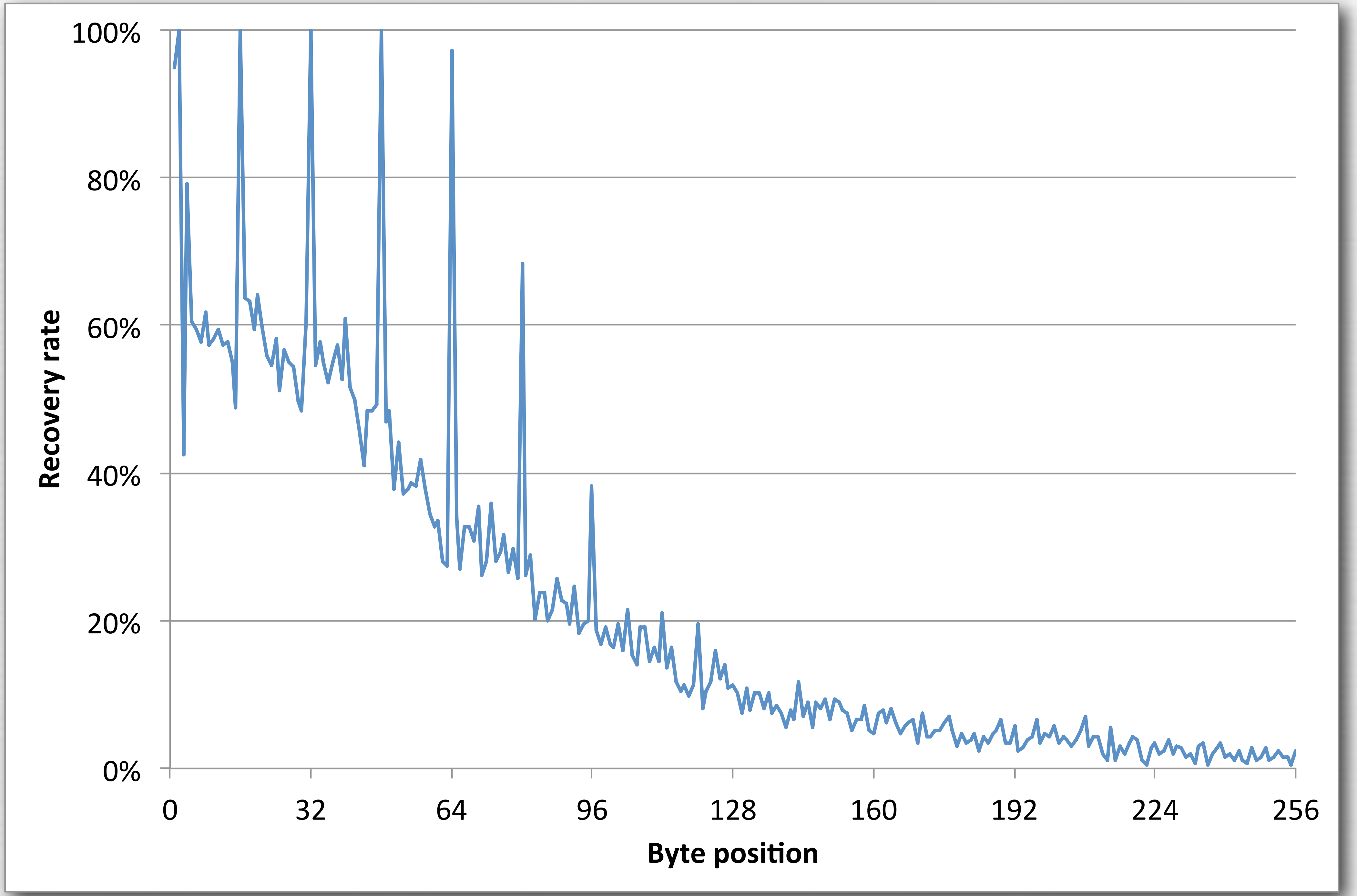
Success Probability
2^{23} Sessions



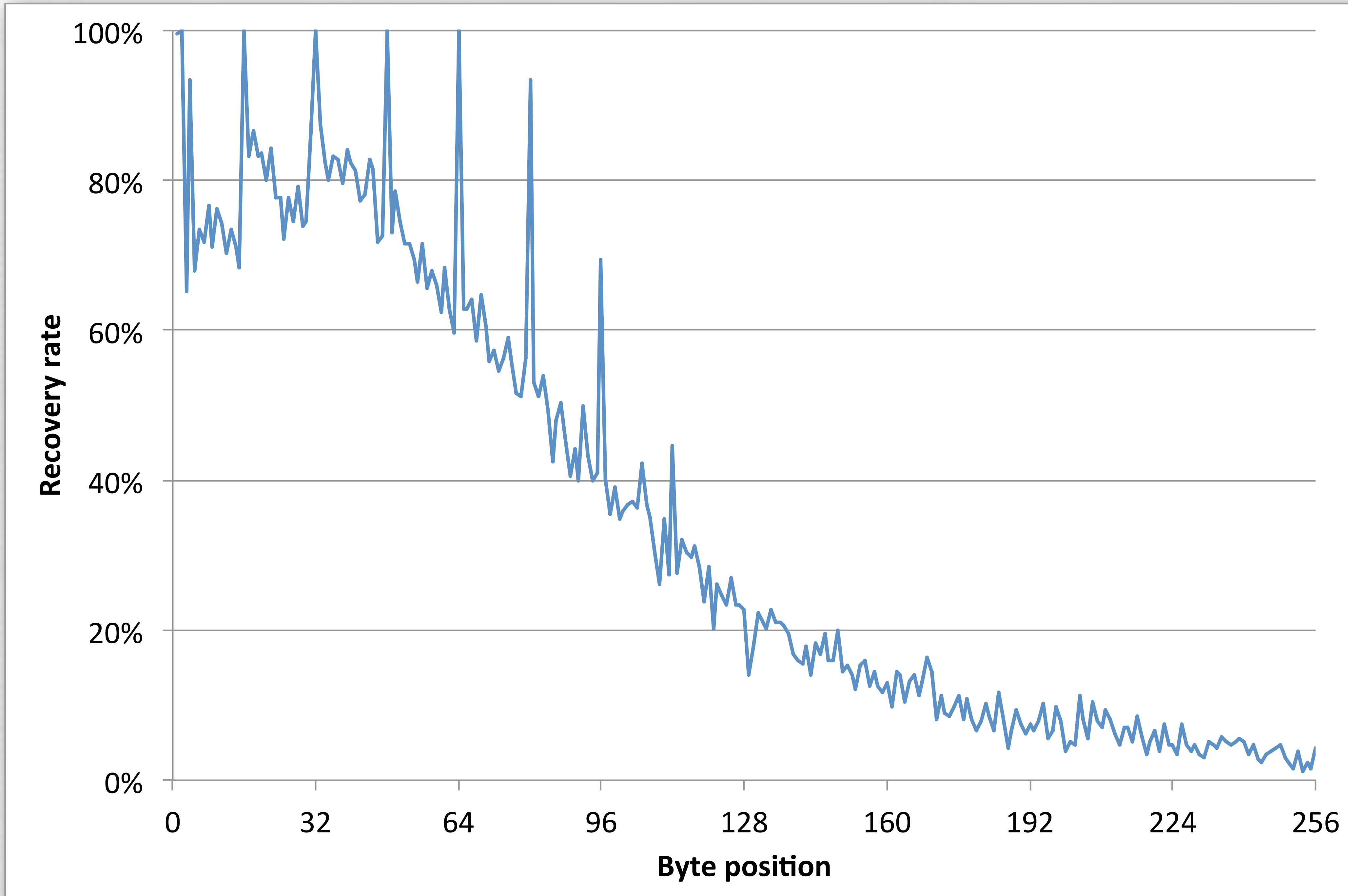
Success Probability
2^{24} Sessions



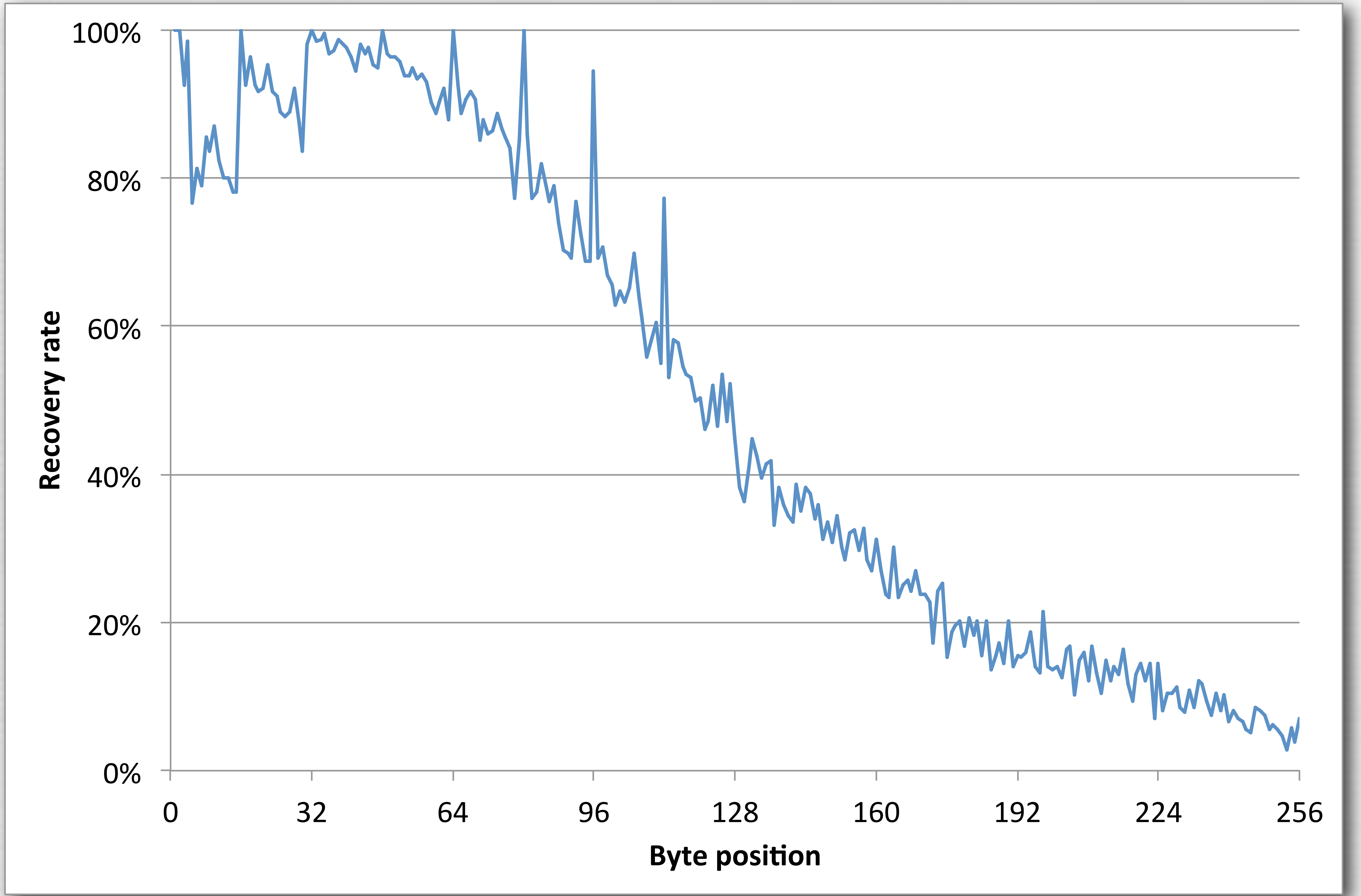
Success Probability
2^{25} Sessions



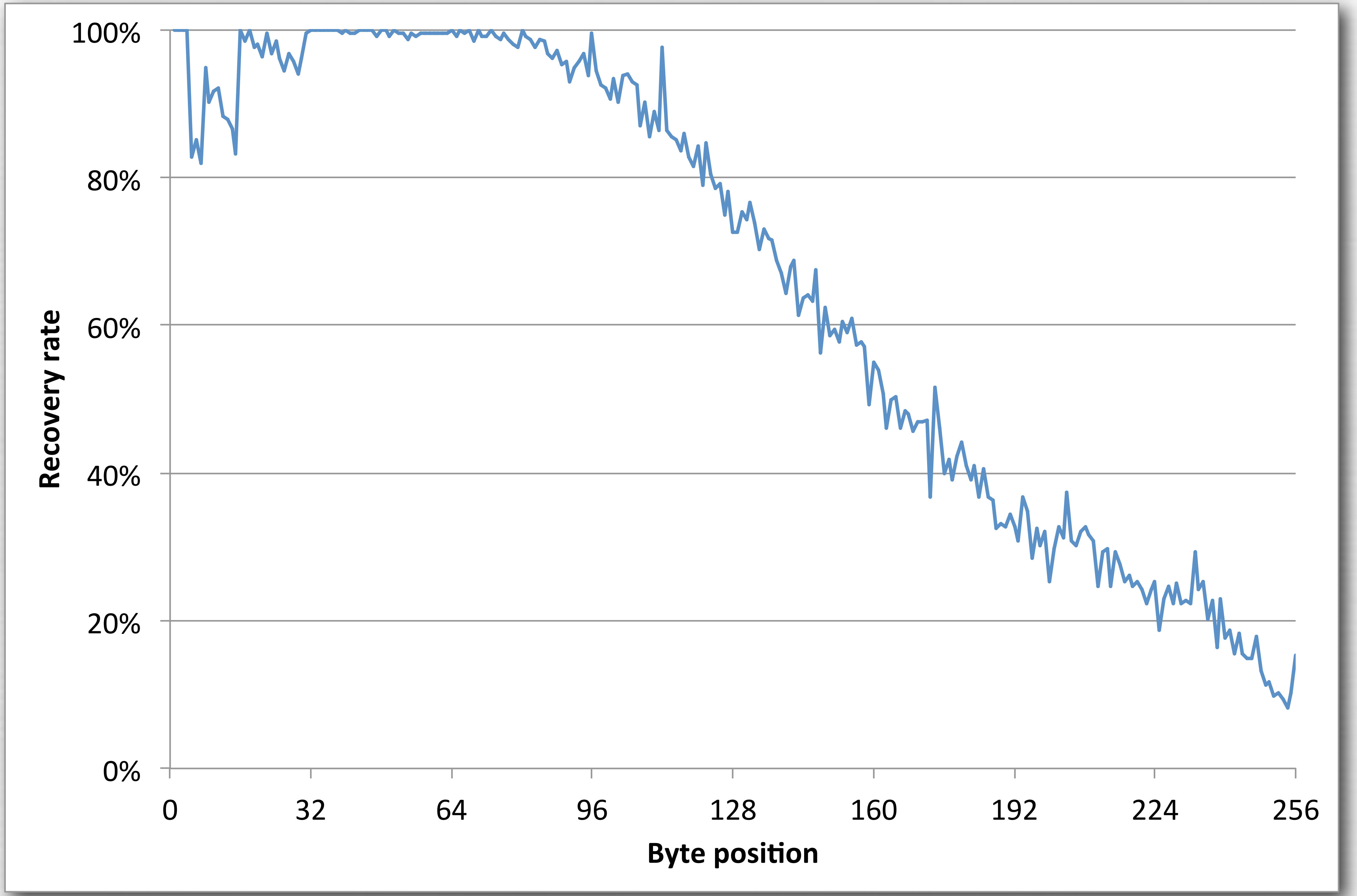
Success Probability
2^{26} Sessions



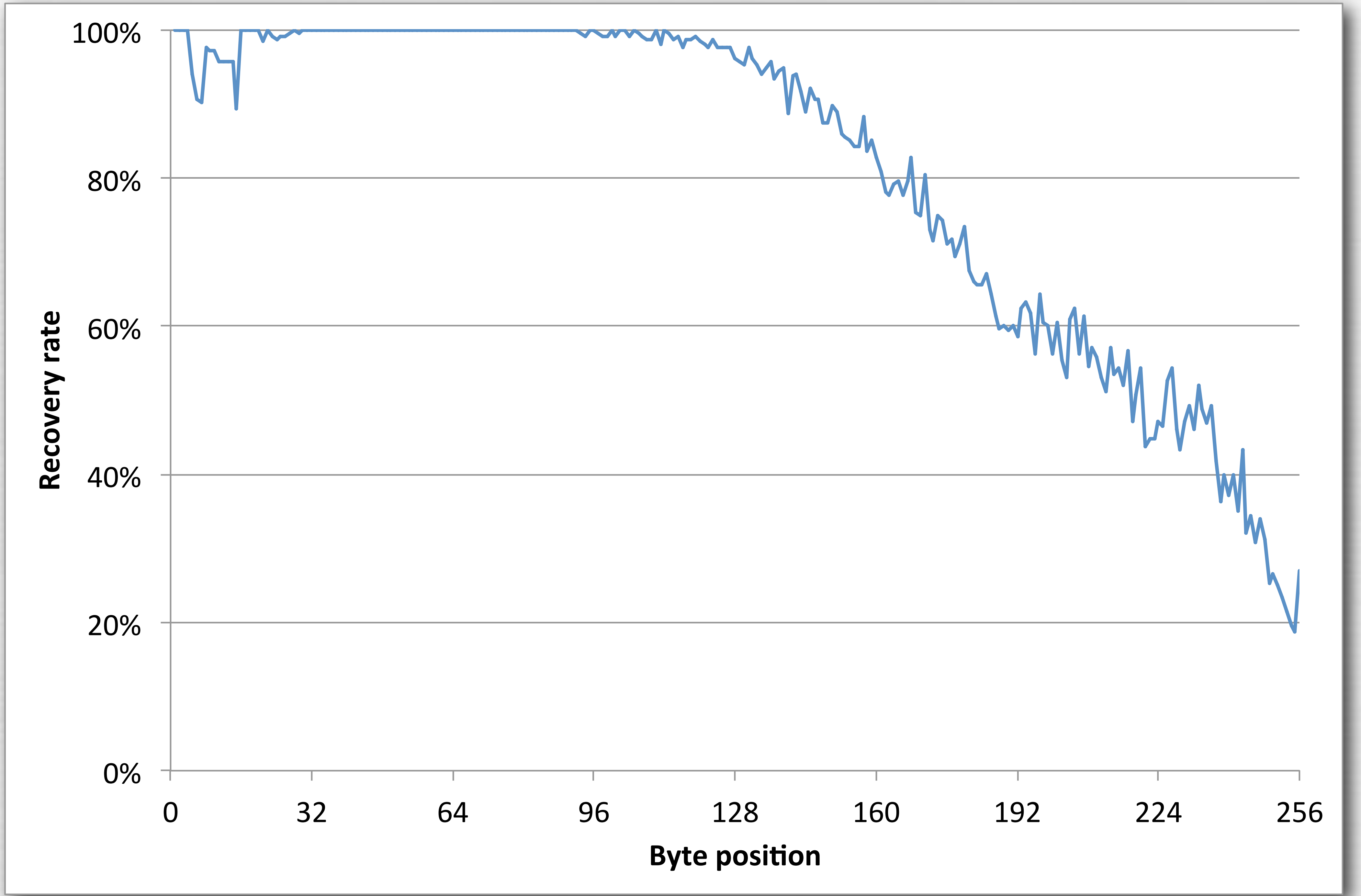
Success Probability
2^{27} Sessions



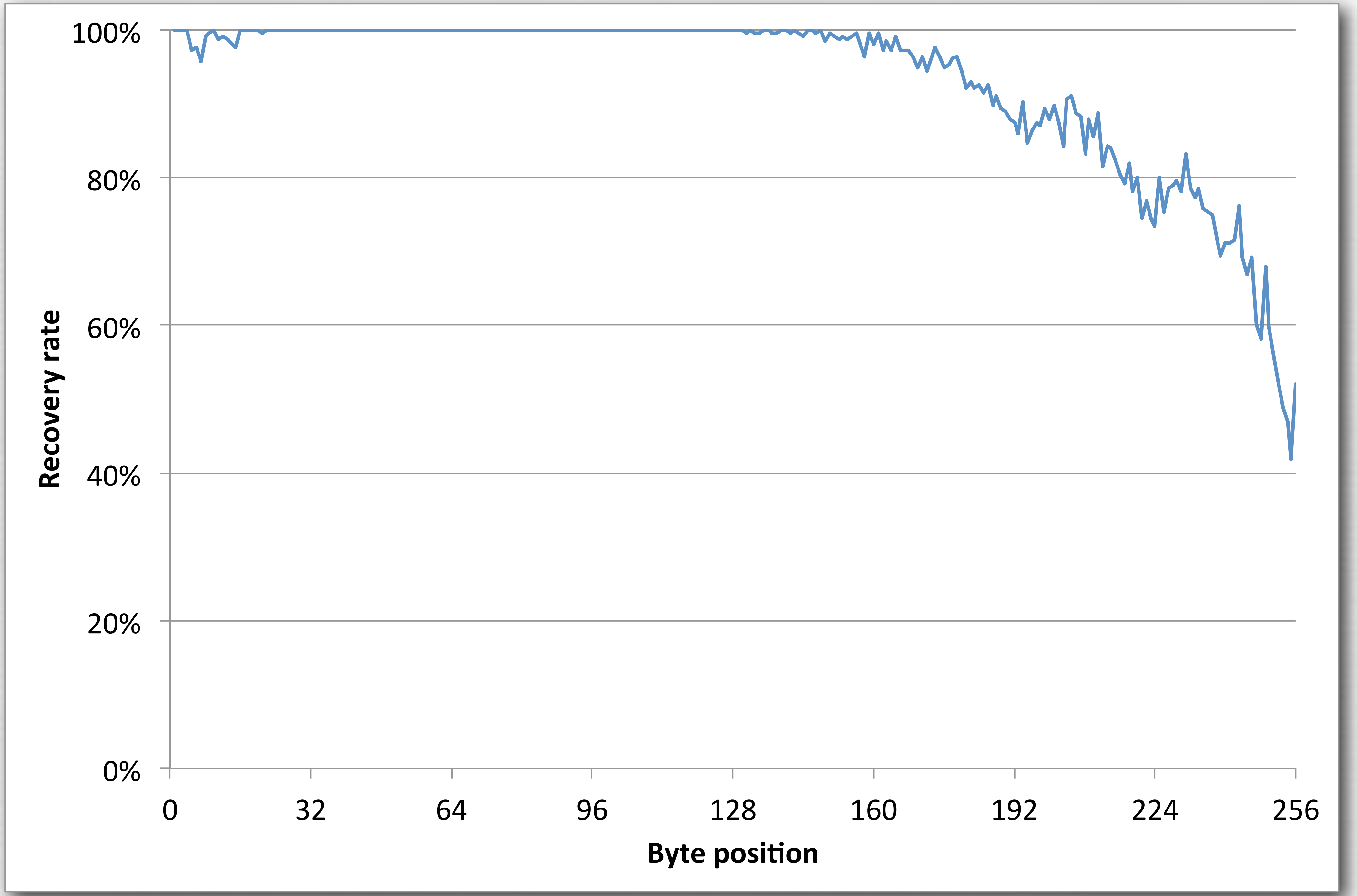
Success Probability
2^{28} Sessions



Success Probability
2^{29} Sessions

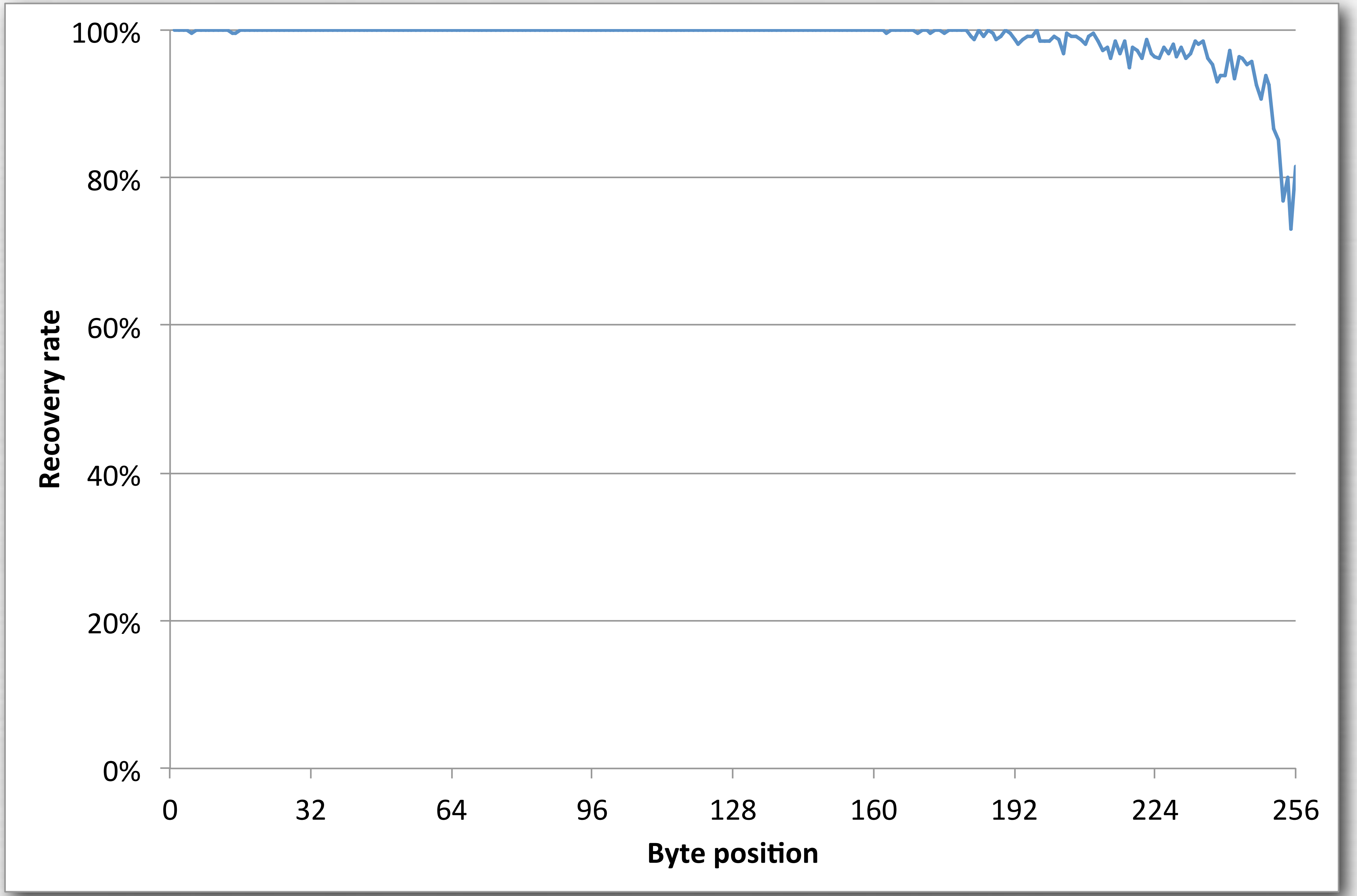


Success Probability
2^{30} Sessions

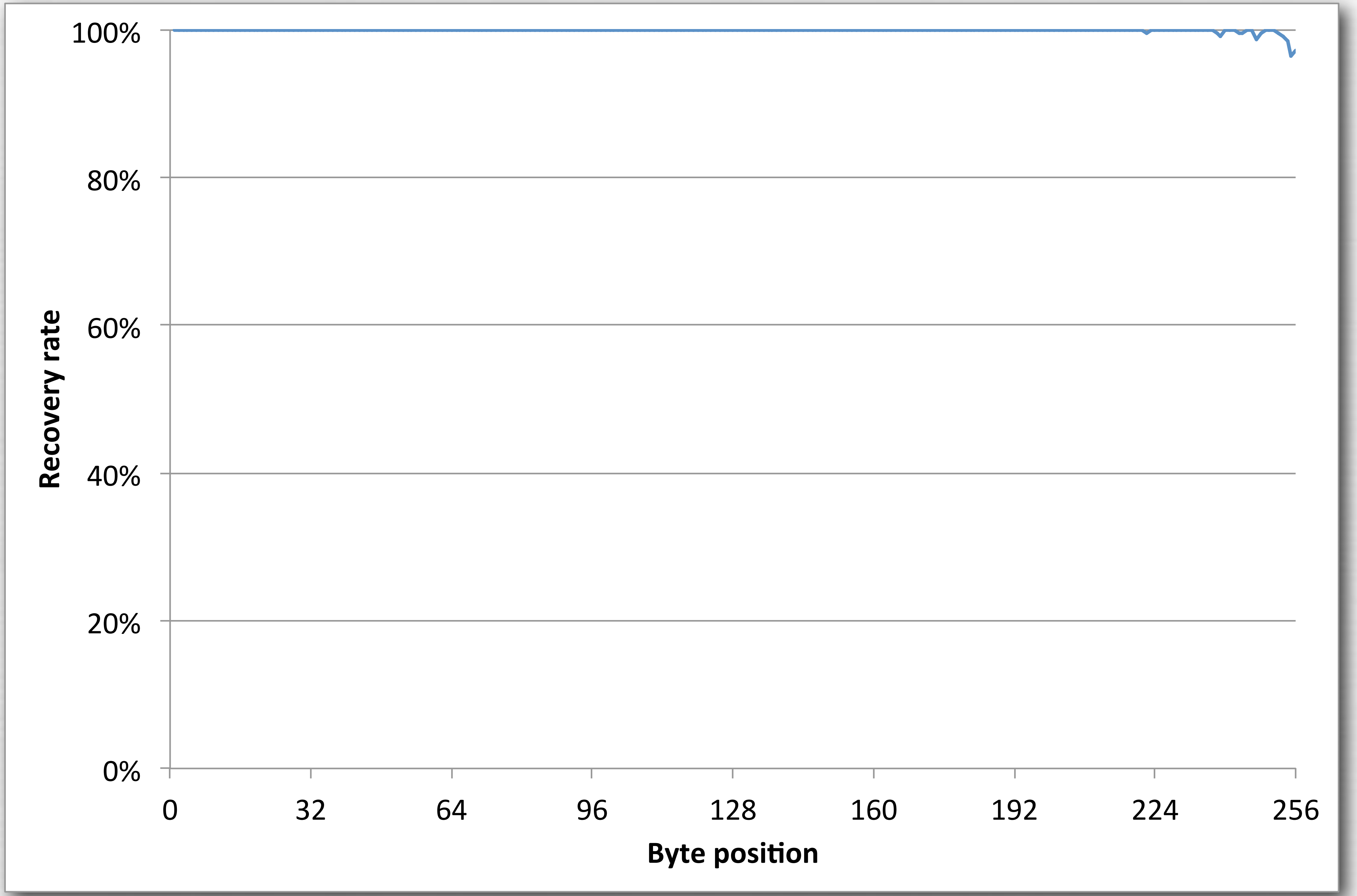


Success Probability
2^{31} Sessions

Information Security Group



Success Probability
2^{32} Sessions



Limitations and Extensions of Attack

Information Security Group

- Limitations of attack
  - Requires $2^{28} \sim 2^{32}$ TLS connections for reliable recovery
  - Attacker has to force TLS session renegotiation / resumption
  - Only first 220 bytes of application data can be targeted
    - Initial 36 bytes used by last message of Handshake protocol
- Extensions:
  - Adapt to take into account a restricted message character space (e.g. base64 encoded plaintexts)
  - Combine with language model for plaintext
  - Consider double-byte biases in the RC4 keystream...

A Second Attack

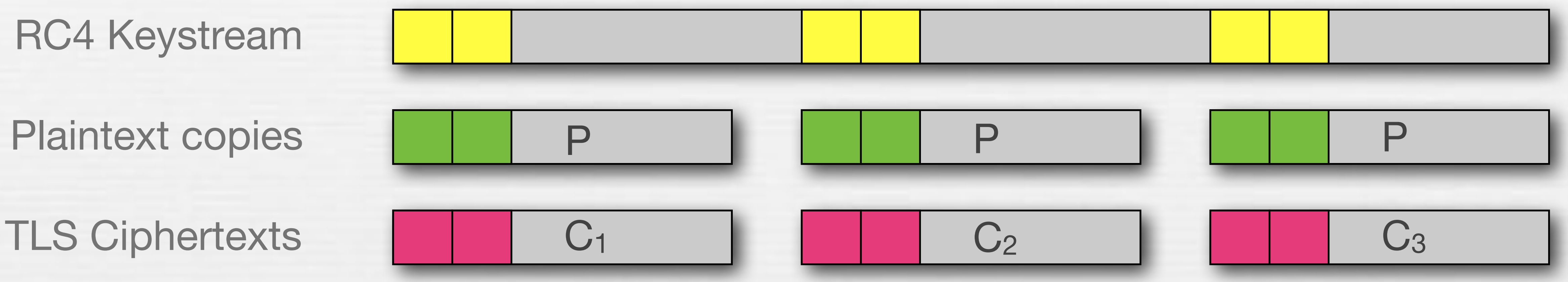
- Fluhrer-McGrew identified biases for consecutive keystream bytes
  - Persistent throughout keystream
- Based on these, we construct an attack which
  - Can target any plaintext byte positions
  - Does not require session renegotiation / resumption

i : keystream byte position mod 256

Byte pair	Condition on i	Probability
$(0,0)$	$i=1$	$2^{-16}(1+2^{-9})$
$(0,0)$	$i \neq 1,255$	$2^{-16}(1+2^{-8})$
$(0,1)$	$i \neq 0,1$	$2^{-16}(1+2^{-8})$
$(i+1,255)$	$i \neq 254$	$2^{-16}(1+2^{-8})$
$(255,i+1)$	$i \neq 1,254$	$2^{-16}(1+2^{-8})$
$(255,i+2)$	$i \neq 0,253,254,255$	$2^{-16}(1+2^{-8})$
$(255,0)$	$i=254$	$2^{-16}(1+2^{-8})$
$(255,1)$	$i=255$	$2^{-16}(1+2^{-8})$
$(255,2)$	$i=0,1$	$2^{-16}(1+2^{-8})$
$(129,129)$	$i=2$	$2^{-16}(1+2^{-8})$
$(255,255)$	$i \neq 254$	$2^{-16}(1-2^{-8})$
$(0,i+1)$	$i \neq 0,255$	$2^{-16}(1-2^{-8})$

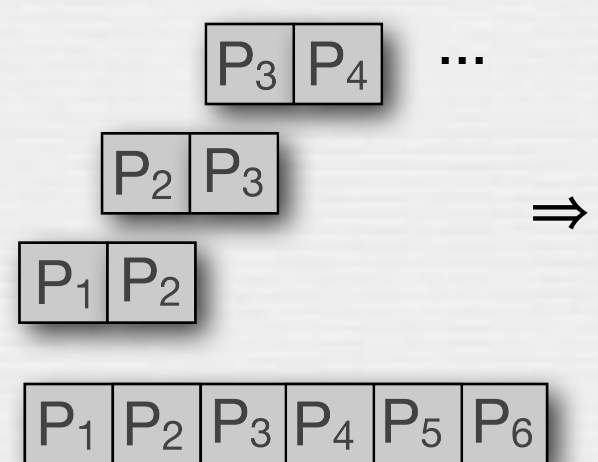
A Second Attack

- Align plaintext with repeating Fluhrer-McGrew biases



- Consider overlapping biases to obtain more accurate likelihood estimate of entire plaintext candidate

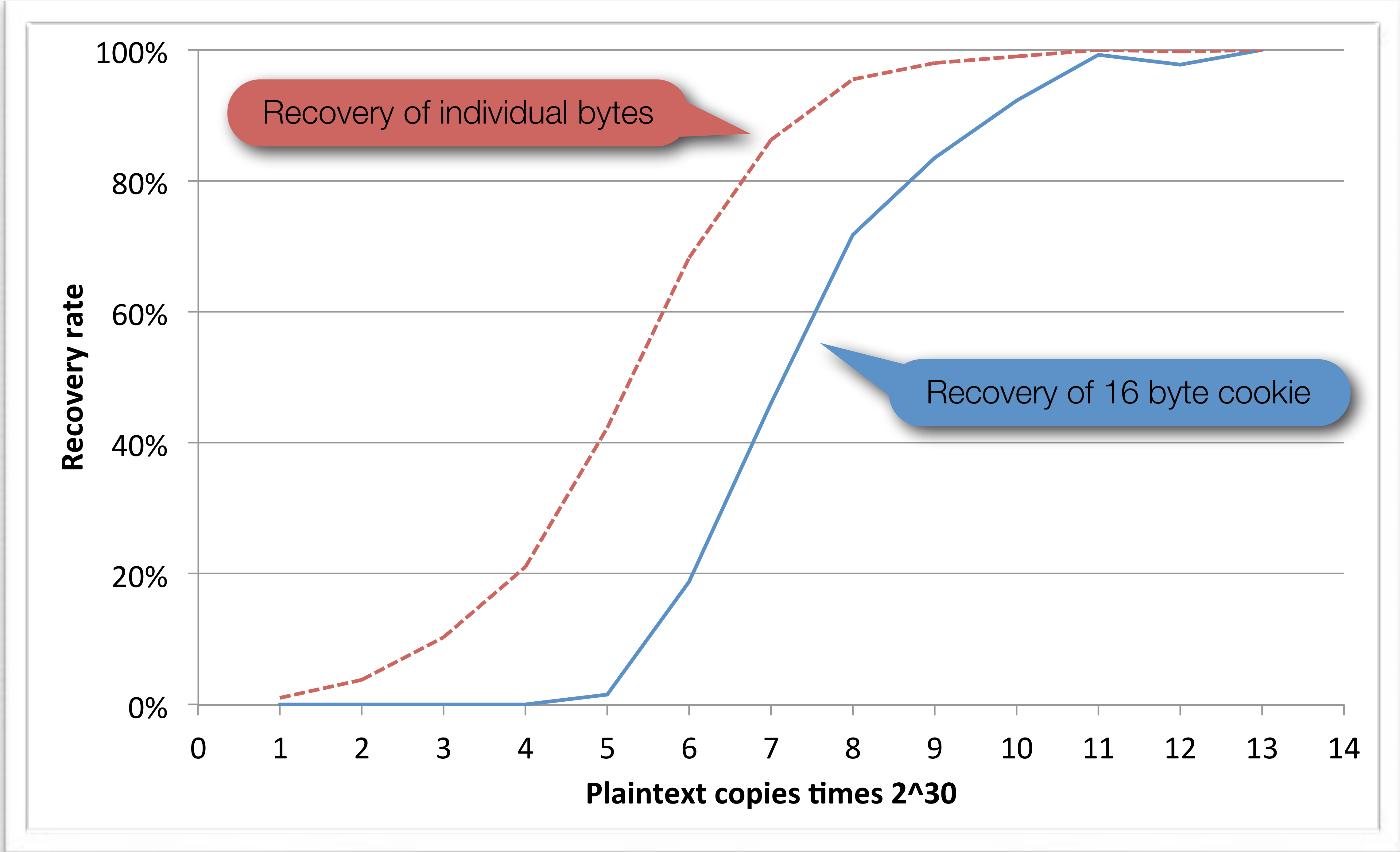
<u>Recovery algorithm:</u>
Optimal Viterbi-style algorithm to determine P with highest likelihood



Likelihood estimate of $P = P_1P_2P_3P_4P_5P_6$

Success Probability

Information Security Group



Limitations and Extensions of Attack

- Limitations
  - Requires $2^{33} \sim 2^{34}$ copies of plaintext to be transmitted for reliable recovery of 16 bytes of plaintext
- Techniques to reduce attack complexity:
  - Adapt to take into account a restricted message character space (e.g. base64 encoded plaintexts)
  - Combine with language model for plaintext

Countermeasures

- Possible countermeasures against our attacks
  - Discard initial keystream bytes
  - Fragment initial records at the application layer
  - Add random length padding to records
  - Limit lifetime of cookies or number of times cookies can be sent
  - Stop using RC4 in TLS
- Vendor response
  - Opera has been implementing a combination of countermeasures
  - Google seems focused on implementing TLS 1.2 and AES-GCM in Chrome
  - RC4 is disabled by default for TLS in Windows Preview 8.1

Conclusions

- Plaintext recovery attacks against RC4 in TLS are feasible although not truly practical
  - $2^{28} \sim 2^{32}$ sessions for reliable recovery of initial bytes
  - $2^{33} \sim 2^{34}$ encryptions for reliable recovery of 16 bytes anywhere in plaintext
- Illustrates that RC4 in TLS provides a security level far below the strength suggested by the used key size (128 bits)
- Furthermore, attacks only becomes better with time...
- Our recommendation: phase out the use of RC4 in TLS as soon as possible

More Information / Future Work

- For the full paper, graphs of RC4 keystream distribution, and raw data, see

http://www.isg.rhul.ac.uk/tls/

- Interested in more discussion on the use of RC4 in TLS? CRYPTO invited talk:
    - “Why the web still runs on RC4”, Adam Langley, Google.

- Future work -- many other security protocols make use of RC4:
    - WPA, Bit-Torrent, Microsoft Point-to-Point Encryption, SSH, Kerberos, Remote Desktop Protocol, etc.
    - Similar analysis and attacks might be applicable...

Questions?

WPA and RC4: Distribution of Z_1