TLS Security - Where Do We Stand?

Kenny Paterson

Royal Holloway
University of London

Information Security Group

Outline

- TLS overview
- TLS attacks and proofs
- Lucky 13
- TLS Record Protocol and RC4
- Discussion

TLS Overview

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv1 broken at birth.
  - SSLv2 flawed, now IETF-deprecated (RFC 6176).
  - SSLv3 still widely supported.

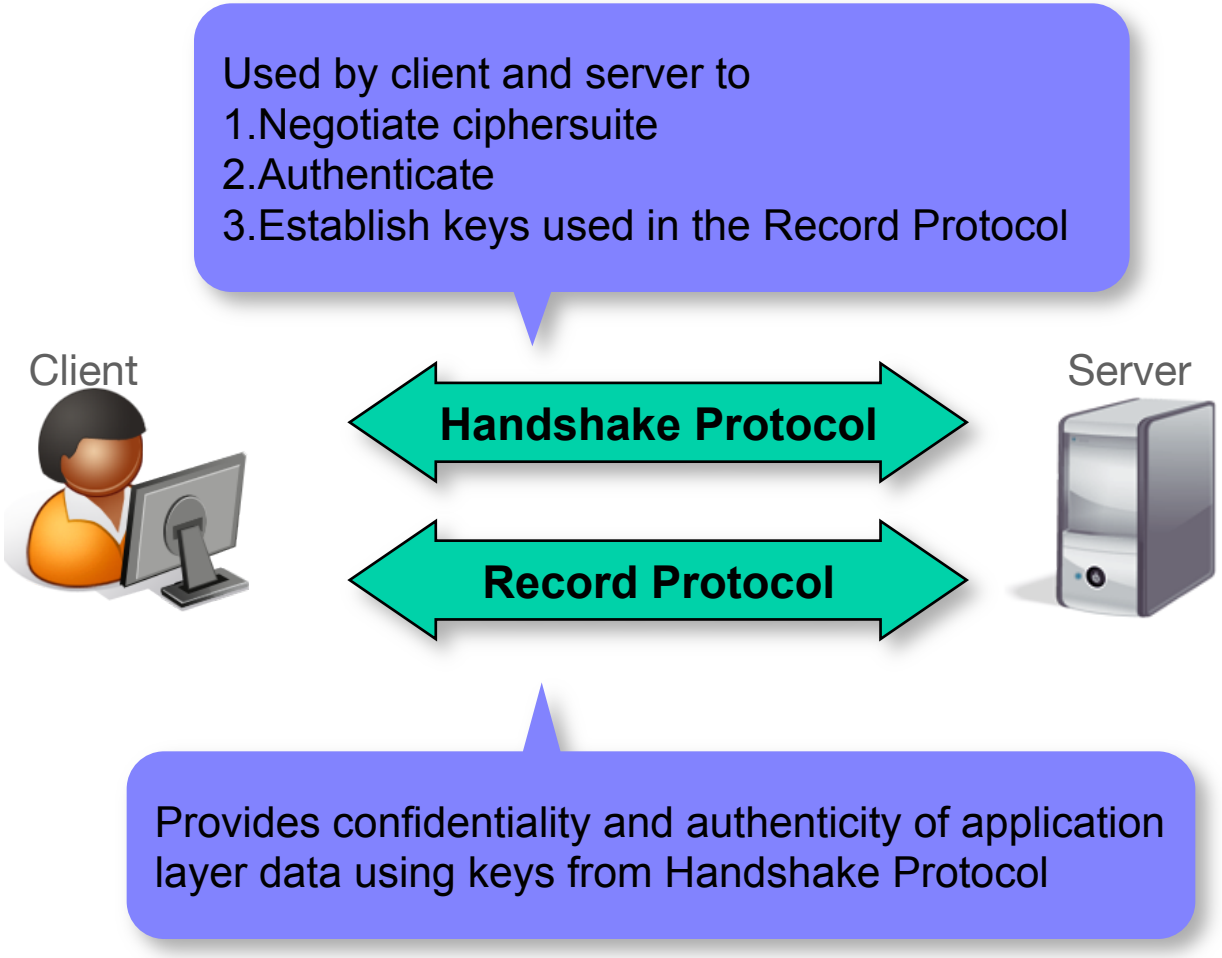
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 in RFC 2246 (1999).
  - TLS 1.1 in RFC 4346 (2006).
  - TLS 1.2 in RFC 5246 (2008).

Importance of TLS

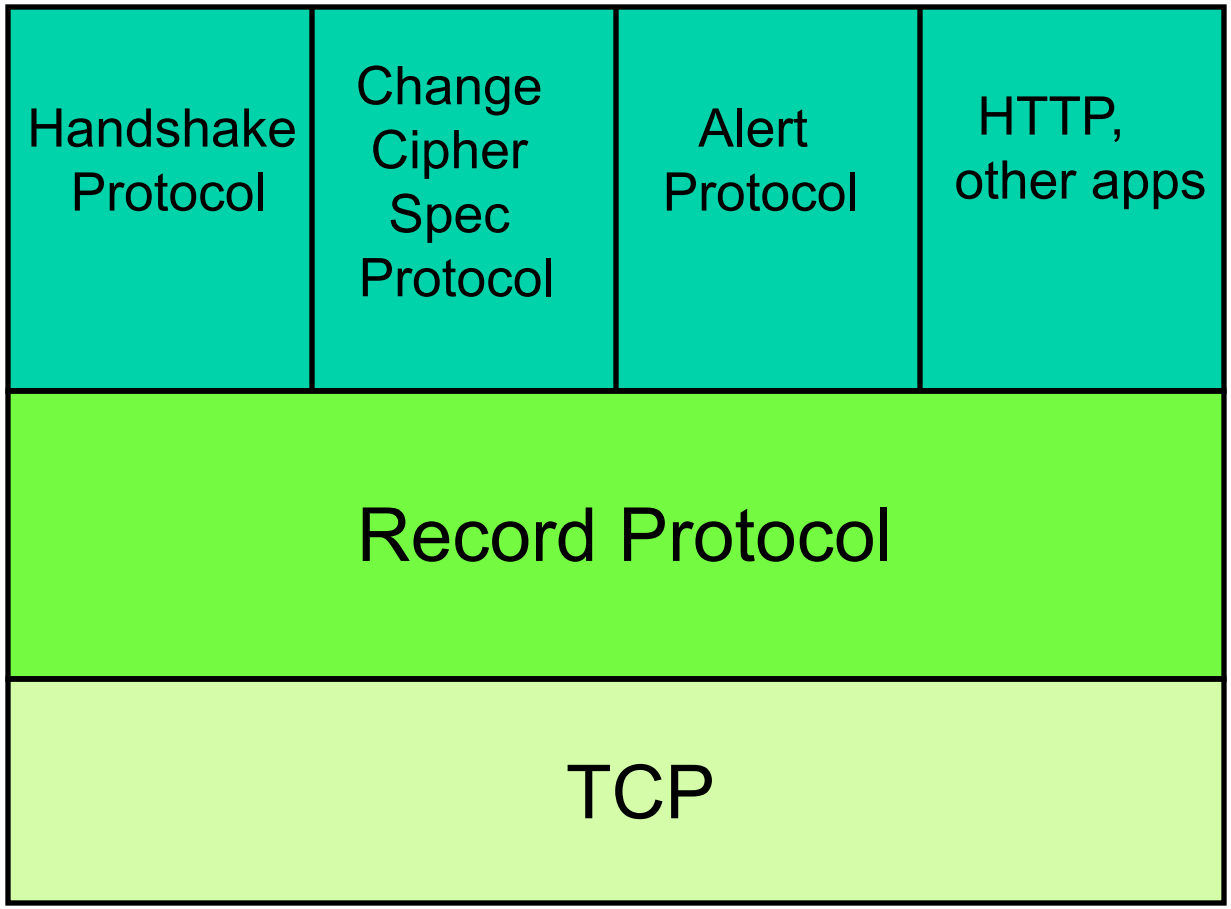
- Originally designed for secure e-commerce, now used much more widely.
  - Retail customer access to online banking facilities.
  - Access to gmail, facebook, Yahoo, etc.
  - Mobile applications, including banking apps.
  - Payment infrastructures.

- TLS has become the *de facto* secure protocol of choice.
  - Used by hundreds of millions of people and devices every day.
  - A serious attack could be catastrophic, both in real terms and in terms of perception/confidence.

Simplified View of TLS



TLS Protocol Architecture



TLS Complexity

- Multiple interacting sub-protocols.
- Many different options for each sub-protocol.
- Which option is used is negotiated during the Handshake Protocol itself.
  - And can be renegotiated using that protocol too.
- Different versions of the protocol.
- Many extensions of the basic protocol.
- Specification versus implementation.
  - Plenty of room for imprecision and error.
  - Spec. and code now contain layers of attack-specific fixes.

Common Researcher Responses to TLS Complexity

- Prove security of parts of the protocol.
  - For example, just the cryptographic transform used in the Record Protocol.
  - Ignoring padding, compression, statefulness, fragementation,…
- Idealise the protocol.
  - For example, assume that Handshake Protocol uses CCA-secure public key encryption (it doesn't!)
  - Or use a symbolic approach to cryptography.

Positive Security Results for TLS

Theory for symmetric encryption in TLS:

- [K01] Krawczyk, Crypto 2001
- [MT10] Maurer and Tackmann, ACM-CCS 2010
- [PRS11] Paterson, Ristenpart, Shrimpton, Asiacrypt 2011

Theory for key exchange in TLS:

- [BR93] Bellare and Rogaway, Crypto 1993
- [JK02] Jonsson and Kaliski Jr., Crypto 2002
- [MSW08] Morrissey et al., Asiacrypt 2008
- [JKSS12], Jager *et al.*, Crypto 2012
- [GKS13] Giesen *et al.*, ACM-CCS 2013
- [KPW13] Krawczyk *et al.*, Crypto 2013

Common Researcher Responses to TLS Complexity

- Prove security of parts of the protocol.
  - For example, just the cryptographic transform used in the Record Protocol.
  - Ignoring padding, compression, statefulness, fragementation,…
- Idealise the protocol.
  - For example, assume that Handshake Protocol uses CCA-secure public key encryption (it doesn't!)
  - Or use a symbolic approach to cryptography.
- **Break (some aspect of) the protocol.**

TLS Attack Literature

- [B98] Bleichenbacher, Crypto 1998
- [V02] Vaudenay, Eurocrypt 2002
- [M02] Moeller, http://www.openssl.org/~bodo/tls-cbc.txt, 2002
- [CHVV03] Canvel *et al.*, Crypto 2003
- [B04] Bard, eprint 2004
- [B06] Bard, SECRYPT 2006
- [RD09] Ray and Dispensa, TLS renegotiation attack, 2009
- [PRS11] Paterson *et al.*, Asiacrypt 2011
- [DR11] Duong and Rizzo, “Here come the XOR Ninjas”, 2011
- [AP12] AlFardan and Paterson, NDSS 2012
- [DR12] Duong and Rizzo, CRIME, 2012
- [MVVP12] Mavrogiannopoulos *et al*.,CCS 2012
- [AP13] N.J. AlFardan and K.G. Paterson, IEEE S&P, 2013
- [ABPPS13] N.J. AlFardan *et al.*, USENIX Security, 2013
- [BFKPS13] Bhargavan *et al.,* IEEE S&P, 2013

TLS Attack Literature

- [B98] Bleichenbacher, Crypto 1998
- [V02] Vaudenay, Eurocrypt 2002
- [M02] Moeller, http://www.openssl.org/~bodo/tls-cbc.txt, 2002
- [CHVV03] Canvel *et al.*, Crypto 2003
- [B04] Bard, eprint 2004
- [B06] Bard, SECRYPT 2006
- [RD09] Ray and Dispensa, TLS renegotiation attack, 2009
- [PRS11] Paterson *et al.*, Asiacrypt 2011
- [DR11] Duong and Rizzo, “Here come the XOR Ninjas”, 2011
- [AP12] AlFardan and Paterson, NDSS 2012
- [DR12] Duong and Rizzo, CRIME, 2012
- [MVVP12] Mavrogiannopoulos *et al*.,CCS 2012
- [AP13] N.J. AlFardan and K.G. Paterson, IEEE S&P, 2013
- [ABPPS13] N.J. AlFardan *et al.*, USENIX Security, 2013
- [BFKPS13] Bhargavan *et al.,* IEEE S&P, 2013

TLS Attack Literature

- [B98] Bleichenbacher, Crypto 1998
- [V02] Vaudenay, Eurocrypt 2002
- [M02] Moeller, http://www.openssl.org/~bodo/tls-cbc.txt, 2002
- [CHVV03] Canvel *et al.*, Crypto 2003
- [B04] Bard, eprint 2004
- [B06] Bard, SECRYPT 2006
- [RD09] Ray and Dispensa, TLS renegotiation attack, 2009
- [PRS11] Paterson *et al.*, Asiacrypt 2011
- [DR11] Duong and Rizzo, “Here come the XOR Ninjas”, 2011
- [AP12] AlFardan and Paterson, NDSS 2012
- [DR12] Duong and Rizzo, CRIME, 2012
- [MVVP12] Mavrogiannopoulos *et al.*,CCS 2012
- [AP13] N.J. AlFardan and K.G. Paterson, IEEE S&P, 2013
- [ABPPS13] N.J. AlFardan *et al.*, USENIX Security, 2013
- [BFKPS13] Bhargavan *et al.,* IEEE S&P, 2013

TLS Attack Literature

- [B98] Bleichenbacher, Crypto 1998
- [V02] Vaudenay, Eurocrypt 2002
- [M02] Moeller, http://www.openssl.org/~bodo/tls-cbc.txt, 2002
- [CHVV03] Canvel *et al.*, Crypto 2003
- [B04] Bard, eprint 2004
- [B06] Bard, SECRYPT 2006
- [RD09] Ray and Dispensa, TLS renegotiation attack, 2009
- [PRS11] Paterson *et al.*, Asiacrypt 2011
- [DR11] Duong and Rizzo, “Here come the XOR Ninjas”, 2011
- [AP12] AlFardan and Paterson, NDSS 2012
- [DR12] Duong and Rizzo, CRIME, 2012
- [MVVP12] Mavrogiannopoulos *et al.*,CCS 2012
- [AP13] N.J. AlFardan and K.G. Paterson, IEEE S&P, 2013
- [ABPPS13] N.J. AlFardan *et al.*, USENIX Security, 2013
- [BFKPS13] Bhargavan *et al.,* IEEE S&P, 2013

TLS Attack Literature

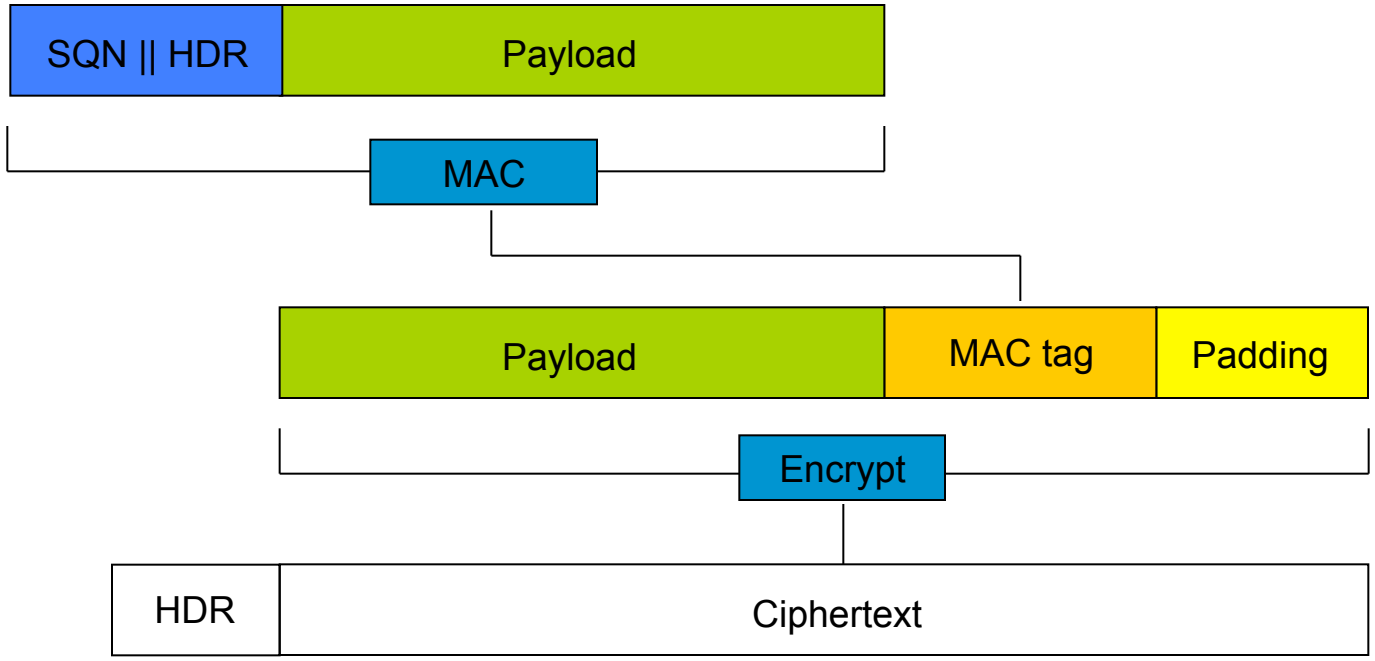
- [B98] Bleichenbacher, Crypto 1998
- [V02] Vaudenay, Eurocrypt 2002
- [M02] Moeller, http://www.openssl.org/~bodo/tls-cbc.txt, 2002
- [CHVV03] Canvel *et al.*, Crypto 2003
- [B04] Bard, eprint 2004
- [B06] Bard, SECRYPT 2006
- [RD09] Ray and Dispensa, TLS renegotiation attack, 2009
- [PRS11] Paterson *et al.*, Asiacrypt 2011
- [DR11] Duong and Rizzo, “Here come the XOR Ninjas”, 2011
- [AP12] AlFardan and Paterson, NDSS 2012
- [DR12] Duong and Rizzo, CRIME, 2012
- [MVVP12] Mavrogiannopoulos *et al.*,CCS 2012
- **[AP13] N.J. AlFardan and K.G. Paterson, IEEE S&P, 2013**
- **[ABPPS13] N.J. AlFardan *et al.*, USENIX Security, 2013**
- [BFKPS13] Bhargavan *et al.,* IEEE S&P, 2013

Outline

- TLS overview
- TLS attacks and proofs
- Lucky 13*
- TLS Record Protocol and RC4
- Discussion

*AlFardan and Paterson, *Lucky 13: Breaking the TLS and DTLS Record Protocols*, IEEE Security and Privacy Symposium, 2013. (www.isg.rhul.ac.uk/tls/lucky13.html)

TLS Record Protocol: MAC-Encode-Encrypt (MEE)

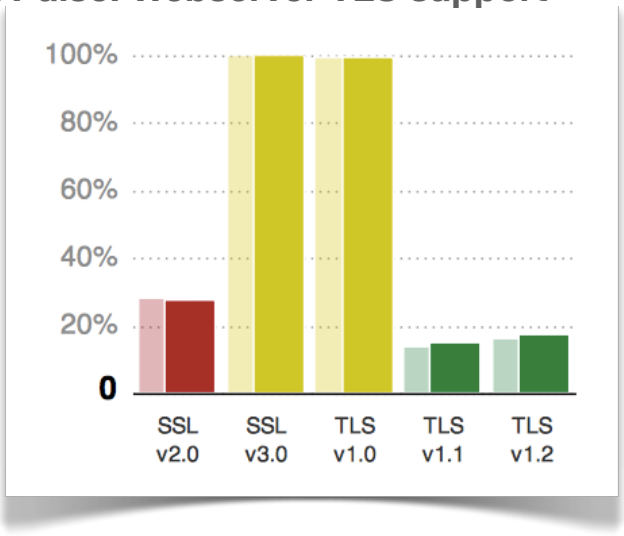


MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128
Padding	“00” or “01 01” or “02 02 02” or …. or “FF FF….FF”

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 yet 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

Lucky 13

- Distinguishing attacks and full plaintext recovery attacks against TLS-CBC implementations following implementation advice in the TLS 1.2 spec.
  - And variant attacks against those that do not.
- Applies to all versions of SSL/TLS.
  - SSLv3.0, TLS 1.0, 1.1, 1.2.
  - And DTLS.
- Demonstrated in the lab against OpenSSL and GnuTLS.

History: TLS and Padding Oracles

[V02,CHVV03]:

- Specifics of TLS padding format can be exploited to mount a plaintext recovery attack.
- The attack depends on being able to distinguish *good* from *bad* padding.
  - In practice, this is done via a timing side-channel.
  - The MAC is only checked if the padding is good, and the MAC is always bad in the attack.
  - Distinguish cases by timing TLS error messages.

Countermeasures?

- Redesign TLS:
  - Pad-MAC-Encrypt or Pad-Encrypt-MAC.
  - Too invasive, did not happen.
- Switch to RC4?
- Or add a fix to ensure *uniform errors*?
  - If attacker can't tell difference between MAC and pad errors, then maybe TLS's MEE construction is secure?
  - So how should TLS implementations ensure uniform errors?

Ensuring Uniform Errors

From the TLS 1.2 specification:

…implementations MUST ensure that record processing time is essentially the same whether or not the padding is correct.

In general, the best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet.

Compute the MAC on what though?

Ensuring Uniform Errors

For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC.

- This approach is adopted in many implementations, including OpenSSL, NSS (Chrome, Firefox), BouncyCastle, OpenJDK, …

- One alternative (GnuTLS and others) is to remove as many bytes as are indicated by the last byte of plaintext and compute the MAC on what's left.

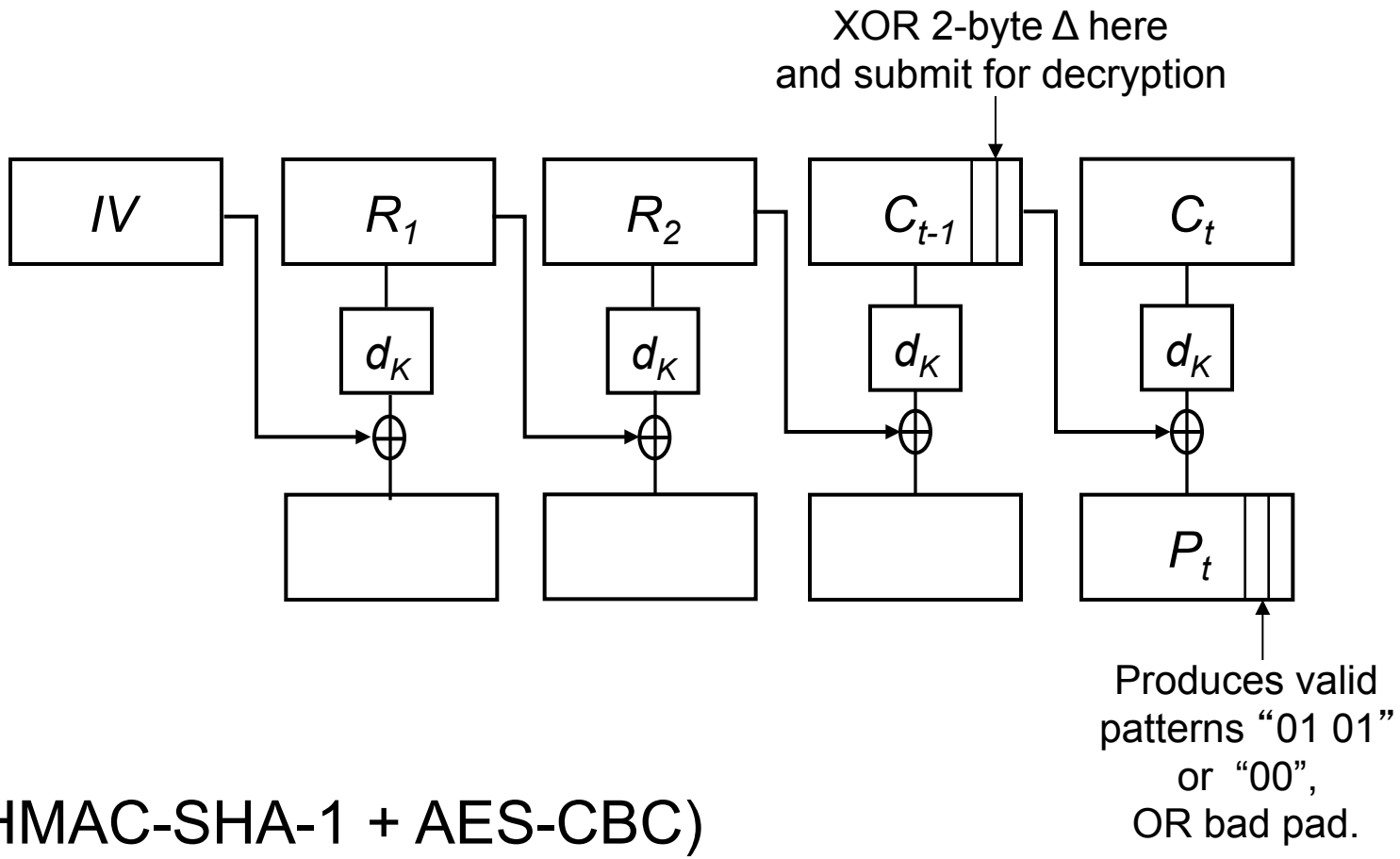
Ensuring Uniform Errors

… This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.

Ensuring Uniform Errors

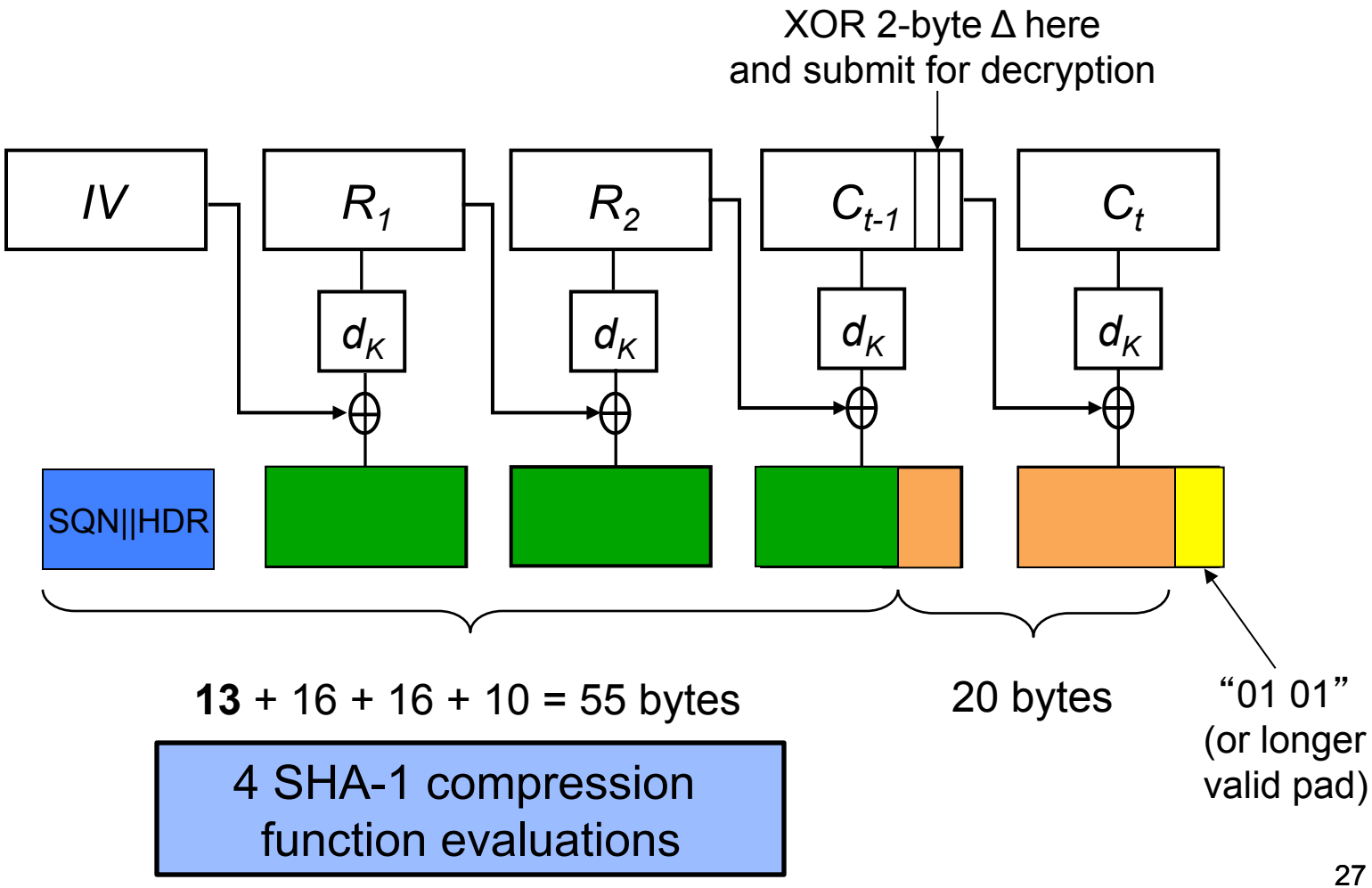
… This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.

Lucky 13 – Plaintext Recovery

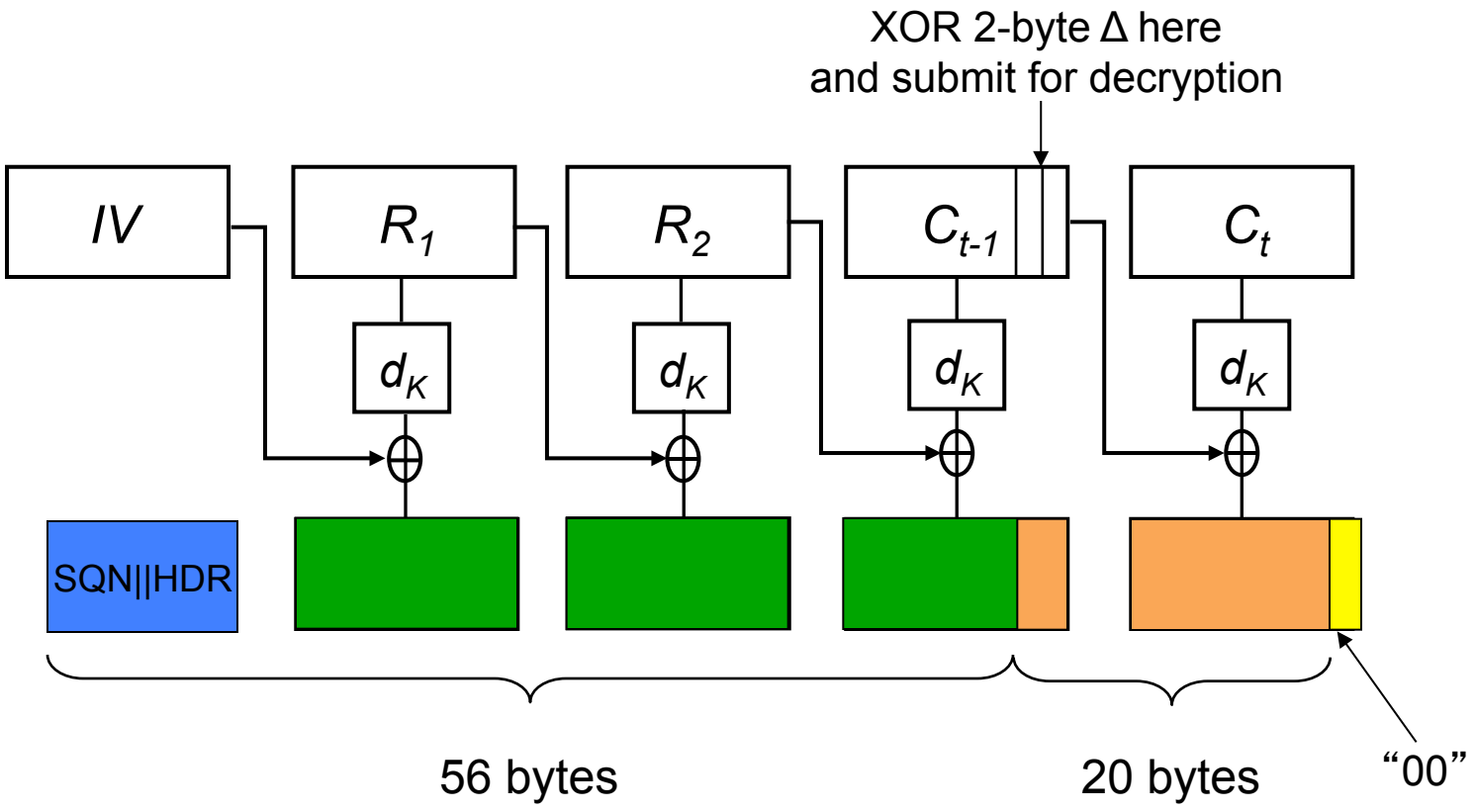


(HMAC-SHA-1 + AES-CBC)

Case: “01 01” (or longer valid pad)

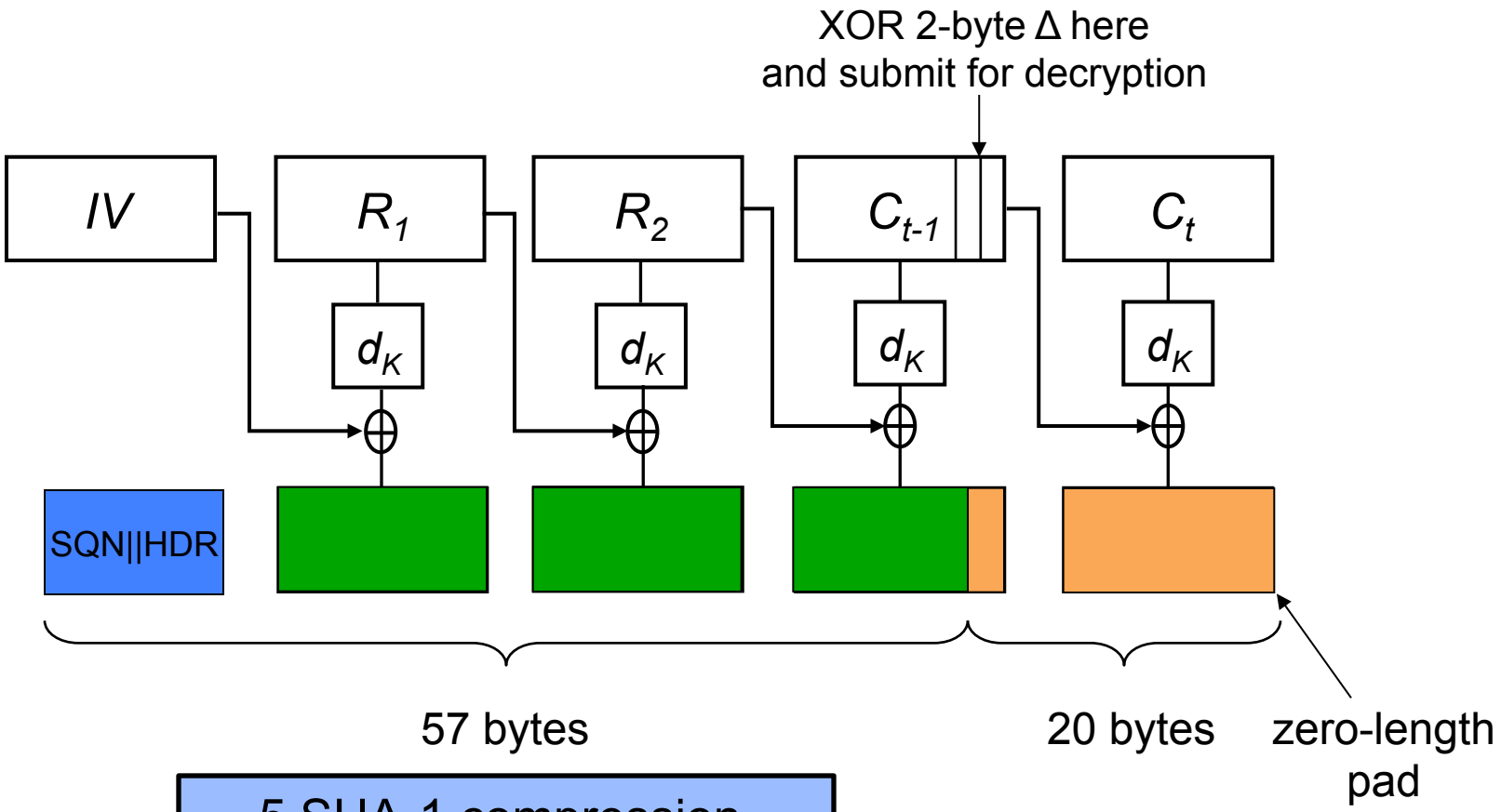


Case: “00”



5 SHA-1 compression function evaluations

Case: Bad padding



5 SHA-1 compression function evaluations

Lucky 13 – Plaintext Recovery

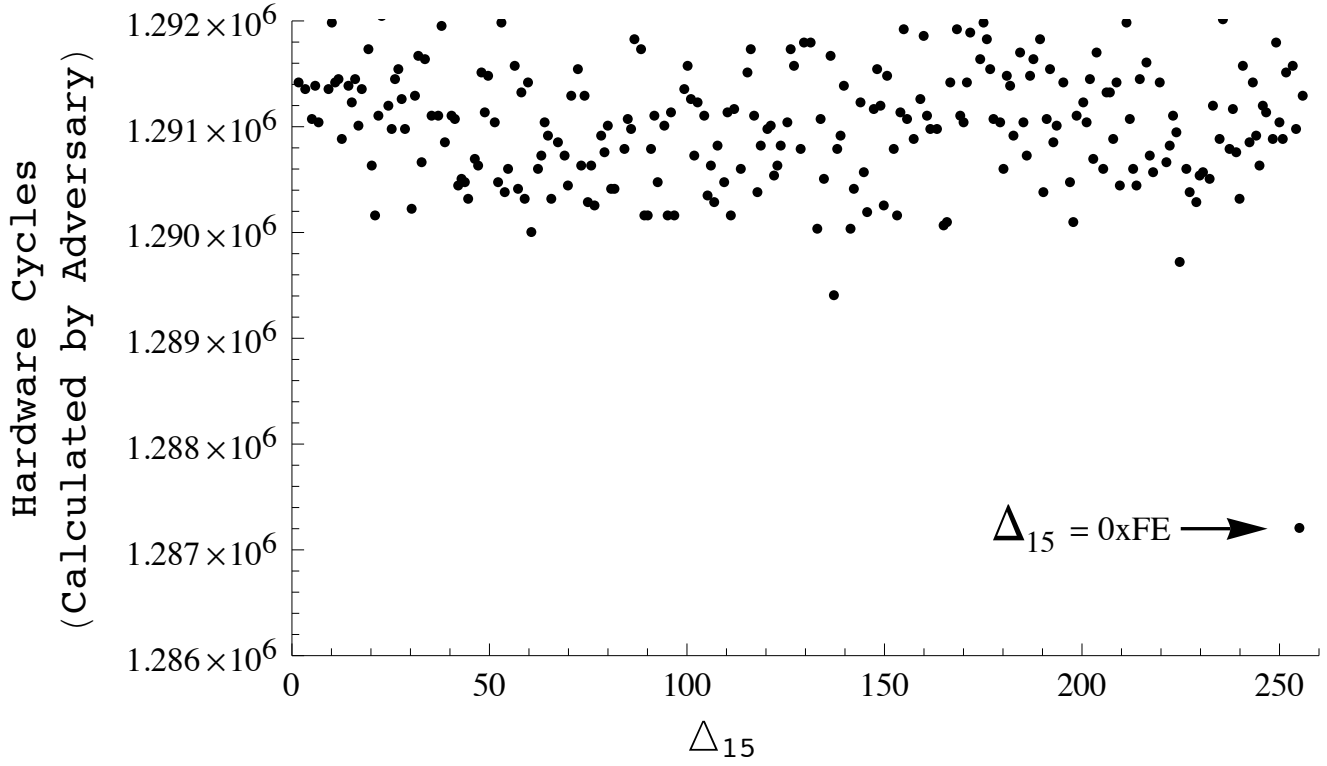
- The injected ciphertext causes bad padding and/or a bad MAC.
  - This leads to a TLS error message
  - The attacker times its appearance on the network.

- There is a timing *difference* between “01 01” case and the other 2 cases.
  - A single SHA-1 compression function evaluation.
  - Roughly 1000 clock cycles, 1μs range on typical processor.
  - Measurable difference on same host, LAN, or a few hops away.

- Detecting the “01 01” case allows last 2 plaintext bytes in the *target* block C_t to be recovered.
  - Using some standard CBC algebra.

- Attack then extends easily to all bytes.

Experimental Results



- Byte 14 of plaintext set to 01; byte 15 set to FF.
- OpenSSLv1.0.1 on server running at 1.87Ghz.
- 100 Mbit LAN.
- Median times (noise not shown).

Lucky 13 – Countermeasures

- We really need constant-time decryption for TLS-CBC.
- Add dummy hash compression function computations when padding is *good* to ensure total is the same as when padding is *bad*.
- Add dummy padding checks to ensure number of iterations done is independent of padding length and/or correctness of padding.
- Watch out for length sanity checks too.
  - Need to ensure there's enough space for *some* plaintext after removing padding and MAC, but without leaking any information about amount of padding removed.
- TL;DR:
  - It's a bit of a nightmare.

Lucky 13 – Impact

- **OpenSSL** patched in versions 1.0.1d, 1.0.0k and 0.9.8y, released 05/02/2013.
- **NSS** (Firefox, Chrome) patched in version 3.14.3, released 15/02/2013.
- **Opera** patched in version 12.13, released 30/01/2013
- **Oracle** released a special critical patch update of JavaSE, 19/02/2013.
- **BouncyCastle** patched in version 1.48, 10/02/2013
- Also **GnuTLS**, **PolarSSL**, **CyaSSL**, **MatrixSSL**.
- **Microsoft** "determined that the issue had been adequately addressed in previous modifications to their TLS and DTLS implementation".
- **Apple:** status unknown.

(Full details at: www.isg.rhul.ac.uk/tls/lucky13.html)

Other Lucky 13 Countermeasures?

- Introduce random delays during decryption.
  - Surprisingly *in*effective, analysis in [AP13].
- Redesign TLS:
  - Pad-MAC-Encrypt or Pad-Encrypt-MAC.
  - Currently, some discussion on TLS mailing lists.
  - TLS 1.3?
- Switch to TLS 1.2
  - Has support for AES-GCM and AES-CCM.
  - Some encouraging signs of increasing adoption.
- Use RC4?

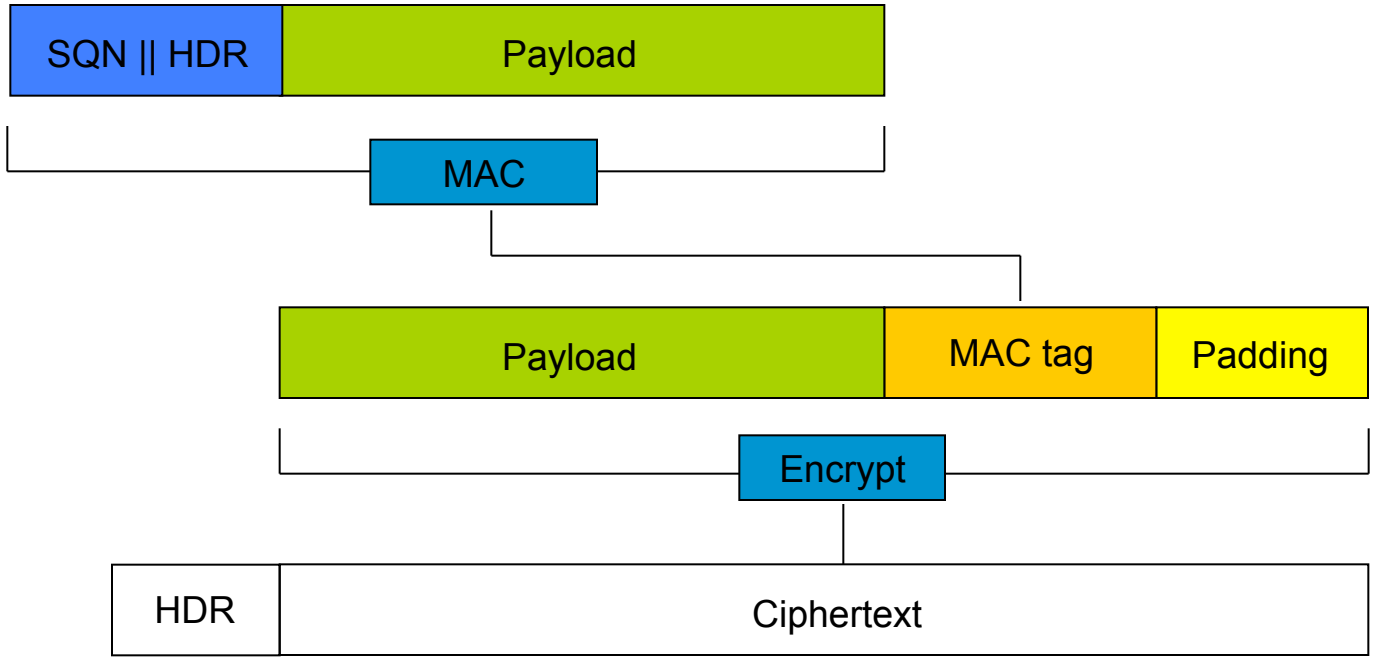
Outline

- TLS overview
- TLS attacks and proofs
- Lucky 13
- TLS Record Protocol and RC4*
- Discussion

*AlFardan, Bernstein, Paterson, Poettering, Schudlt, *On the Security of RC4 in TLS.* USENIX Security Symposium, 2013.

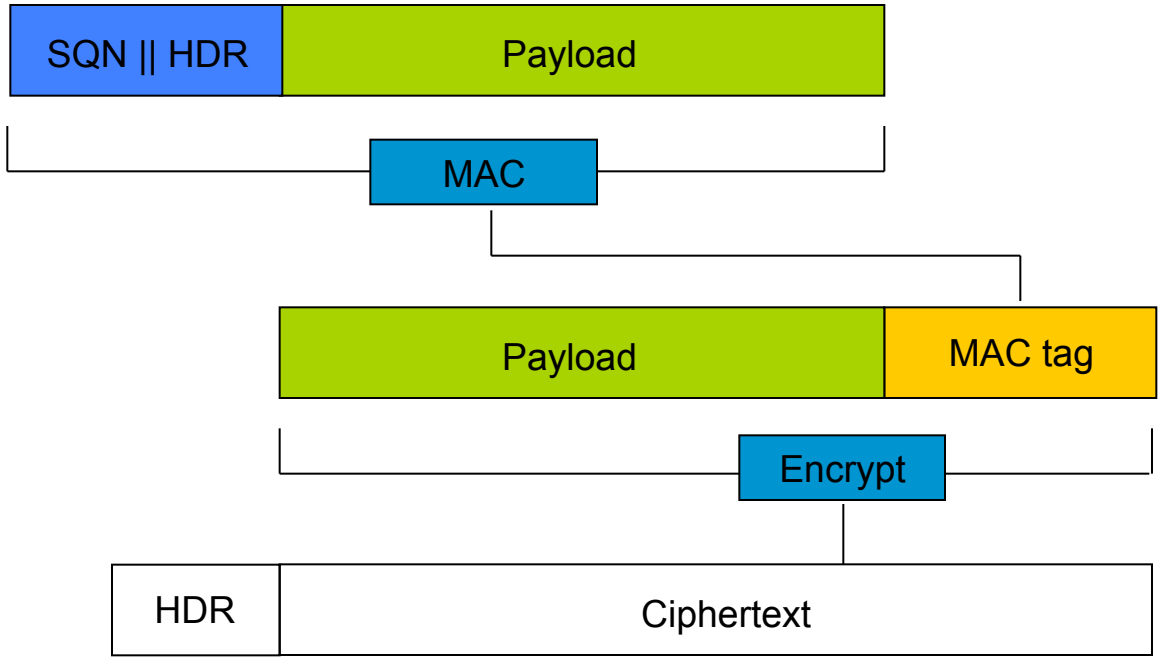
(www.isg.rhul.ac.uk/tls)

TLS Record Protocol: MEE



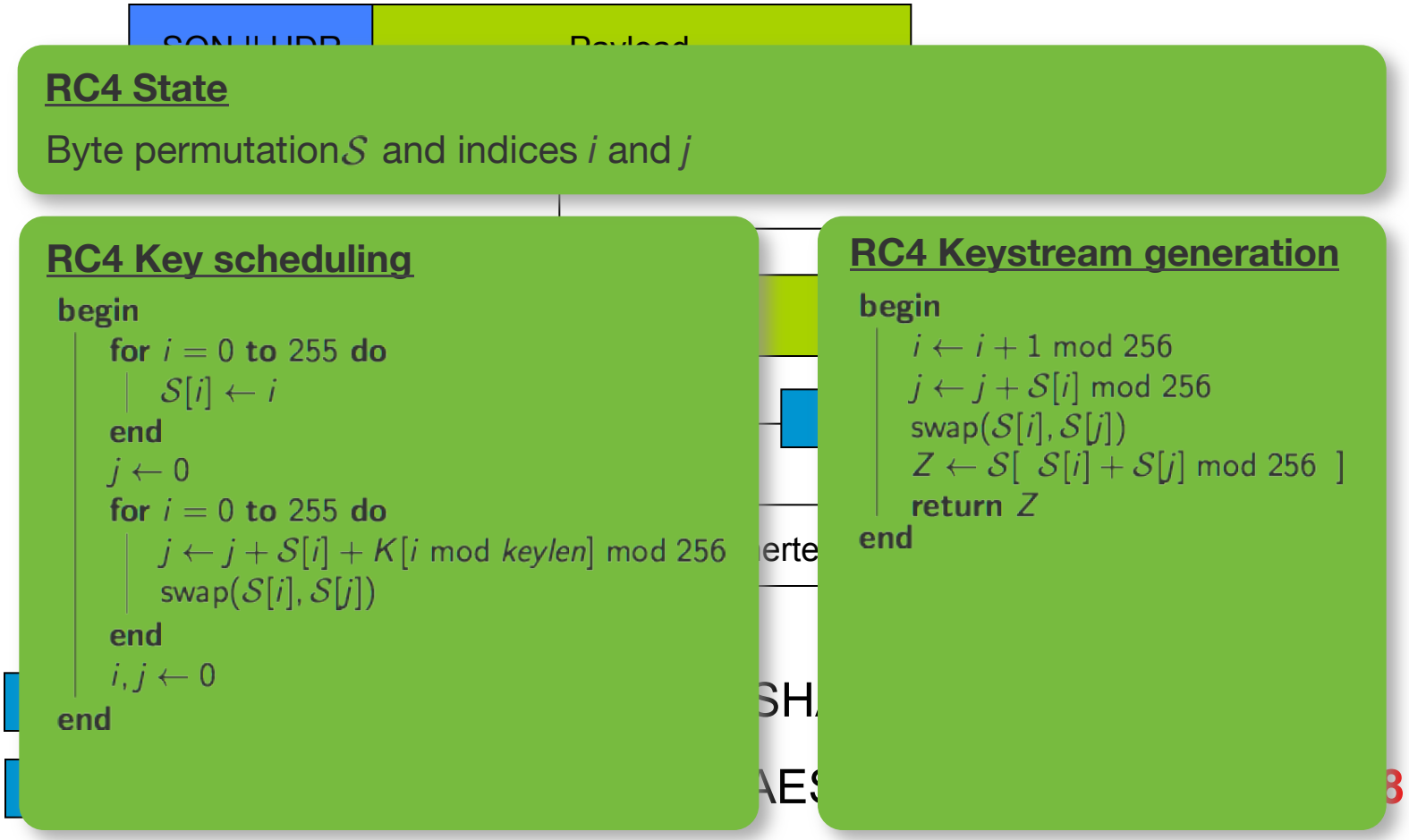
MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128
Padding	“00” or “01 01” or “02 02 02” or …. or “FF FF….FF”

TLS Record Protocol: RC4-128



MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, **RC4-128**

TLS Record Protocol: RC4-128



Use of RC4 in TLS

- In the face of the attacks on CBC-based ciphersuites in TLS, switching to RC4 has been a recommended mitigation (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.
  - (RC4 widely considered weak because of WEP debacle.)

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]:

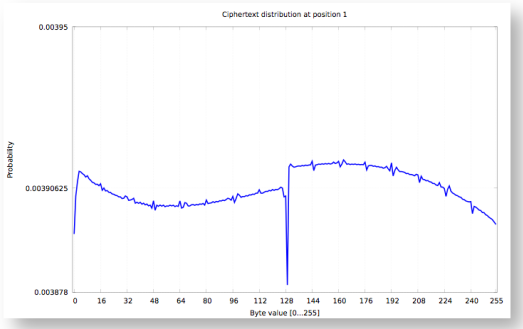
$$\Pr[Z_l = 256 - l] \geq \frac{1}{256} + \frac{1}{256^2}$$

l = keylength

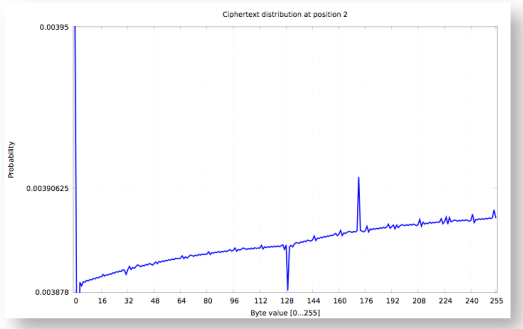
Complete Keystream Byte Distributions

- Our approach in [ABPPS13]:
  - Based on the output from 2^{45} random independent 128-bit RC4 keys, estimate the keystream byte distribution of the first 256 bytes

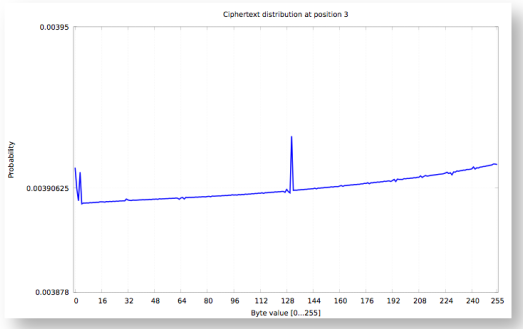
    ..



Z_1



Z_2

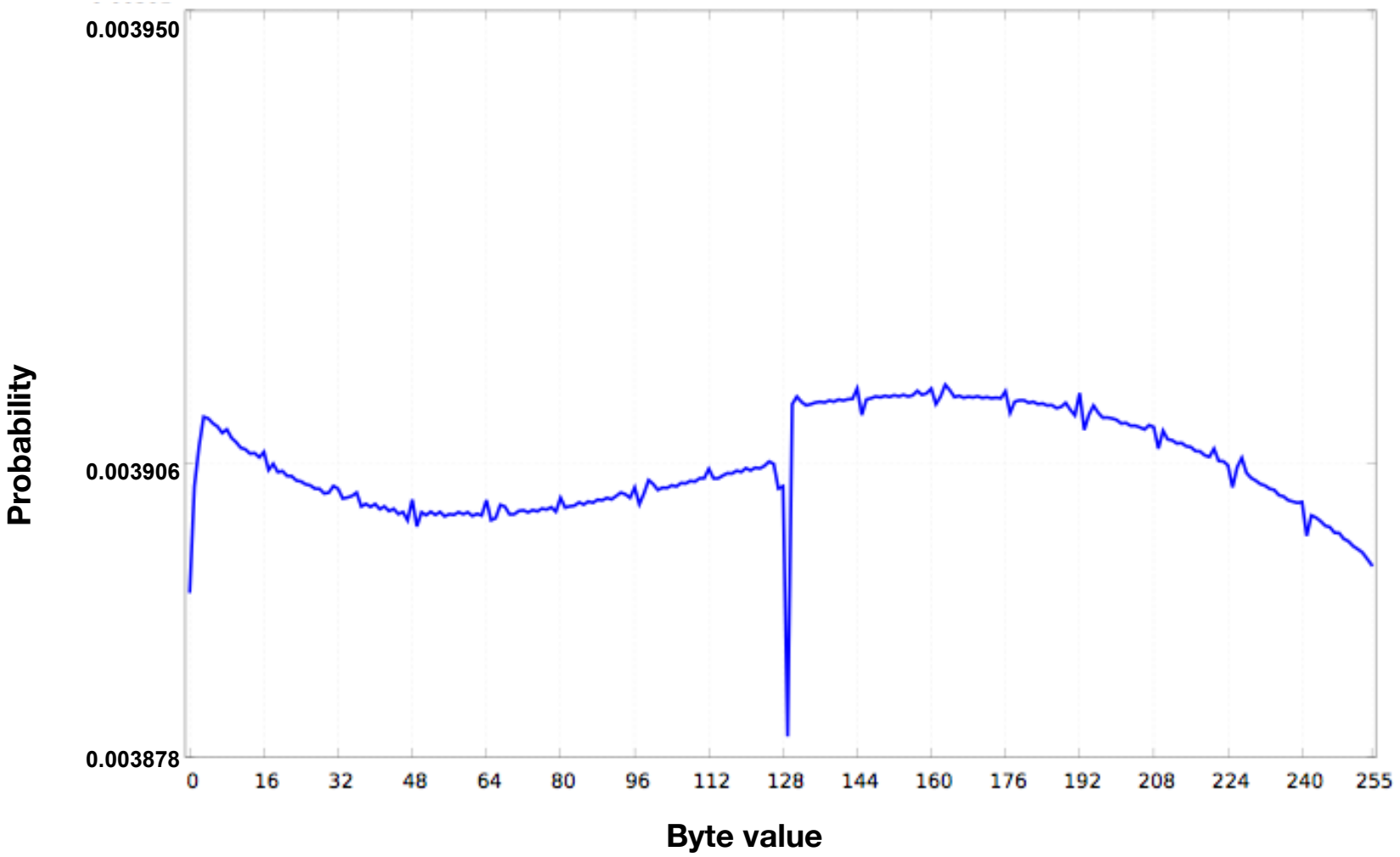


Z_3

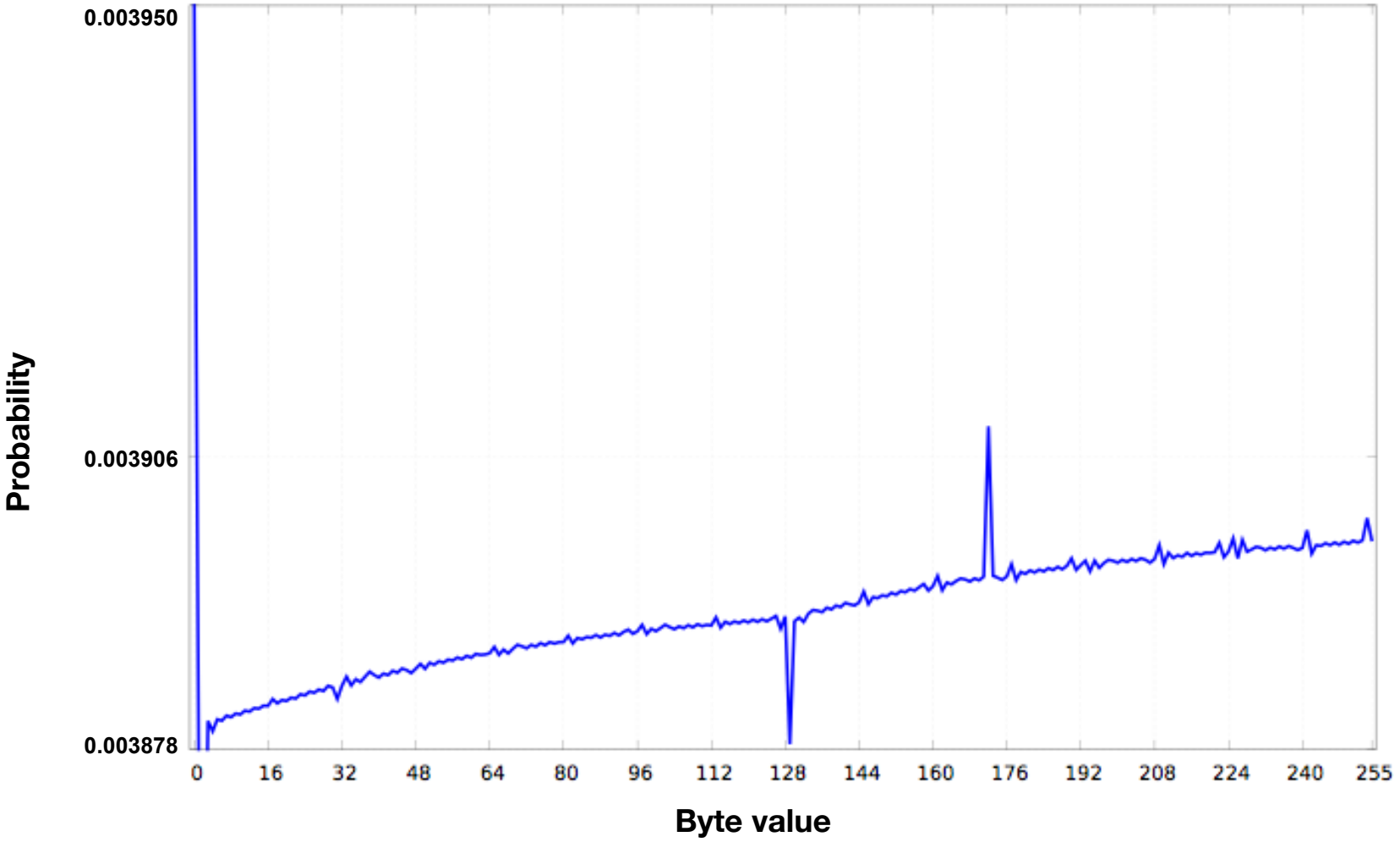
... ...

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

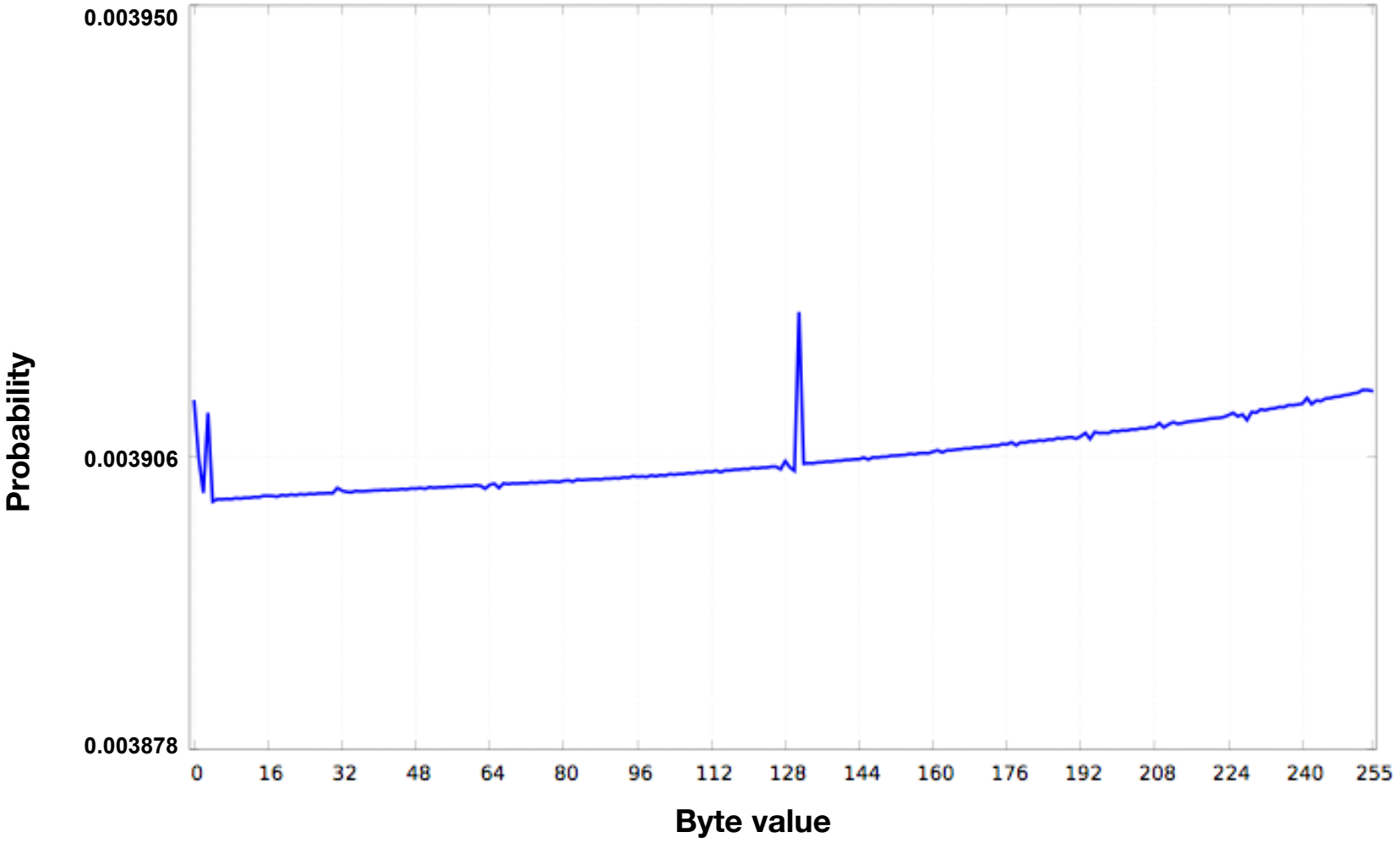
Keystream Distribution at Position 1



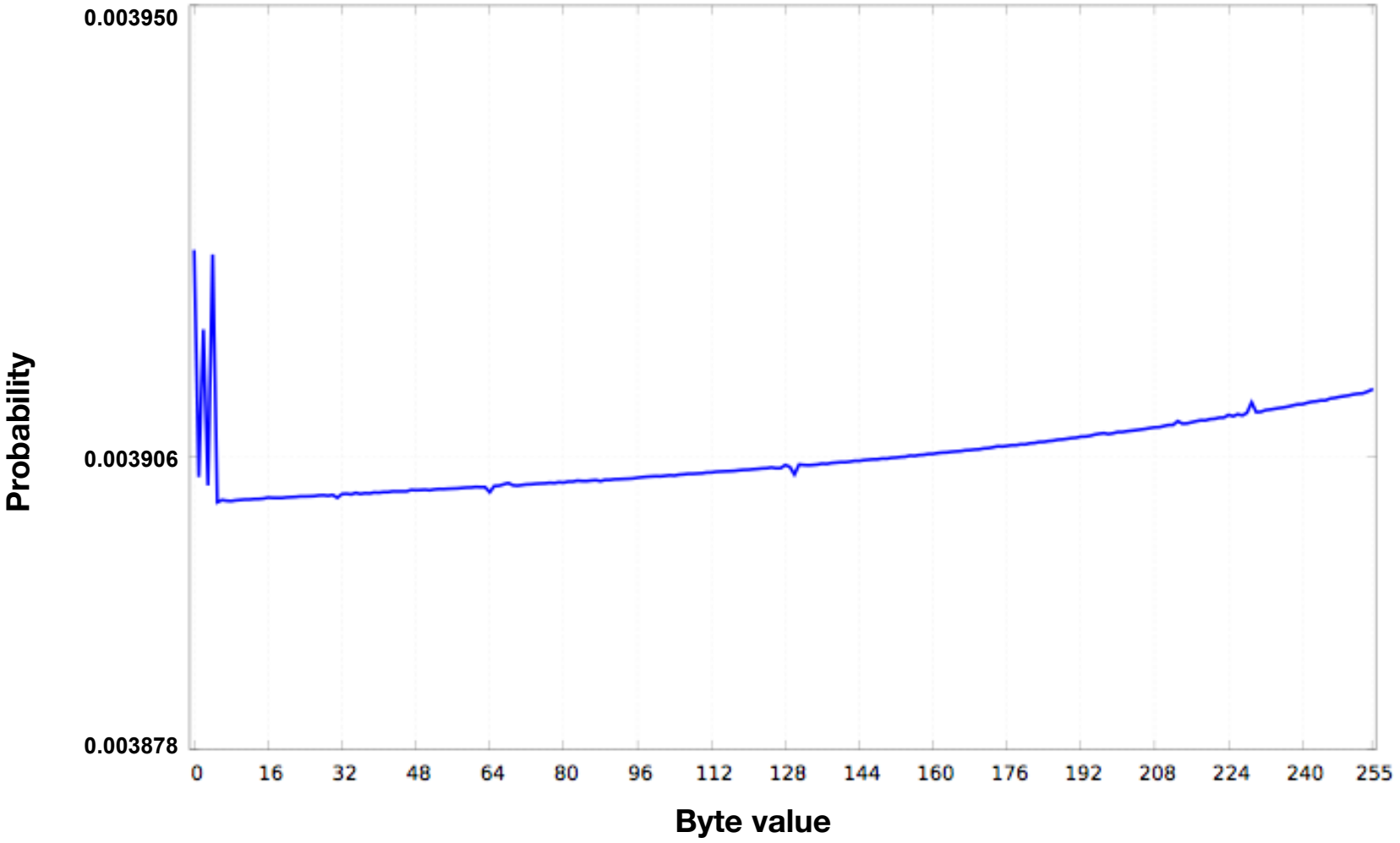
Keystream Distribution at Position 2



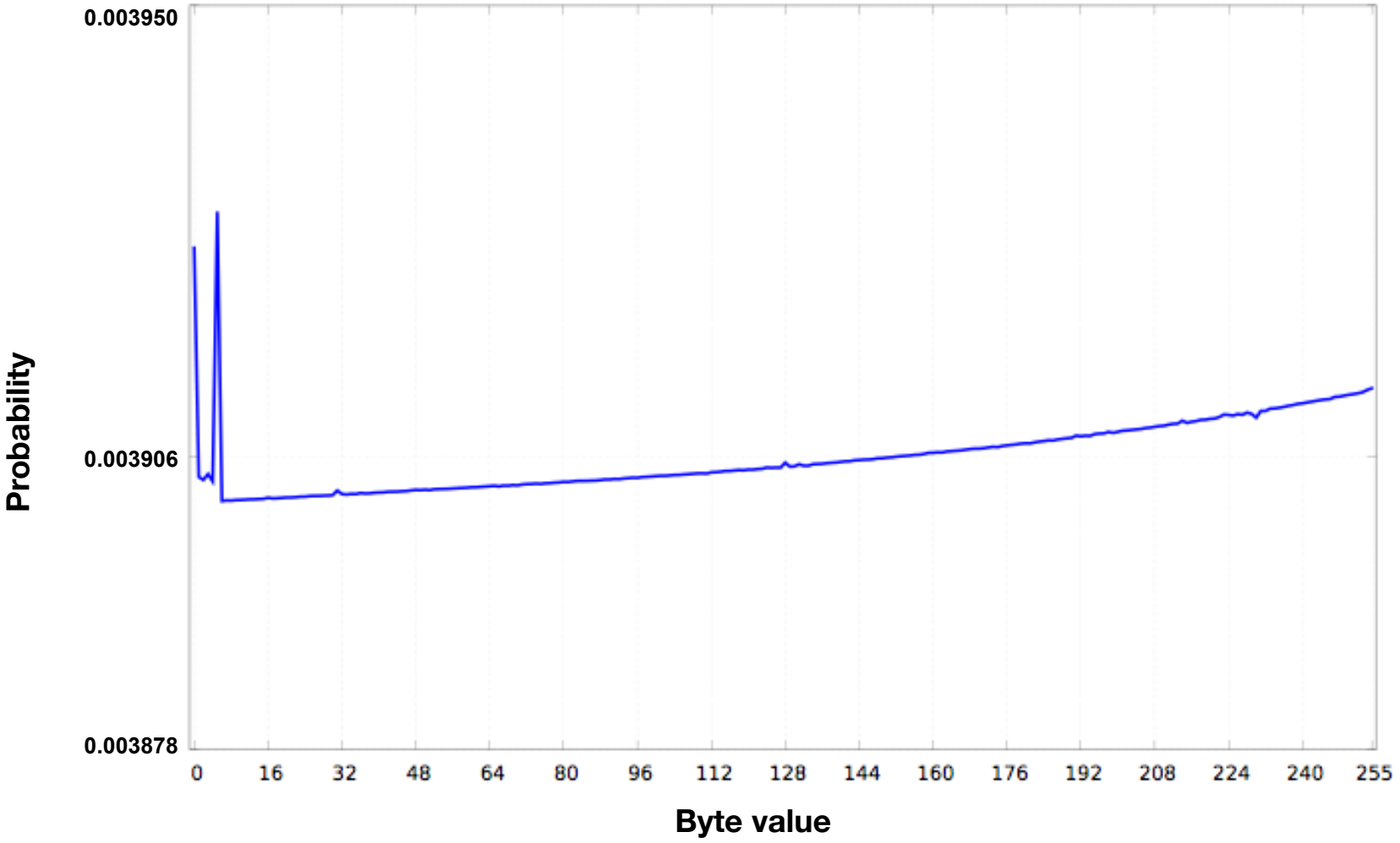
Keystream Distribution at Position 3



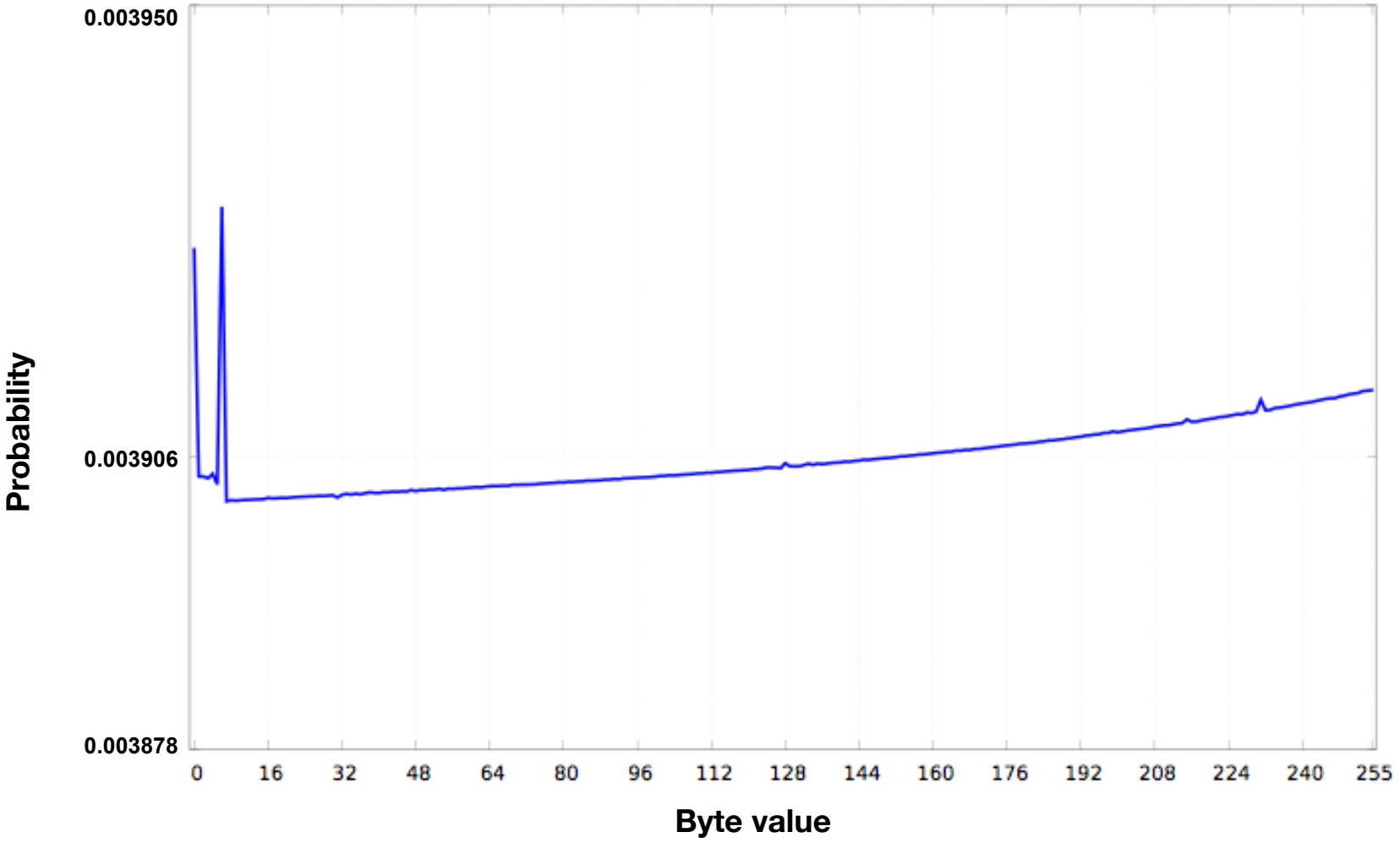
Keystream Distribution at Position 4



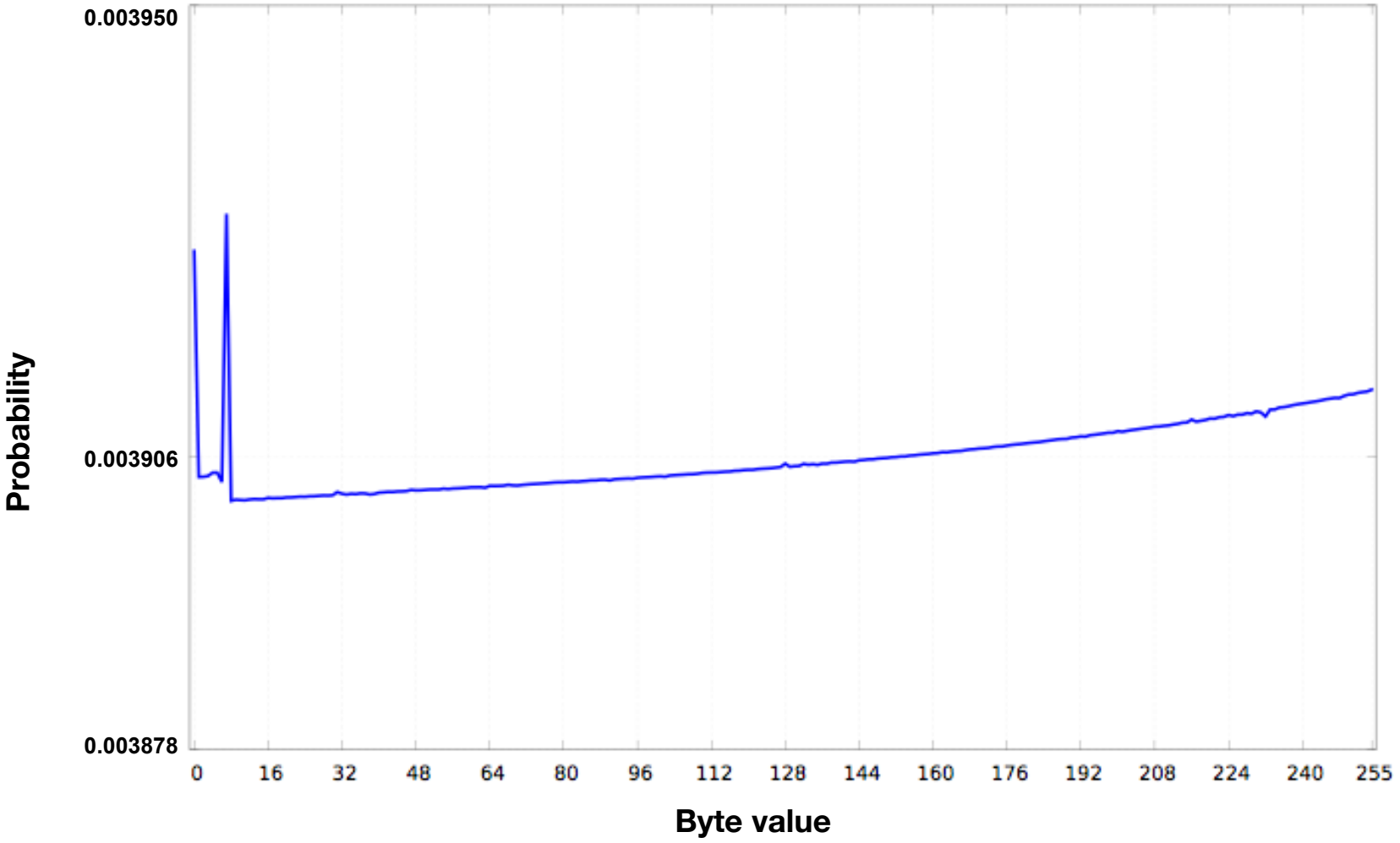
Keystream Distribution at Position 5



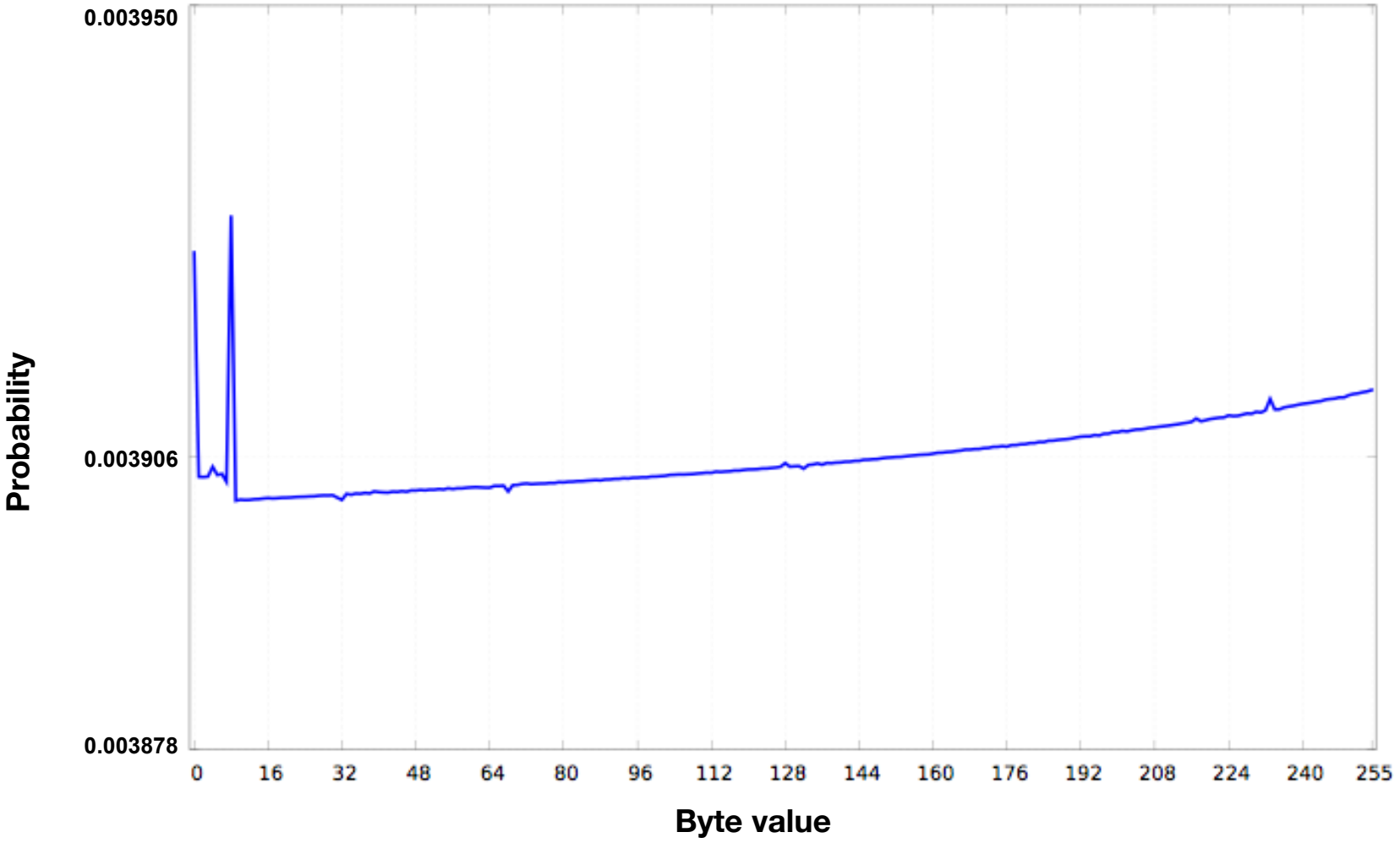
Keystream Distribution at Position 6



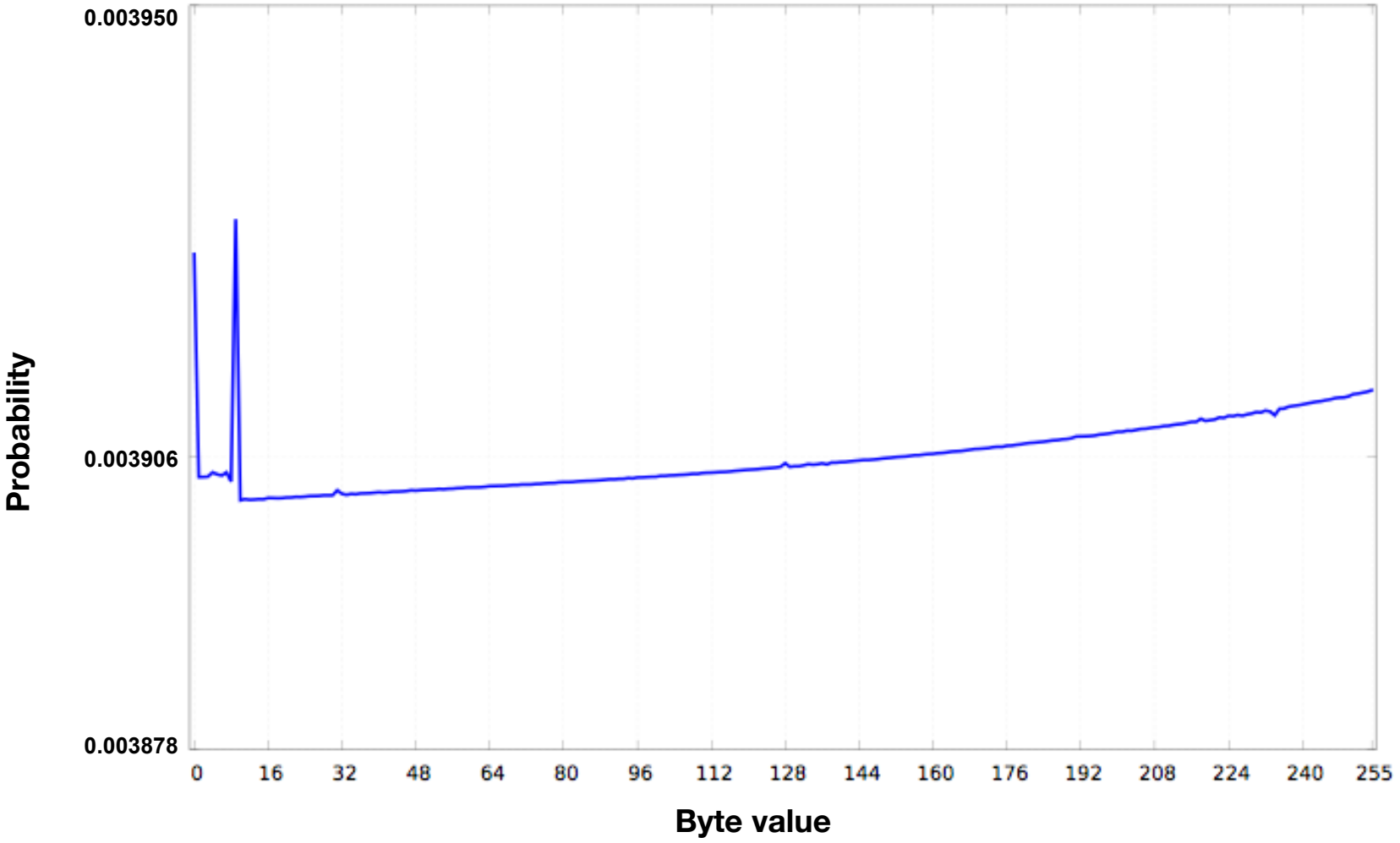
Keystream Distribution at Position 7



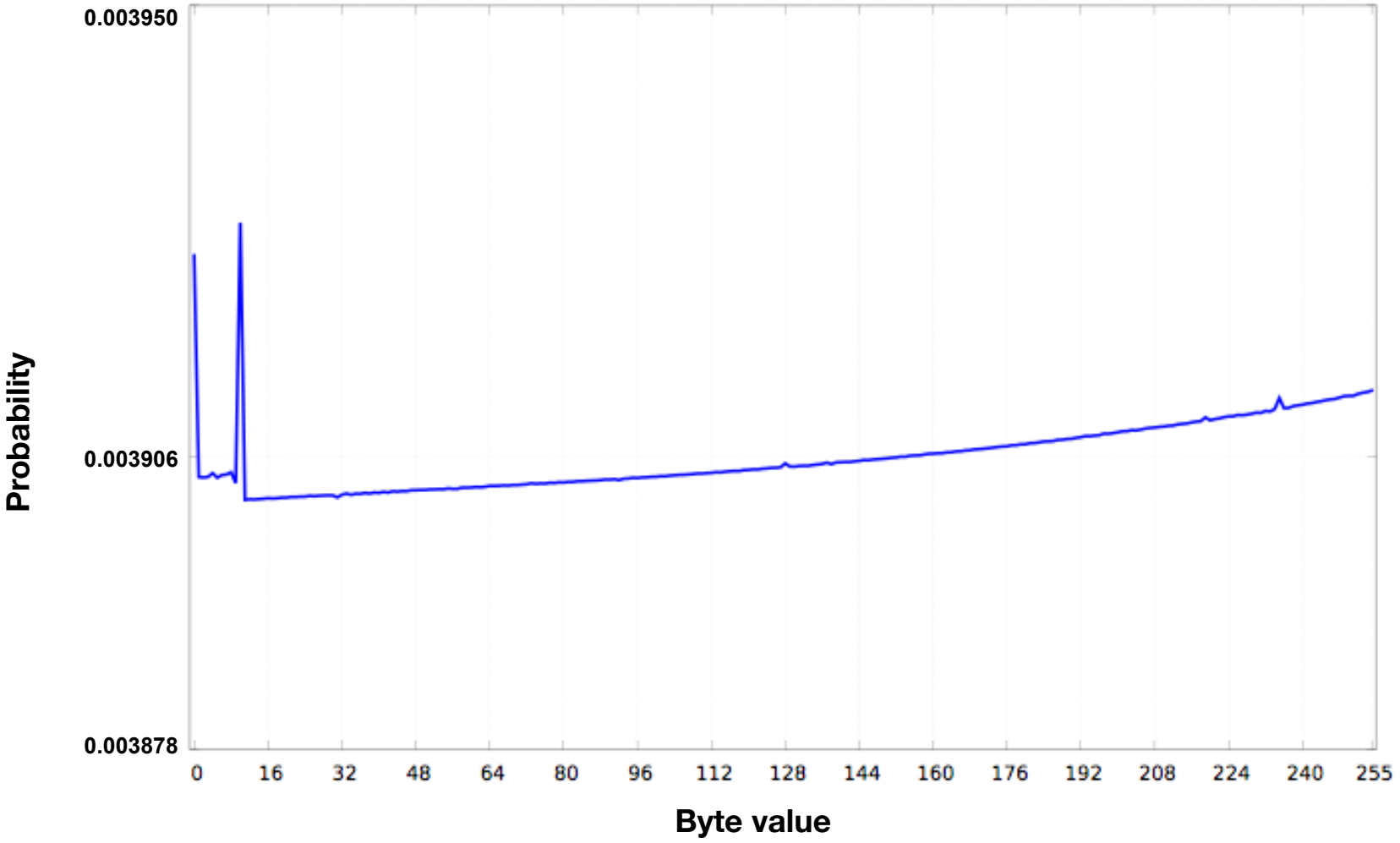
Keystream Distribution at Position 8



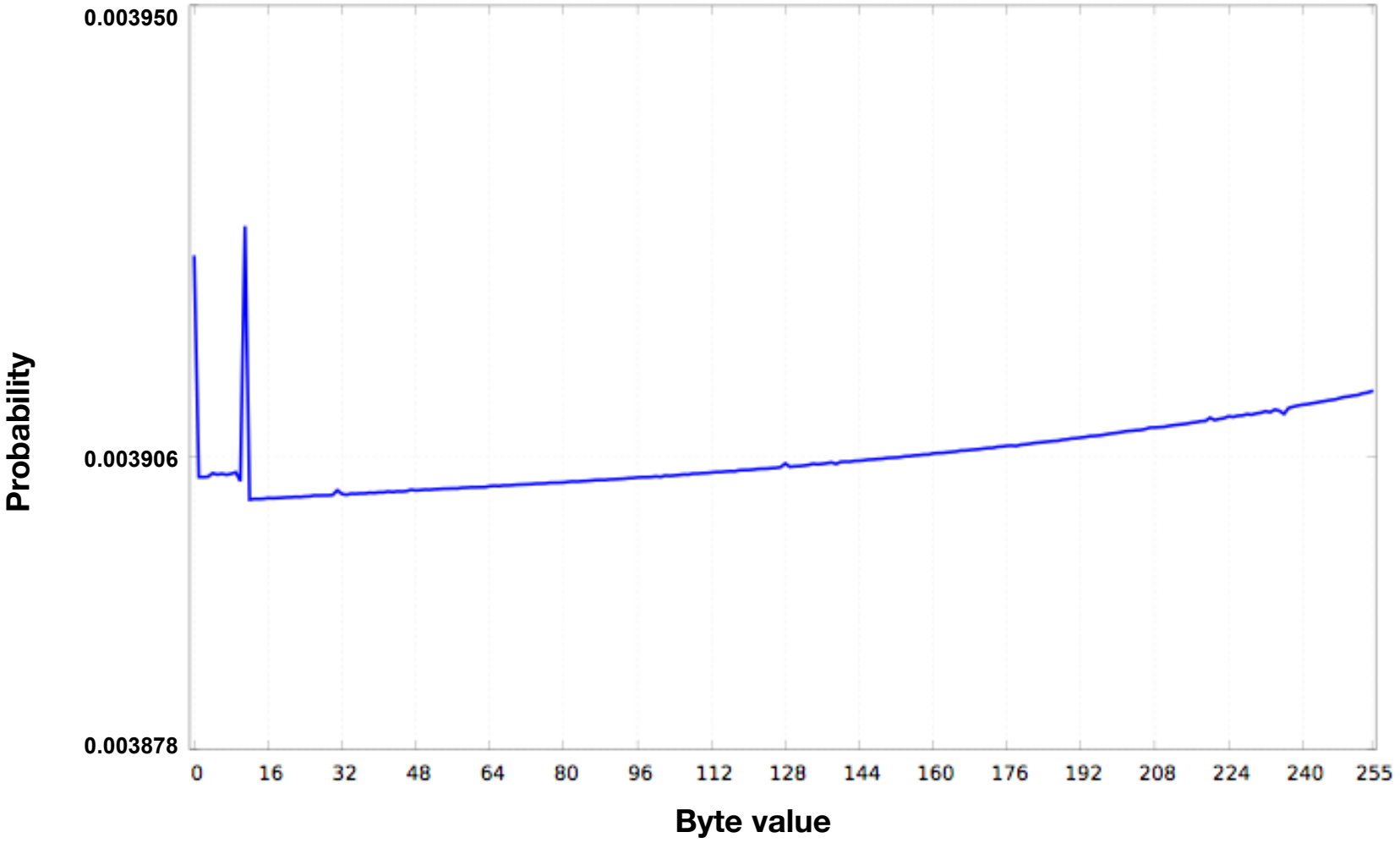
Keystream Distribution at Position 9



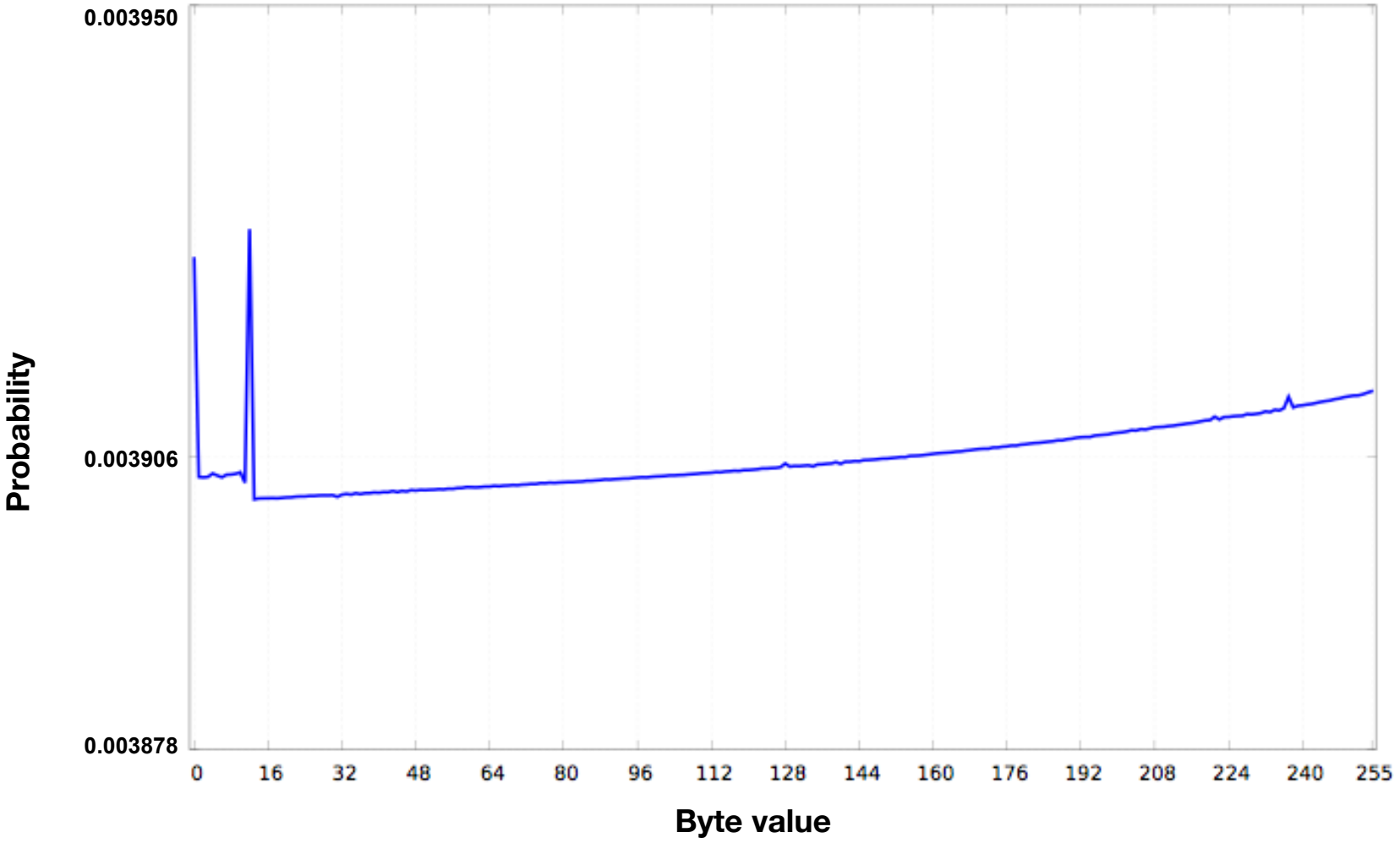
Keystream Distribution at Position 10



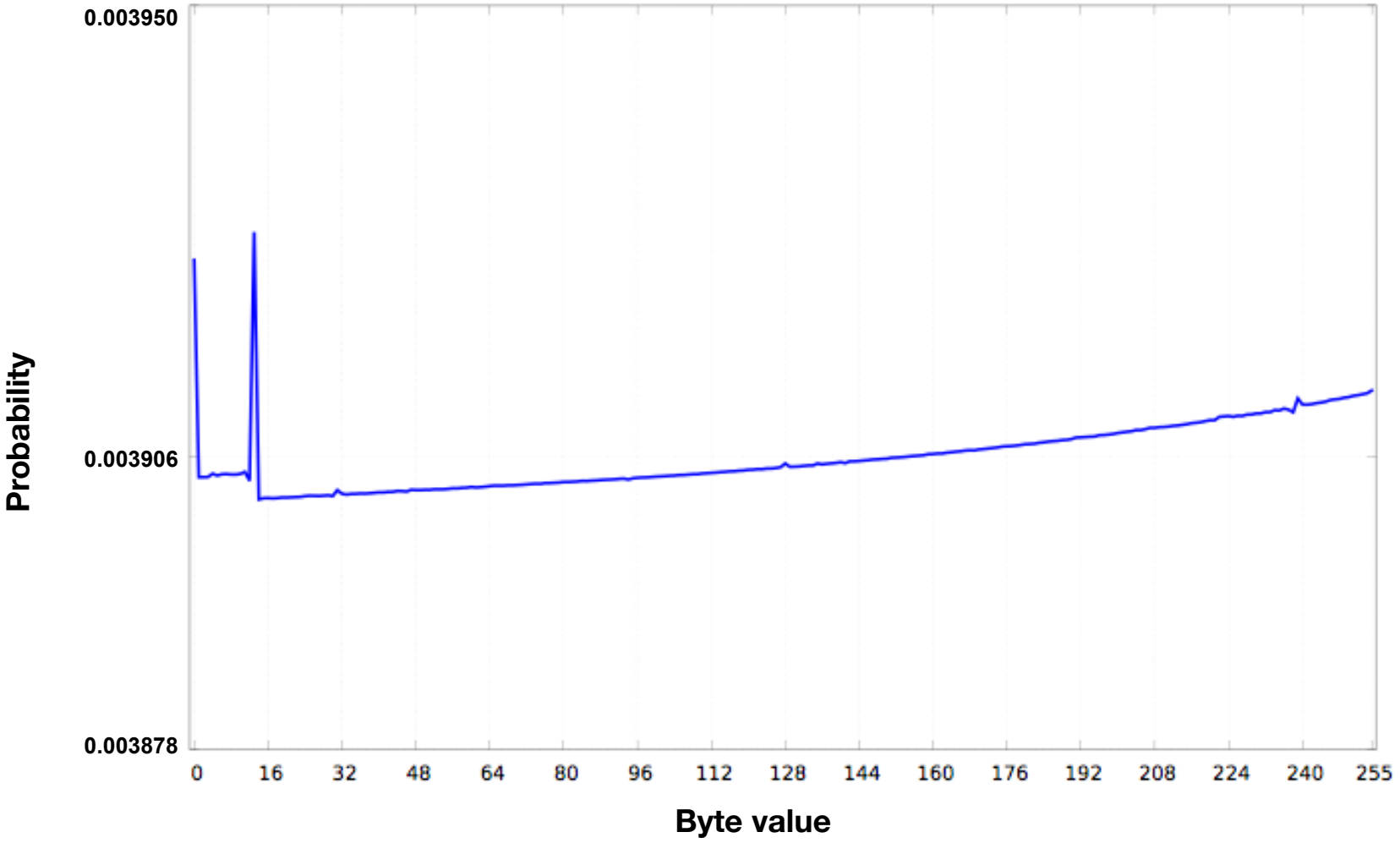
Keystream Distribution at Position 11



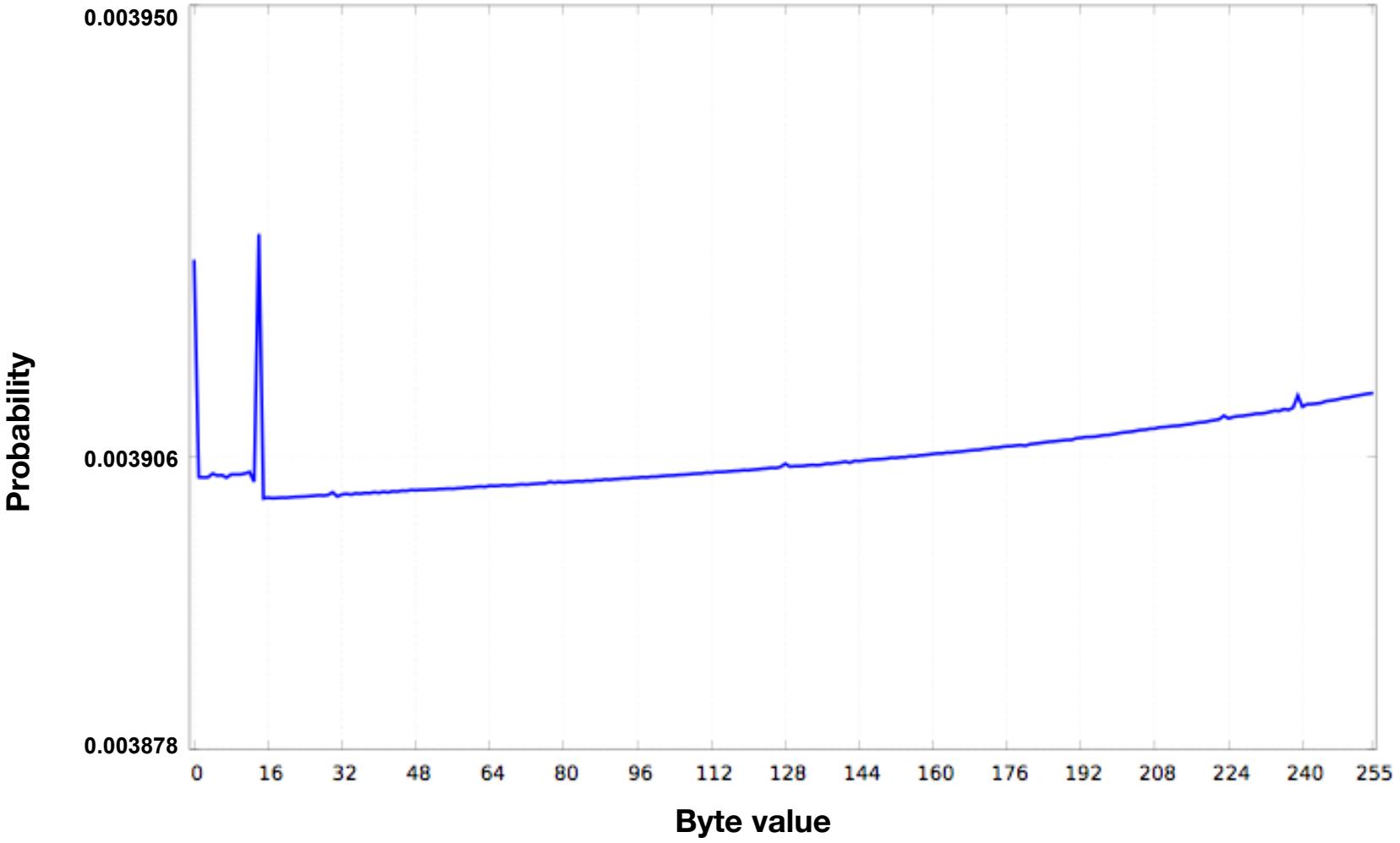
Keystream Distribution at Position 12



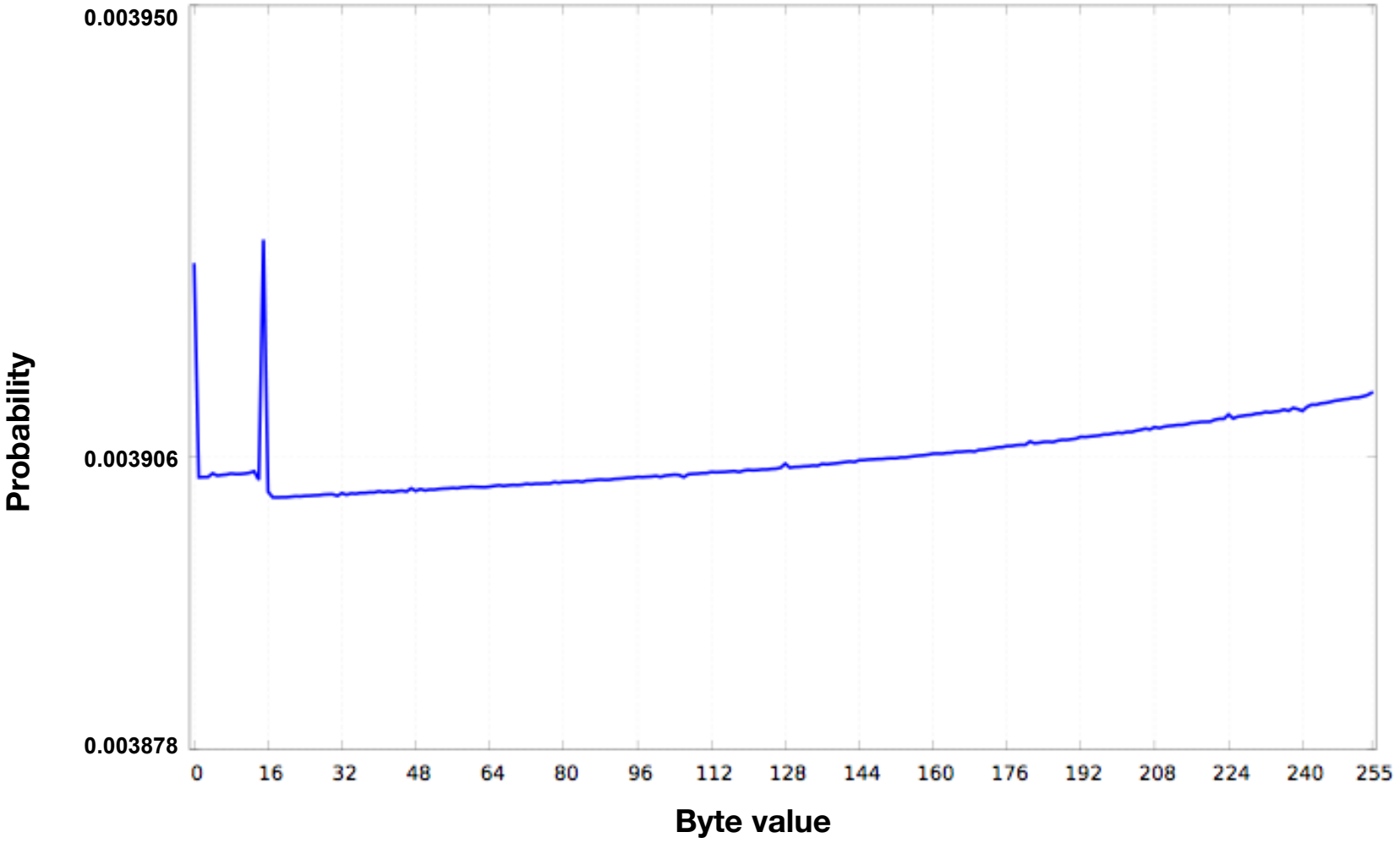
Keystream Distribution at Position 13



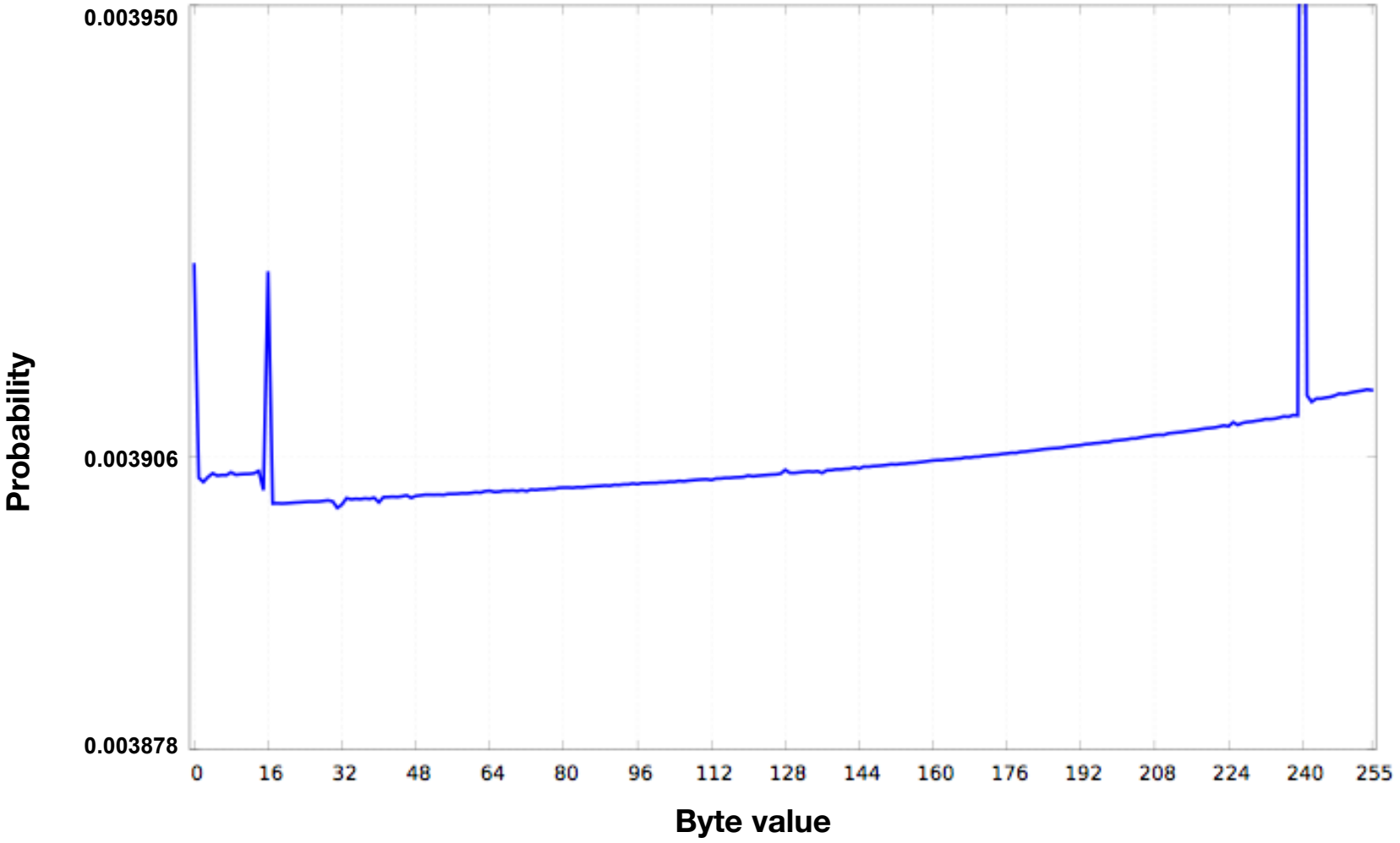
Keystream Distribution at Position 14



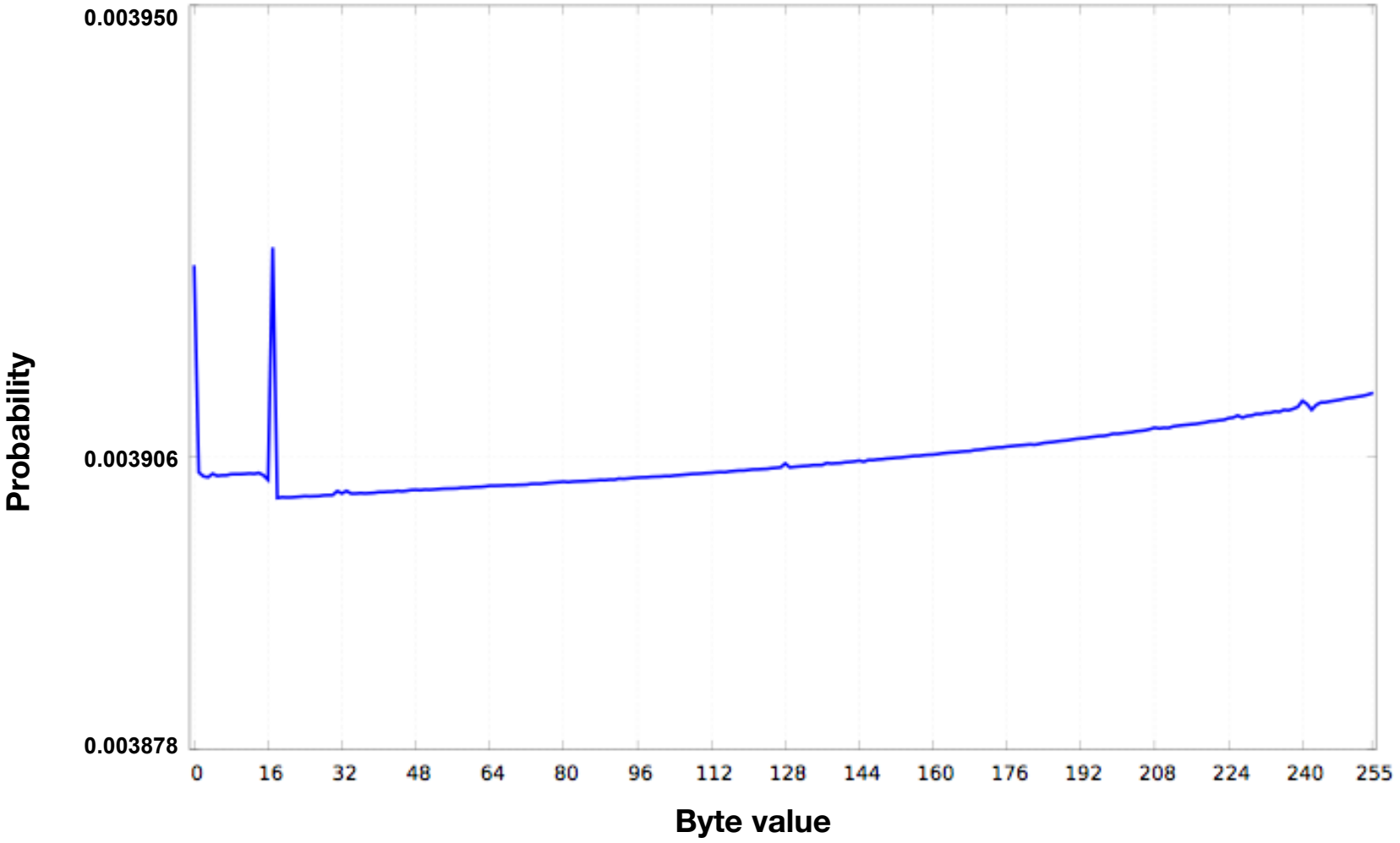
Keystream Distribution at Position 15



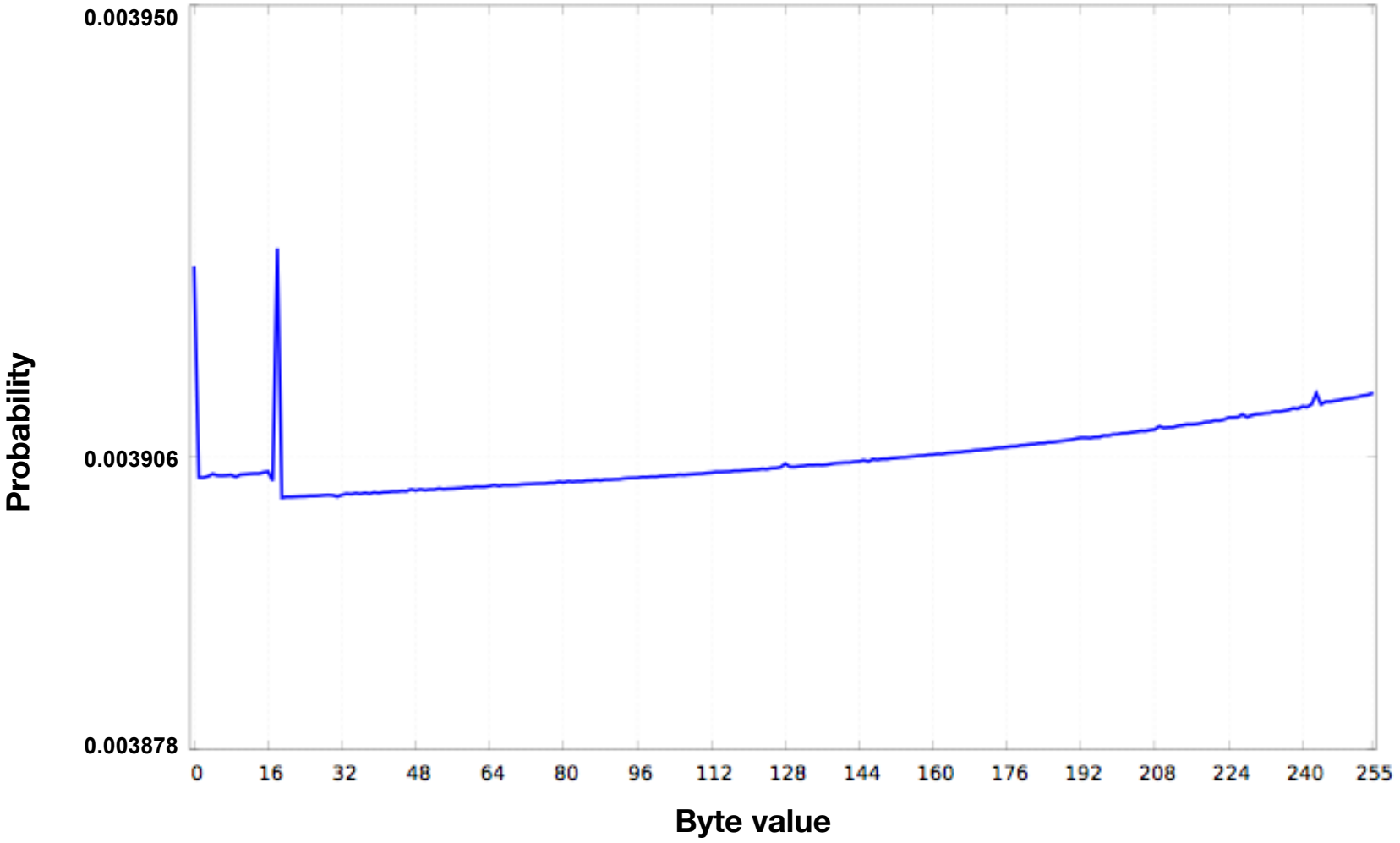
Keystream Distribution at Position 16



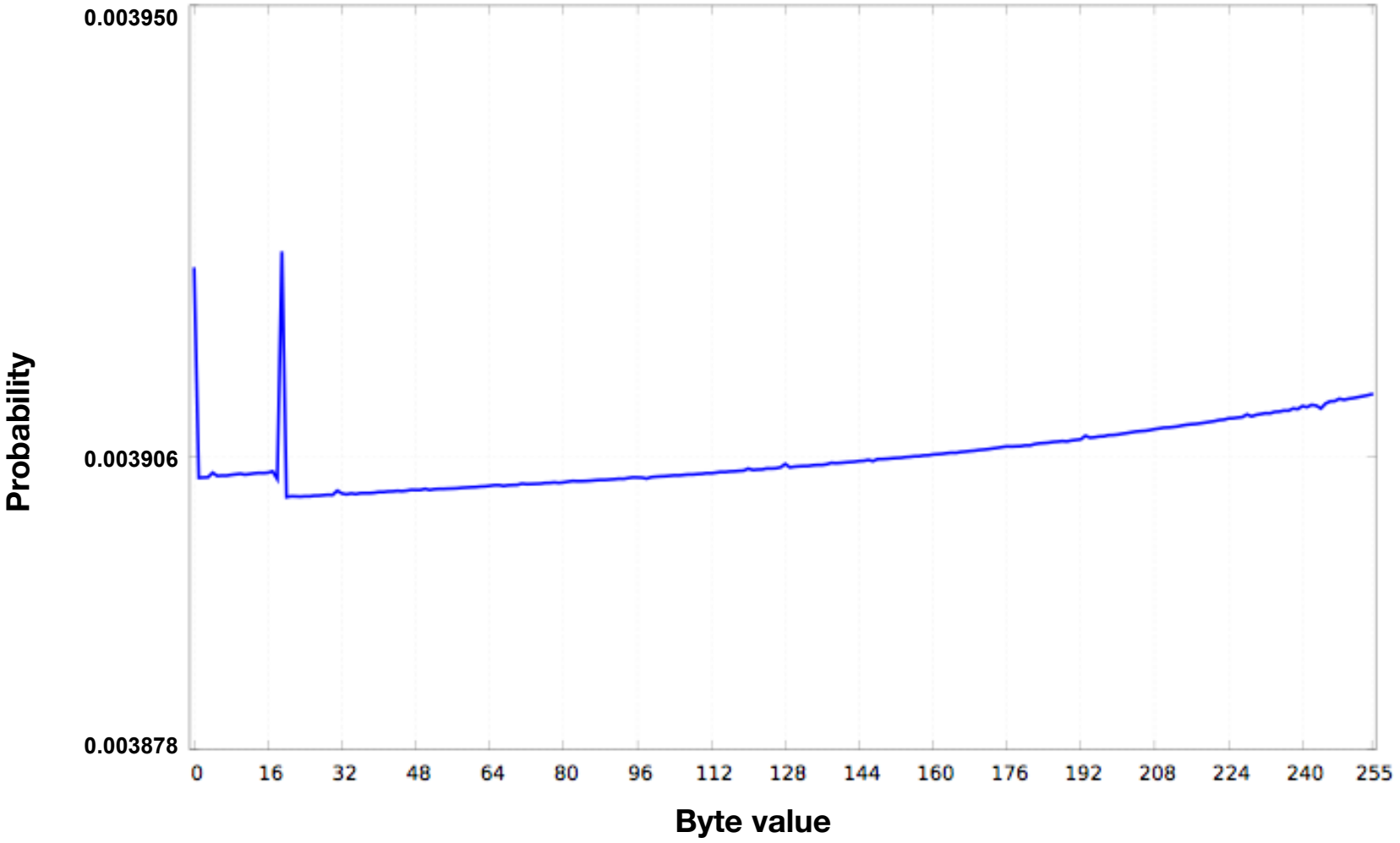
Keystream Distribution at Position 17



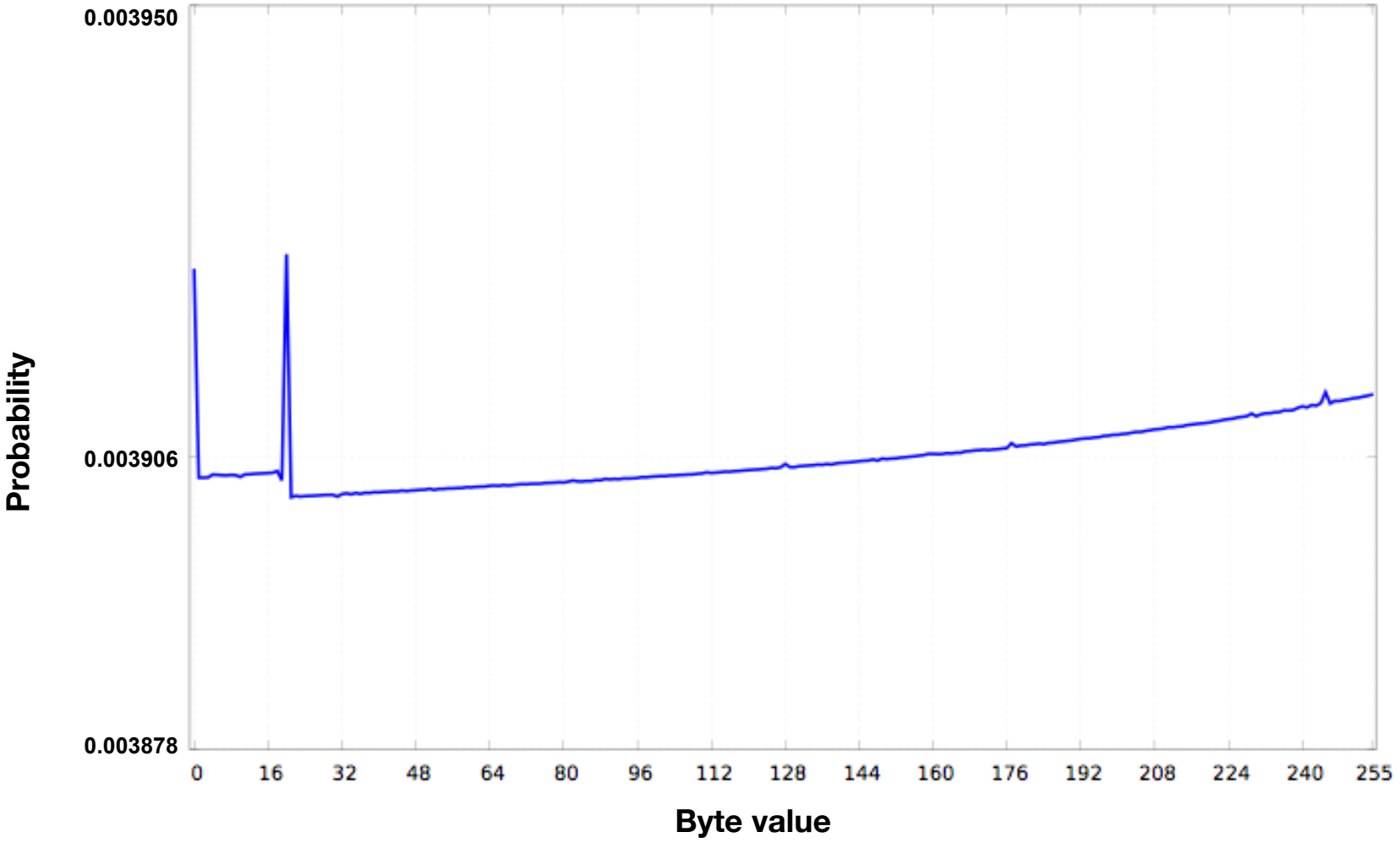
Keystream Distribution at Position 18



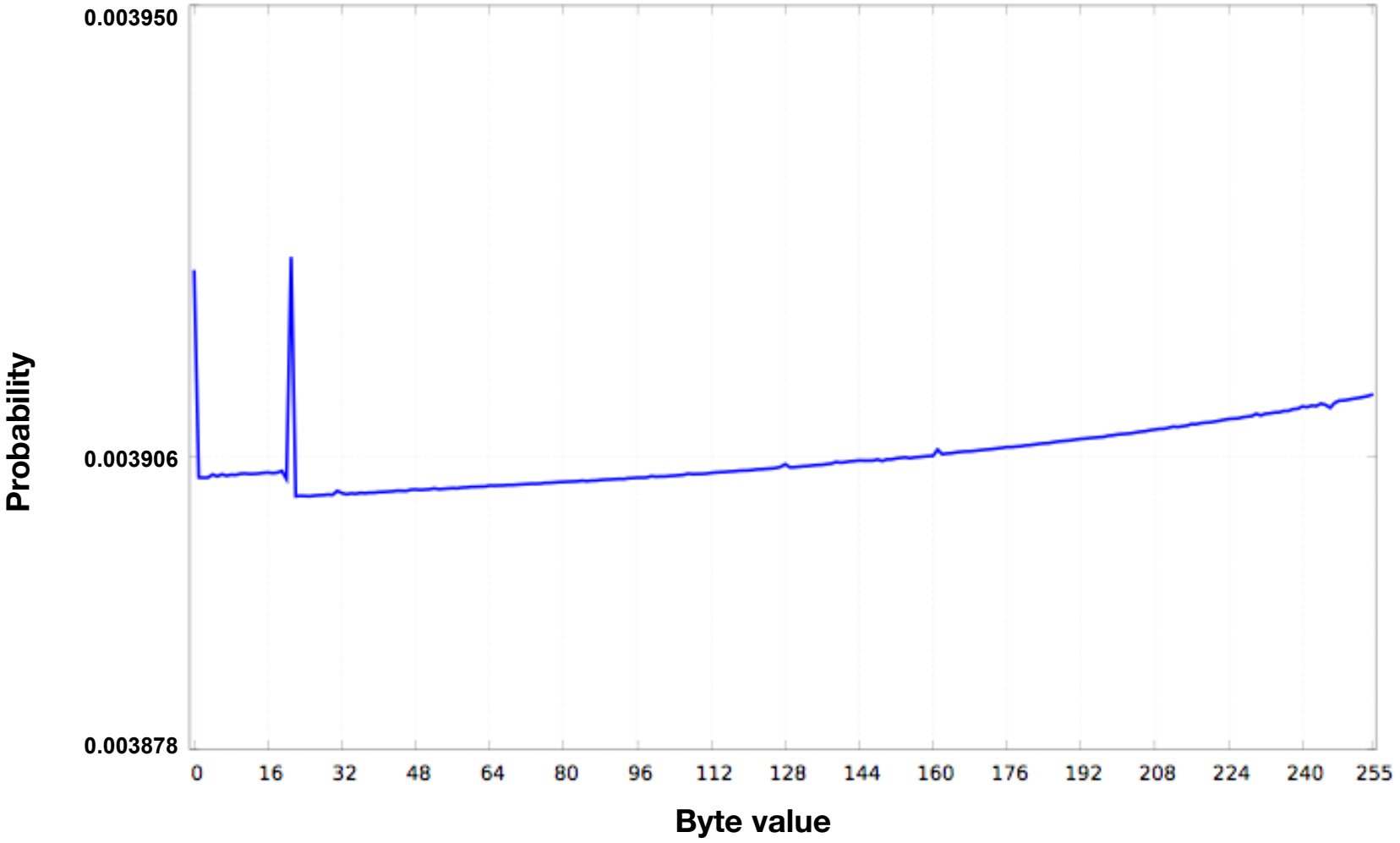
Keystream Distribution at Position 19



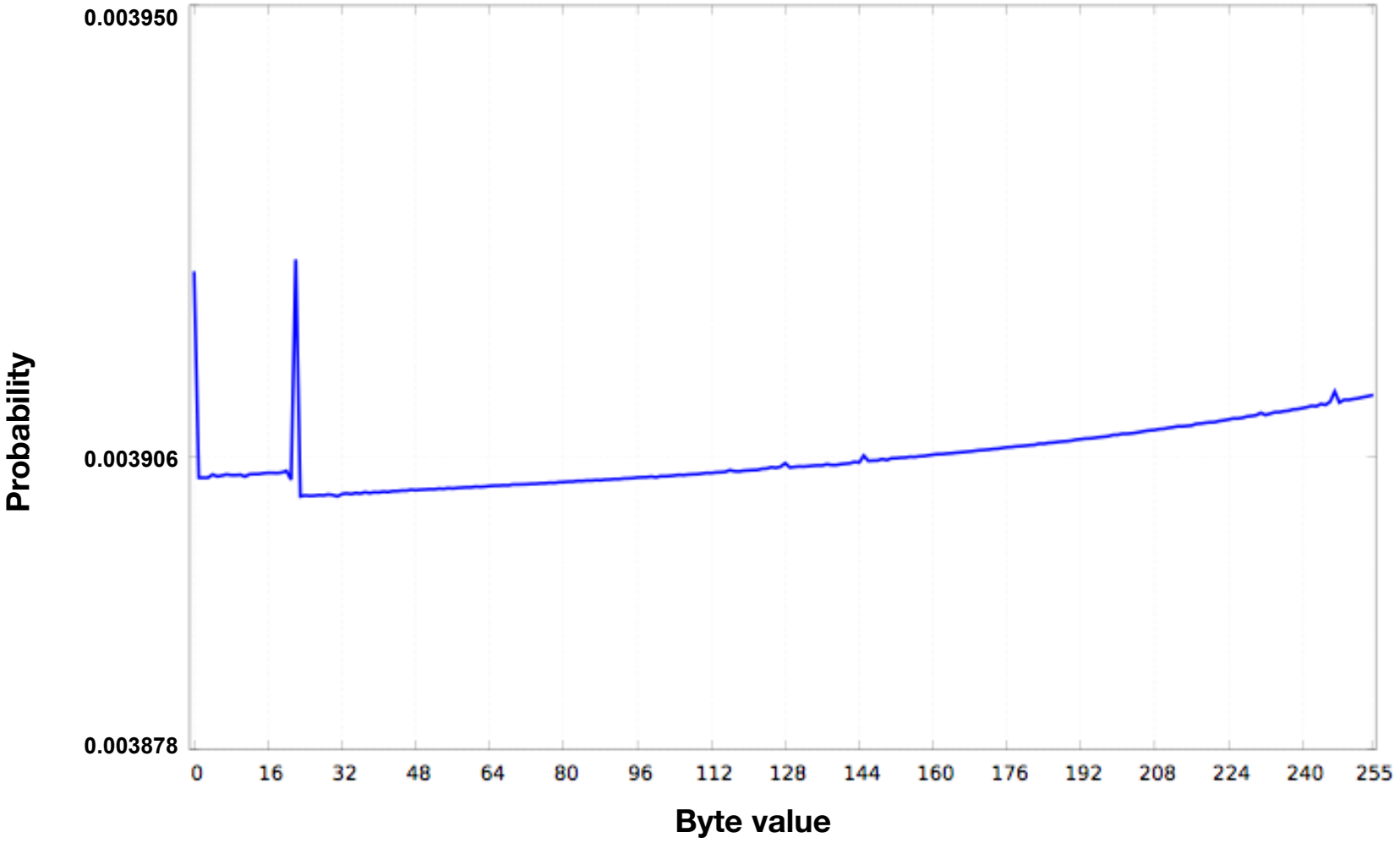
Keystream Distribution at Position 20



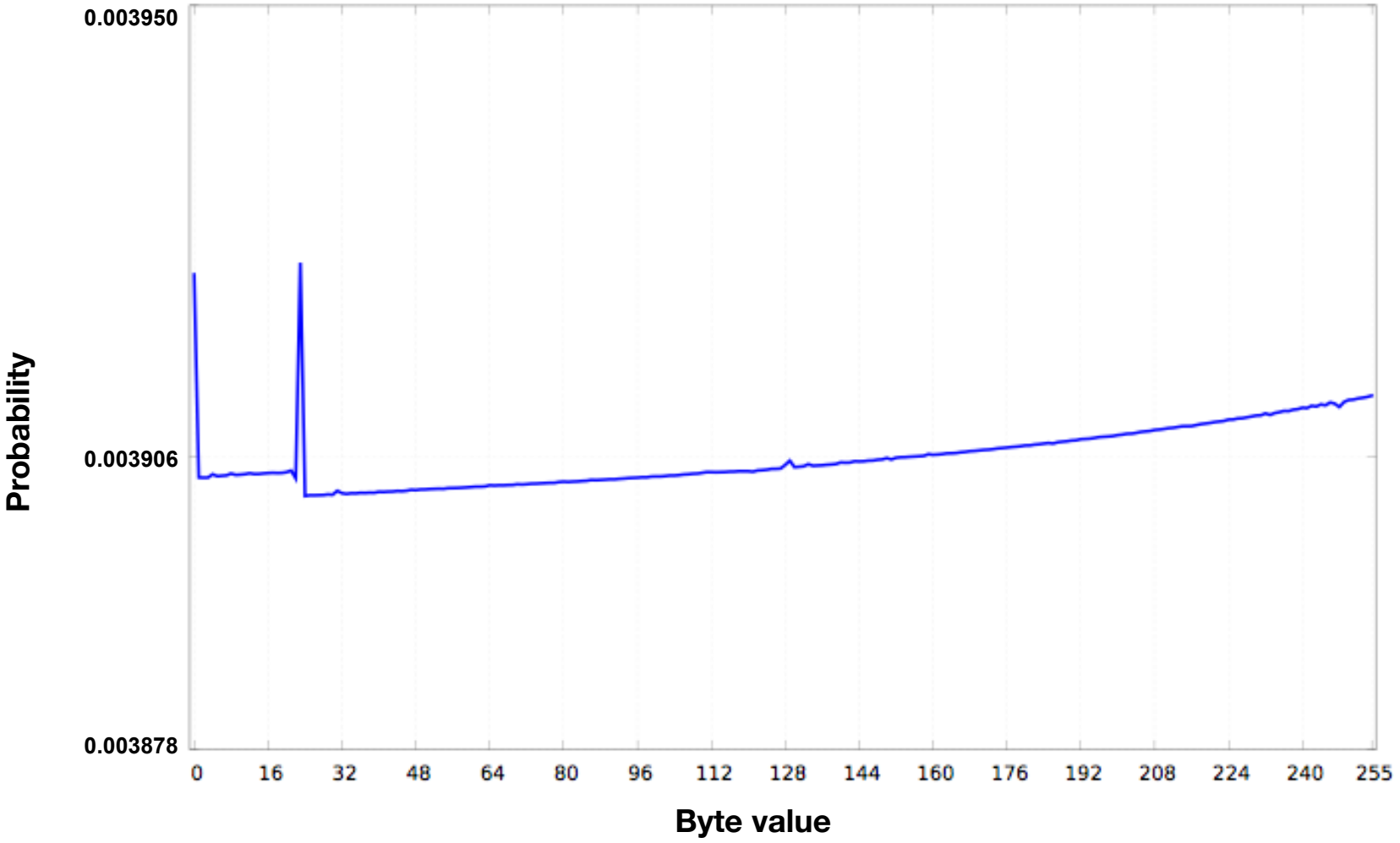
Keystream Distribution at Position 21



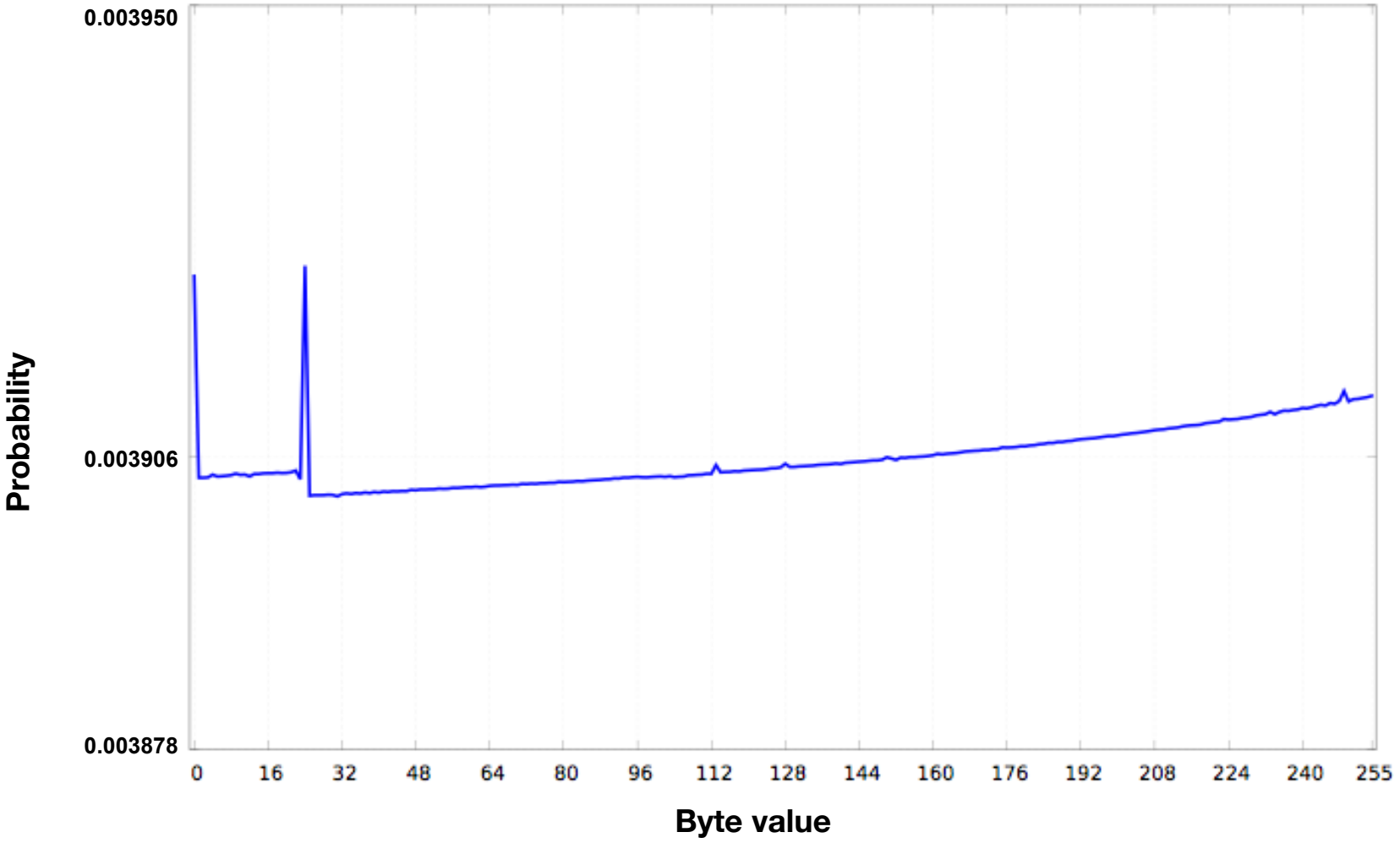
Keystream Distribution at Position 22



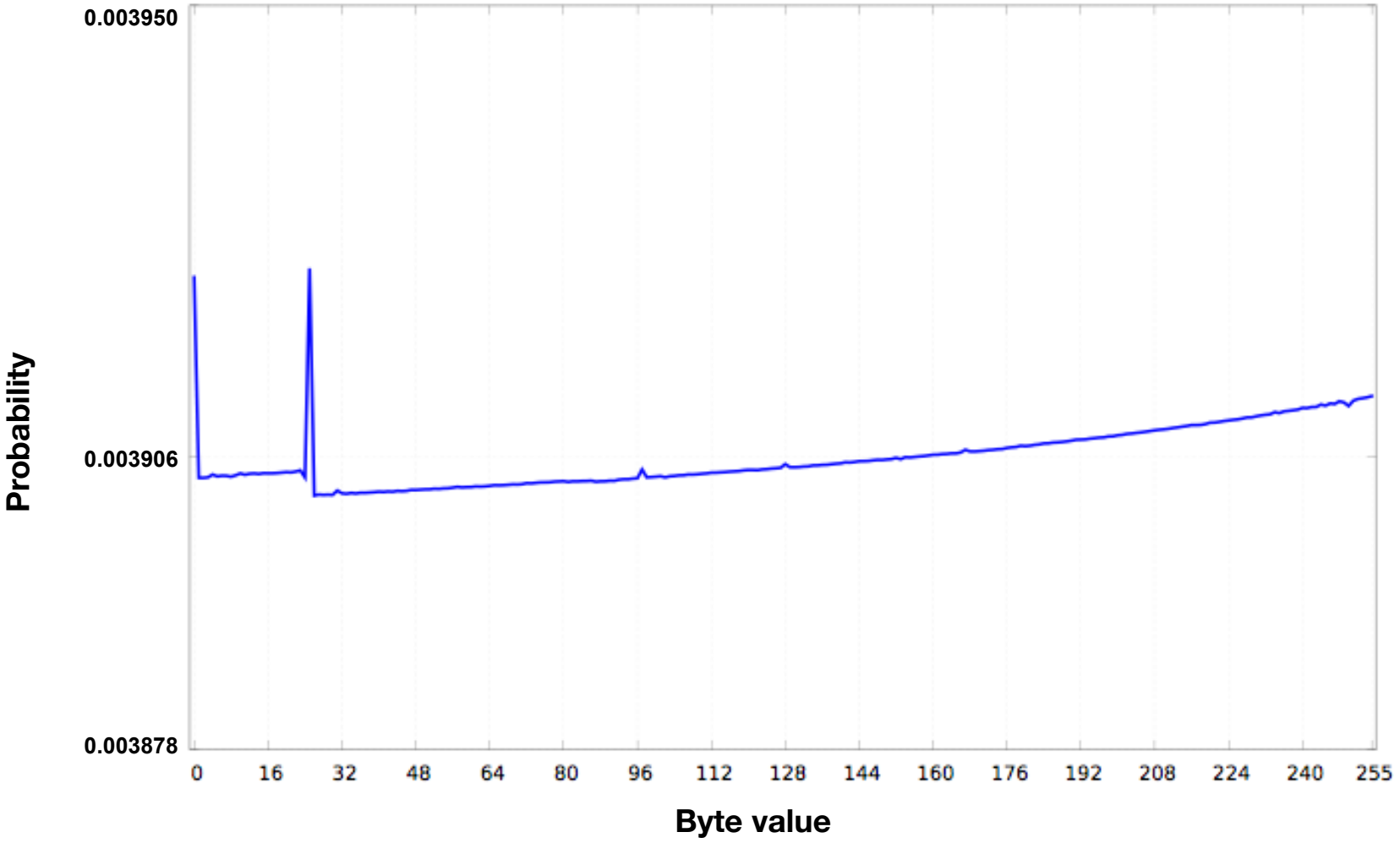
Keystream Distribution at Position 23



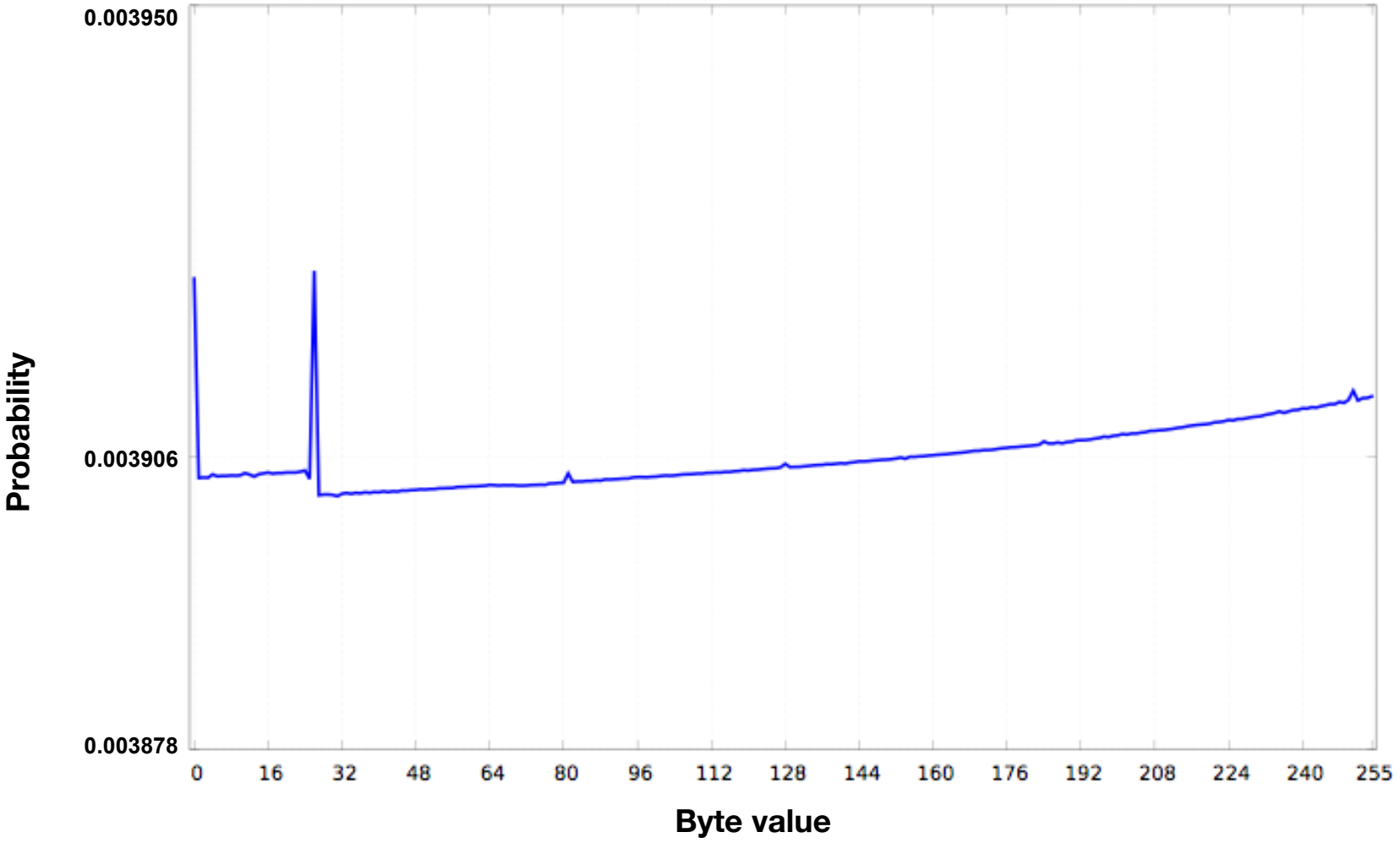
Keystream Distribution at Position 24



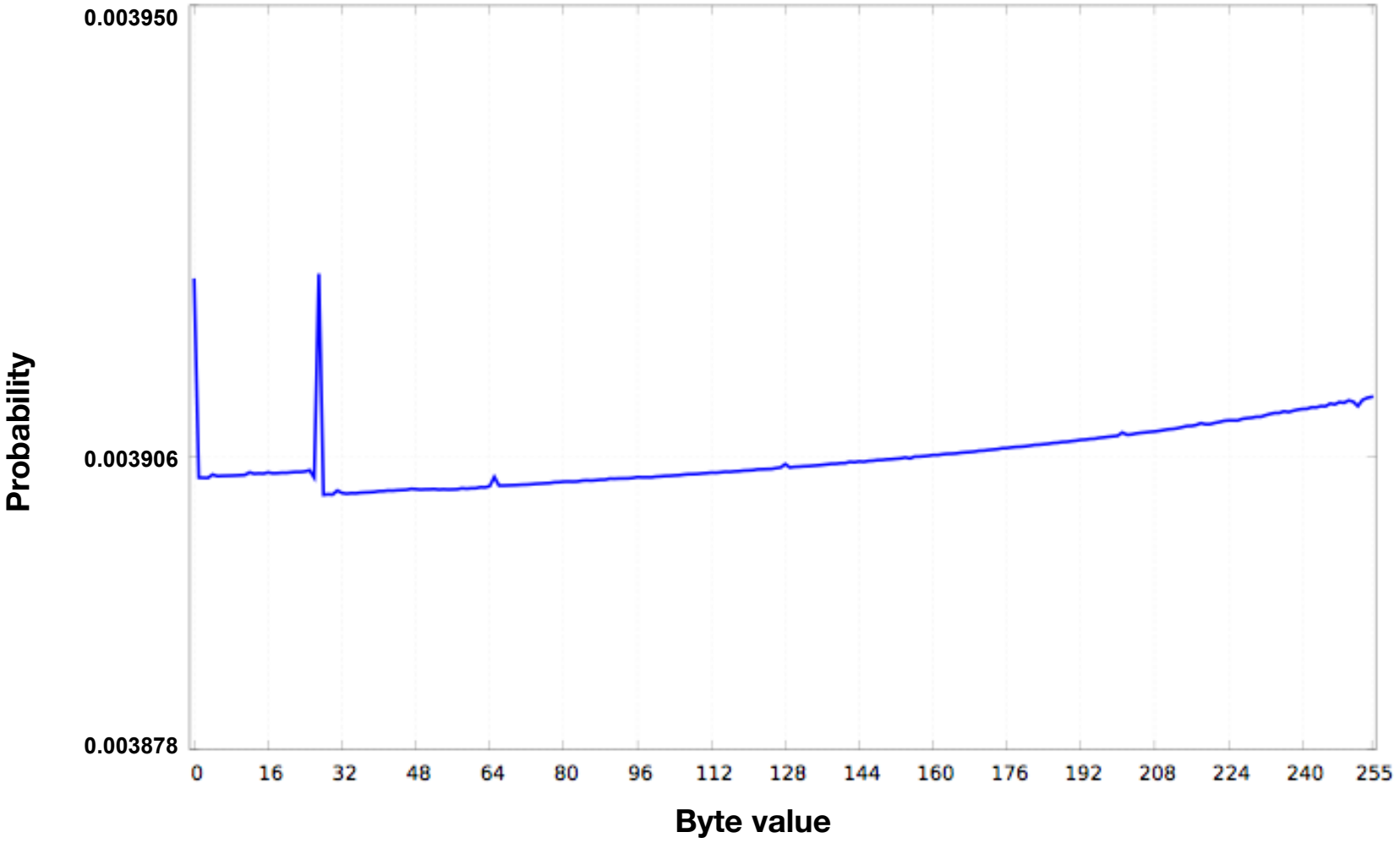
Keystream Distribution at Position 25



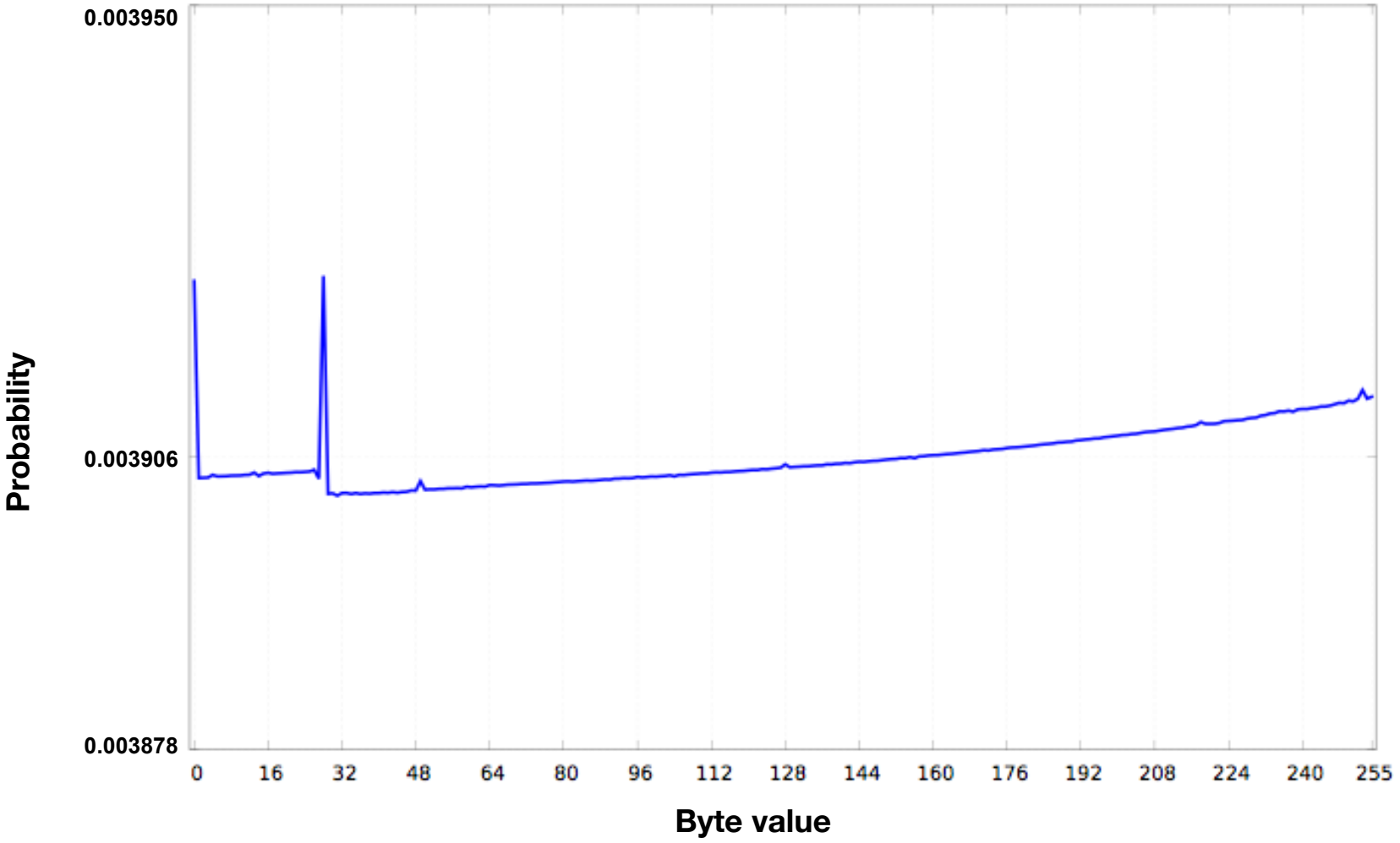
Keystream Distribution at Position 26



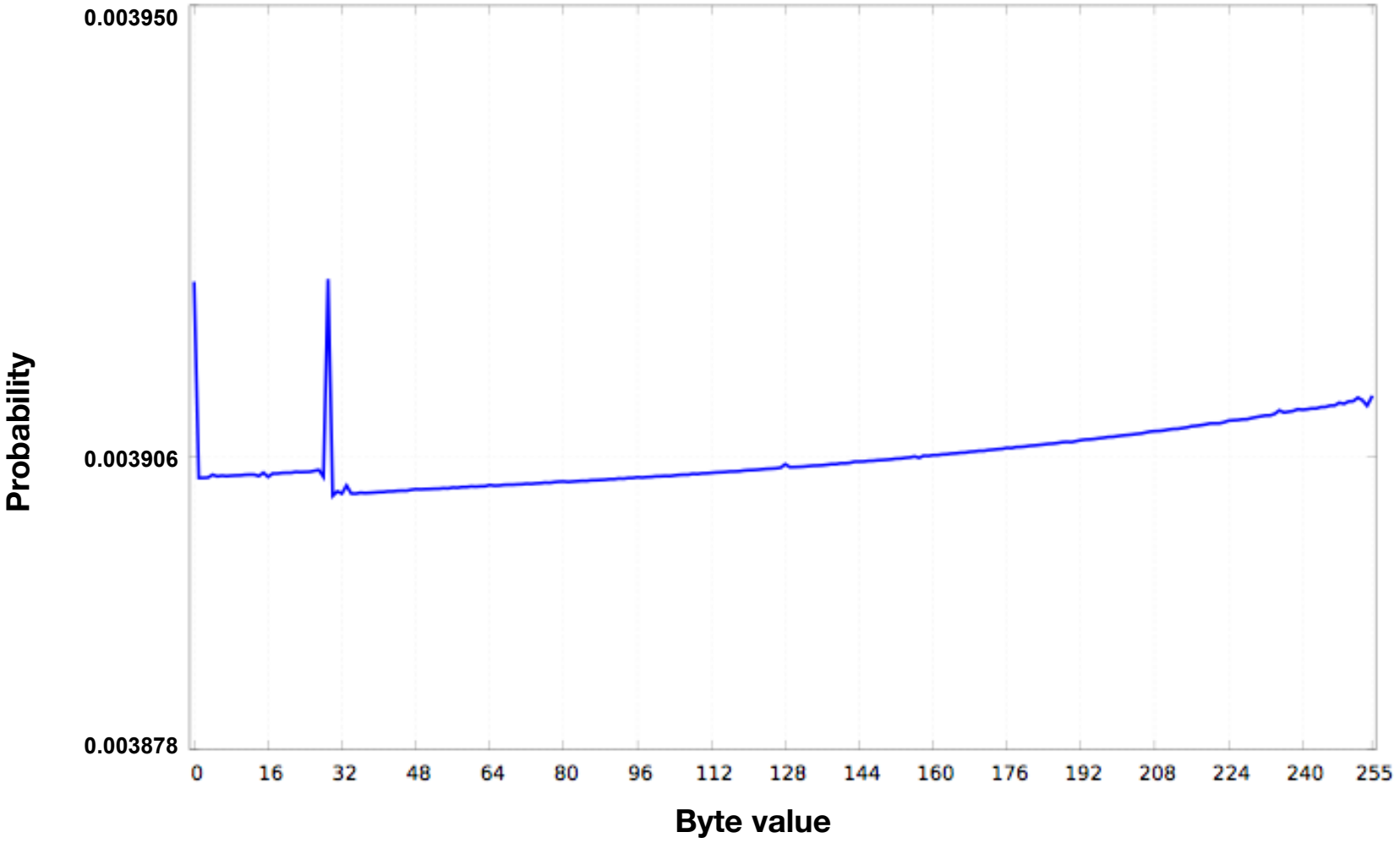
Keystream Distribution at Position 27



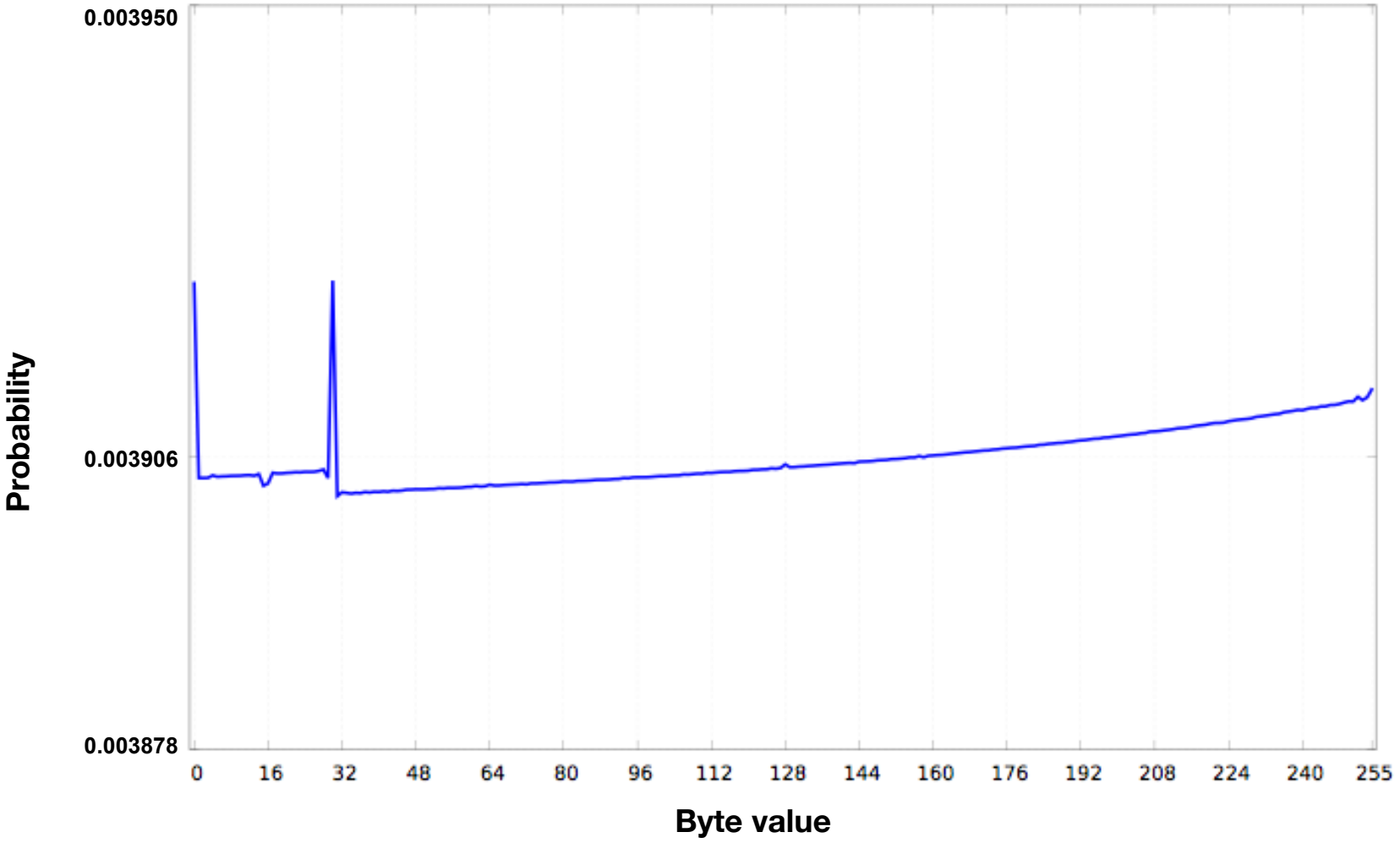
Keystream Distribution at Position 28



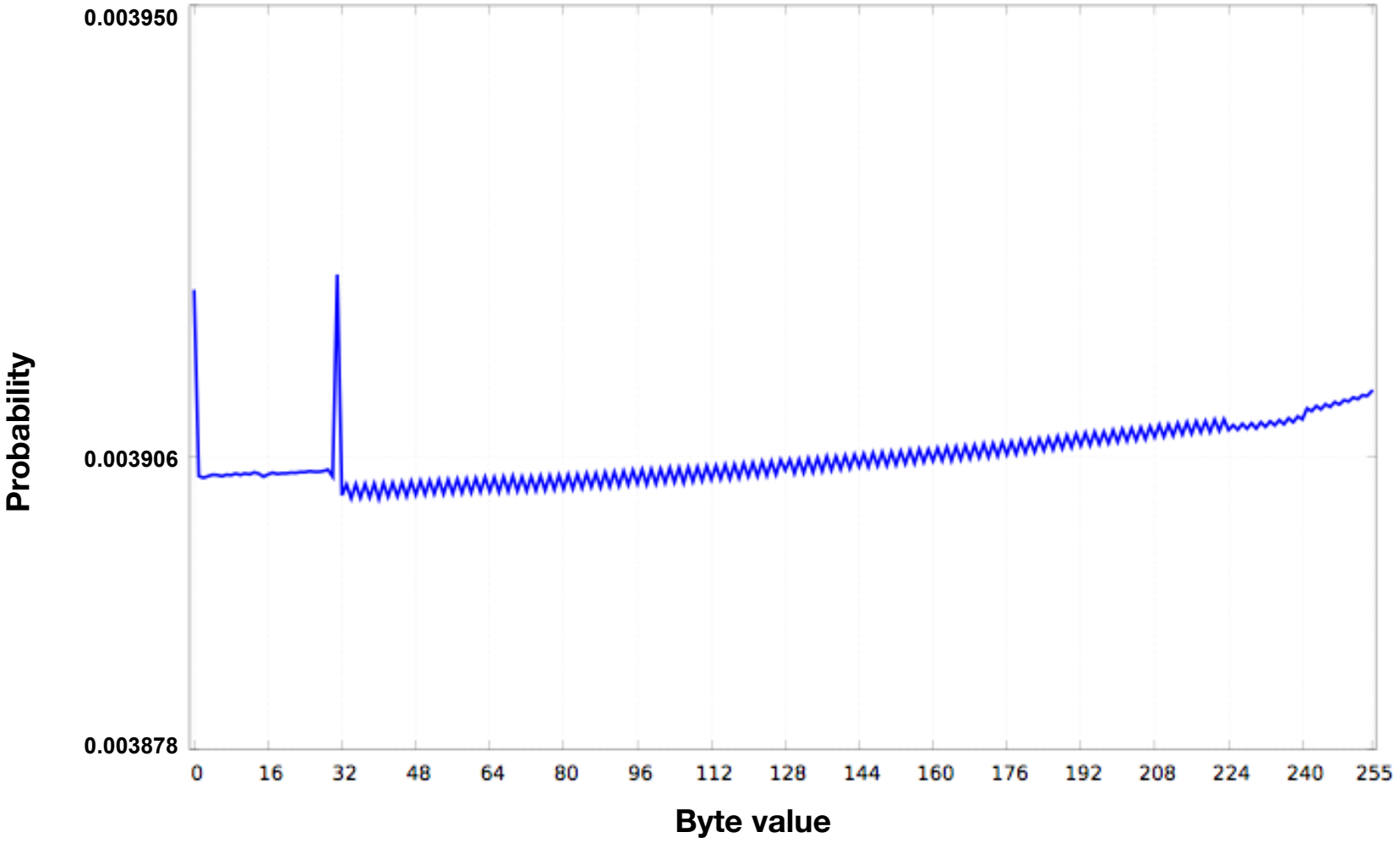
Keystream Distribution at Position 29



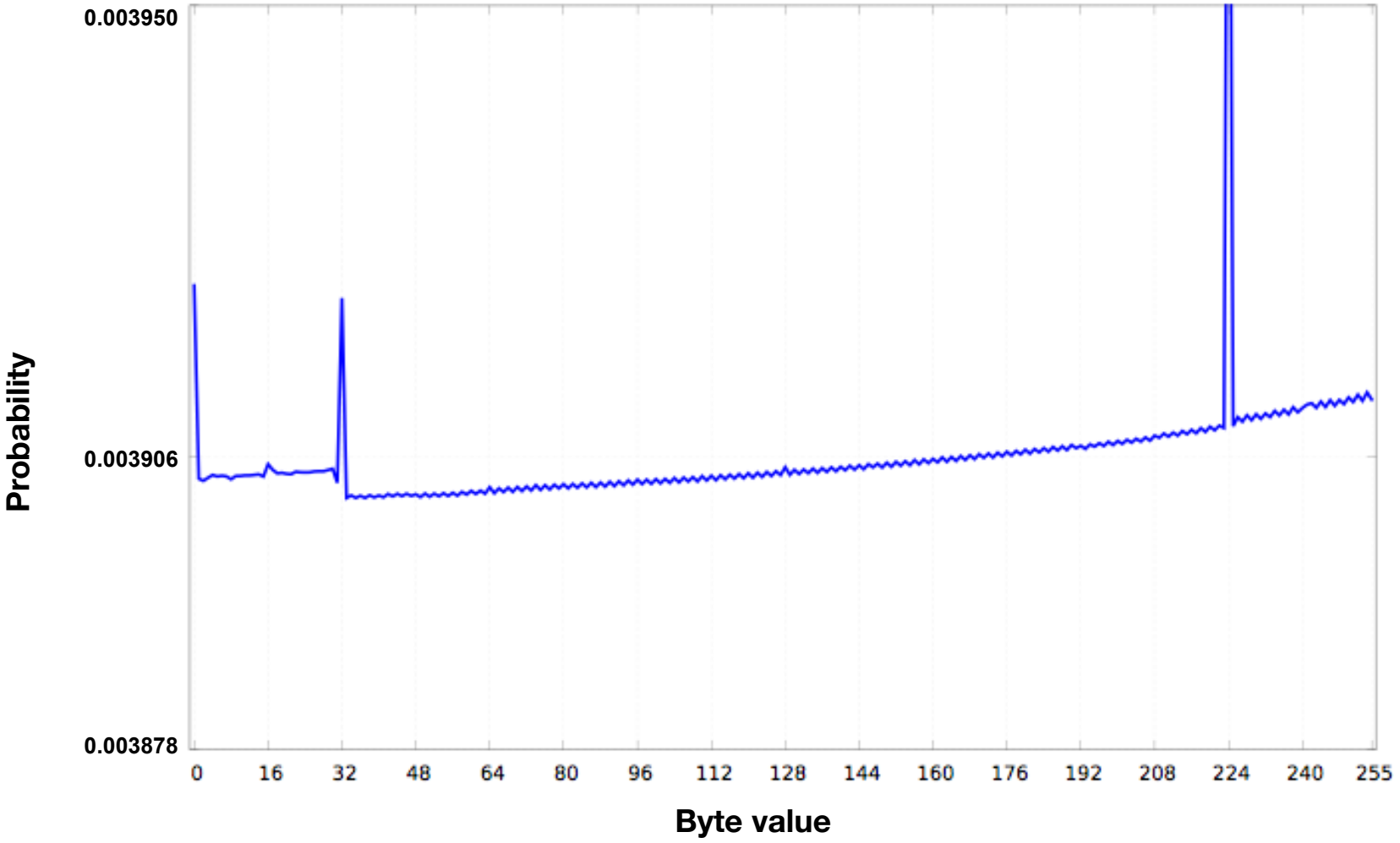
Keystream Distribution at Position 30



Keystream Distribution at Position 31



Keystream Distribution at Position 32

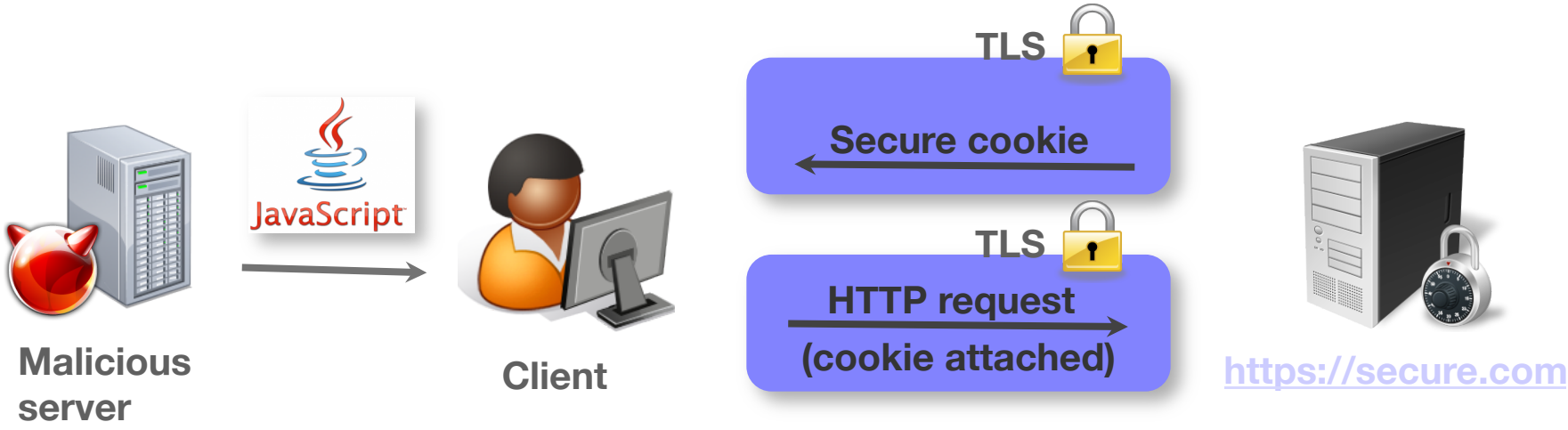


Plaintext Recovery

- 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.

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

Targeting Secure HTTP Cookies

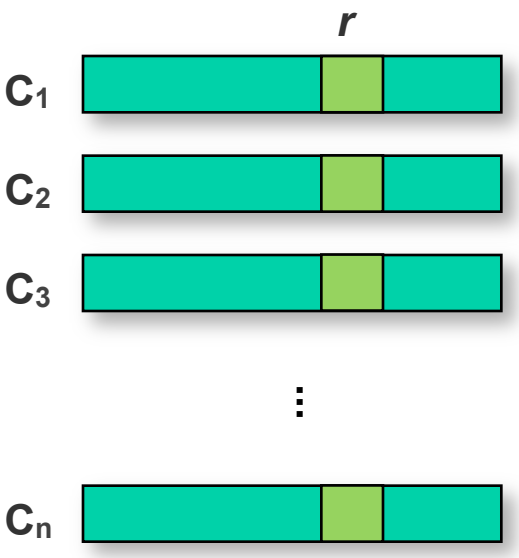


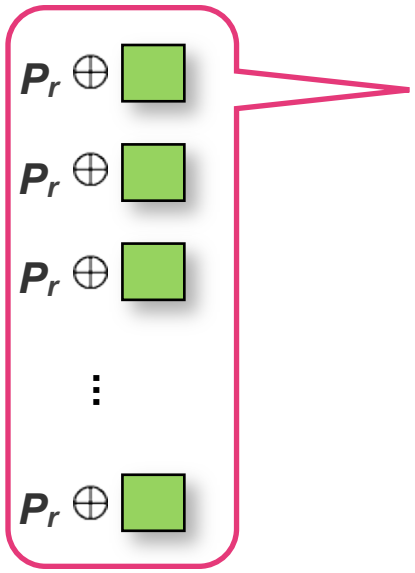
- Javascript
  - Uses XMLHttpRequest objects to generate POST requests.
  - Request to secure site possible due to Cross-Origin Resource Sharing.
  - Number of requests generated by script must be balanced to avoid browser overload.

Plaintext Recovery

Encryptions of plaintext under different keys

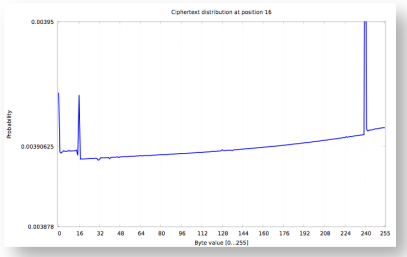
Plaintext candidate byte P_r





Induced distribution on Z_r

combine with

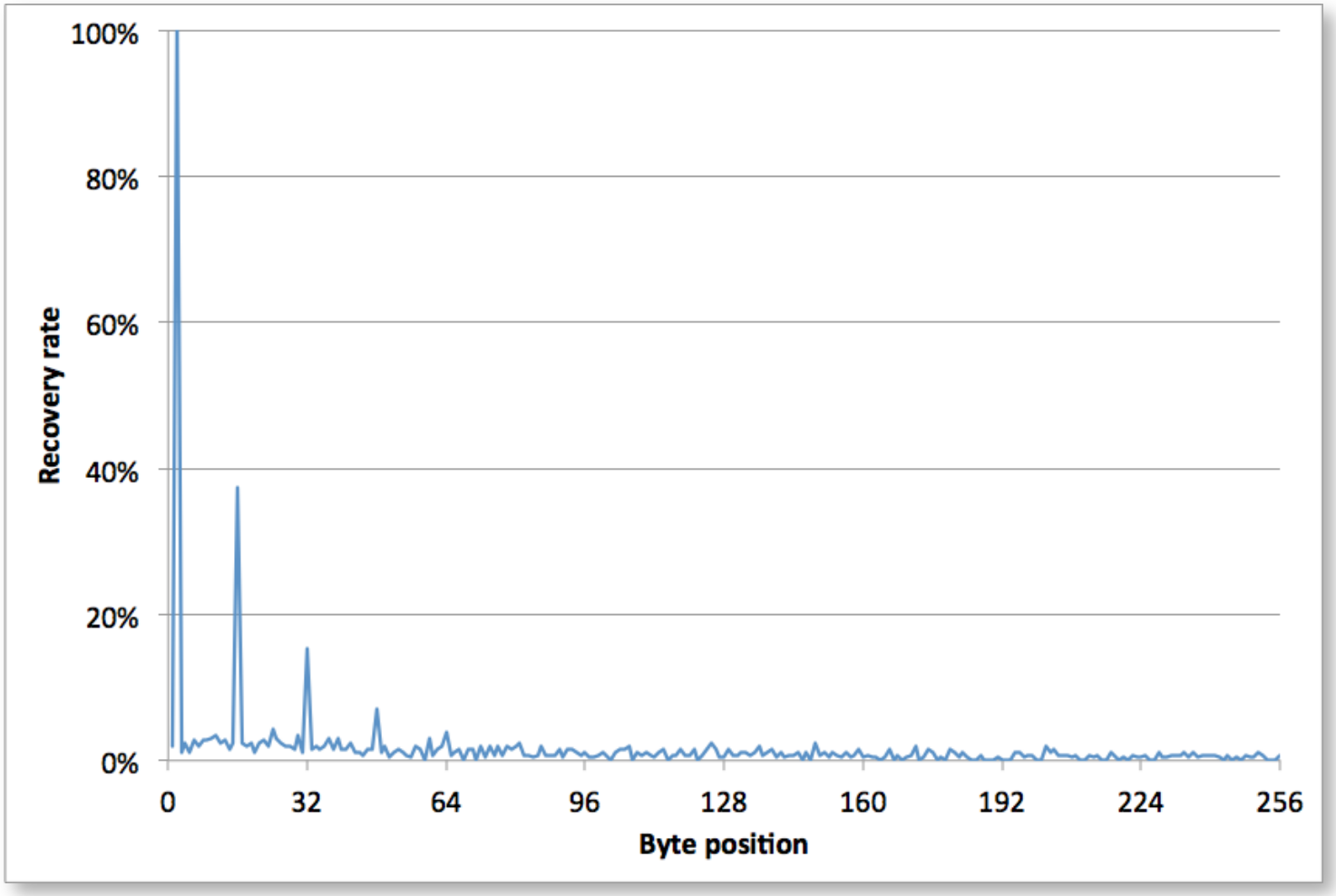


⇓

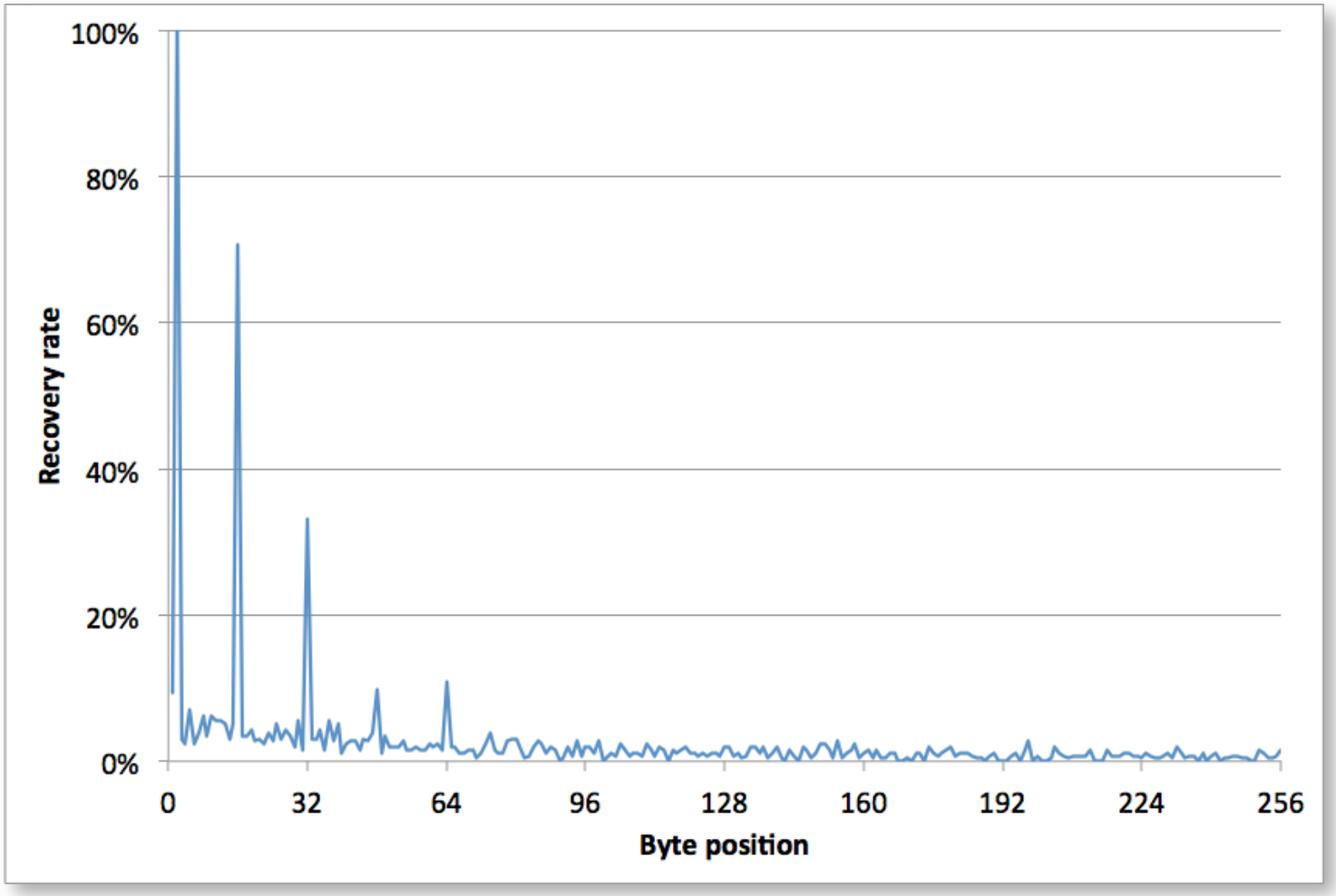
Likelihood of P_r being correct plaintext byte

<u>Recovery algorithm:</u>
Compute most likely plaintext byte

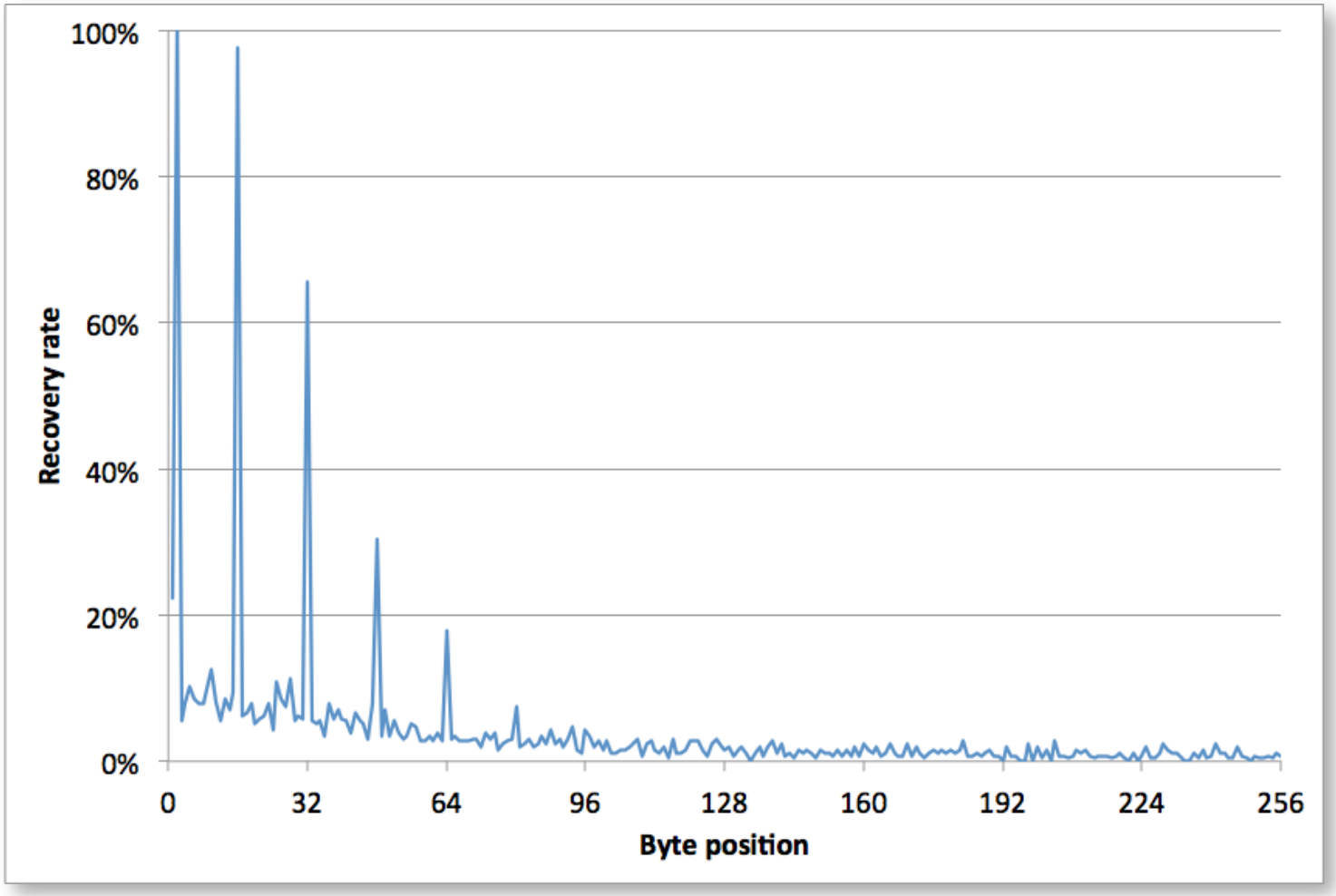
Success Probability
2^{20} Sessions



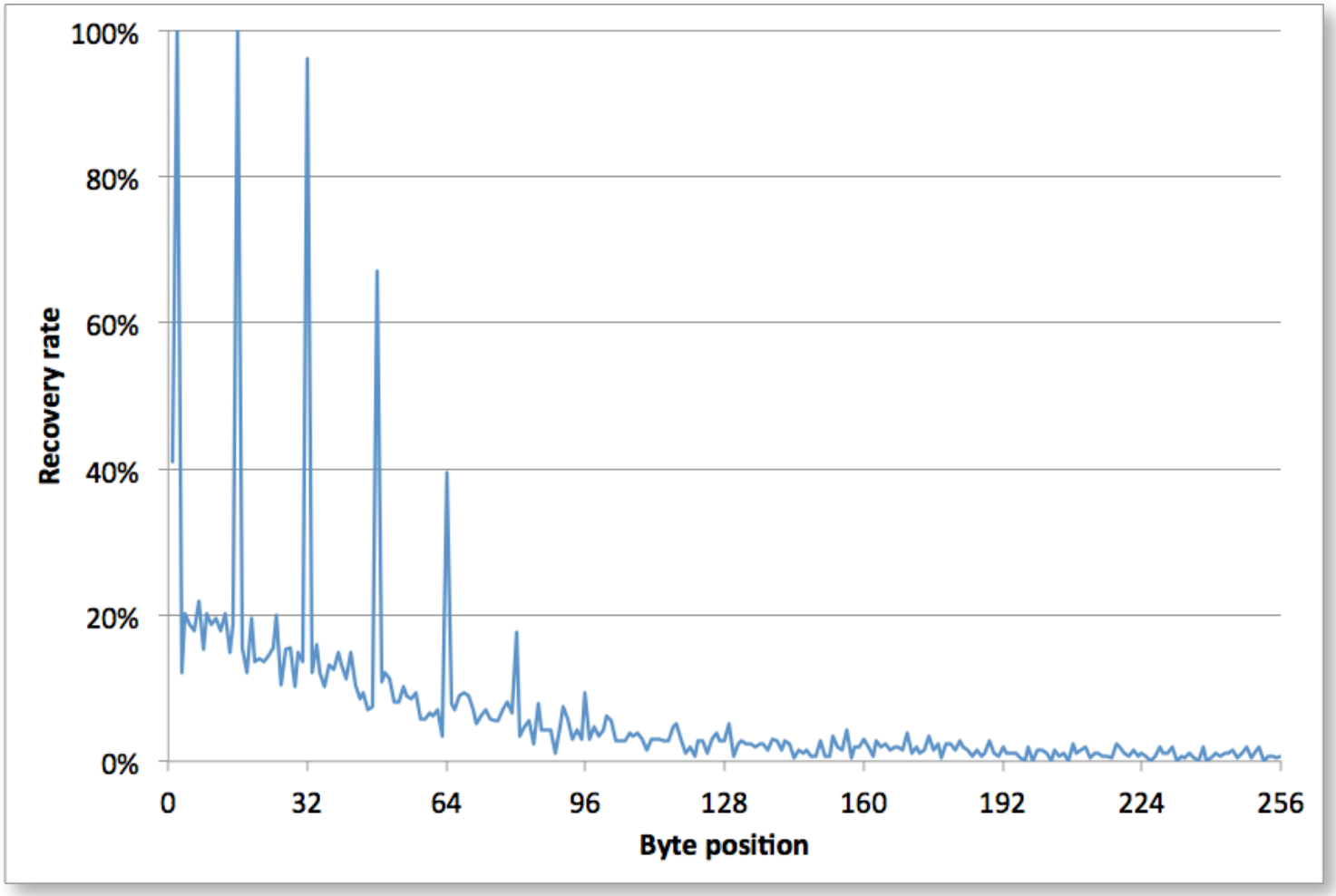
Success Probability
2^{21} Sessions



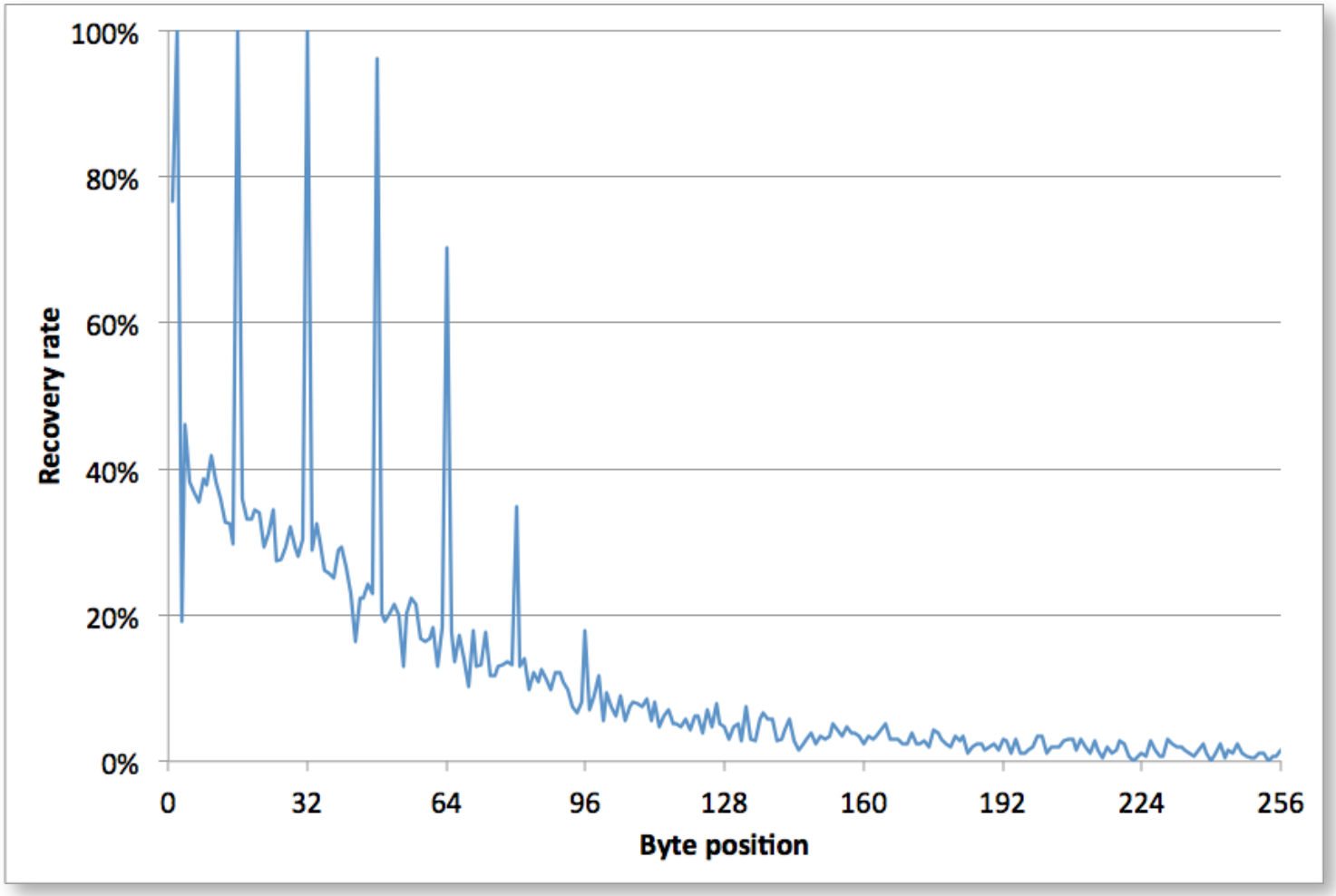
Success Probability
2^{22} Sessions



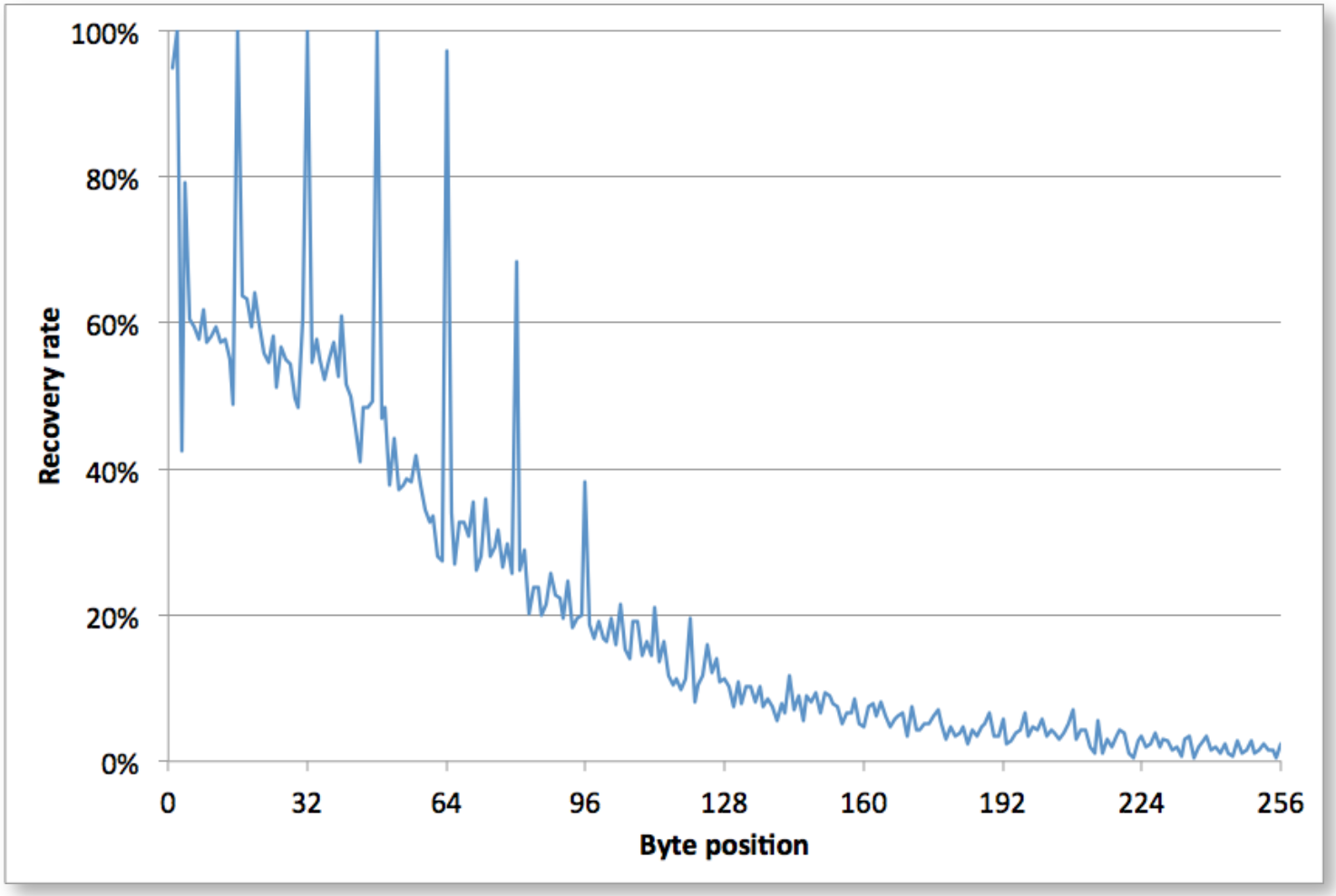
Success Probability
2^{23} Sessions



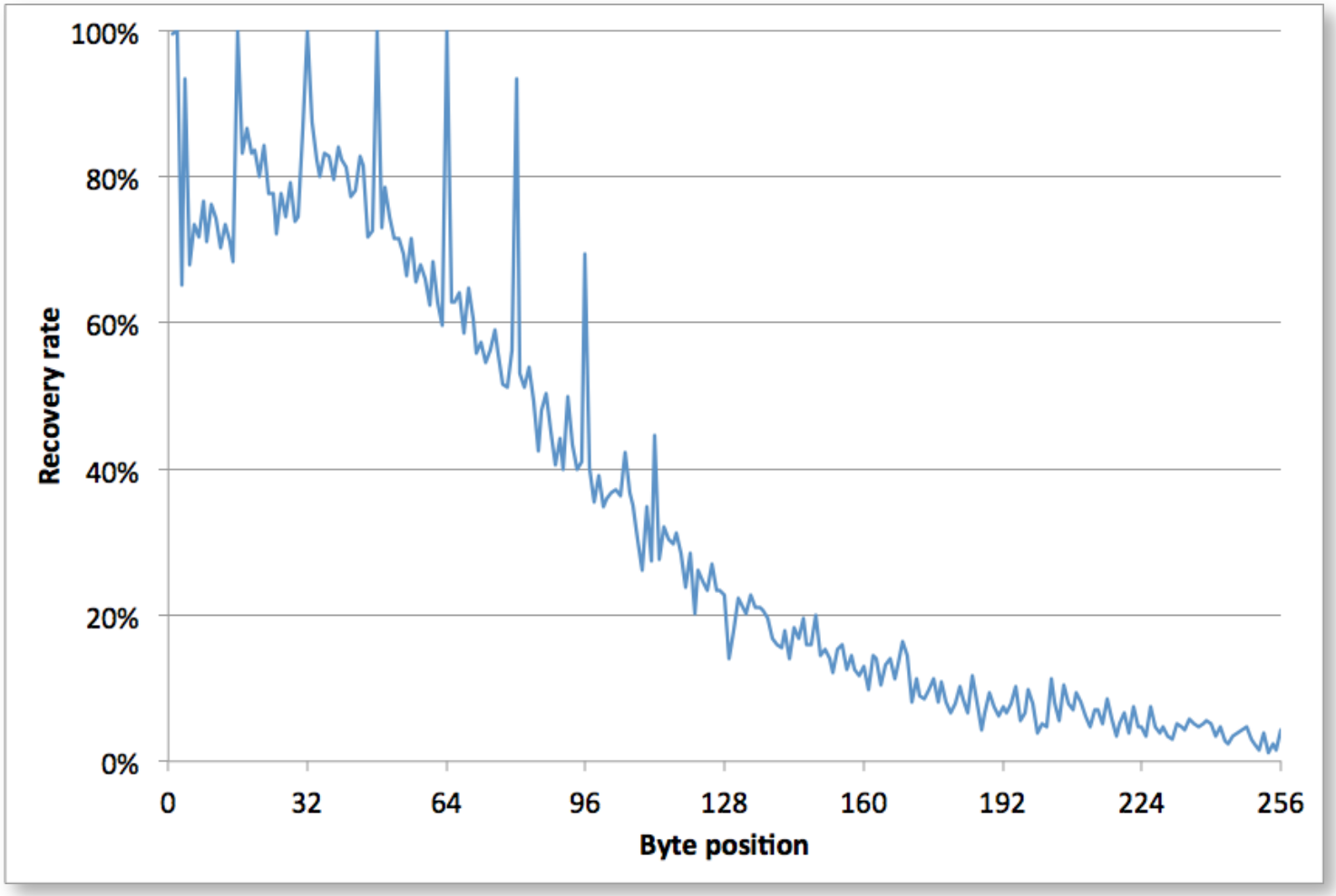
Success Probability
2^{24} Sessions



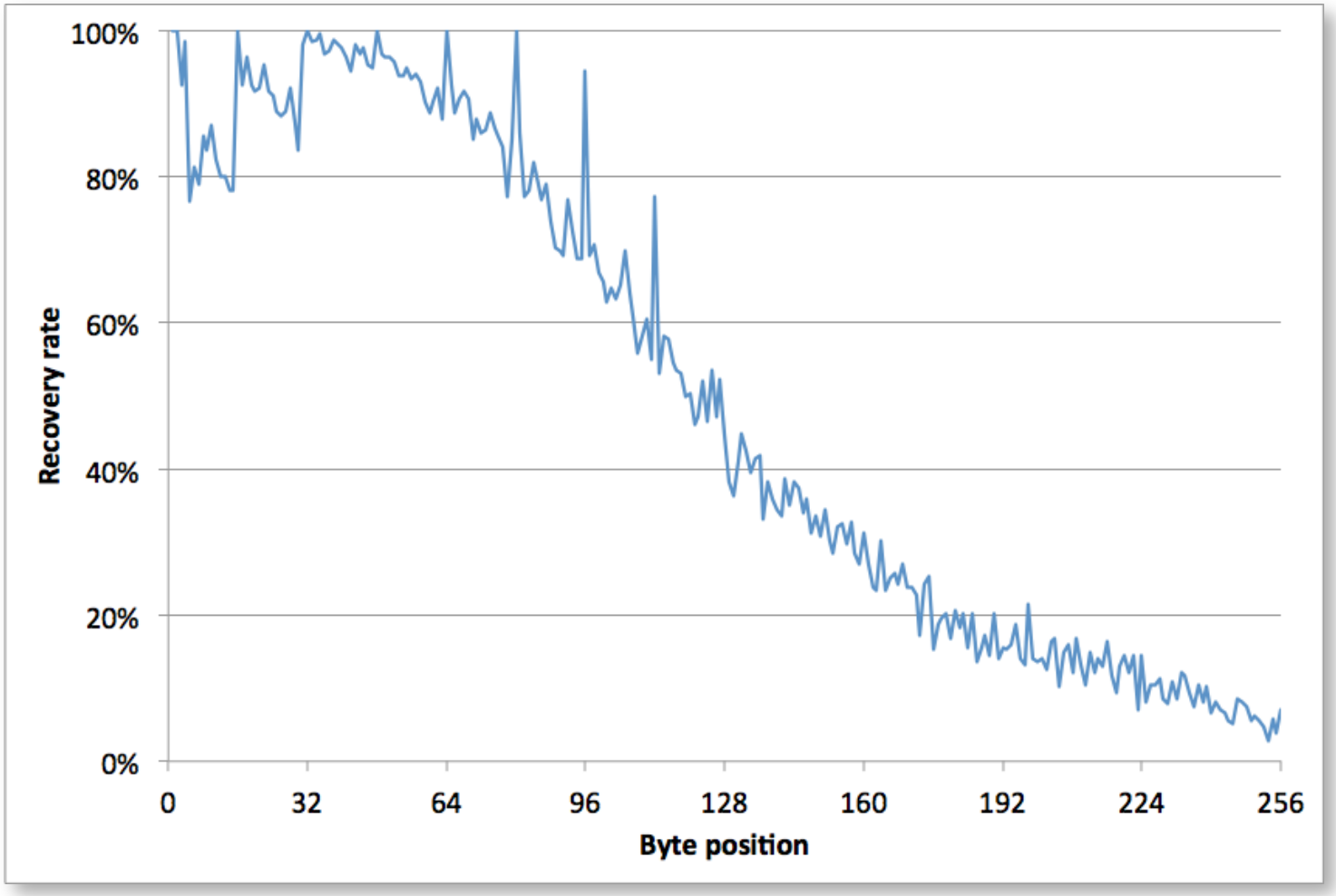
Success Probability
2^{25} Sessions



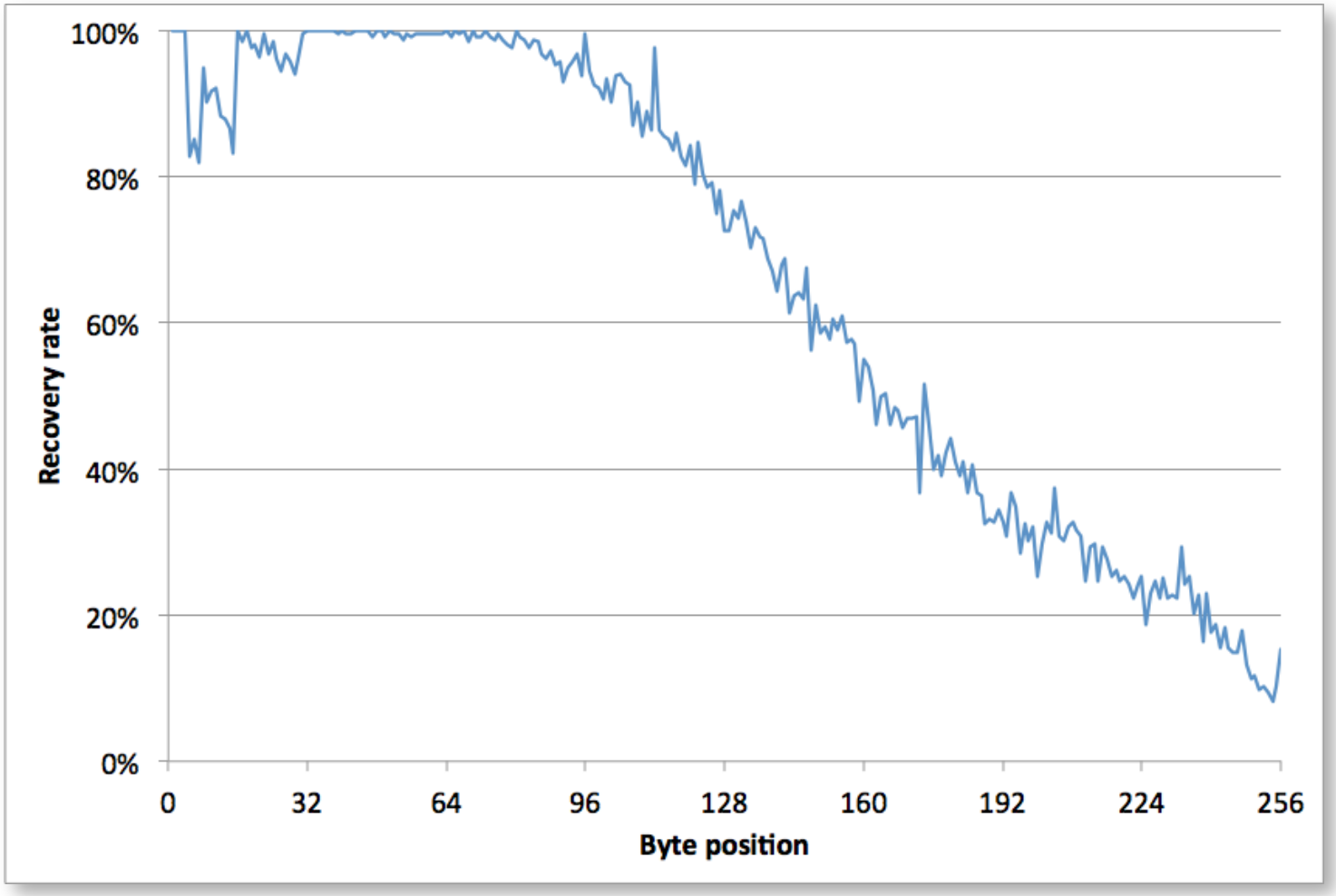
Success Probability
2^{26} Sessions



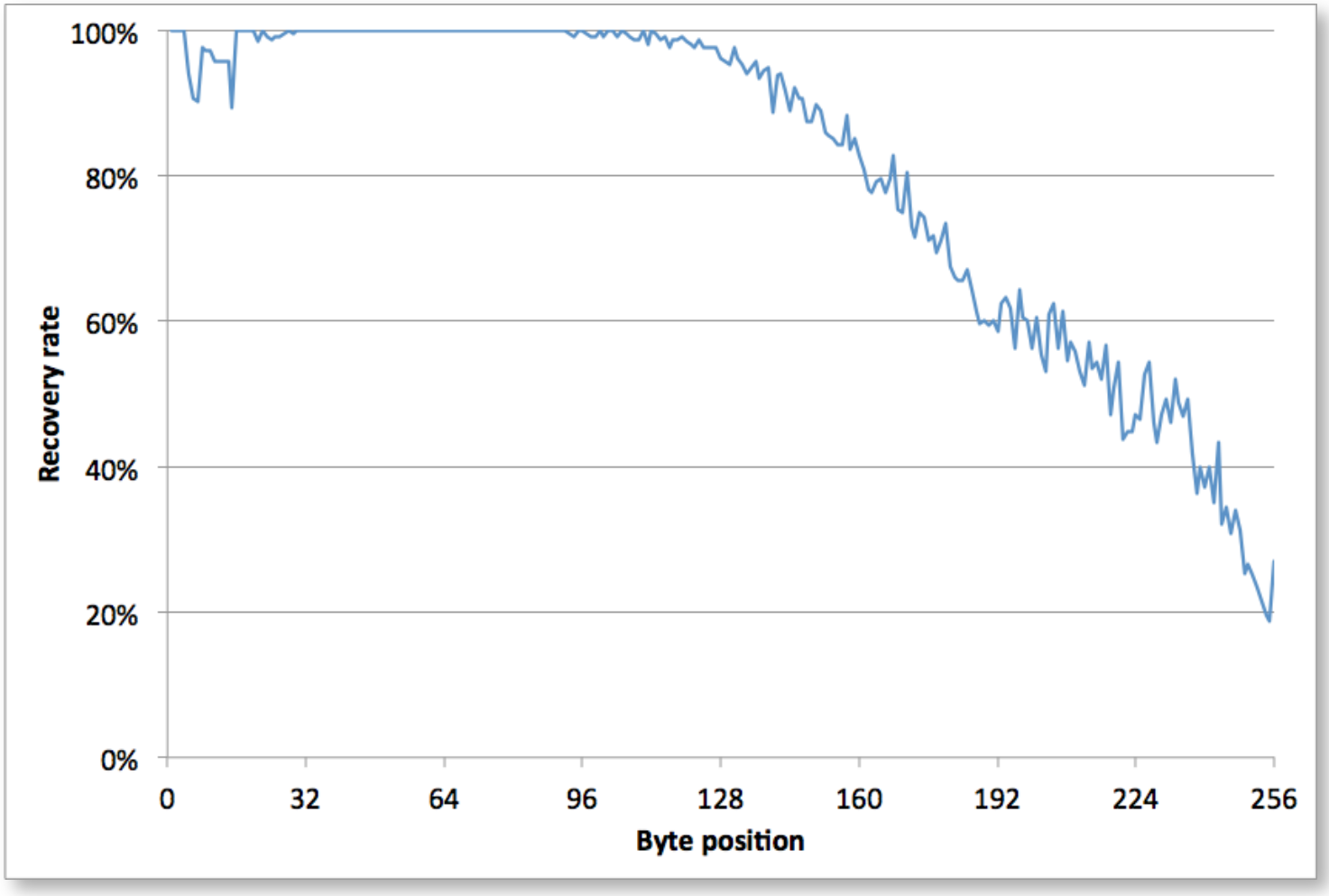
Success Probability
2^{27} Sessions



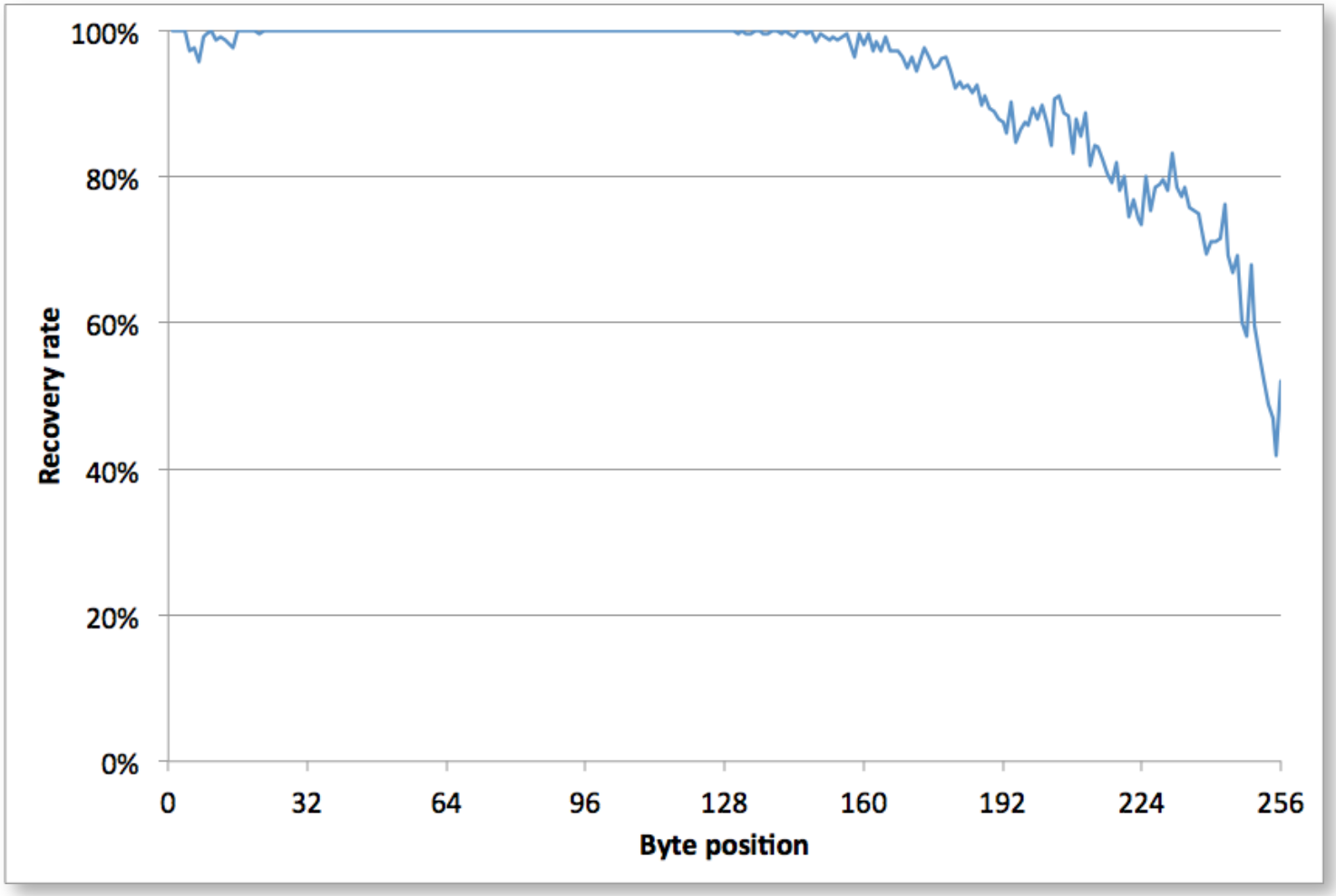
Success Probability
2^{28} Sessions



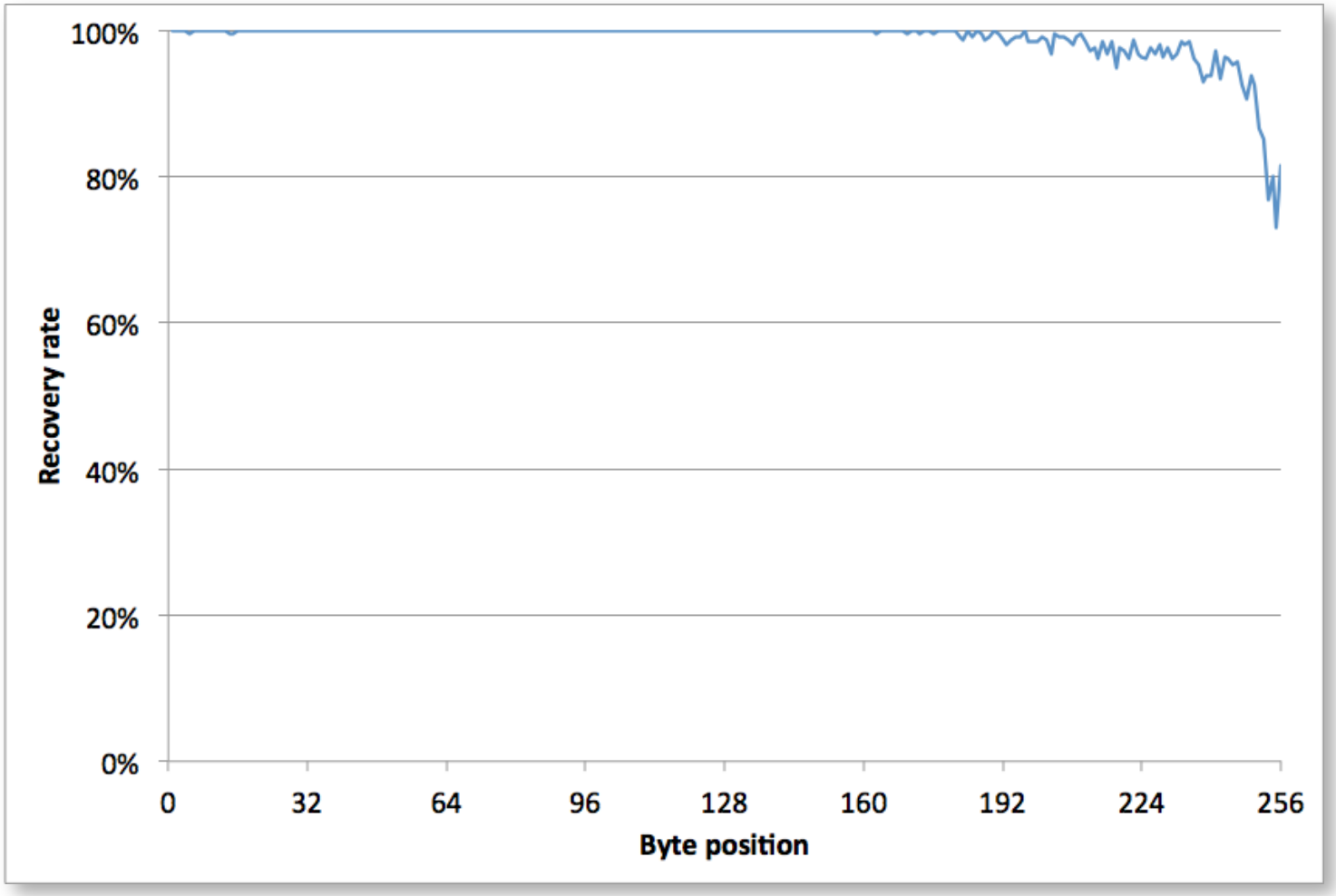
Success Probability
2^{29} Sessions



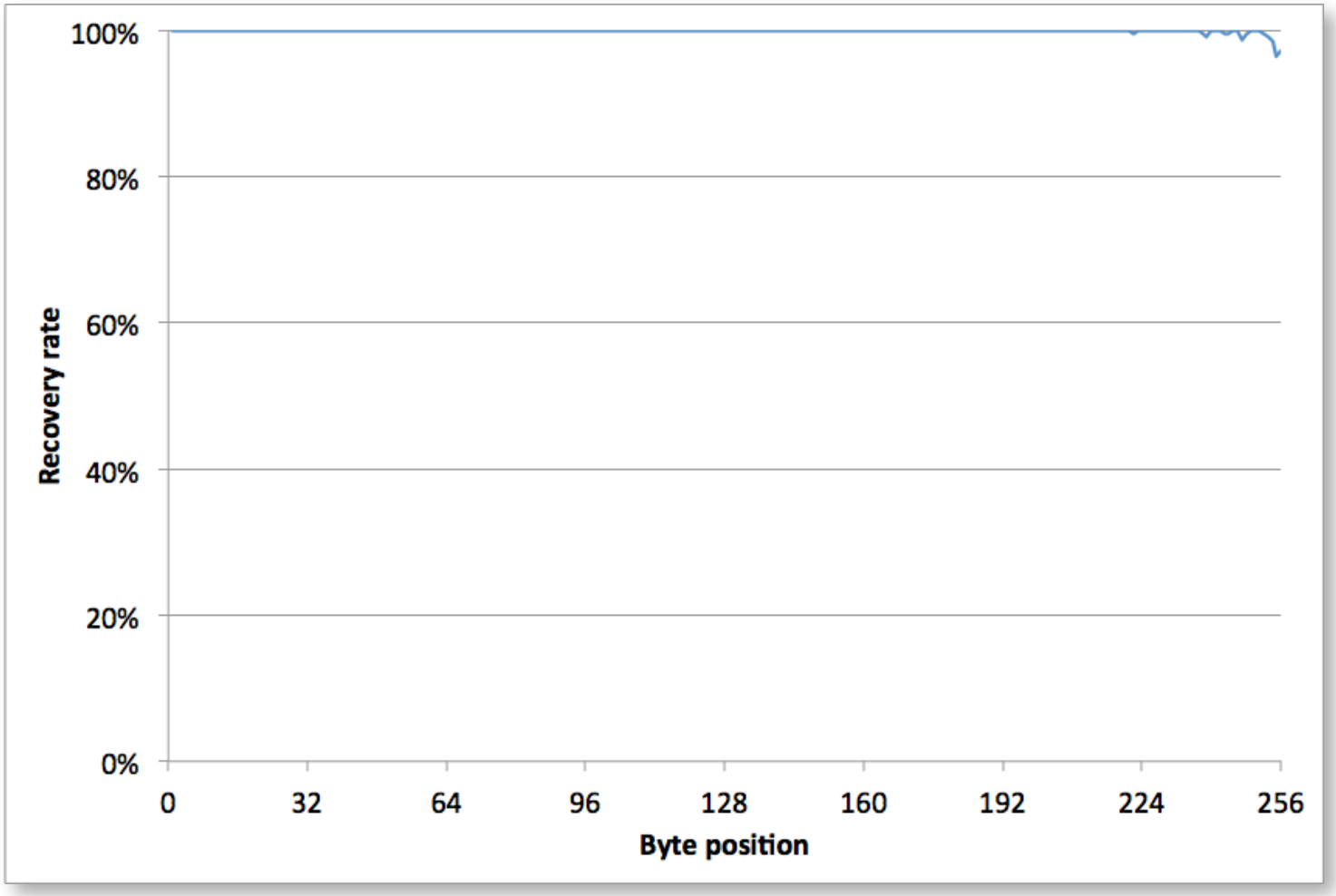
Success Probability
2^{30} Sessions



Success Probability
2^{31} Sessions



Success Probability
2^{32} Sessions



Limitations of Attack

- Requires 2^{28} ~ 2^{32} TLS connections for reliable recovery.
- Attacker has to force TLS session renegotiation / resumption.
  - No known mechanism from within Javascript.
- Only the first 220 bytes of application data can be targeted.
  - Initial 36 bytes used to encrypt last message of Handshake protocol.
  - In reality, first 220 bytes of application data usually contain uninteresting HTTP headers.

A Second Attack

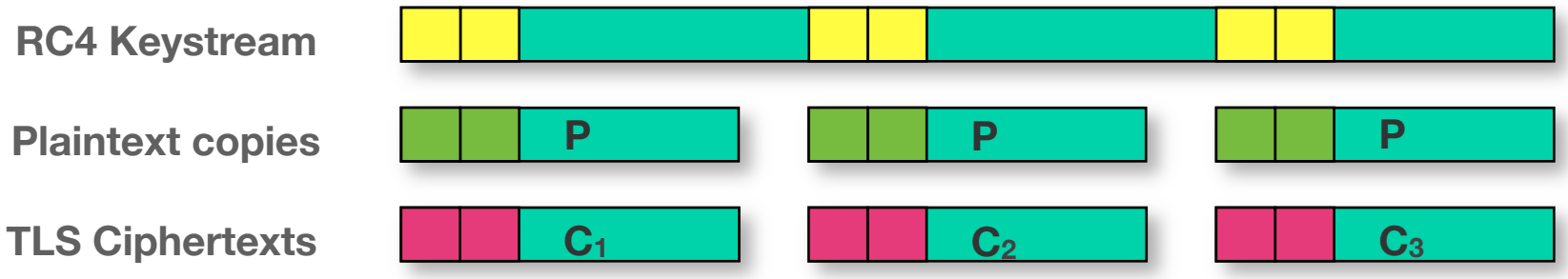
- Fluhrer and McGrew identified biases for consecutive keystream bytes.
  - Persistent throughout keystream.
- Based on these, we construct an attack which:
  - Can target any plaintext byte positions;
  - Does not require session renegotiation / resumption.

i : keystream byte position mod 256

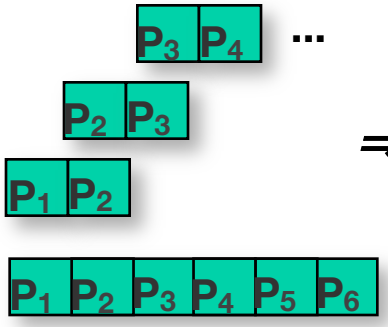
Byte pair	Condition on i	Probability
$(0, 0)$	$i = 1$	$2^{-16}(1 + 2^{-9})$
$(0, 0)$	$i \neq 1, 255$	$2^{-16}(1 + 2^{-8})$
$(0, 1)$	$i \neq 0, 1$	$2^{-16}(1 + 2^{-8})$
$(i + 1, 255)$	$i \neq 254$	$2^{-16}(1 + 2^{-8})$
$(255, i + 1)$	$i \neq 1, 254$	$2^{-16}(1 + 2^{-8})$
$(255, i + 2)$	$i \neq 0, 253,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

A Second Attack

- Align plaintext with repeating Fluhrer-McGrew biases



- Exploit overlapping nature of plaintext byte pairs to obtain approximate likelihood for plaintext candidates.

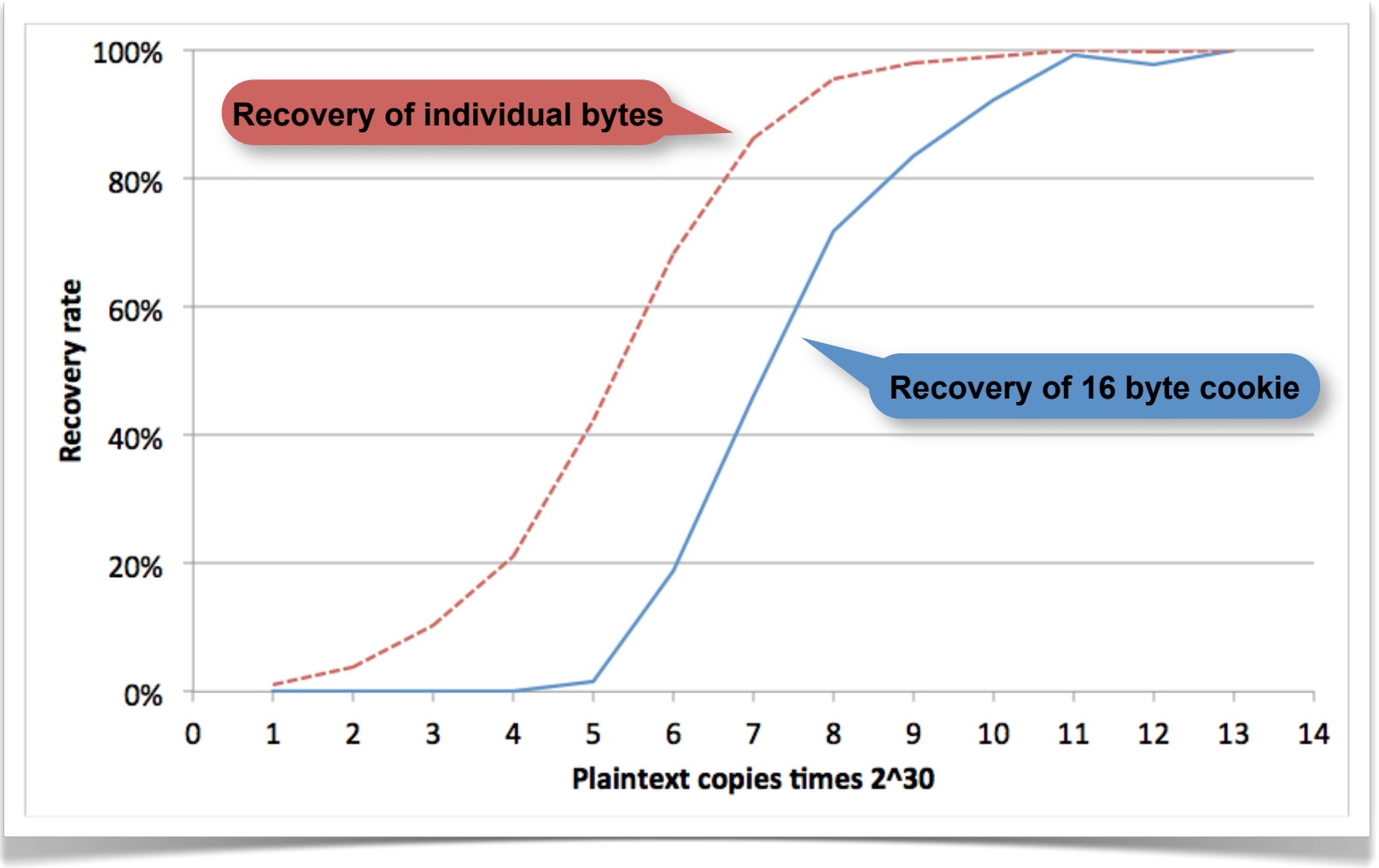


⇒

Approximate likelihood for $P = P_1P_2P_3P_4P_5P_6$

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

Success Probability



Countermeasures

- Possible countermeasures against the 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.
  - Draft RFC for Salsa20 just published.

Summary of RC4 in TLS

- 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.
- The attacks illustrate that RC4 in TLS provides a security level far below the strength suggested by the key size of 128 bits.
- Furthermore, attacks only becomes better with time.
- **Our recommendation: phase out the use of RC4 in TLS as soon as possible.**

Outline

- TLS overview
- TLS attacks and proofs
- Lucky 13
- TLS Record Protocol and RC4
- Discussion

Discussion

- TLS's *ad hoc* MAC-Encode-Encrypt construction is hard to implement securely and hard to prove positive security results about.
  - Long history of attacks and fixes for CBC-mode, culminating in this year's "Lucky 13" attack.
  - Each fix was the "easiest option at the time".
  - Now reached point where a 500 line patch to OpenSSL was needed to fully eliminate the Lucky 13 attack on CBC-mode.
  - Attacks show that small details matter.
  - Suitably detailed analysis for MEE-TLS-CBC only completed in 2011.

Discussion

- RC4 was known to be weak for many years.
  - Actual exploitation of its weaknesses in a TLS context went unexplored.
  - Needed multi-session mechanism (BEAST technology) to make the attack plausible.
  - Borrowing tools from outside cryptography.

- Once a bad cryptographic choice is out there in implementations, it's very hard to undo.
  - Old versions of TLS hang around for a long time.
  - There is no TLS product recall programme!
  - Slow uptake of TLS 1.1, 1.2.

Discussion

- TLS is coming under sustained pressure from attacks.
  - BEAST, Lucky 13 and RC4 attacks are providing incentives to move to TLS 1.2.
  - Good vendor response to BEAST, CRIME, Lucky 13, less so to RC4 attack.
    - First three are fixable, the other not (really).
- Having a cool name for your attack is important.
- Attacks really do improve with age.
  - BEAST (1995 – 2011), Lucky 13 (Feb. 2013 – Mar. 2013).
  - RC4 attacks are currently only “semi-practical” but we ignore such attacks at our peril.

Research Directions?

- TLS Record Protocol cryptography has now been heavily analysed.
  - Still some mileage in looking at AE implementations?
- Major recent progress in analysing TLS Handshake protocol.
  - [JKSS12], [KPW13], [GKS12].
- Can still expect implementation issues to emerge.
  - Check the “OpenSSL Fact” twitter feed regularly!
- TLS’s complex system of interacting protocols can still throw up surprises.
  - Alert Protocol desynchronisation attack [BFKPS13].
  - TLS Renegotiation attack [RD09].

TLS – Current Status?

"This is a dead parrot."

"He's not dead. He's just resting."