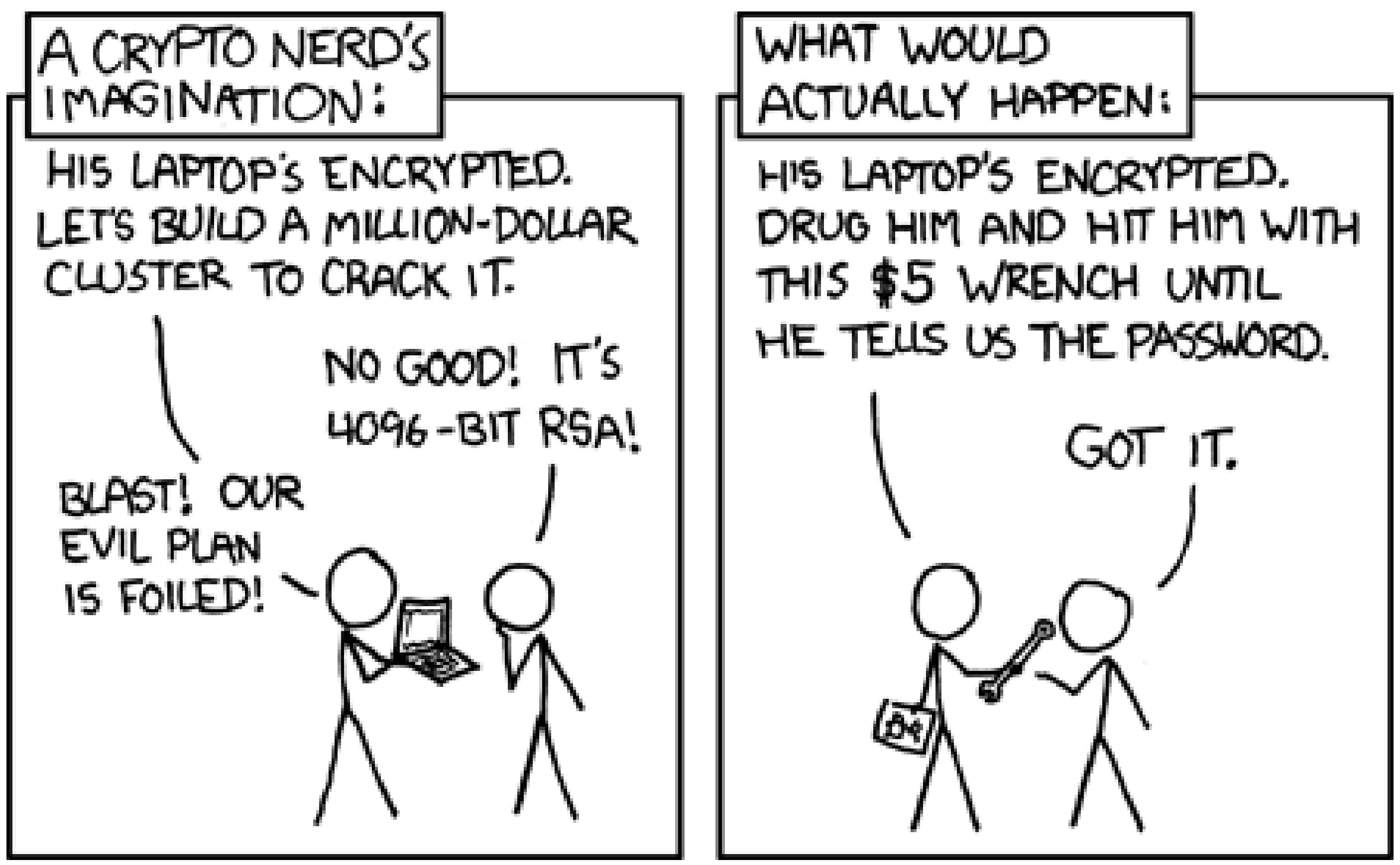
Failures of
secret-key cryptography

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

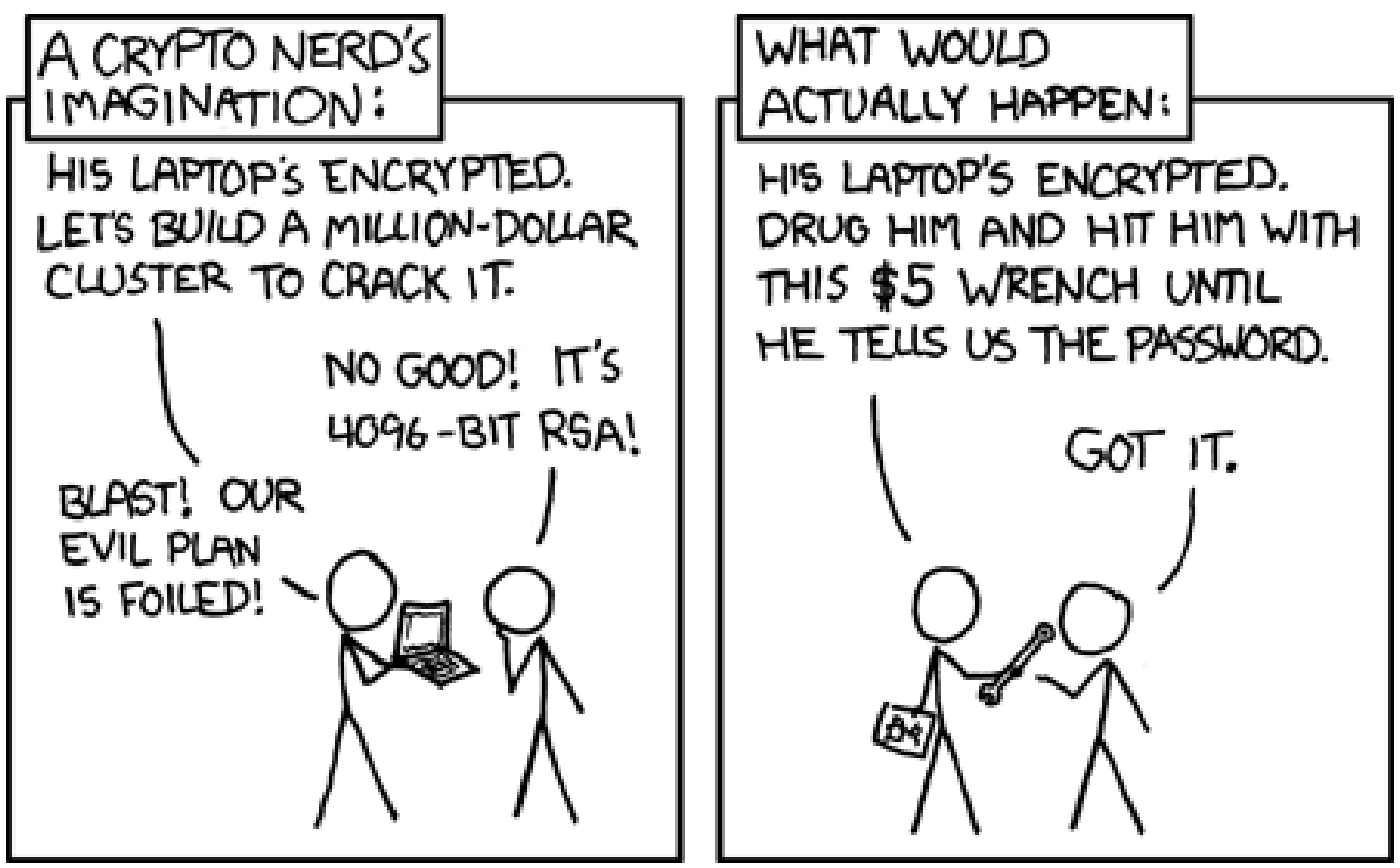


http://xkcd.com/538/

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years “no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds”.

Failures of
secret-key cryptography

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven



http://xkcd.com/538/

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

of
ey cryptography

ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

/xkcd.com/538/

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03
"Do the
it's actu
to break

raphy

is at Chicago &
siteit Eindhoven

n/538/

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein
"Do these people
it's actually infeas
to break real-worlc

ago &
hoven

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

Or are they simply wrong: real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security?"

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years "no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn't use homebrew crypto or toy keys) in the Internet or commercial worlds".

2002 Shamir: "Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

Or are they simply wrong: real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security?"

Let's look at some examples.

igg–Gutmann (and again
tmann): In the past 15
o one ever lost money to
k on a properly designed
stem (meaning one that
se homebrew crypto
eys) in the Internet or
cial worlds".

amir: "Cryptography is
ypassed. I am not aware
ajor world-class security
employing cryptography in
e hackers penetrated the
y actually going through
tanalysis."

2013.03 Bernstein:
"Do these people mean that
it's actually infeasible
to break real-world crypto?

Or do they mean that
breaks are feasible
but still not worthwhile
for the attackers?

Or are they simply wrong:
real-world crypto is breakable;
is in fact being broken;
is one of many ongoing
disaster areas in security?"

Let's look at some examples.

Window

Flame b
spied on

2012.06.
"We rec
of a com
malware
immedia
issue. . .
through
compon
been sig
allow so
was proc

ann (and again
n the past 15
r lost money to
operly designed
aning one that
ew crypto
e Internet or
s".

yptography is
I am not aware
d-class security
cryptography in
penetrated the
y going through

2013.03 Bernstein:
"Do these people mean that
it's actually infeasible
to break real-world crypto?

Or do they mean that
breaks are feasible
but still not worthwhile
for the attackers?

Or are they simply wrong:
real-world crypto is breakable;
is in fact being broken;
is one of many ongoing
disaster areas in security?"

Let's look at some examples.

Windows code sig

Flame broke into c
spied on audio, ke

2012.06.03 Micros
"We recently beca
of a complex piece
malware known as
immediately began
issue. . . . We have
through our analys
components of the
been signed by cer
allow software to a
was produced by M

again
t 15
ey to
gned
that

or

y is
aware
urity
phy in
d the
rough

2013.03 Bernstein:
"Do these people mean that
it's actually infeasible
to break real-world crypto?

Or do they mean that
breaks are feasible
but still not worthwhile
for the attackers?

Or are they simply wrong:
real-world crypto is breakable;
is in fact being broken;
is one of many ongoing
disaster areas in security?"

Let's look at some examples.

Windows code signatures

Flame broke into computers
spied on audio, keystrokes, e

2012.06.03 Microsoft:
"We recently became aware
of a complex piece of target
malware known as 'Flame' a
immediately began examinin
issue. . . . We have discovere
through our analysis that so
components of the malware
been signed by certificates t
allow software to appear as
was produced by Microsoft."

2013.03 Bernstein:
"Do these people mean that it's actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

Or are they simply wrong: real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security?"

Let's look at some examples.

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft:
"We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

Bernstein:
se people mean that
ally infeasible
real-world crypto?

ey mean that
re feasible
not worthwhile
ttackers?

hey simply wrong:
ld crypto is breakable;
t being broken;
f many ongoing
areas in security?”

ok at some examples.

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft:
“We recently became aware of a complex piece of targeted malware known as ‘Flame’ and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft.”

2012.06.
prefix co
MD5 ha
More int
publishe
attack w
new and

:
mean that
ible
d crypto?

that

while

wrong:
s breakable;
oken;
going
ecurity?"

e examples.

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft:
"We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

2012.06.07 Steven
prefix collision att
MD5 has been use
More interestingly
published chosen-p
attack was used, b
new and unknown

t

e;

s.

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft: "We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

2012.06.07 Stevens: "A cho
prefix collision attack agains
MD5 has been used for Flan
More interestingly . . . not o
published chosen-prefix collis
attack was used, but an ent
new and unknown variant."

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft: "We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft: "We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft: "We recently became aware of a complex piece of targeted malware known as 'Flame' and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Was MD5 "homebrew crypto"? No. Standardized, widely used. Worthwhile to attack? Yes.

s code signatures

roke into computers,
 audio, keystrokes, etc.

.03 Microsoft:
ently became aware
nplex piece of targeted
 known as 'Flame' and
tely began examining the
. We have discovered
 our analysis that some
ents of the malware have
ned by certificates that
ftware to appear as if it
duced by Microsoft."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Was MD5 "homebrew crypto"? No. Standardized, widely used. Worthwhile to attack? Yes.

Compare
"Cryptos
magnitu

natures

computers,
ystrokes, etc.

soft:
me aware
e of targeted
'Flame' and
n examining the
e discovered
sis that some
e malware have
rtificates that
appear as if it
Microsoft."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Was MD5 "homebrew crypto"? No. Standardized, widely used. Worthwhile to attack? Yes.

Compare to 2011
"Cryptosystem fai
magnitude below a

,
etc.

ed
nd
g the
ed
me
have
hat
if it
”

2012.06.07 Stevens: “A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant.”

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame “may have been active for as long as five to eight years”.

Was MD5 “homebrew crypto”? No. Standardized, widely used. Worthwhile to attack? Yes.

Compare to 2011 Grigg–Gut
“Cryptosystem failure is ord
magnitude below any other

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Was MD5 "homebrew crypto"? No. Standardized, widely used. Worthwhile to attack? Yes.

Compare to 2011 Grigg–Gutmann: "Cryptosystem failure is orders of magnitude below any other risk."

2012.06.07 Stevens: "A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant."

CrySyS: Flame file `wavesup3.drv` appeared in logs in 2007; Flame "may have been active for as long as five to eight years".

Was MD5 "homebrew crypto"? No. Standardized, widely used. Worthwhile to attack? Yes.

Compare to 2011 Grigg–Gutmann: "Cryptosystem failure is orders of magnitude below any other risk."

http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

07 Stevens: "A chosen-
llision attack against
s been used for Flame.
terestingly . . . not our
d chosen-prefix collision
as used, but an entirely
unknown variant."

Flame file `wavesup3.drv`
d in logs in 2007; Flame
ve been active for as long
o eight years".

05 "homebrew crypto"?
ndardized, widely used.
hile to attack? Yes.

Compare to 2011 Grigg–Gutmann:
"Cryptosystem failure is orders of magnitude below any other risk."

http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

WEP

WEP int
in 802.1

2001 Bo
24-bit "
leaking
allowing

2001 Ar
this also

2001 Flu
WEP bu
from sec
RC4 out

s: "A chosen-
ack against
ed for Flame.
. . . not our
prefix collision
out an entirely
variant."

e wavesup3.drv
n 2007; Flame
ctive for as long
ars".

brew crypto"?
widely used.
ack? Yes.

Compare to 2011 Grigg–Gutmann:
"Cryptosystem failure is orders of magnitude below any other risk."

http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

WEP

WEP introduced i
in 802.11 wireless

2001 Borisov–Gold
24-bit "nonce" fre
leaking plaintext x
allowing very easy

2001 Arbaugh–Sha
this also breaks us

2001 Fluhrer–Man
WEP builds RC4 k
from secret k, "no
RC4 outputs leak

sen-
t
ne.
ur
sion
irely

3.drv
ame
s long

o"?
ed.

Compare to 2011 Grigg–Gutmann:
"Cryptosystem failure is orders of magnitude below any other risk."

http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wag
24-bit "nonce" frequently re
leaking plaintext xor and
allowing very easy forgeries.

2001 Arbaugh–Shankar–Wa
this also breaks user auth.

2001 Fluhrer–Mantin–Shami
WEP builds RC4 key (k, n)
from secret k, "nonce" n;
RC4 outputs leak bytes of k

Compare to 2011 Grigg–Gutmann:
"Cryptosystem failure is orders of magnitude below any other risk."

http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner: 24-bit "nonce" frequently repeats, leaking plaintext xor and allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan: this also breaks user auth.

2001 Fluhrer–Mantin–Shamir: WEP builds RC4 key (k, n) from secret k, "nonce" n; RC4 outputs leak bytes of k.

e to 2011 Grigg–Gutmann:
system failure is orders of
de below any other risk."

/en.wikipedia.org/wiki
Mission_Accomplished
n

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner: 24-bit "nonce" frequently repeats, leaking plaintext xor and allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan: this also breaks user auth.

2001 Fluhrer–Mantin–Shamir: WEP builds RC4 key (k, n) from secret k, "nonce" n; RC4 outputs leak bytes of k.

Impleme
of k-reco
Stubblef
2004 Ko
d'Otrepp
Tews–W
Sepehrd
2013 S–

Grigg–Gutmann:
lure is orders of
any other risk.”

pedia.org/wiki
Accomplished

WEP

WEP introduced in 1997
in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner:
24-bit “nonce” frequently repeats,
leaking plaintext xor and
allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan:
this also breaks user auth.

2001 Fluhrer–Mantin–Shamir:
WEP builds RC4 key (k, n)
from secret k, “nonce” n;
RC4 outputs leak bytes of k.

Implementations,
of k-recovery atta
Stubblefield–Ioann
2004 KoreK, 2004
d’Otreppe, 2006 K
Tews–Weinmann–
Sepehrdad–Vaude
2013 S–Sušil–V–V

tmann:
ers of
risk.”

g/wiki
shed

WEP

WEP introduced in 1997
in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner:
24-bit “nonce” frequently repeats,
leaking plaintext xor and
allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan:
this also breaks user auth.

2001 Fluhrer–Mantin–Shamir:
WEP builds RC4 key (k, n)
from secret k, “nonce” n;
RC4 outputs leak bytes of k.

Implementations, optimizati
of k-recovery attack: 2001
Stubblefield–Ioannidis–Rubin
2004 KoreK, 2004 Devine, 2
d’Otreppe, 2006 Klein, 2007
Tews–Weinmann–Pyshkin, 2
Sepehrdad–Vaudenay–Vuagn
2013 S–Sušil–V–V, . . .

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner: 24-bit “nonce” frequently repeats, leaking plaintext xor and allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan: this also breaks user auth.

2001 Fluhrer–Mantin–Shamir: WEP builds RC4 key (k, n) from secret k, “nonce” n; RC4 outputs leak bytes of k.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner: 24-bit "nonce" frequently repeats, leaking plaintext xor and allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan: this also breaks user auth.

2001 Fluhrer–Mantin–Shamir: WEP builds RC4 key (k, n) from secret k, "nonce" n; RC4 outputs leak bytes of k.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

"These are academic papers! Nobody was actually attacked."

WEP

WEP introduced in 1997 in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner: 24-bit “nonce” frequently repeats, leaking plaintext xor and allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan: this also breaks user auth.

2001 Fluhrer–Mantin–Shamir: WEP builds RC4 key (k, n) from secret k, “nonce” n; RC4 outputs leak bytes of k.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d’Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

“These are academic papers! Nobody was actually attacked.”

Fact: WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for \$40900000.

troduced in 1997
1 wireless standard.

risov–Goldberg–Wagner:
nonce" frequently repeats,
plaintext xor and
very easy forgeries.

baugh–Shankar–Wan:
breaks user auth.

uhrer–Mantin–Shamir:
ilds RC4 key (k, n)
cret k, "nonce" n;
tputs leak bytes of k.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

"These are academic papers! Nobody was actually attacked."

Fact: WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for \$40900000.

Keeloq

Wikiped
used in
entry sys
as Chrys
GM, Ho
Volkswa
Shurlok,

2007 Ind
Biham–D
"How to
recover
using 2[1
only 2[44

n 1997
standard.

dberg–Wagner:
equently repeats,
or and
forgeries.

ankar–Wan:
er auth.

tin–Shamir:
key (k, n)
nce" n;
bytes of k.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

"These are academic papers! Nobody was actually attacked."

Fact: WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Keeloq

Wikipedia: "KeeL
used in many rem
entry systems by s
as Chrysler, Daew
GM, Honda, Toyo
Volkswagen Group
Shurlok, Jaguar, e

2007 Indesteege–K
Biham–Dunkelma
"How to steal car
recover 64-bit Kee
using 2^{16} known p
only $2^{44.5}$ encrypti

gner:
epeats,

n:

ir:

.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

"These are academic papers! Nobody was actually attacked."

Fact: WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Keeloq

Wikipedia: "KeeLoq is or wa used in many remote keyless entry systems by such comp as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

Implementations, optimizations of k-recovery attack: 2001 Stubblefield–Ioannidis–Rubin, 2004 KoreK, 2004 Devine, 2005 d'Otreppe, 2006 Klein, 2007 Tews–Weinmann–Pyshkin, 2010 Sepehrdad–Vaudenay–Vuagnoux, 2013 S–Sušil–V–V, . . .

"These are academic papers! Nobody was actually attacked."

Fact: WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Keeloq

Wikipedia: "KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

entations, optimizations
overy attack: 2001
ield–Ioannidis–Rubin,
oreK, 2004 Devine, 2005
pe, 2006 Klein, 2007
Veinmann–Pyshkin, 2010
ad–Vaudenay–Vuagnoux,
Sušil–V–V, . . .

are academic papers!
was actually attacked.”

VEP blamed for 2007 theft
llion credit-card numbers
J. Maxx. Subsequent
settled for $40900000.

Keeloq

Wikipedia: “KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc.”

2007 Indesteege–Keller–Biham–Dunkelman–Preneel “How to steal cars”: recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eis
Paar–Sa
recovere
allowing
cloning

optimizations
ck: 2001
idis–Rubin,
Devine, 2005
Klein, 2007
Pyshkin, 2010
nay–Vuagnoux,
, . . .

mic papers!
lly attacked."

d for 2007 theft
t-card numbers
Subsequent
$40900000.

Keeloq

Wikipedia: "KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eisenbarth–K
Paar–Salmasizadel
recovered system's
allowing practically
cloning of KeeLoq

ons

n,
2005
7
2010
houx,

!
ed.”

7 theft
mbers
nt
0.

Keeloq

Wikipedia: “KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc.”

2007 Indesteege–Keller–Biham–Dunkelman–Preneel
“How to steal cars”: recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eisenbarth–Kasper–Mo
Paar–Salmasizadeh–Shalmar
recovered system’s *master* k
allowing practically instantar
cloning of KeeLoq keys.

Keeloq

Wikipedia: "KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

Keeloq

Wikipedia: "KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

Keeloq

Wikipedia: "KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc."

2007 Indesteege–Keller–Biham–Dunkelman–Preneel "How to steal cars": recover 64-bit KeeLoq key using 2^{16} known plaintexts, only $2^{44.5}$ encryptions.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

2. If all the "X is weak" press comes from academics, is it safe to conclude that real attackers aren't breaking X? How often do real attackers issue press releases?

ia: “KeeLoq is or was
many remote keyless
stems by such companies
sler, Daewoo, Fiat,
nda, Toyota, Volvo,
gen Group, Clifford,
Jaguar, etc.”

desteege–Keller–
Dunkelman–Preneel
steal cars”:
64-bit KeeLoq key
^{6} known plaintexts,
$^{.5}$ encryptions.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system’s *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this “bypassing” the cryptography?

2. If all the “X is weak” press comes from academics, is it safe to conclude that real attackers aren’t breaking X? How often do real attackers issue press releases?

VMWare

VMWare
desktop
many lo

Recomm
Dell, etc
to “SAL
user exp
slows do

oq is or was
ote keyless
such companies
oo, Fiat,
ta, Volvo,
, Clifford,
tc."

Keller–
n–Preneel
s":
eLoq key
laintexts,
ions.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

2. If all the "X is weak" press comes from academics, is it safe to conclude that real attackers aren't breaking X? How often do real attackers issue press releases?

VMWare View

VMWare View is a
desktop protocol s
many low-cost ter

Recommendation
Dell, etc.: switch
to "SALSA20-256
user experience".
slows down netwo

as
s
anies

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

2. If all the "X is weak" press comes from academics, is it safe to conclude that real attackers aren't breaking X? How often do real attackers issue press releases?

VMWare View

VMWare View is a remote desktop protocol supported many low-cost terminals.

Recommendation from VMW
Dell, etc.: switch from "AES
to "SALSA20-256" for the "
user experience". Apparentl
slows down network graphic

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

2. If all the "X is weak" press comes from academics, is it safe to conclude that real attackers aren't breaking X? How often do real attackers issue press releases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

2008 Eisenbarth–Kasper–Moradi–Paar–Salmasizadeh–Shalmani recovered system's *master* key, allowing practically instantaneous cloning of KeeLoq keys.

1. Setup phase of this attack watches power consumption of Keeloq device. Is this "bypassing" the cryptography?

2. If all the "X is weak" press comes from academics, is it safe to conclude that real attackers aren't breaking X? How often do real attackers issue press releases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

Closer look at documentation: "AES-128" and "SALSA20-256" are actually "AES-128-GCM" and "Salsa20-256-Round12".

senbarth–Kasper–Moradi–
lmasizadeh–Shalmani
d system's *master* key,
practically instantaneous
of KeeLoq keys.

phase of this attack
power consumption
q device. Is this
ng" the cryptography?

the "X is weak" press
rom academics, is it safe
ude that real attackers
reaking X? How often do
ckers issue press releases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

Closer look at documentation: "AES-128" and "SALSA20-256" are actually "AES-128-GCM" and "Salsa20-256-Round12".

AES-128
and mes
No indic
"Salsa20
any mes
Can atta
One *can*
with me
but *does*

Salsa20
advanta
both Sal
unauthe
User nee

Kasper–Moradi–
h–Shalmani
s *master* key,
y instantaneous
keys.

this attack
nsumption
Is this
ryptography?

weak" press
mics, is it safe
eal attackers
? How often do
e press releases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

Closer look at documentation: "AES-128" and "SALSA20-256" are actually "AES-128-GCM" and "Salsa20-256-Round12".

AES-128-GCM inc
and message auth

No indication that
"Salsa20-256-Rou
any message authe
Can attacker forge
One *can* easily co
with message auth
but *does* VMWare

Salsa20 has speed
advantages over A
both Salsa20 and
unauthenticated c
User needs *authen*

oradi–
ni
ey,
neous

k

ny?

ess
safe
ers
en do
eases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

Closer look at documentation: "AES-128" and "SALSA20-256" are actually "AES-128-GCM" and "Salsa20-256-Round12".

AES-128-GCM includes AES
and message authentication

No indication that VMWare
"Salsa20-256-Round12" incl
any message authentication.
Can attacker forge packets?
One *can* easily combine Sals
with message authentication
but *does* VMWare do this?

Salsa20 has speed and secur
advantages over AES, but
both Salsa20 and AES are
unauthenticated ciphers.
User needs *authenticated* ci

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from VMWare, Dell, etc.: switch from "AES-128" to "SALSA20-256" for the "best user experience". Apparently AES slows down network graphics.

Closer look at documentation: "AES-128" and "SALSA20-256" are actually "AES-128-GCM" and "Salsa20-256-Round12".

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication. Can attacker forge packets? One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers. User needs *authenticated* cipher.

e View

e View is a remote
protocol supported by
w-cost terminals.

nendation from VMWare,
:.: switch from "AES-128"
SA20-256" for the "best
erience". Apparently AES
own network graphics.

ook at documentation:
28" and "SALSA20-256"
ally "AES-128-GCM"
lsa20-256-Round12".

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication.
Can attacker forge packets?
One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.
User needs *authenticated* cipher.

SSL/TL

Standard
of a pac
send ran
$c_0 =$ AE
$c_1 =$ AE
$c_2 =$ AE

a remote
supported by
minals.

from VMWare,
from "AES-128"
" for the "best
Apparently AES
rk graphics.

umentation:
SALSA20-256"
-128-GCM"
Round12".

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication.
Can attacker forge packets?
One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.
User needs *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CB
of a packet $(p_0, p_1$
send random v,
$c_0 = \text{AES}_k(p_0 \oplus v$
$c_1 = \text{AES}_k(p_1 \oplus c$
$c_2 = \text{AES}_k(p_2 \oplus c$

by

Ware,
S-128”
‘best
y AES
s.

on:
256”
”
.

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare’s “Salsa20-256-Round12” includes any message authentication.
Can attacker forge packets?
One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.
User needs *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CBC encrypti
of a packet (p_0, p_1, p_2):
send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication.
Can attacker forge packets?
One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.
User needs *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CBC encryption of a packet (p_0, p_1, p_2):
send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication.
Can attacker forge packets?
One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.
User needs *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CBC encryption of a packet (p_0, p_1, p_2):
send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-CBC encryption in SSL:
retrieve last block c_{-1} from previous ciphertext; send
$c_0 = \text{AES}_k(p_0 \oplus c_{-1})$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

-GCM includes AES
ssage authentication.

ation that VMWare's
-256-Round12" includes
sage authentication.
acker forge packets?
easily combine Salsa20
ssage authentication,
VMWare do this?

has speed and security
ges over AES, but
lsa20 and AES are
nticated ciphers.
eds *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CBC encryption of a packet (p_0, p_1, p_2): send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-CBC encryption in SSL: retrieve last block c_{-1} from previous ciphertext; send
$c_0 = \text{AES}_k(p_0 \oplus c_{-1})$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

SSL lets
as functi

2002 Mö
To chec
choose p
compare

2006 Ba
maliciou
be able
especiall
is split a

2011 Du
fast atta
including

cludes AES
entication.

: VMWare's
nd12" includes
entication.
e packets?
mbine Salsa20
nentication,
e do this?

and security
AES, but
AES are
iphers.
nticated cipher.

SSL/TLS/HTTPS

Standard AES-CBC encryption
of a packet (p_0, p_1, p_2):
send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-CBC encryption in SSL:
retrieve last block c_{-1}
from previous ciphertext; send
$c_0 = \text{AES}_k(p_0 \oplus c_{-1})$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

SSL lets attacker c
as function of c_{-1}

2002 Möller:
To check a guess
choose $p_0 = c_{-1} \oplus$
compare c_0 to c_{-3}

2006 Bard:
malicious code in
be able to carry ou
especially if high-e
is split across bloc

2011 Duong–Rizzo
fast attack fully in
including controlle

S
.
's
udes
sa20
n,
rity
pher.

SSL/TLS/HTTPS

Standard AES-CBC encryption of a packet (p_0, p_1, p_2): send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-CBC encryption in SSL: retrieve last block c_{-1} from previous ciphertext; send
$c_0 = \text{AES}_k(p_0 \oplus c_{-1})$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

SSL lets attacker choose p_0
as function of c_{-1}! Very bac

2002 Möller:
To check a guess g for (e.g.
choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$,
compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser sh
be able to carry out this att
especially if high-entropy da
is split across blocks.

2011 Duong–Rizzo "BEAST
fast attack fully implemente
including controlled variable

SSL/TLS/HTTPS

Standard AES-CBC encryption of a packet (p_0, p_1, p_2): send random v,
$c_0 = \text{AES}_k(p_0 \oplus v)$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

AES-CBC encryption in SSL: retrieve last block c_{-1} from previous ciphertext; send
$c_0 = \text{AES}_k(p_0 \oplus c_{-1})$,
$c_1 = \text{AES}_k(p_1 \oplus c_0)$,
$c_2 = \text{AES}_k(p_2 \oplus c_1)$.

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo “BEAST”: fast attack fully implemented, including controlled variable split.

S/HTTPS

d AES-CBC encryption
ket (p_0, p_1, p_2):
ndom v,
$\mathrm{S}_k(p_0 \oplus v)$,
$\mathrm{S}_k(p_1 \oplus c_0)$,
$\mathrm{S}_k(p_2 \oplus c_1)$.

C encryption in SSL:
last block c_{-1}
evious ciphertext; send
$\mathrm{S}_k(p_0 \oplus c_{-1})$,
$\mathrm{S}_k(p_1 \oplus c_0)$,
$\mathrm{S}_k(p_2 \oplus c_1)$.

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo "BEAST":
fast attack fully implemented, including controlled variable split.

Counter
send a c
just befo

S

C encryption

$, p_2$):

),

$_0$),

$_1$).

ion in SSL:

c_{-1}

ertext; send

$_{-1}$),

$_0$),

$_1$).

SSL lets attacker choose p_0
as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3},
choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$,
compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should
be able to carry out this attack,
especially if high-entropy data
is split across blocks.

2011 Duong–Rizzo "BEAST":
fast attack fully implemented,
including controlled variable split.

Countermeasure in
send a content-fre
just before sending

ion

:

nd

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo "BEAST": fast attack fully implemented, including controlled variable split.

Countermeasure in browsers send a content-free packet just before sending real pack

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo "BEAST":
fast attack fully implemented, including controlled variable split.

Countermeasure in browsers: send a content-free packet just before sending real packet.

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo "BEAST": fast attack fully implemented, including controlled variable split.

Countermeasure in browsers: send a content-free packet just before sending real packet.

Attacker can also try to attack CBC by forging *ciphertexts*, but each SSL packet includes an authenticator.

"Authenticate-then-encrypt": SSL appends an authenticator, pads reversibly to full block, encrypts with CBC.

SSL lets attacker choose p_0 as function of c_{-1}! Very bad.

2002 Möller:
To check a guess g for (e.g.) p_{-3}, choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$, compare c_0 to c_{-3}.

2006 Bard:
malicious code in browser should be able to carry out this attack, especially if high-entropy data is split across blocks.

2011 Duong–Rizzo "BEAST": fast attack fully implemented, including controlled variable split.

Countermeasure in browsers: send a content-free packet just before sending real packet.

Attacker can also try to attack CBC by forging *ciphertexts*, but each SSL packet includes an authenticator.

"Authenticate-then-encrypt": SSL appends an authenticator, pads reversibly to full block, encrypts with CBC.

2001 Krawczyk:
This is provably secure.

attacker choose p_0
ion of c_{-1}! Very bad.

öller:
k a guess g for (e.g.) p_{-3},
$p_0 = c_{-1} \oplus g \oplus c_{-4}$,
c_0 to c_{-3}.

rd:
s code in browser should
to carry out this attack,
y if high-entropy data
cross blocks.

ong–Rizzo "BEAST":
ck fully implemented,
g controlled variable split.

Countermeasure in browsers: send a content-free packet just before sending real packet.

Attacker can also try to attack CBC by forging *ciphertexts*, but each SSL packet includes an authenticator.

"Authenticate-then-encrypt": SSL appends an authenticator, pads reversibly to full block, encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Va
This is c
if attack
padding

choose p_0
! Very bad.

g for (e.g.) p_{-3},
$\oplus g \oplus c_{-4}$,
$_3$.

browser should
ut this attack,
entropy data
ks.

o "BEAST":
nplemented,
ed variable split.

Countermeasure in browsers: send a content-free packet just before sending real packet.

Attacker can also try to attack CBC by forging *ciphertexts*, but each SSL packet includes an authenticator.

"Authenticate-then-encrypt": SSL appends an authenticator, pads reversibly to full block, encrypts with CBC.

2001 Krawczyk: This is provably secure.

2001 Vaudenay:
This is completely
if attacker can dis
padding failure fro

d.

) p_{-3},

hould
ack,
ta

”:
d,
split.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

“Authenticate-then-encrypt”:
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC fa

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

"Authenticate-then-encrypt":
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack CBC by forging *ciphertexts*, but each SSL packet includes an authenticator.

"Authenticate-then-encrypt": SSL appends an authenticator, pads reversibly to full block, encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Vaudenay:
This is completely broken if attacker can distinguish padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle by observing SSL server timing.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

"Authenticate-then-encrypt":
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

"Authenticate-then-encrypt":
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

2001 Krawczyk:
This is provably secure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

measure in browsers:
ontent-free packet
ore sending real packet.

r can also try to attack
forging *ciphertexts*,
n SSL packet
an authenticator.

ticate-then-encrypt":
pends an authenticator,
ersibly to full block,
with CBC.

awczyk:
provably secure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Crypto

n browsers:
e packet
g real packet.

try to attack
phertexts,
ket
nticator.

n-encrypt":
uthenticator,
full block,
C.

ecure.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic alg

:

ket.

ack

”:

or,

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
“Lucky 13”: watch timing
more closely; attack still works.

“Cryptographic algorithm ag

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic algorithm agility":

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto
is okay if there's a backup plan

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto
is okay if there's a backup plan +
(2) the pretense that there
is in fact a backup plan.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto
is okay if there's a backup plan +
(2) the pretense that there
is in fact a backup plan.

SSL has a crypto switch
that in theory allows
switching to AES-GCM.
But most SSL software
doesn't support AES-GCM.

2001 Vaudenay:
This is completely broken
if attacker can distinguish
padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle
by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing
more closely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto
is okay if there's a backup plan +
(2) the pretense that there
is in fact a backup plan.

SSL has a crypto switch
that in theory allows
switching to AES-GCM.
But most SSL software
doesn't support AES-GCM.

The software does support
one non-CBC option:

2001 Vaudenay:
This is completely broken if attacker can distinguish padding failure from MAC failure.

2003 Canvel:
Obtain such a padding oracle by observing SSL server timing.

Response in OpenSSL etc.: always compute MAC even if padding fails.

2013.02 AlFardan–Paterson
"Lucky 13": watch timing more closely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto is okay if there's a backup plan +
(2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM.
But most SSL software doesn't support AES-GCM.

The software does support one non-CBC option: RC4.
2013.01: widely recommended, used for 50% of SSL traffic.

udenay:
completely broken
ker can distinguish
failure from MAC failure.

nvel:
such a padding oracle
rving SSL server timing.

e in OpenSSL etc.:
ompute MAC
padding fails.

AlFardan–Paterson
13": watch timing
sely; attack still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto
is okay if there's a backup plan +
(2) the pretense that there
is in fact a backup plan.

SSL has a crypto switch
that in theory allows
switching to AES-GCM.
But most SSL software
doesn't support AES-GCM.

The software does support
one non-CBC option: RC4.
2013.01: widely recommended,
used for 50% of SSL traffic.

Not as s
hash to
2001 Riv
do not a
. . . [prot
RC4] sho

broken
tinguish
om MAC failure.

lding oracle
server timing.

SSL etc.:
IAC
ils.

-Paterson
h timing
ck still works.

"Cryptographic algorithm agility":
(1) the pretense that bad crypto is okay if there's a backup plan +
(2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM.
But most SSL software doesn't support AES-GCM.

The software does support one non-CBC option: RC4.
2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as W
hash to avoid relat
2001 Rivest: "The
do not apply to RC
. . . [protocol] desi
RC4] should not b

ailure.

e

ing.

rks.

“Cryptographic algorithm agility”:
(1) the pretense that bad crypto is okay if there’s a backup plan + (2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM. But most SSL software doesn’t support AES-GCM.

The software does support one non-CBC option: RC4. 2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as WEP: SSL
hash to avoid related RC4 k
2001 Rivest: “The new atta
do not apply to RC4-based S
. . . [protocol] designers [usi
RC4] should not be concern

“Cryptographic algorithm agility”:
(1) the pretense that bad crypto is okay if there’s a backup plan + (2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM. But most SSL software doesn’t support AES-GCM.

The software does support one non-CBC option: RC4. 2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: “The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned.”

“Cryptographic algorithm agility”: (1) the pretense that bad crypto is okay if there’s a backup plan + (2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM. But most SSL software doesn’t support AES-GCM.

The software does support one non-CBC option: RC4. 2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: “The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned.”

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

"Cryptographic algorithm agility": (1) the pretense that bad crypto is okay if there's a backup plan + (2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM. But most SSL software doesn't support AES-GCM.

The software does support one non-CBC option: RC4. 2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys. 2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

graphic algorithm agility":
pretense that bad crypto
f there's a backup plan +
pretense that there
t a backup plan.

a crypto switch
theory allows
g to AES-GCM.
st SSL software
support AES-GCM.

tware does support
-CBC option: RC4.
: widely recommended,
50% of SSL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The sing

2001 Ma

$z_2 \rightarrow 0$.

gorithm agility":
hat bad crypto
backup plan +
hat there
plan.

switch
ws
GCM.
tware
ES-GCM.

support
on: RC4.
ecommended,
SL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.
2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte bi

2001 Mantin–Shar
$z_2 \to 0$.

gility":
ypto
lan +

ed,

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys. 2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir: $z_2 \to 0$.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.
2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:

$z_2 \to 0$.

2002 Mironov:

$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$, contrary to Mantin–Shamir claim.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$, contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: "The new attacks do not apply to RC4-based SSL. . . . [protocol] designers [using RC4] should not be concerned."

Problem: many nasty biases in RC4 output bytes $z_1, z_2, \ldots$.

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt, "On the security of RC4 in TLS": Force target cookie into many RC4 sessions. Use RC4 biases to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$, contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

scary as WEP: SSL uses a
avoid related RC4 keys.

vest: "The new attacks
apply to RC4-based SSL.
tocol] designers [using
ould not be concerned."

: many nasty biases in
tput bytes $z_1, z_2, \ldots$.

Fardan–Bernstein–
n–Poettering–Schuldt,
security of RC4 in TLS":
rget cookie into many
sions. Use RC4 biases
cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–
Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlF
Paterson
accurate
for all i
found $\approx$
used *all*
via prop

/EP: SSL uses a
ted RC4 keys.
e new attacks
C4-based SSL.
gners [using
be concerned."

asty biases in
$z_1, z_2, \ldots$.

ernstein–
ng–Schuldt,
of RC4 in TLS":
e into many
e RC4 biases
n ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlFardan–Be
Paterson–Poetteri
accurately comput
for all $i \in \{1, \ldots,$
found ≈**65536** sin
used *all* of them i
via proper Bayesia

uses a
eys.
cks
SSL.
ng
ed.”

in
.

t,
TLS”:
ny
es
xts.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuld
accurately computed $\Pr[z_i$ =
for all $i \in \{1, \ldots, 256\}$, all
found $\approx$**65536** single-byte b
used *all* of them in SSL atta
via proper Bayesian analysis

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \ldots, 256\}$, all j;
found ≈**65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

The single-byte biases:

2001 Mantin–Shamir:
$z_2 \to 0$.

2002 Mironov:
$z_1 \not\to 0$, $z_1 \not\to 1$, $z_1 \to 2$, etc.

2011 Maitra–Paul–Sen Gupta:
$z_3 \to 0$, $z_4 \to 0$, $\ldots$, $z_{255} \to 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–Sarkar: $z_{16} \to 240$.
(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \ldots, 256\}$, all j;
found $\approx$**65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

$\approx$256 of these biases were found
independently (slightly earlier)
by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii:
$z_{32} \to 224$, $z_{48} \to 208$, etc.;
$z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

gle-byte biases:

antin–Shamir:

ronov:

$z_1 \not\to 1$, $z_1 \to 2$, etc.

aitra–Paul–Sen Gupta:

$z_4 \to 0, \ldots, z_{255} \to 0$,

to Mantin–Shamir claim.

n Gupta–Maitra–Paul–

$z_{16} \to 240$.

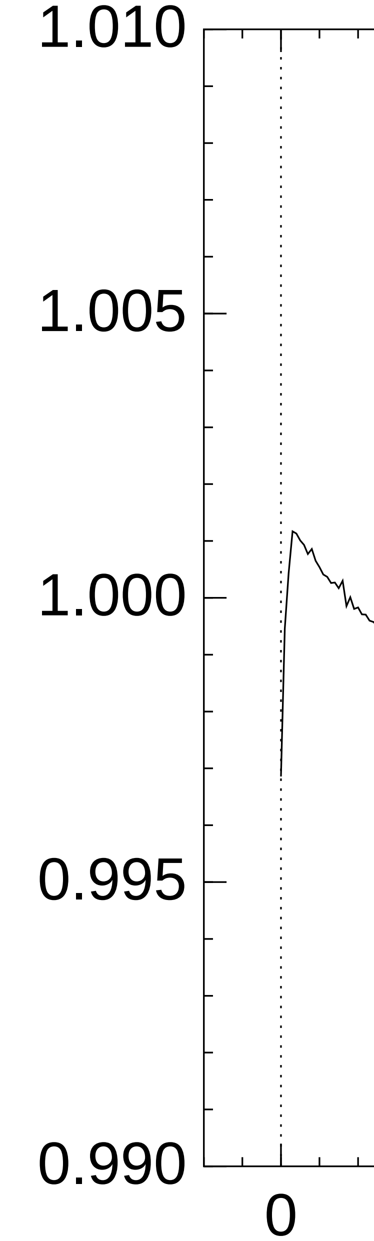
specific to 128-bit keys.)

t: there's more!

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \ldots, 256\}$, all j;
found ≈**65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

≈256 of these biases were found
independently (slightly earlier)
by 2013 Watanabe–Isobe–
Ohigashi–Morii, 2013 Isobe–
Ohigashi–Watanabe–Morii:
$z_{32} \to 224$, $z_{48} \to 208$, etc.;
$z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

Graph o



ases:

mir:

$_1 \to 2$, etc.

–Sen Gupta:

. . . , $z_{255} \to 0$,

n–Shamir claim.

Maitra–Paul–

0.

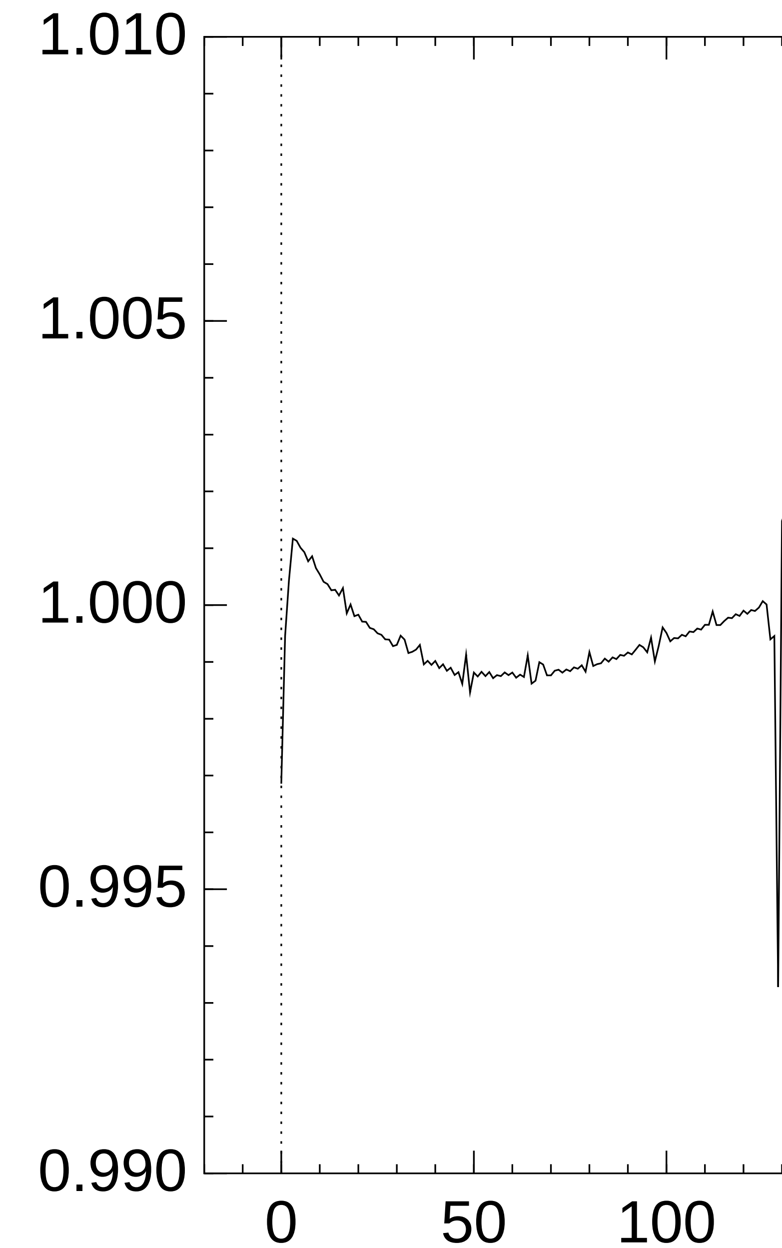
128-bit keys.)

more!

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \ldots, 256\}$, all j;
found ≈**65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

≈256 of these biases were found
independently (slightly earlier)
by 2013 Watanabe–Isobe–
Ohigashi–Morii, 2013 Isobe–
Ohigashi–Watanabe–Morii:
$z_{32} \to 224$, $z_{48} \to 208$, etc.;
$z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

Graph of $256 \Pr[z$



c.

ta:

→ 0,

claim.

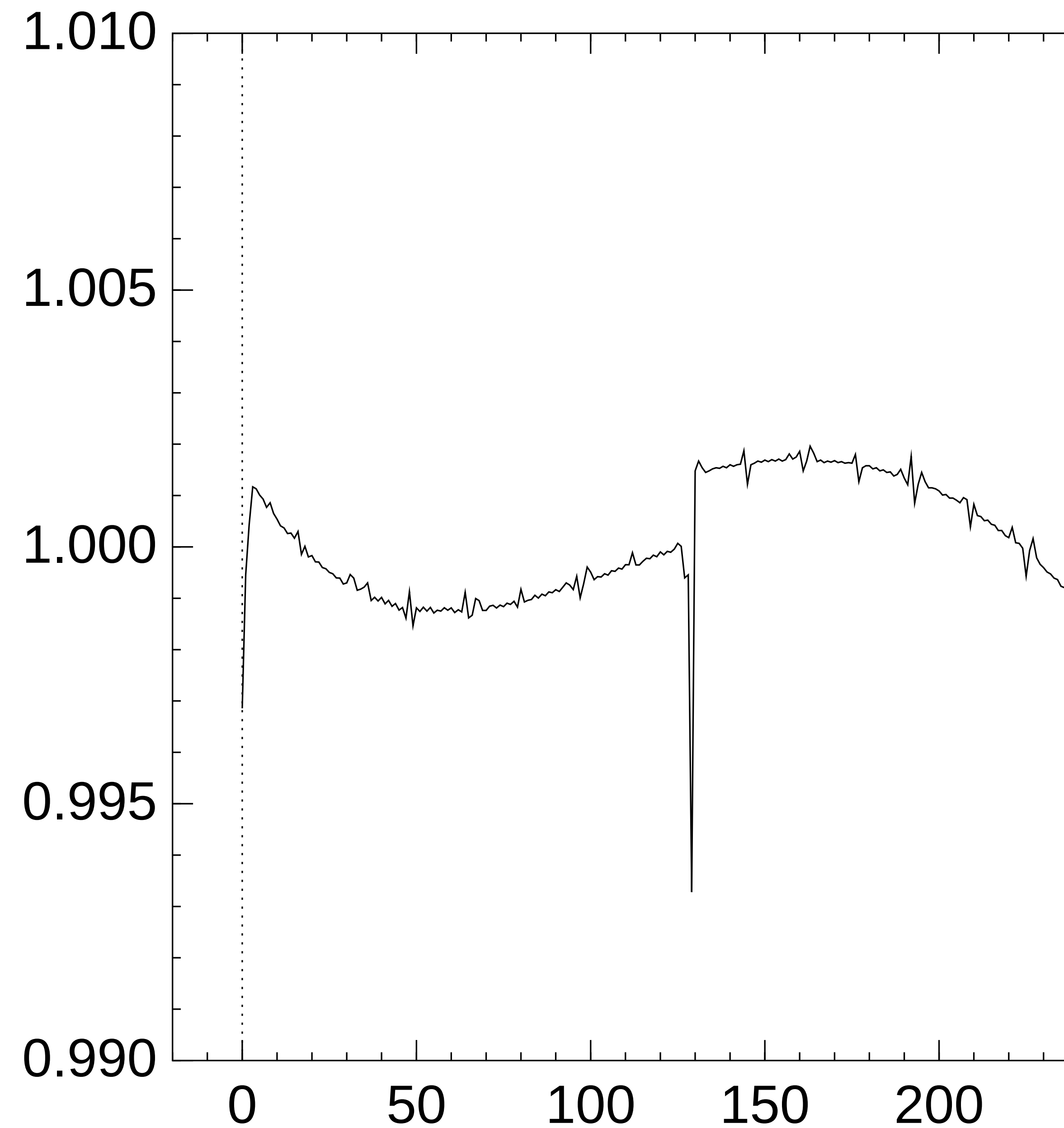
ul–

keys.)

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

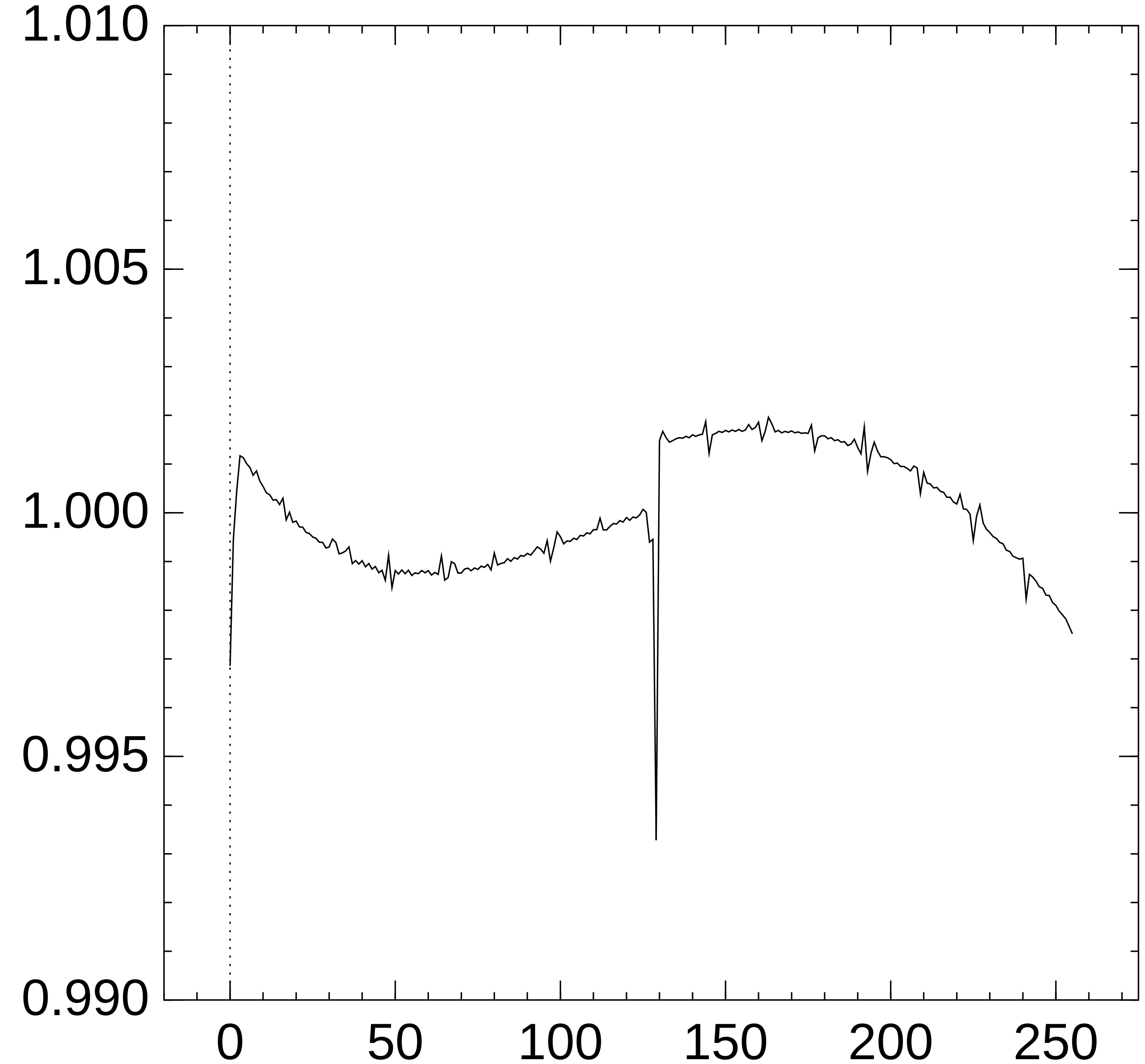
Graph of $256\,\Pr[z_1 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

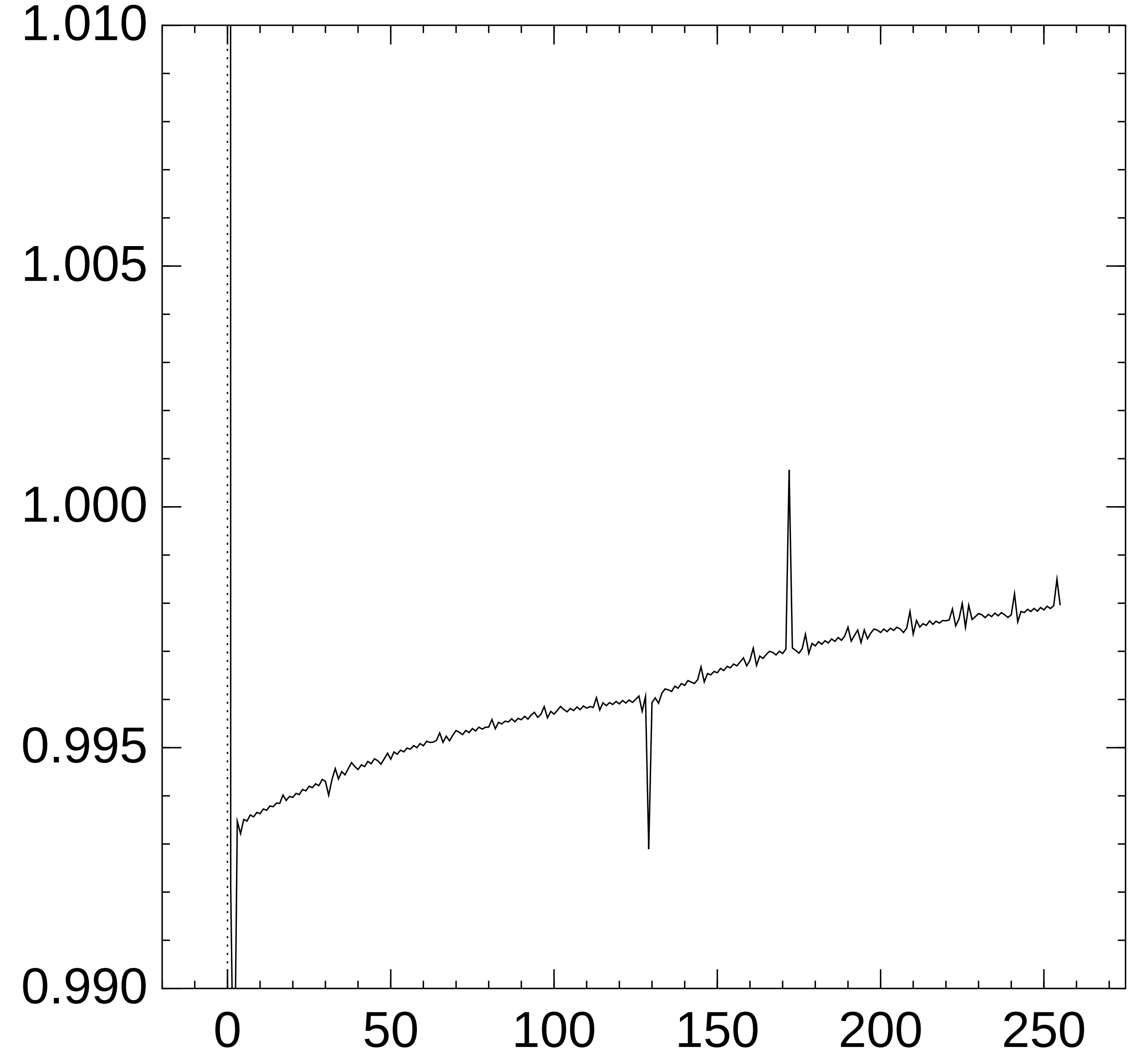
Graph of $256 \Pr[z_1 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

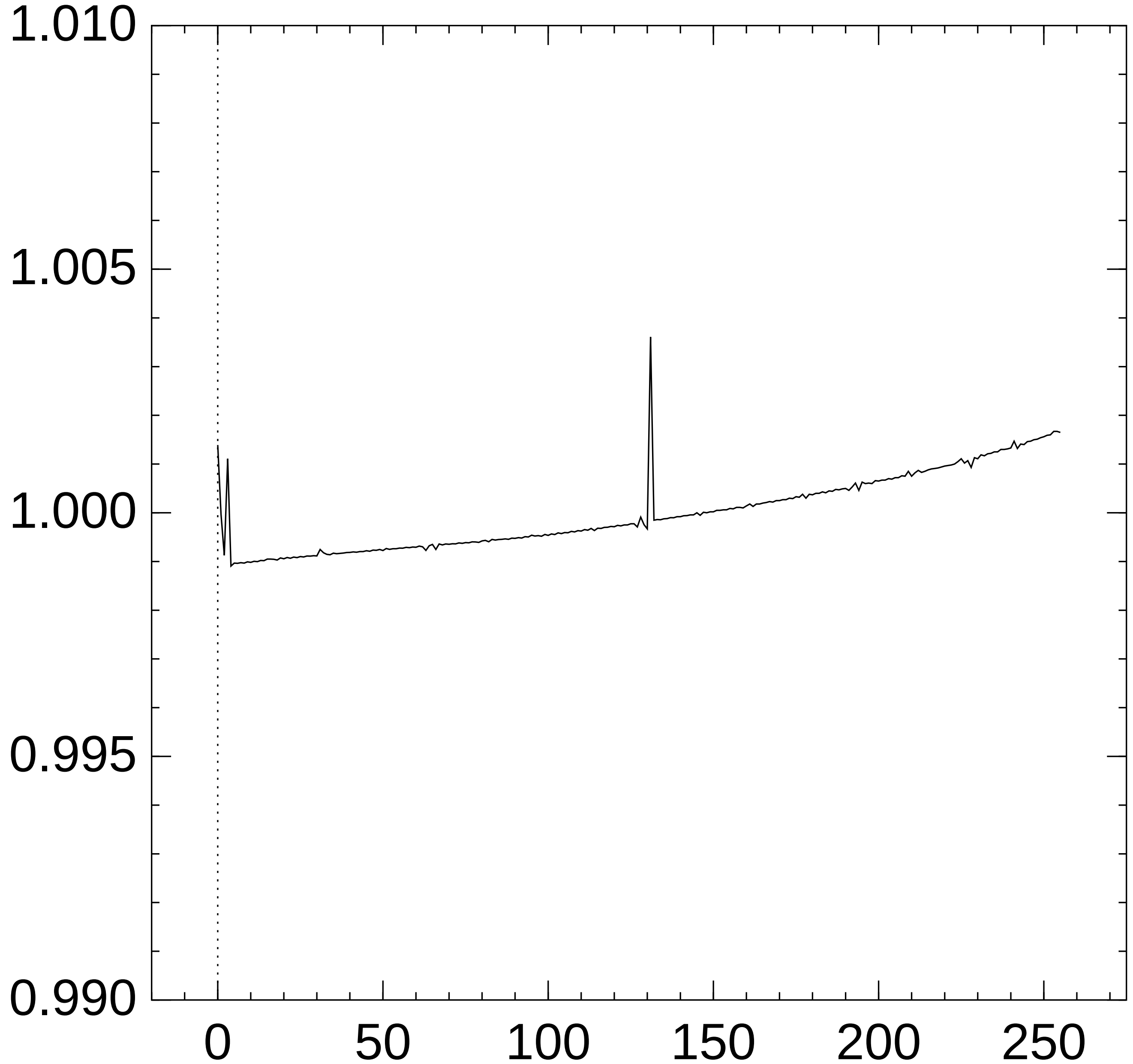
Graph of $256 \Pr[z_2 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

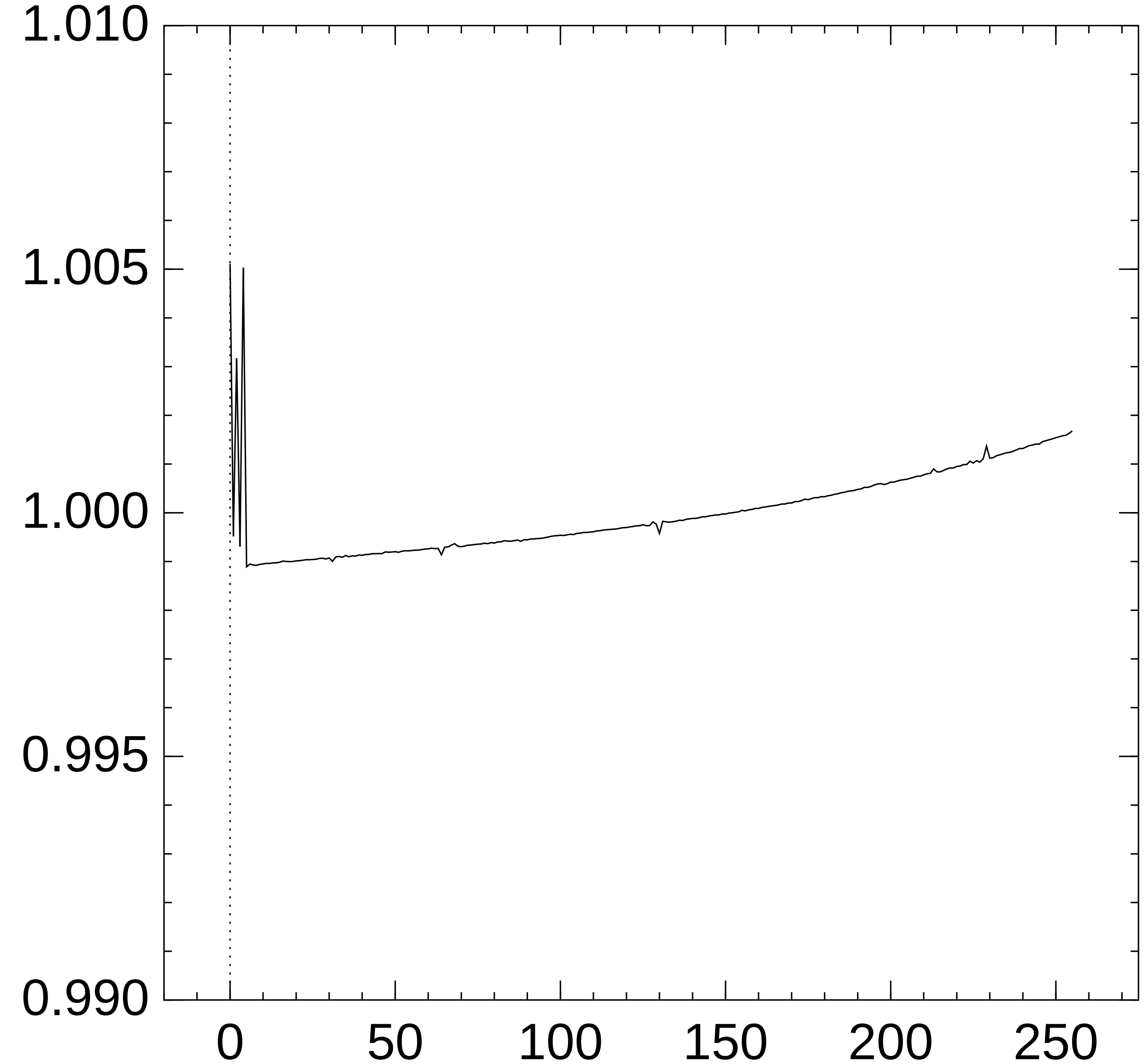
Graph of $256 \Pr[z_3 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

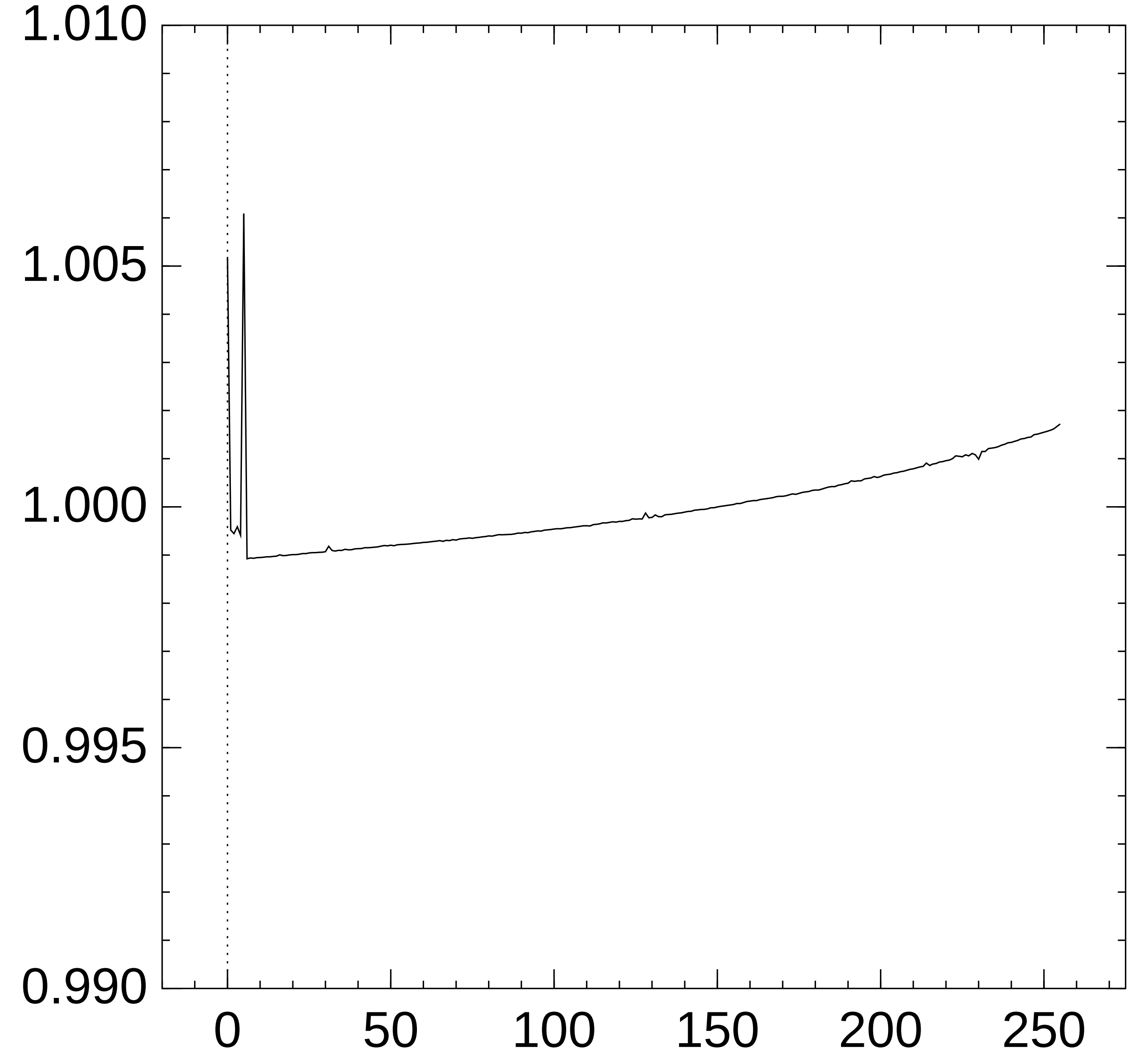
Graph of $256 \Pr[z_4 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

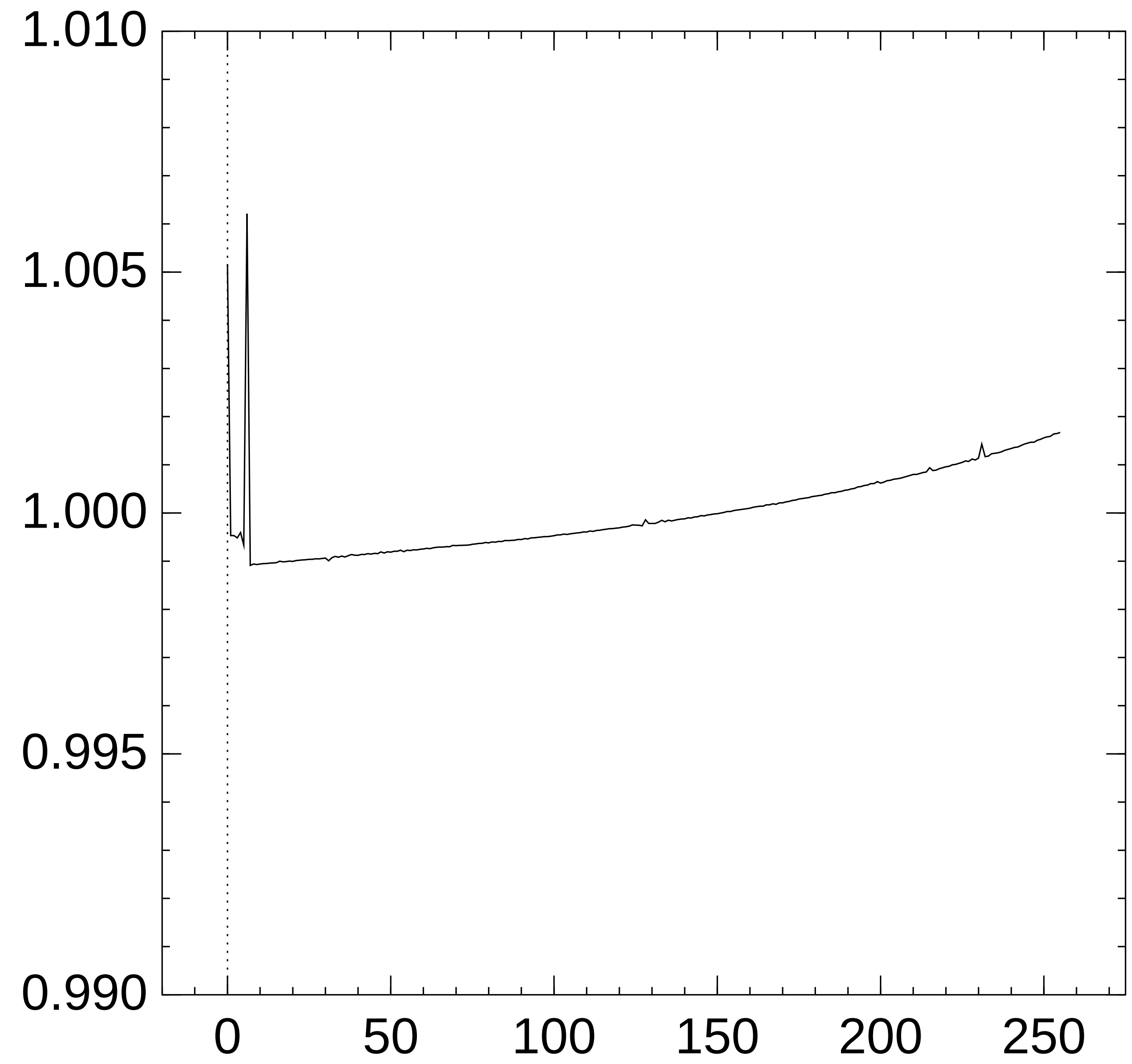
Graph of $256 \Pr[z_5 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

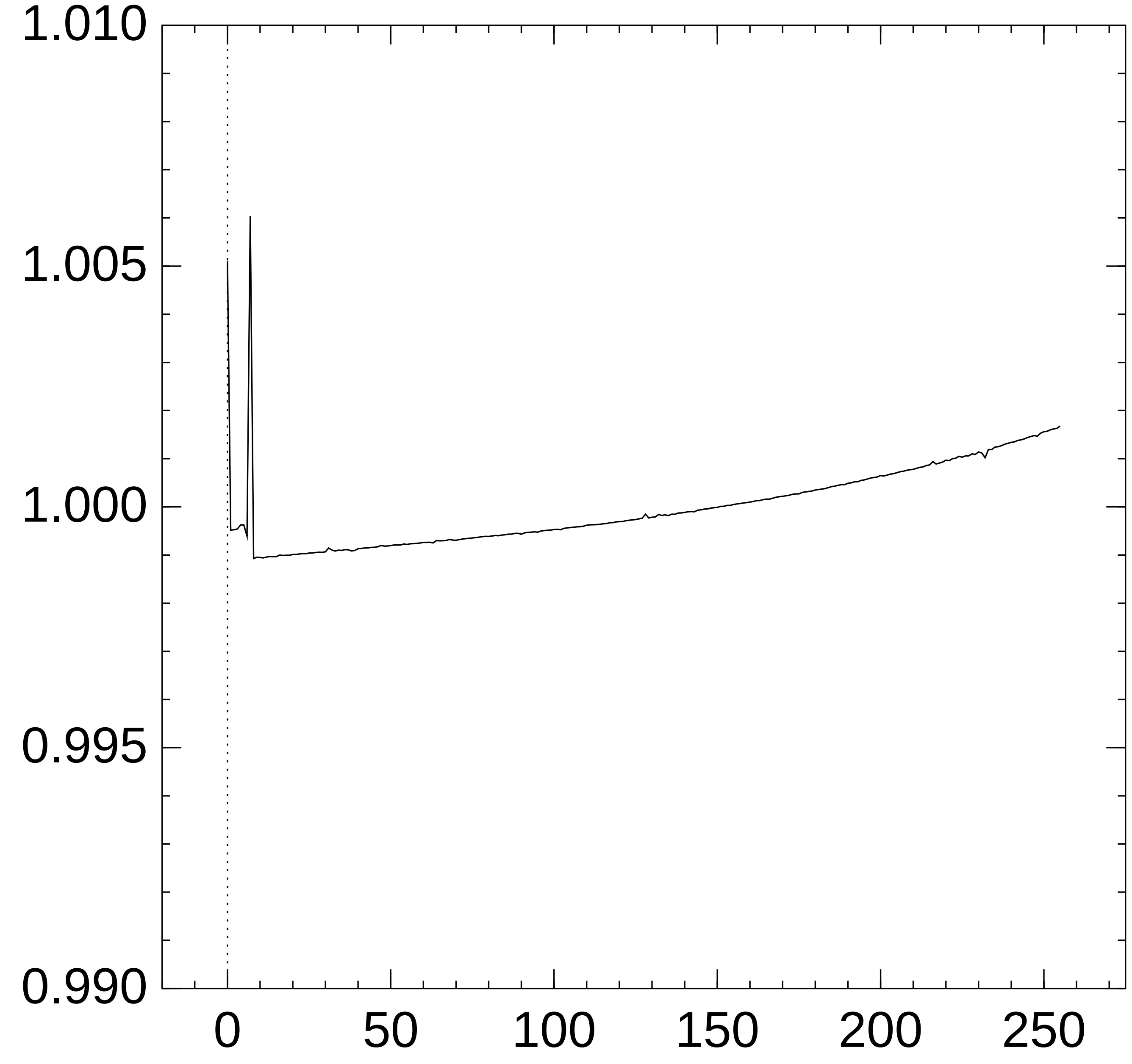
Graph of $256 \Pr[z_6 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

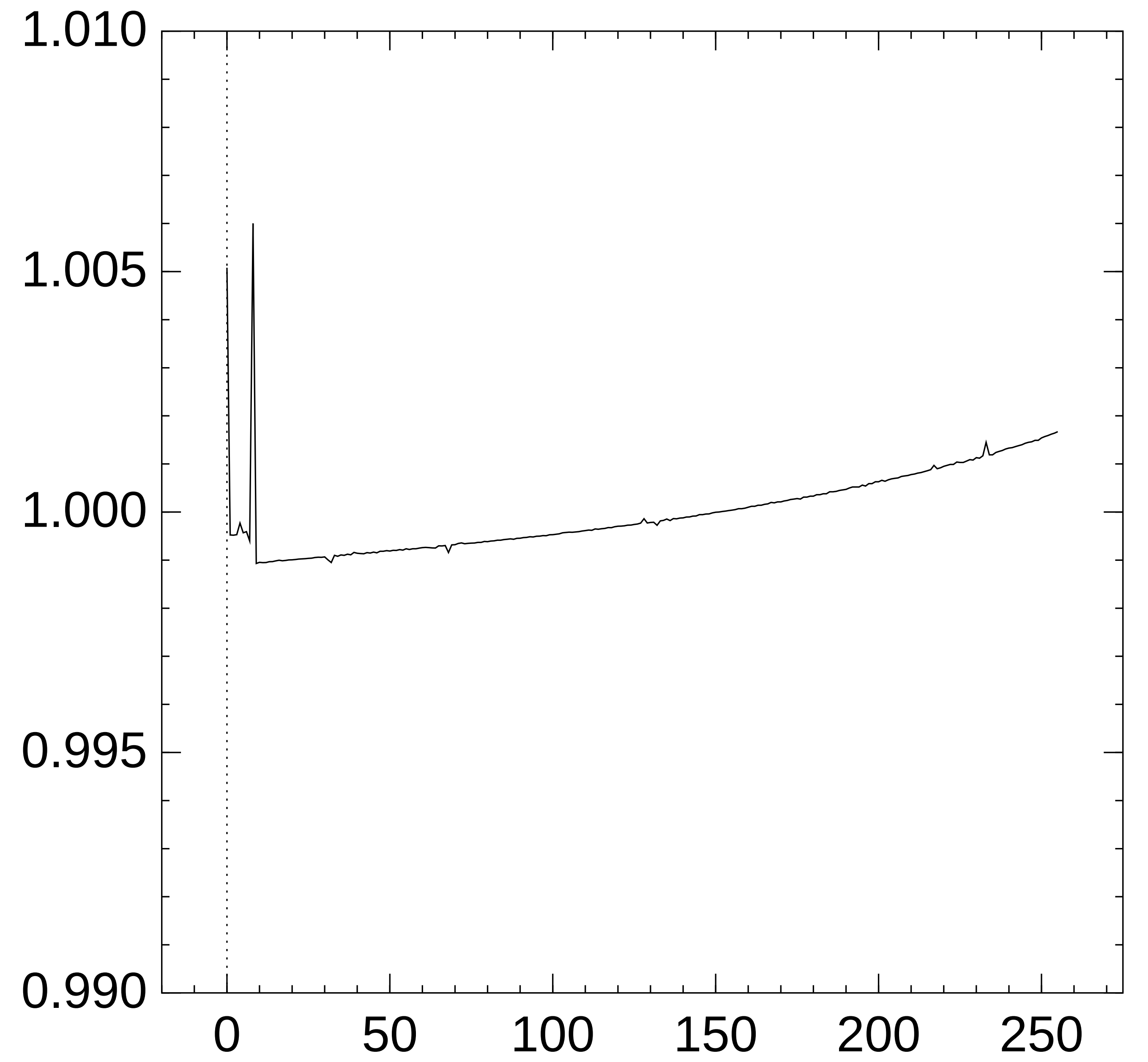
Graph of $256 \Pr[z_7 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

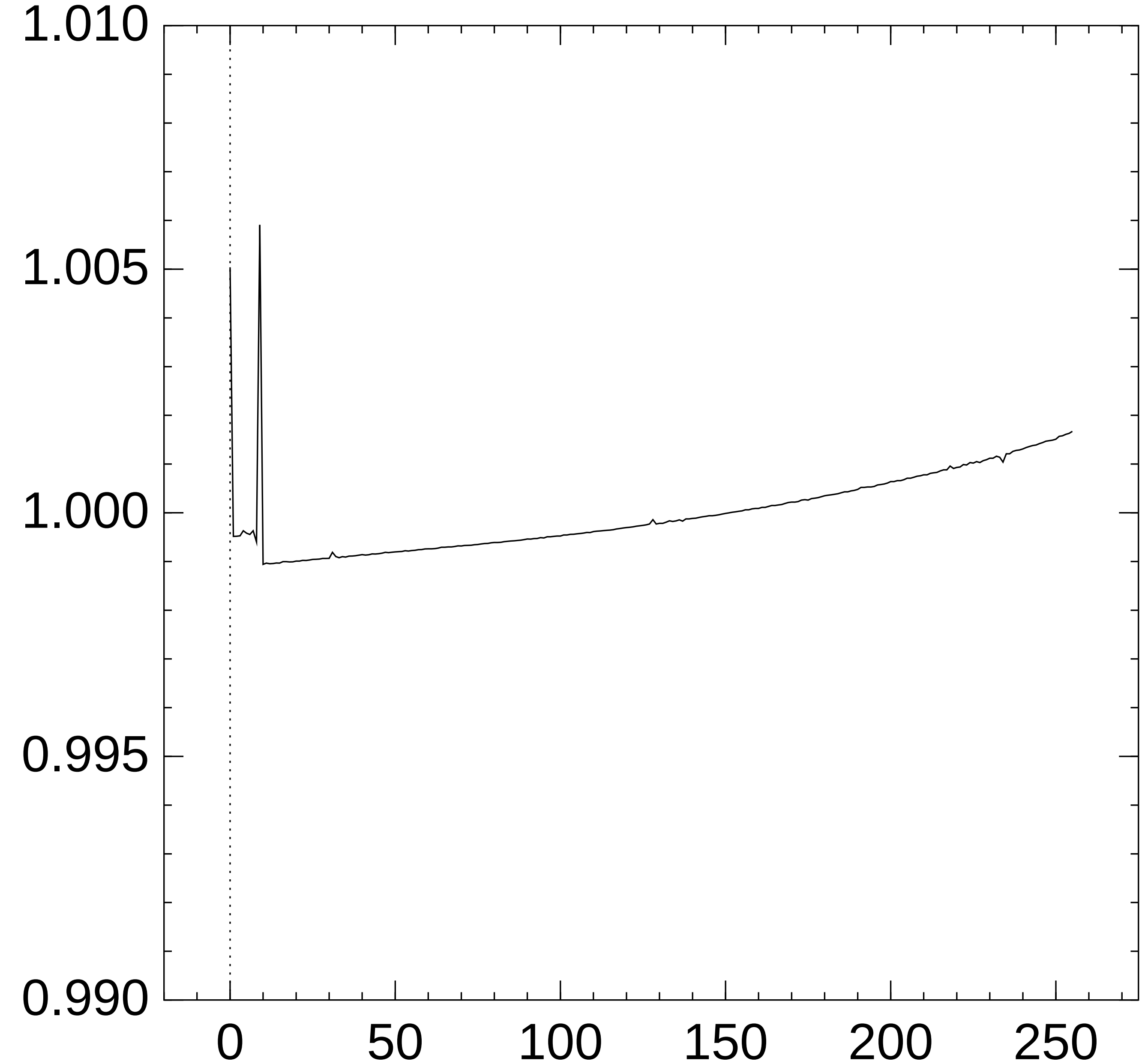
Graph of $256 \Pr[z_8 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

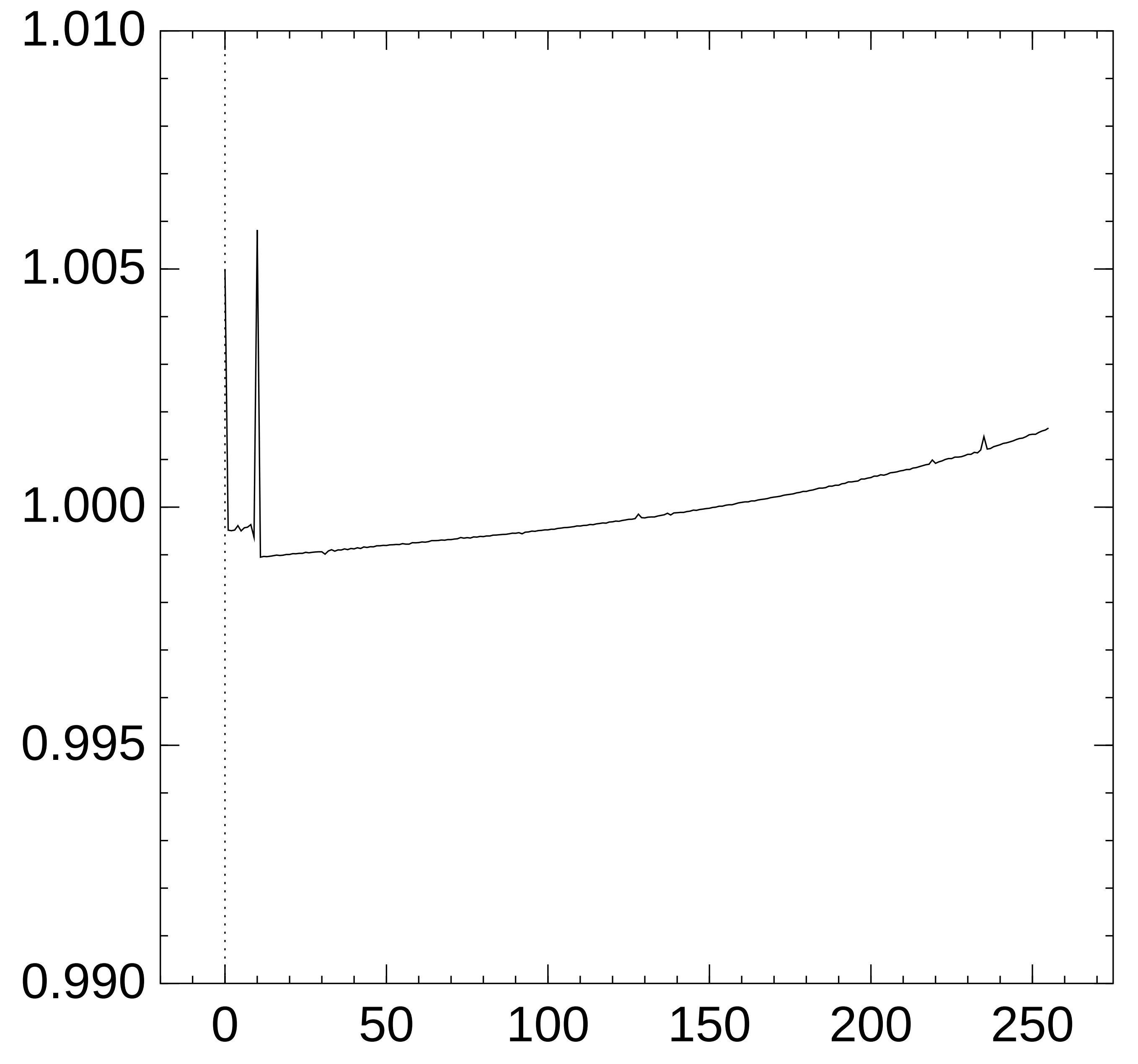
Graph of $256 \Pr[z_9 = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

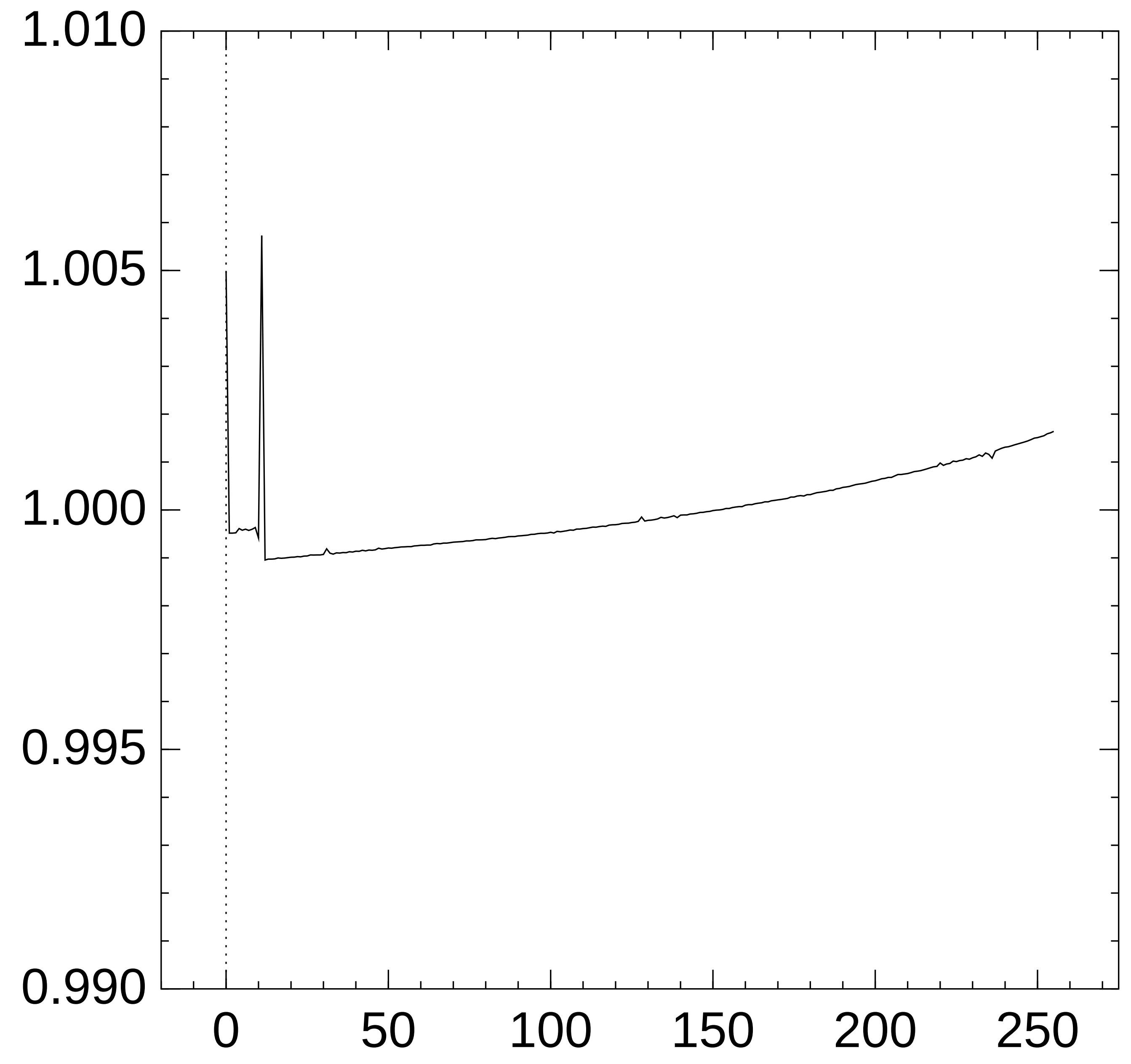
Graph of $256 \Pr[z_{10} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

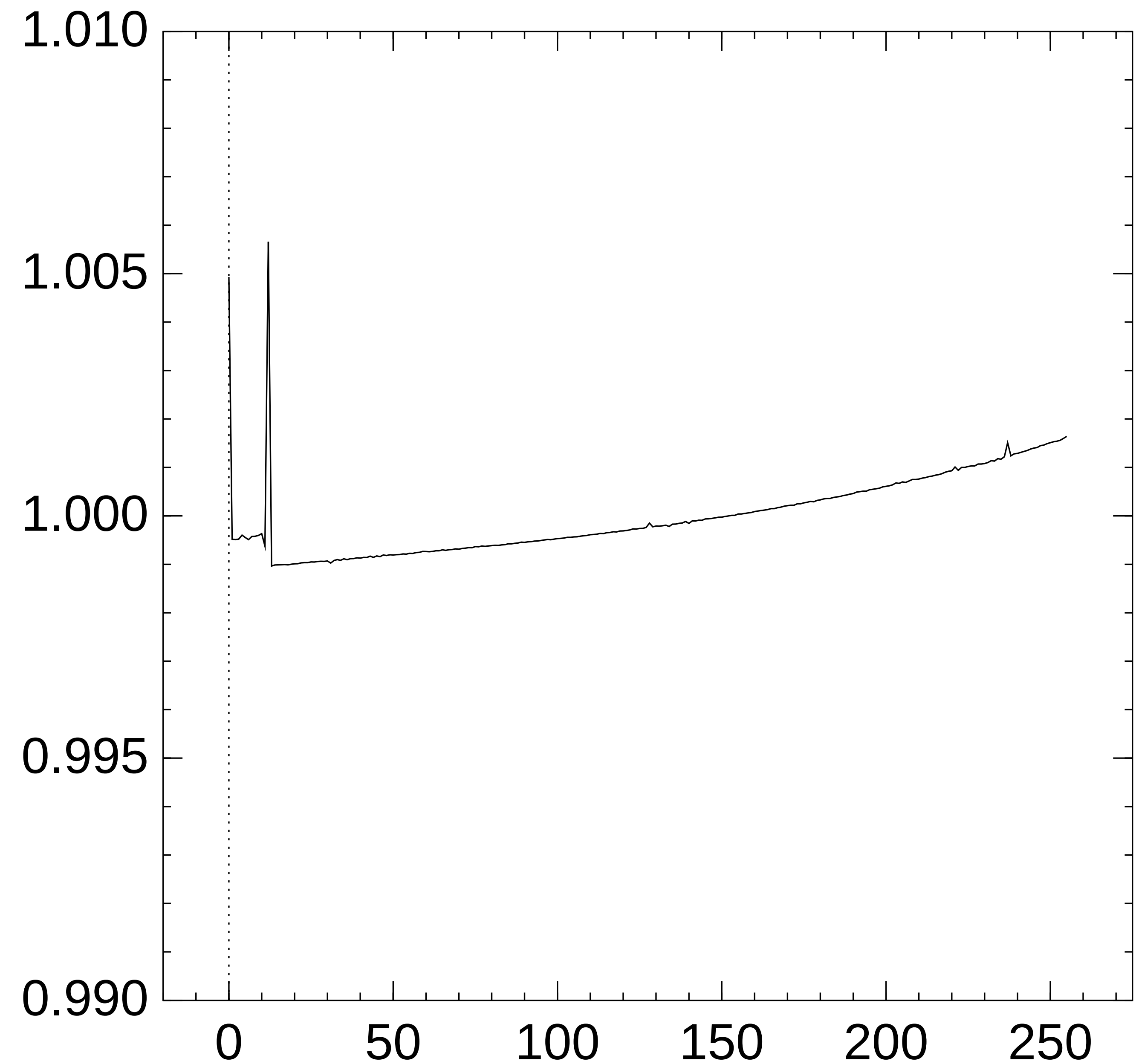
Graph of $256 \Pr[z_{11} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

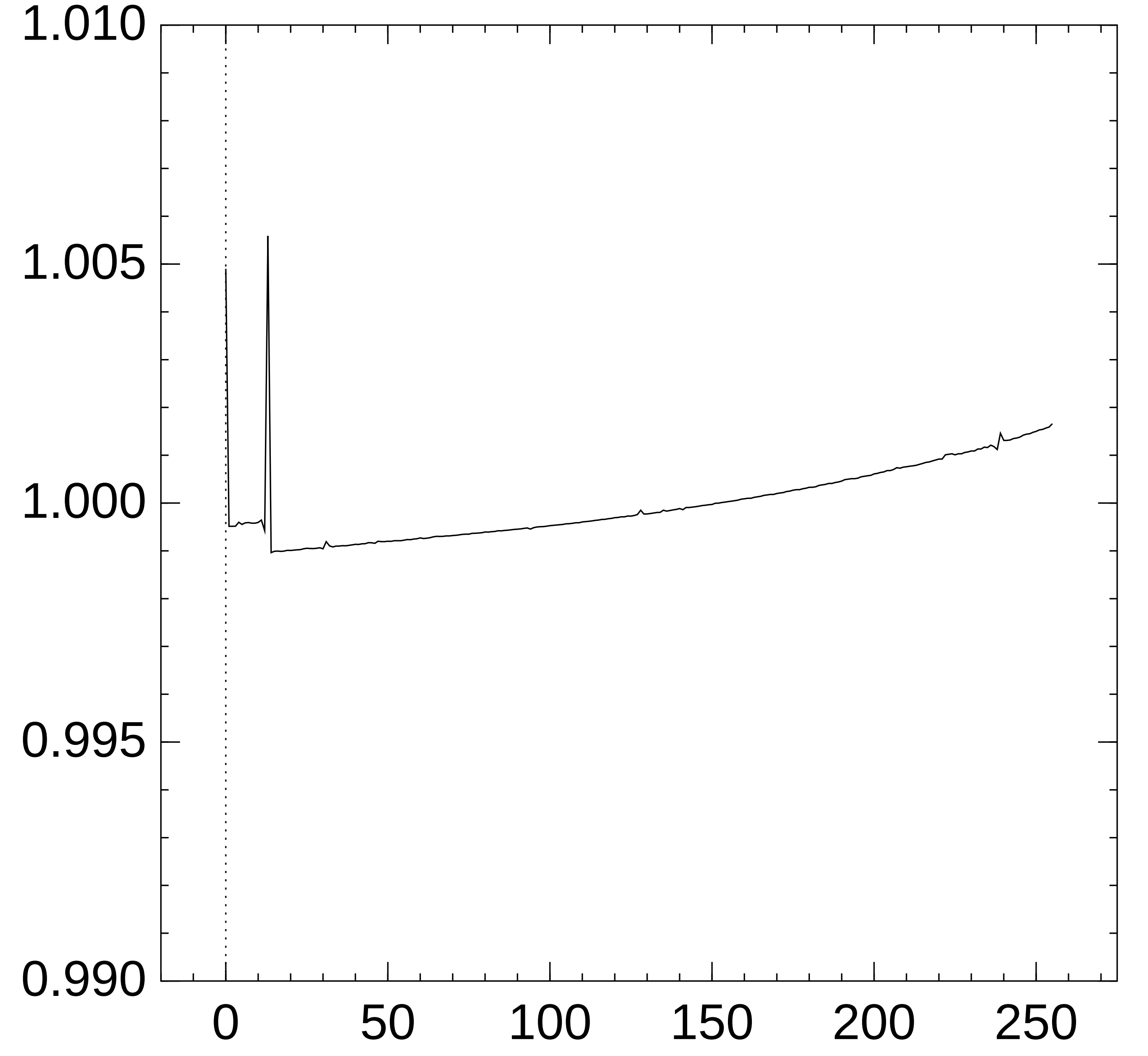
Graph of $256 \Pr[z_{12} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

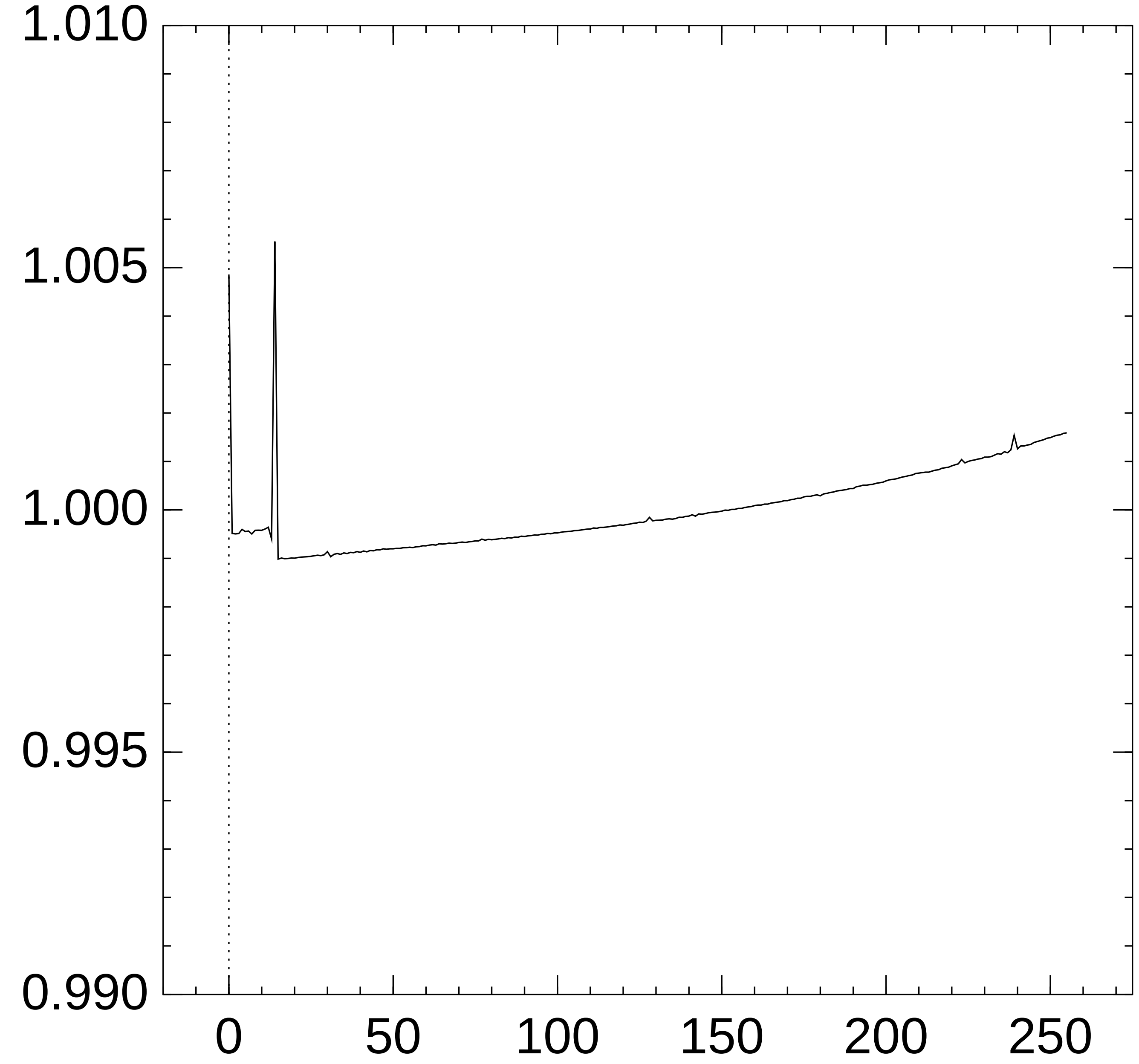
Graph of $256 \Pr[z_{13} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

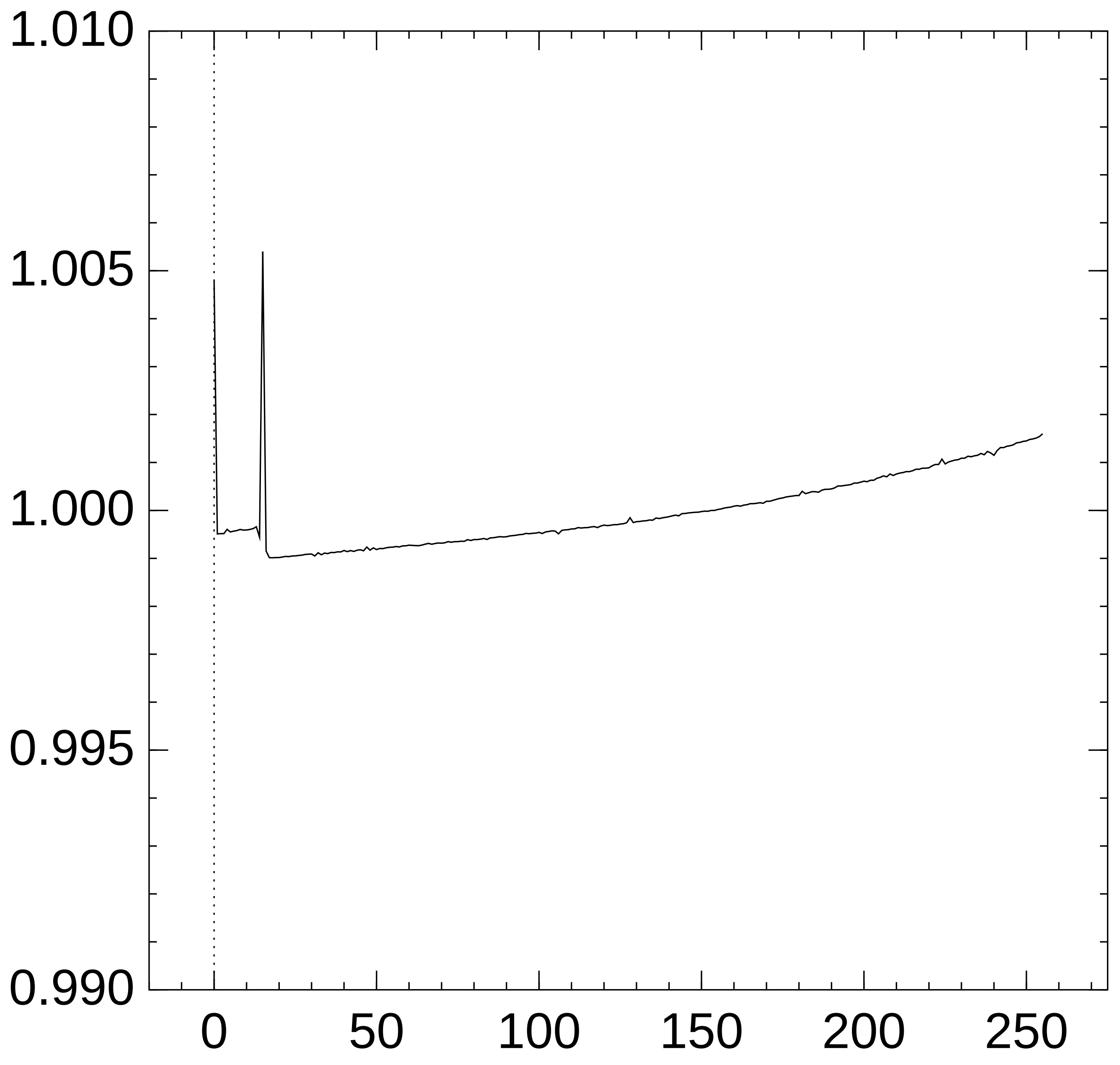
Graph of $256 \Pr[z_{14} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

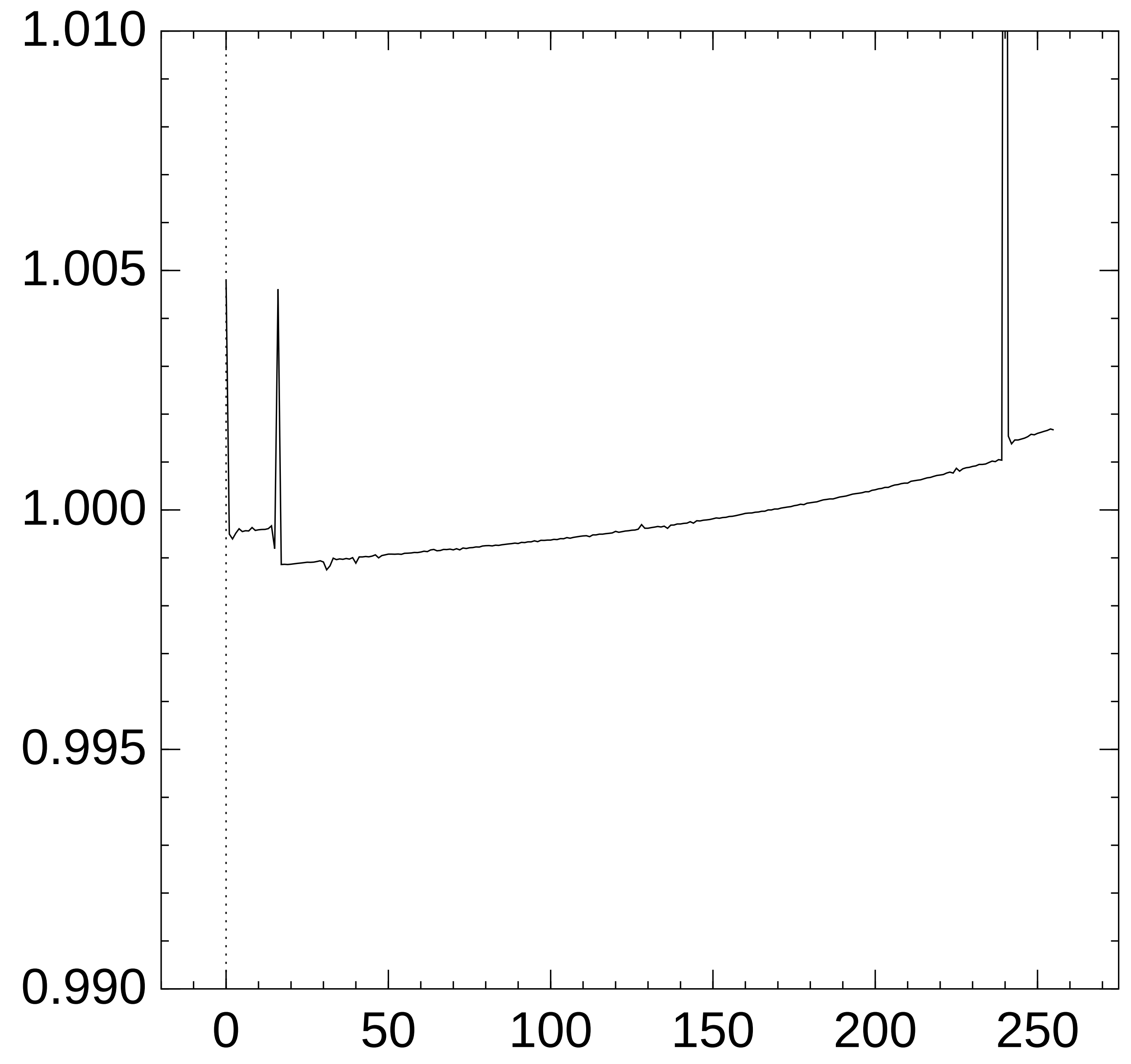
Graph of $256 \Pr[z_{15} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

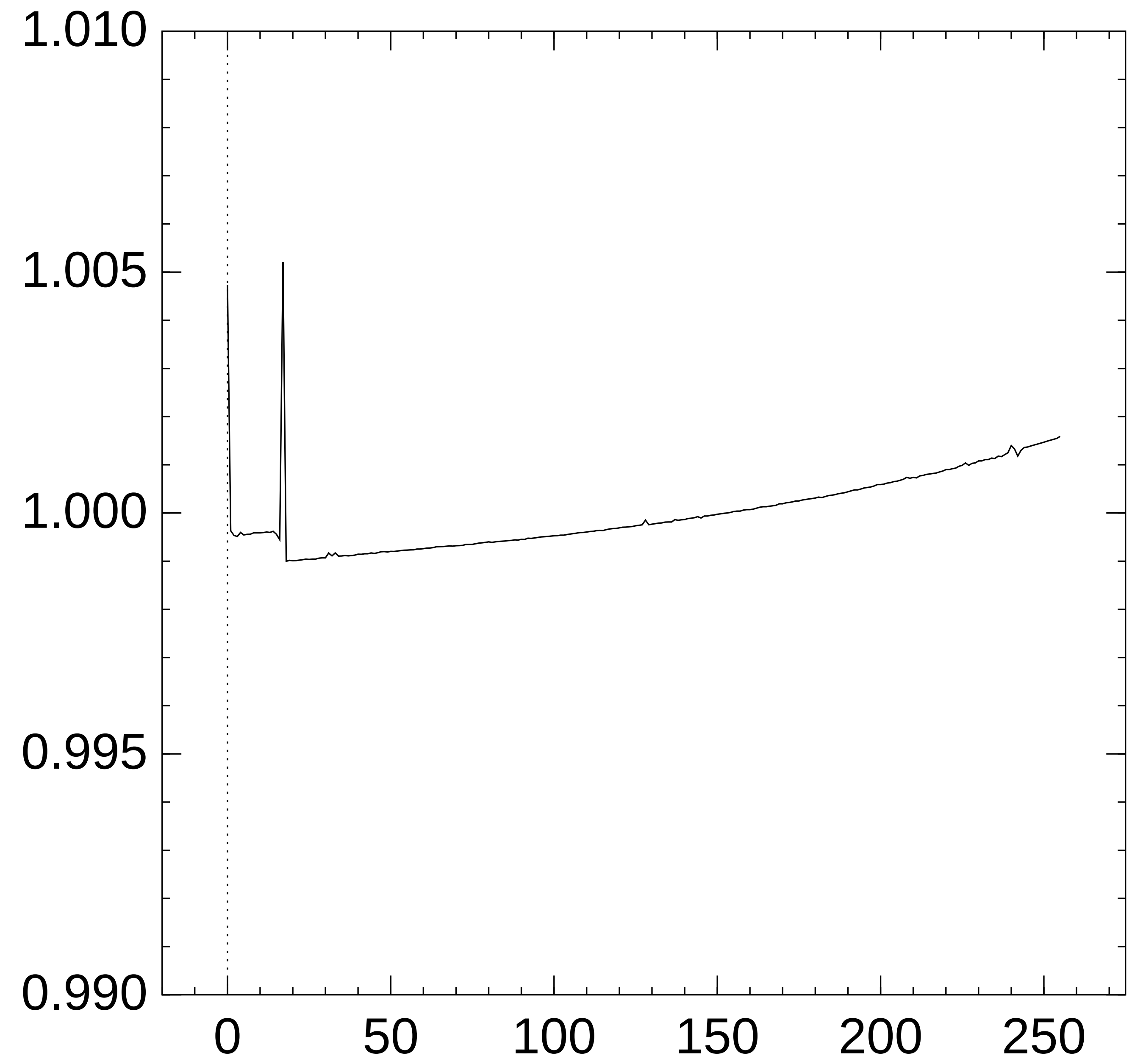
Graph of $256 \Pr[z_{16} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

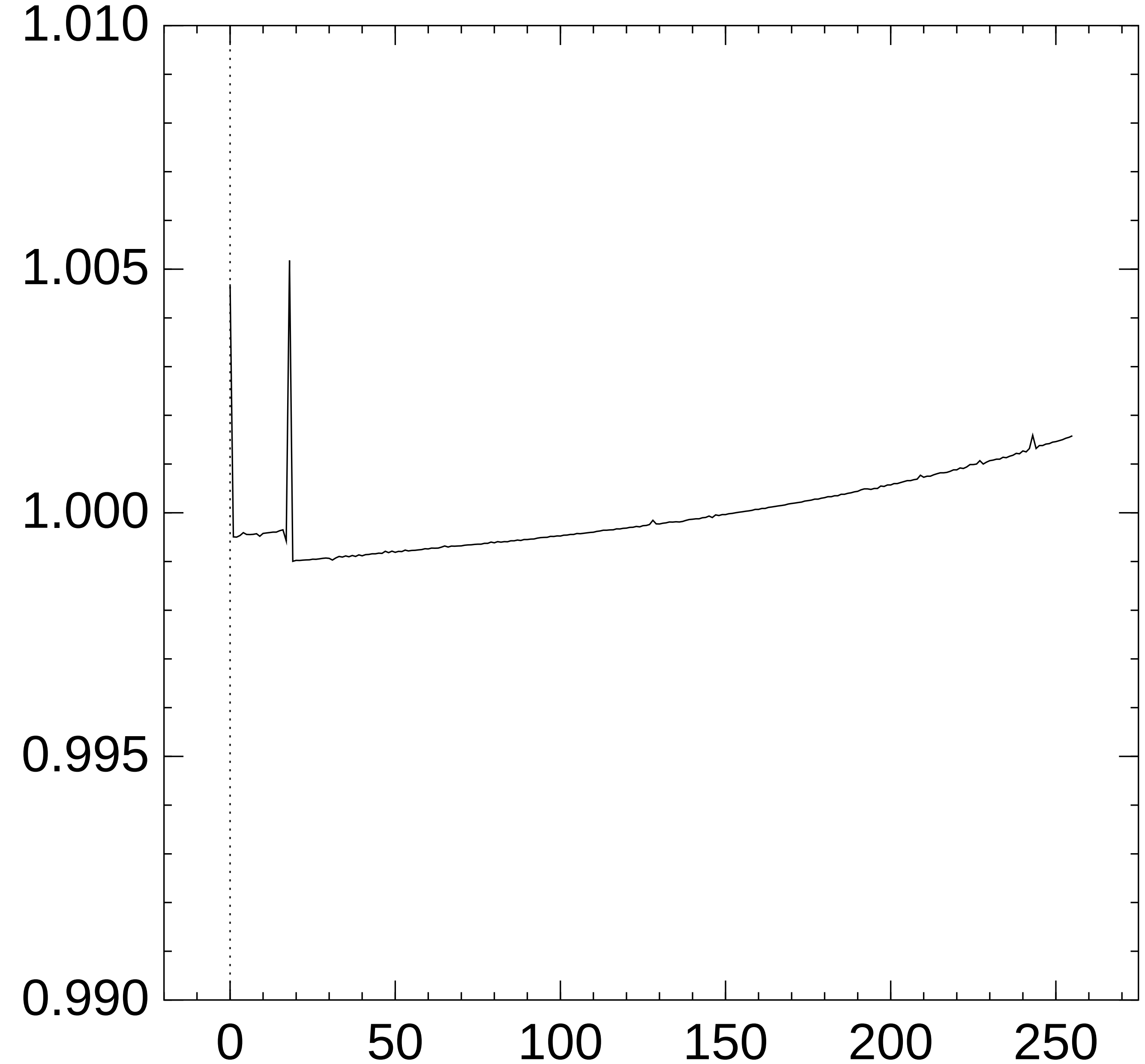
Graph of $256 \Pr[z_{17} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

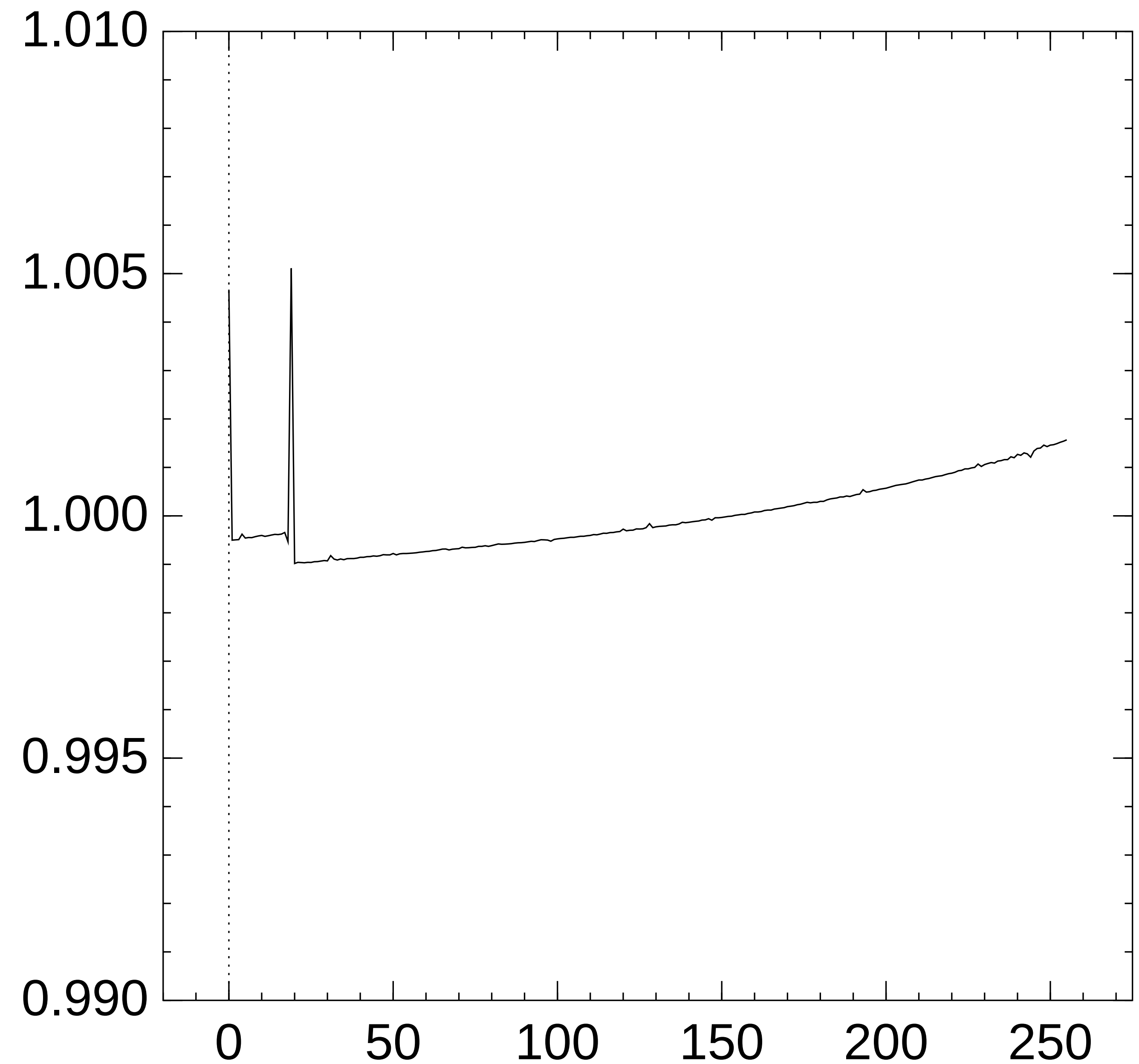
Graph of $256 \Pr[z_{18} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

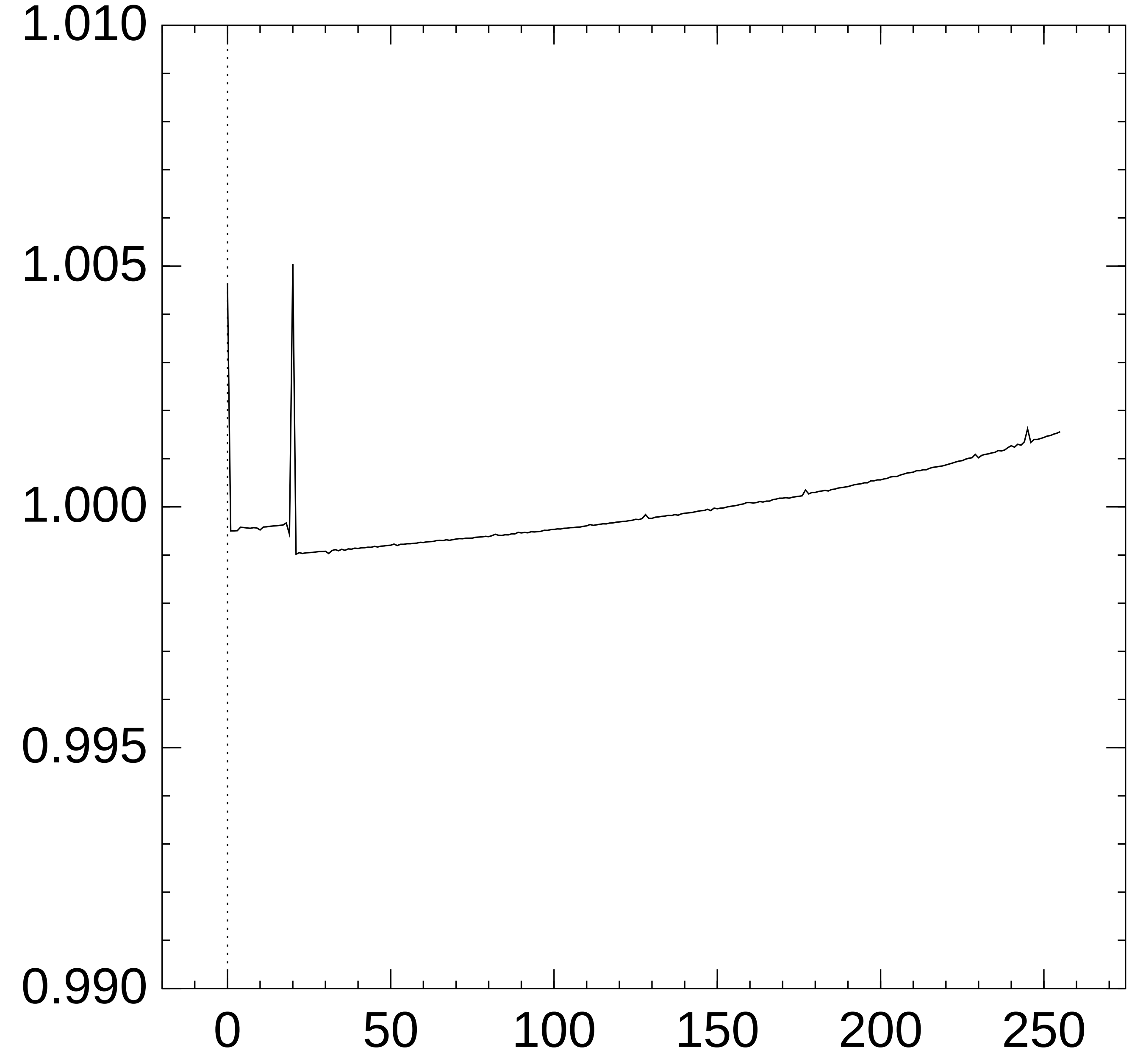
Graph of $256 \Pr[z_{19} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

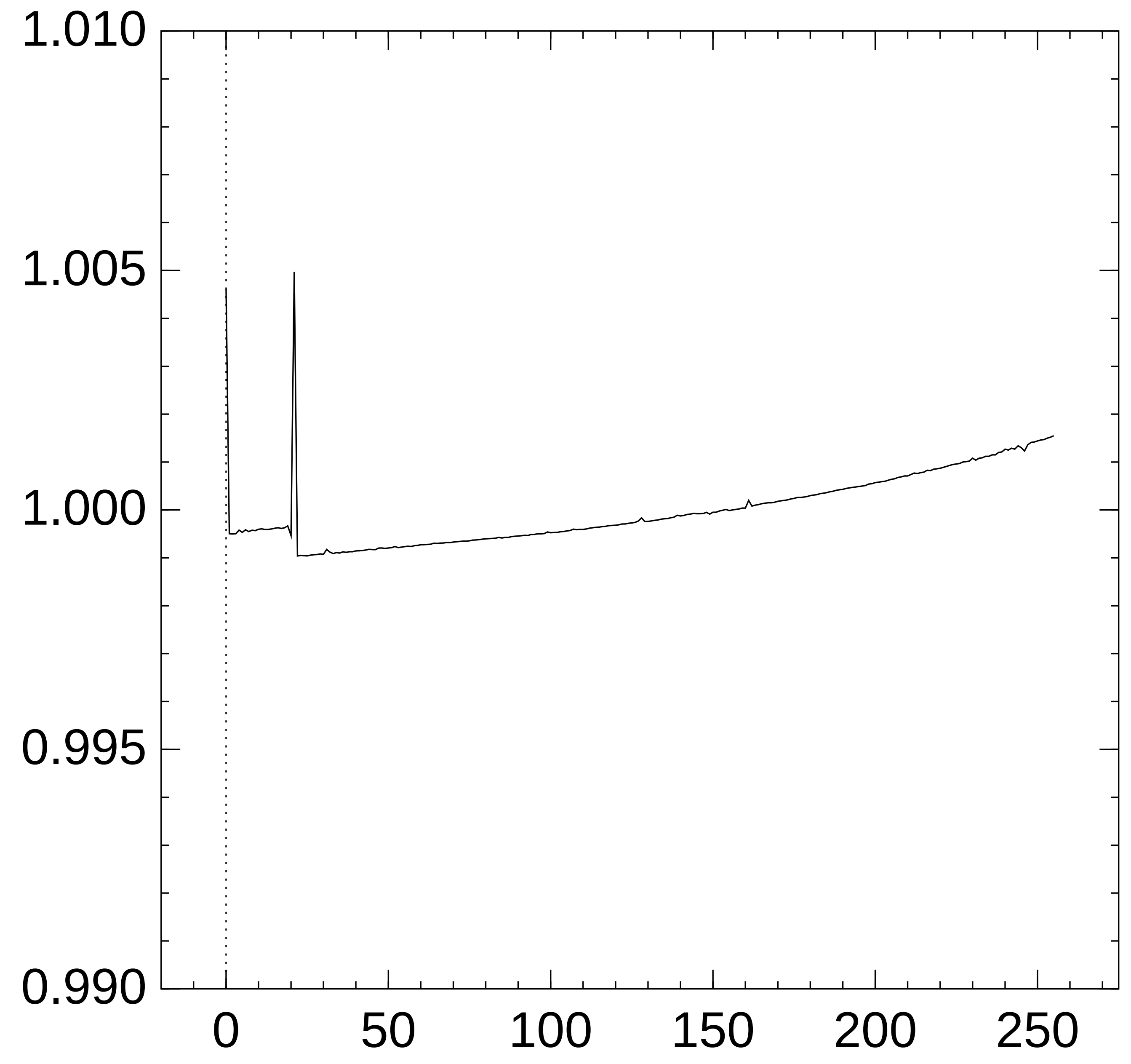
Graph of $256 \Pr[z_{20} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

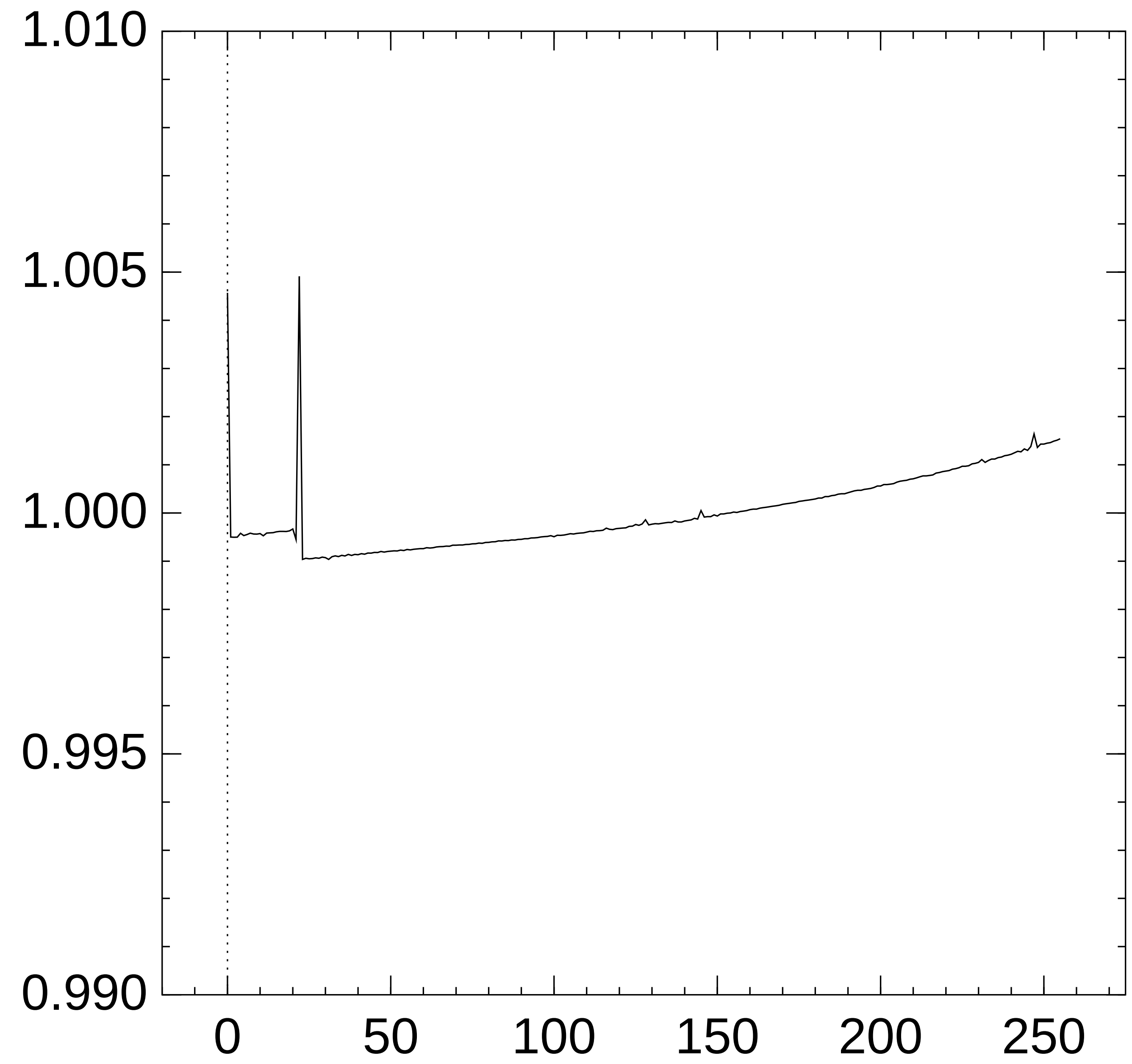
Graph of $256 \Pr[z_{21} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

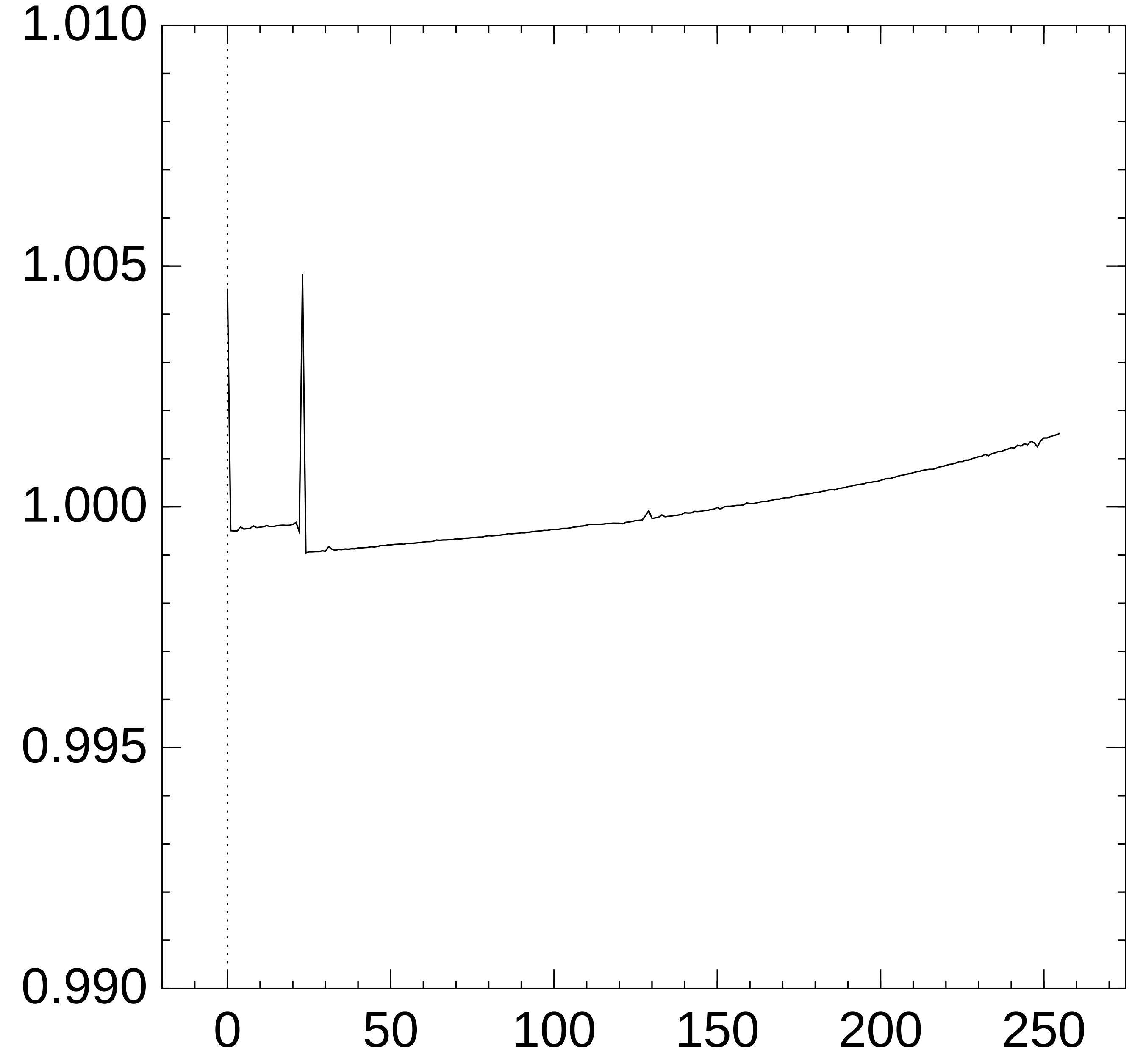
Graph of $256 \Pr[z_{22} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

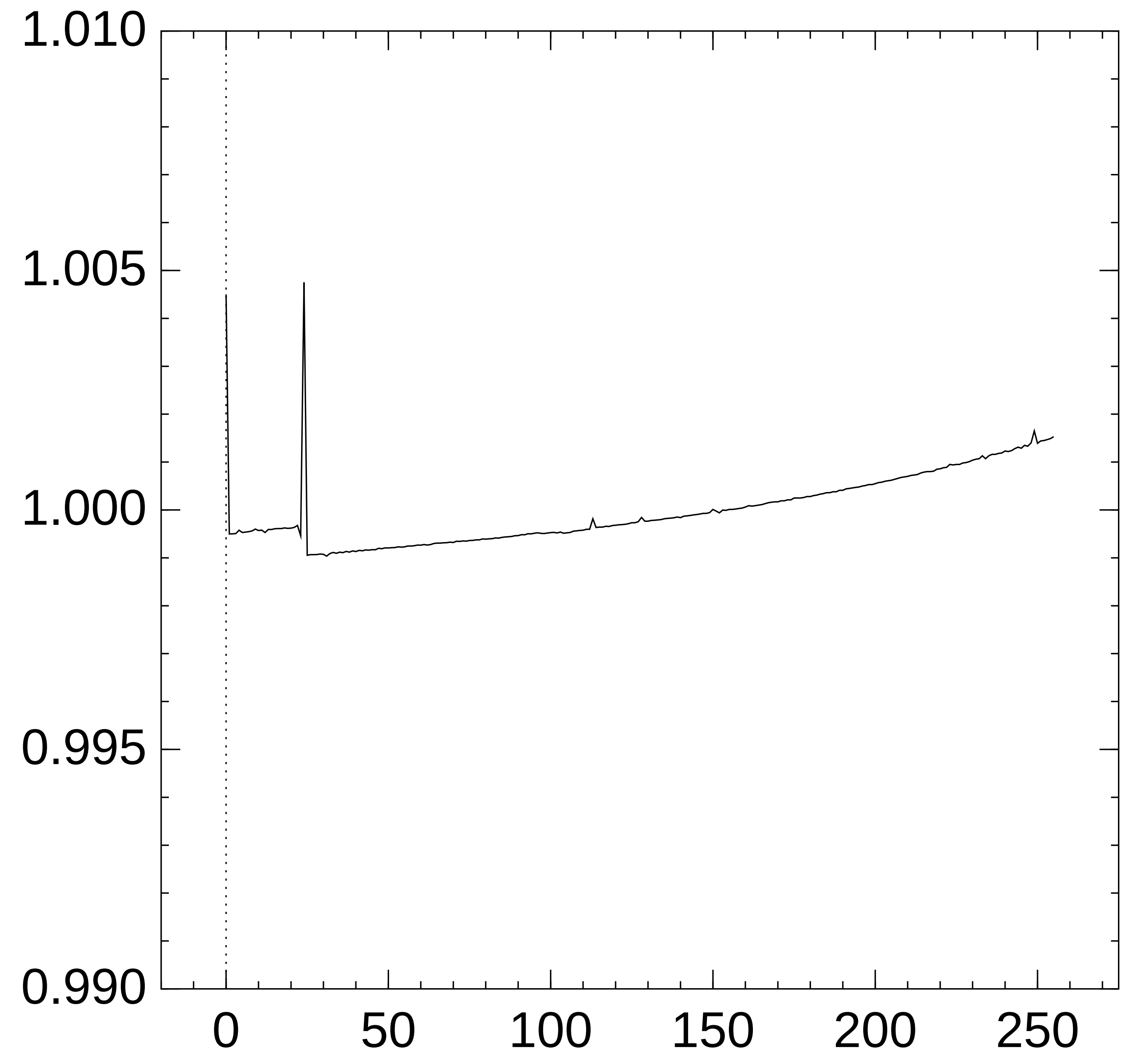
Graph of $256 \Pr[z_{23} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

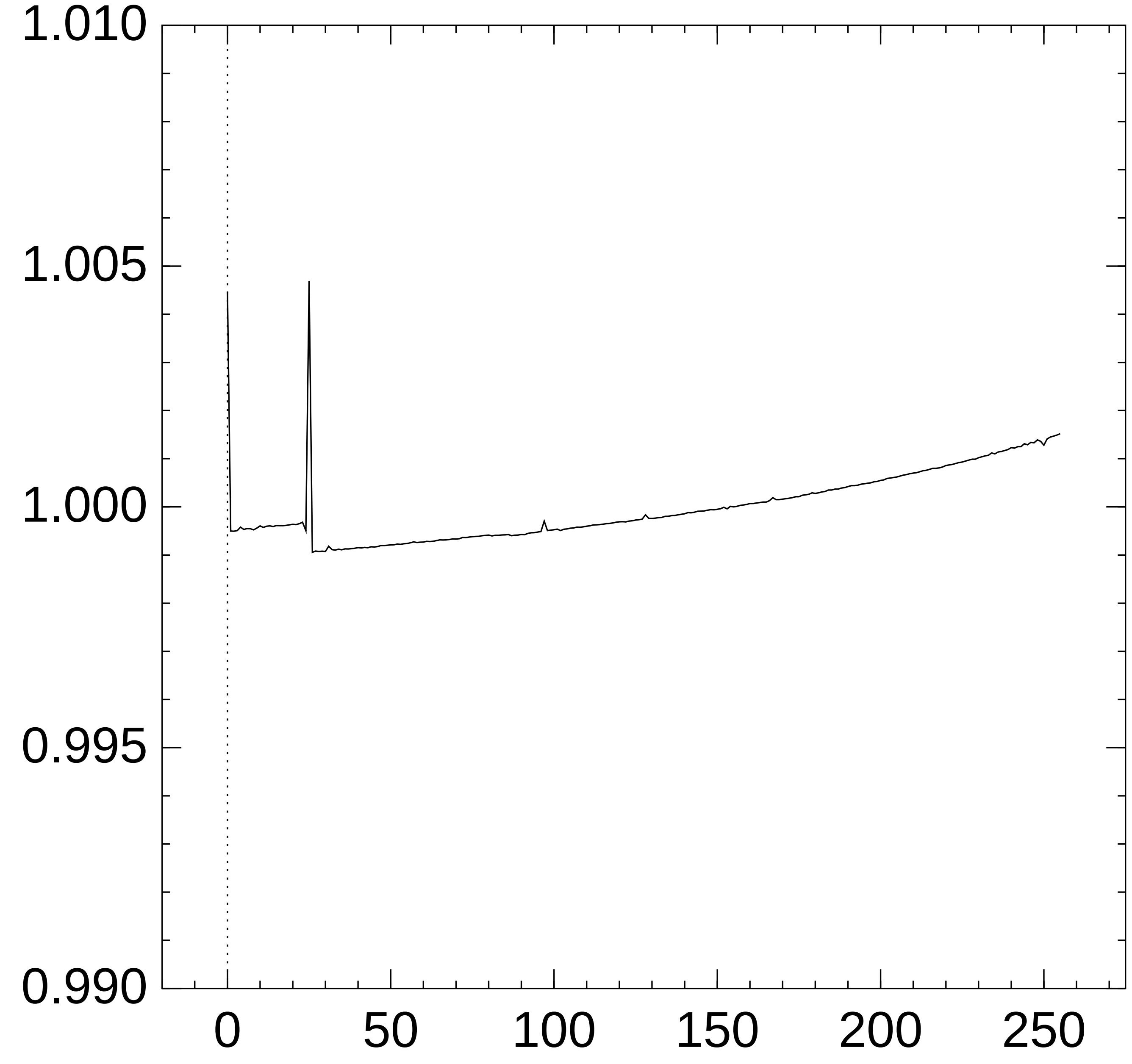
Graph of $256 \Pr[z_{24} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

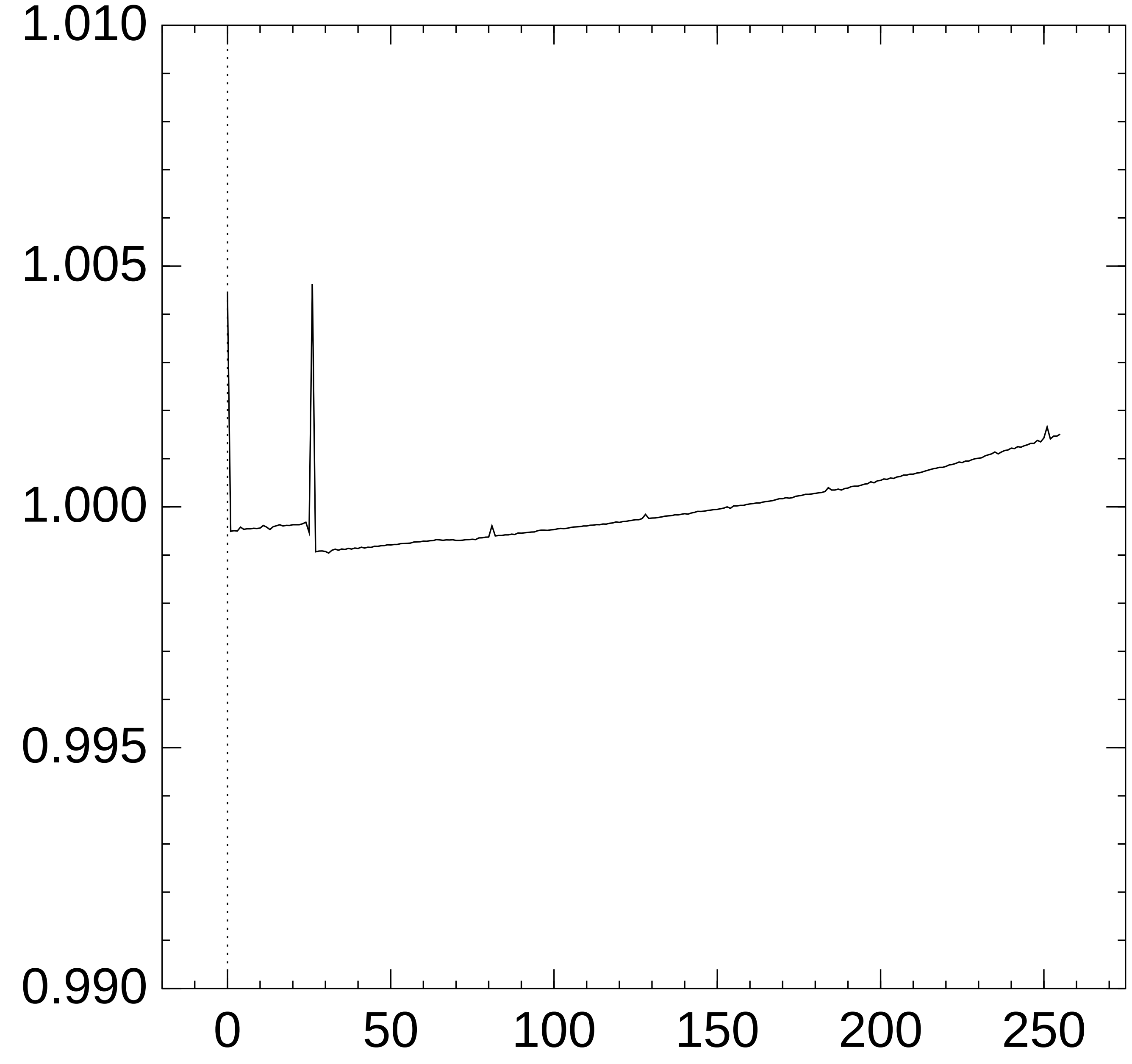
Graph of $256 \Pr[z_{25} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

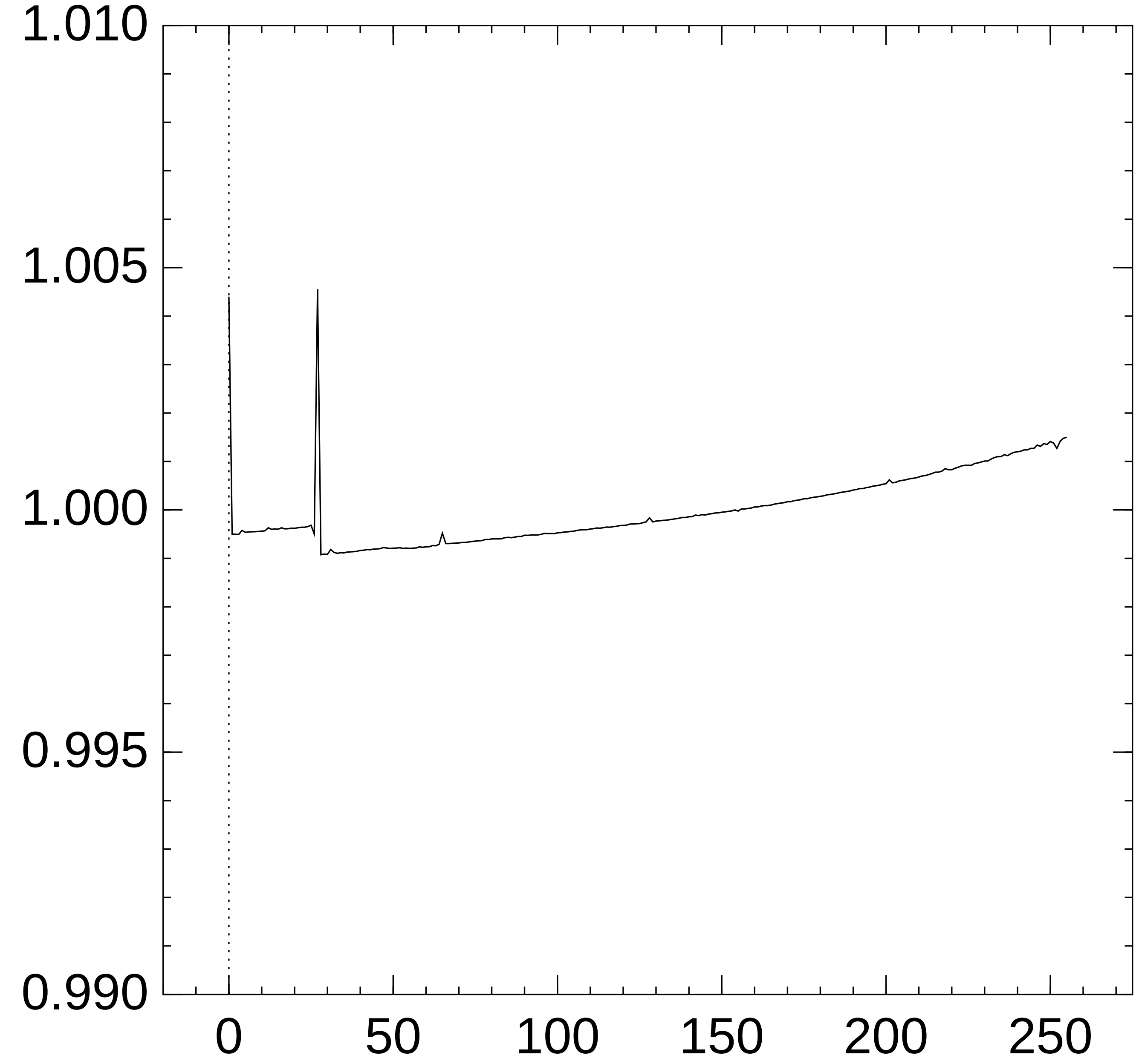
Graph of $256 \Pr[z_{26} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

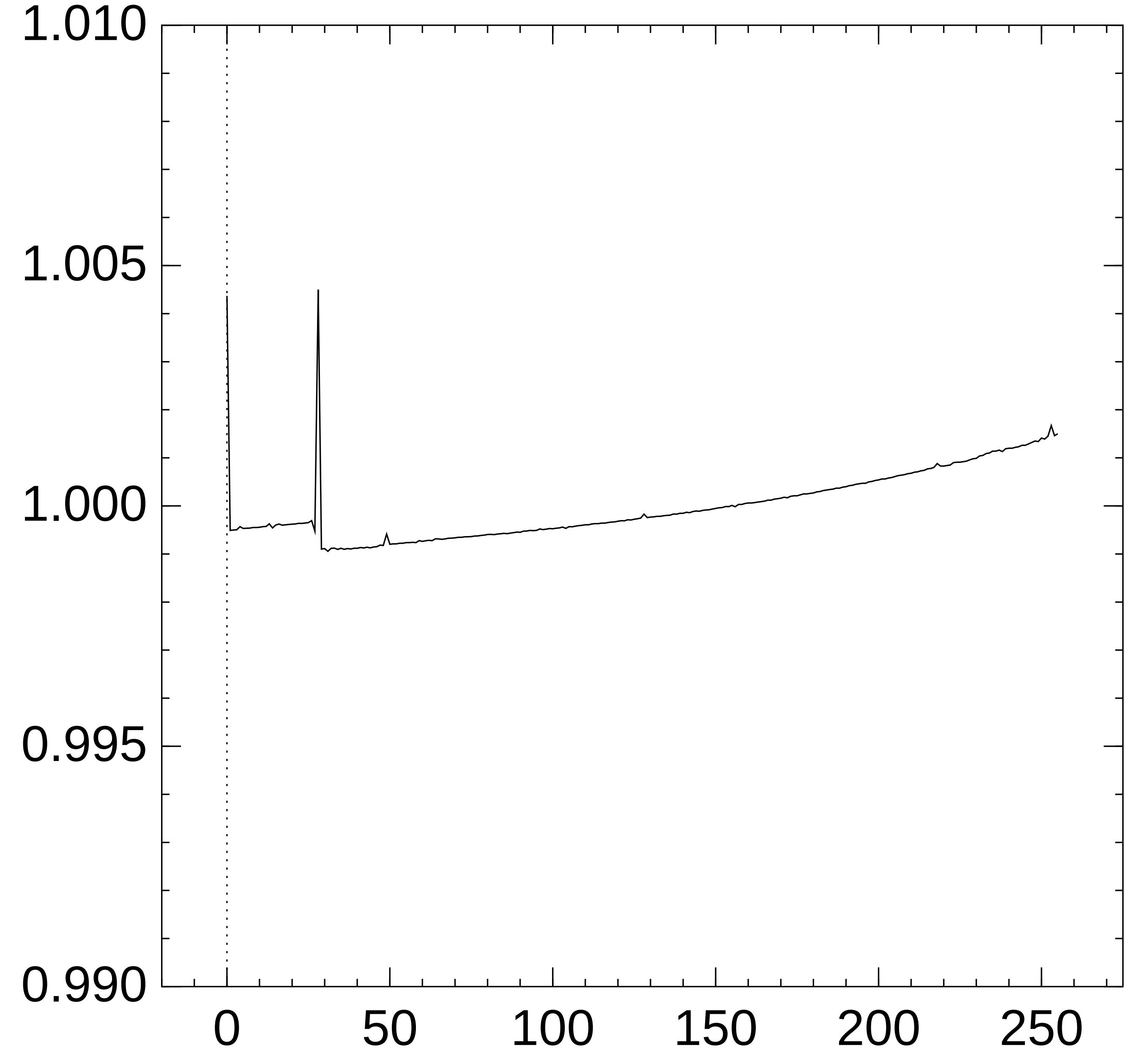
Graph of $256 \Pr[z_{27} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

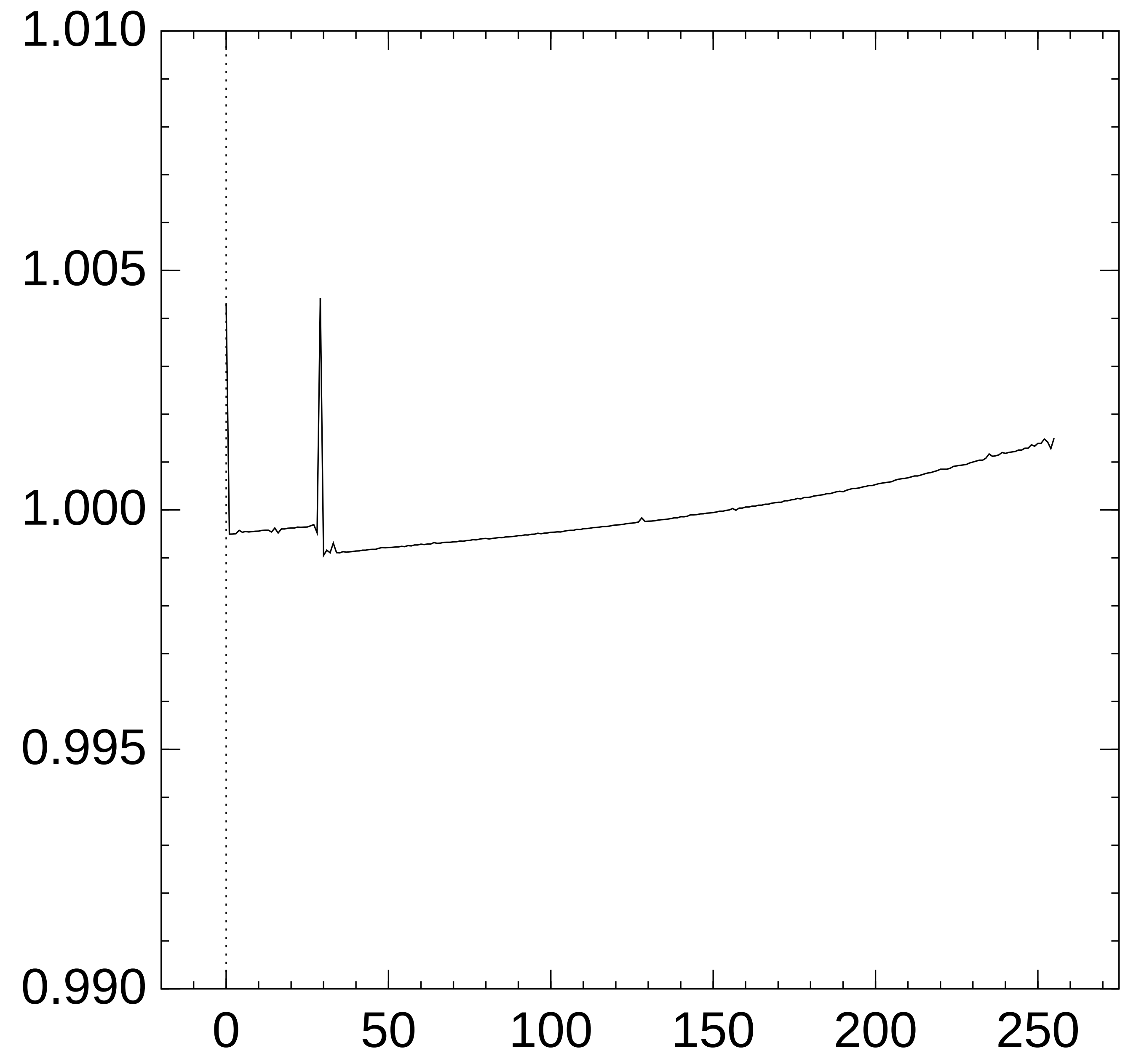
Graph of $256 \Pr[z_{28} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

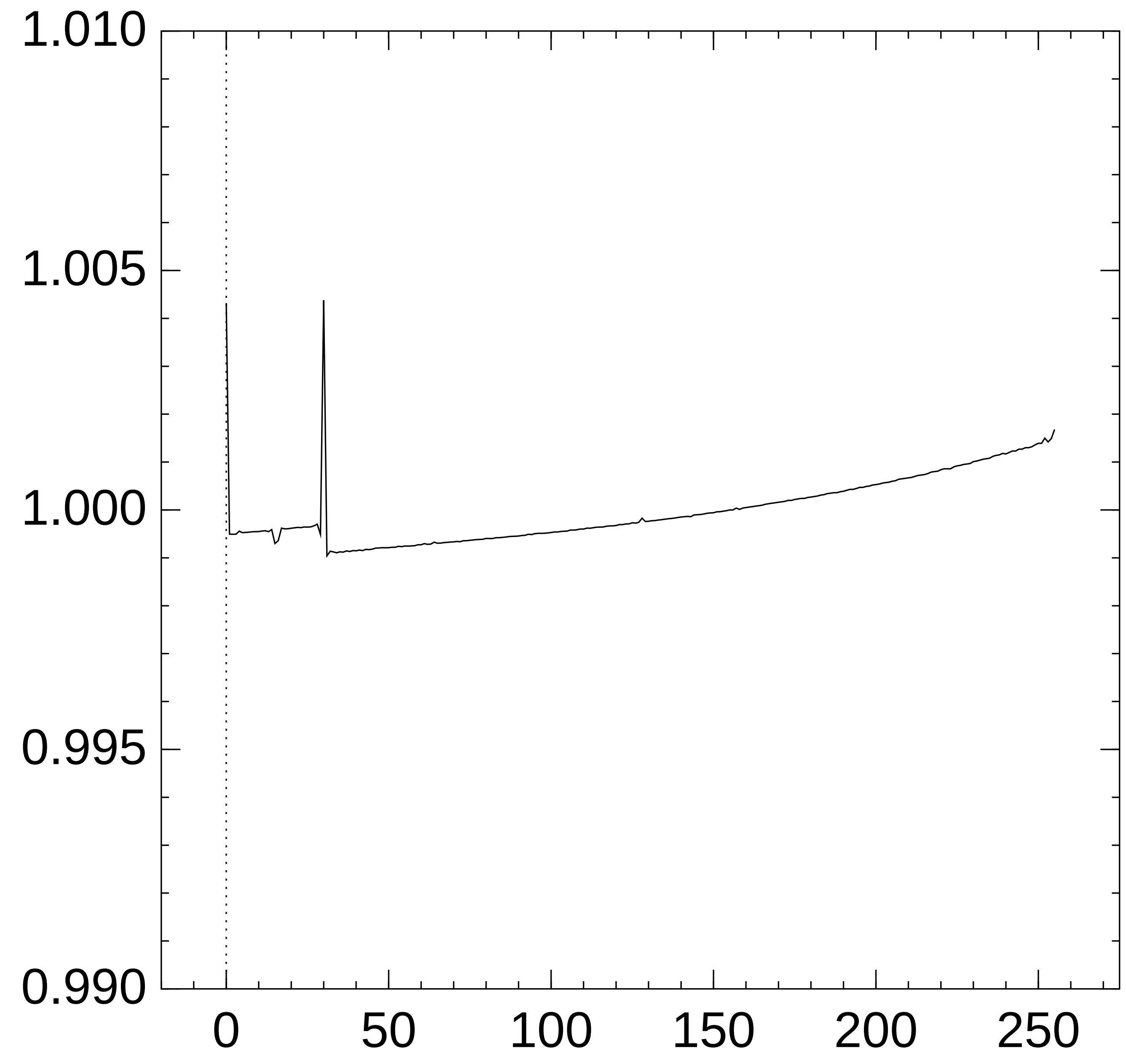
Graph of $256 \Pr[z_{29} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

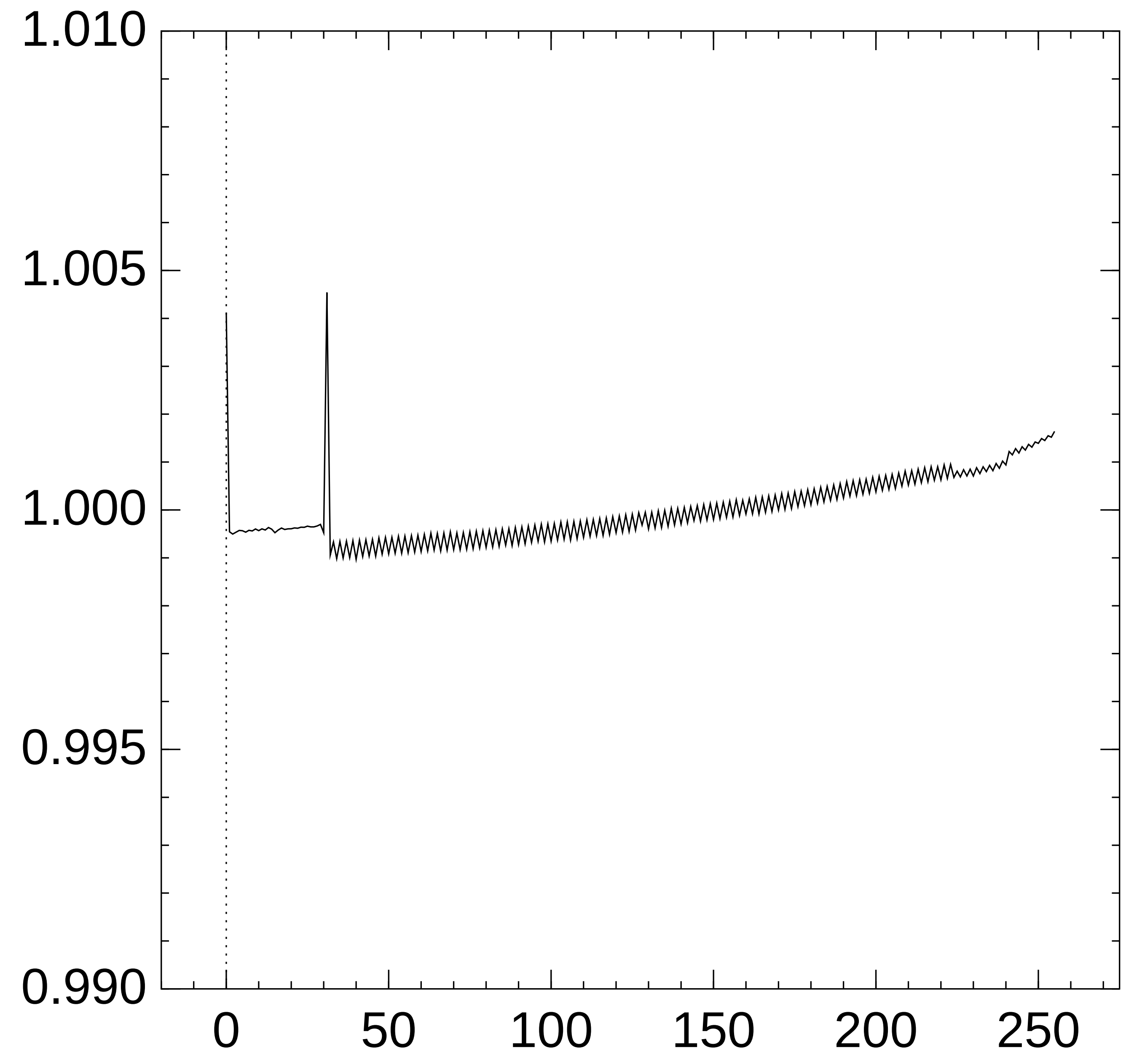
Graph of $256 \Pr[z_{30} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

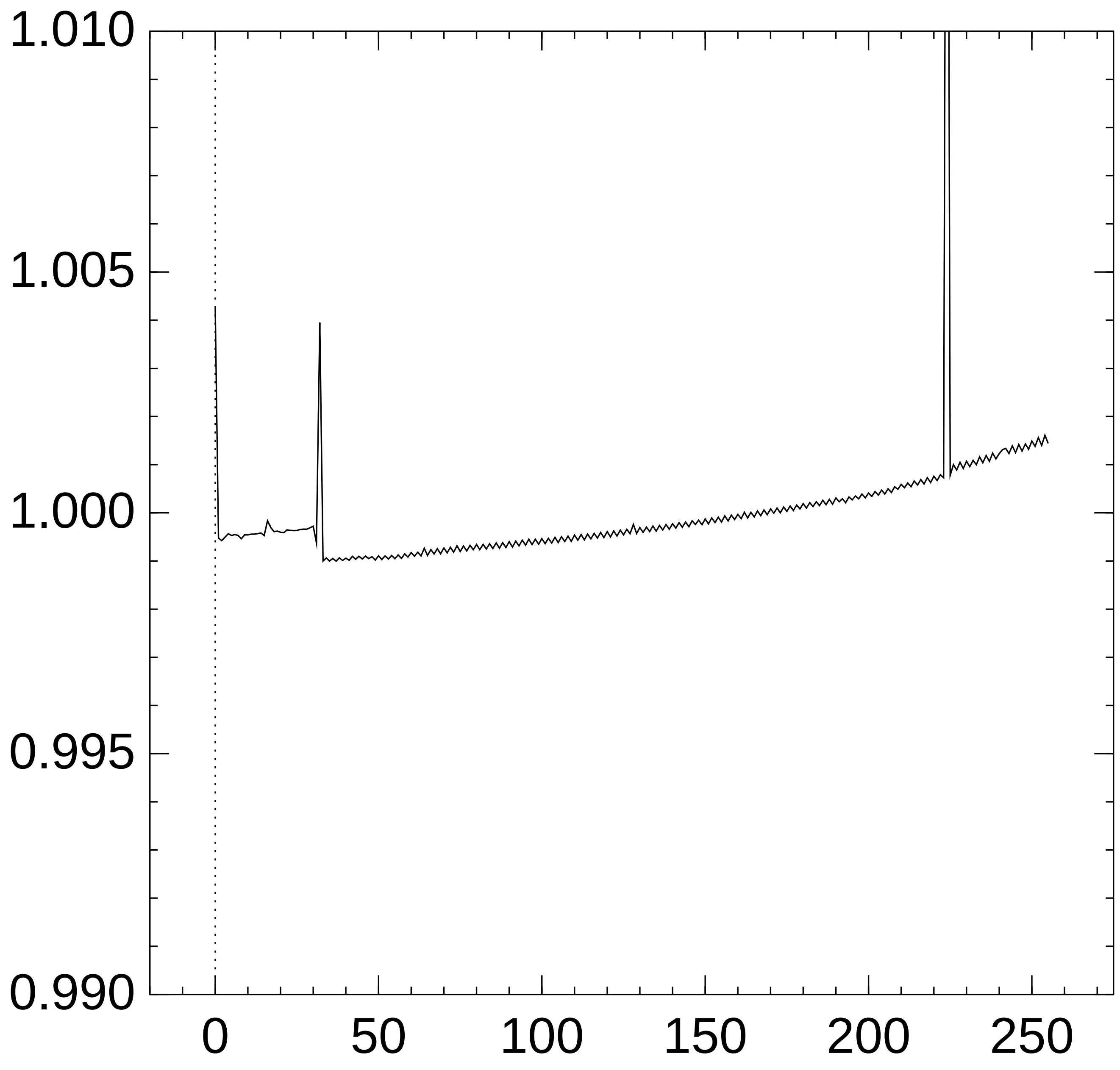
Graph of $256 \Pr[z_{31} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

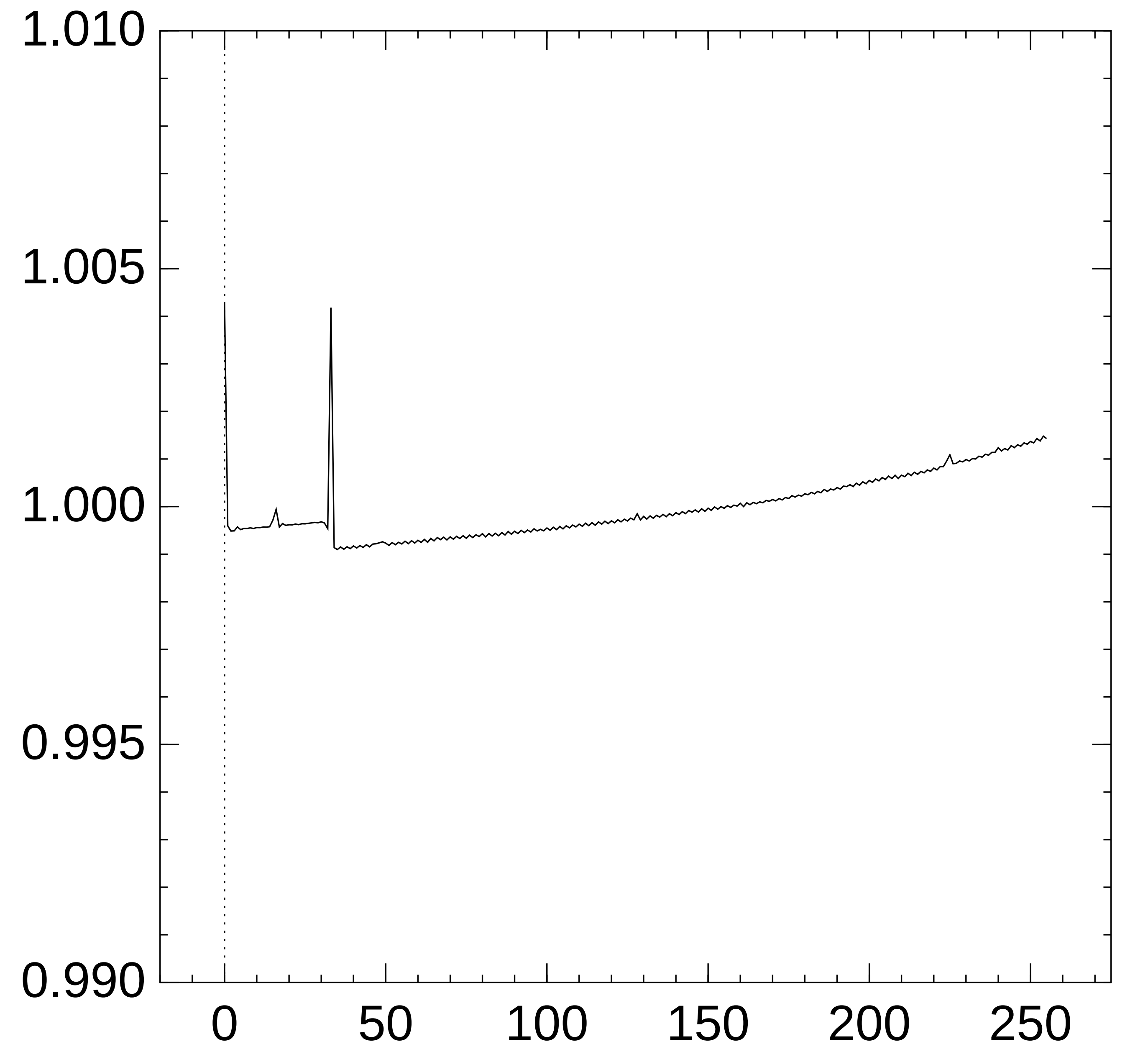
Graph of $256 \Pr[z_{32} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

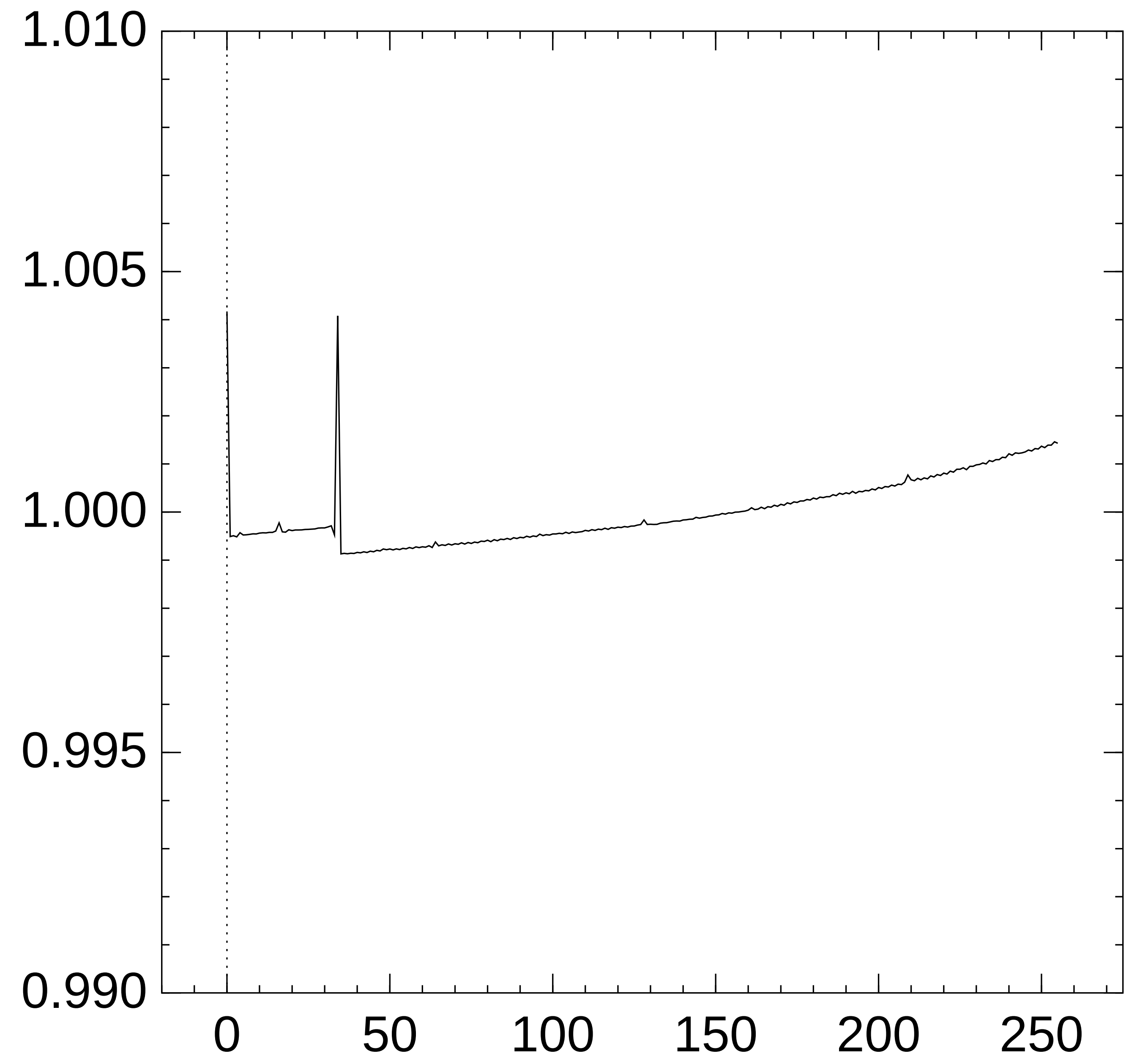
Graph of $256 \Pr[z_{33} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

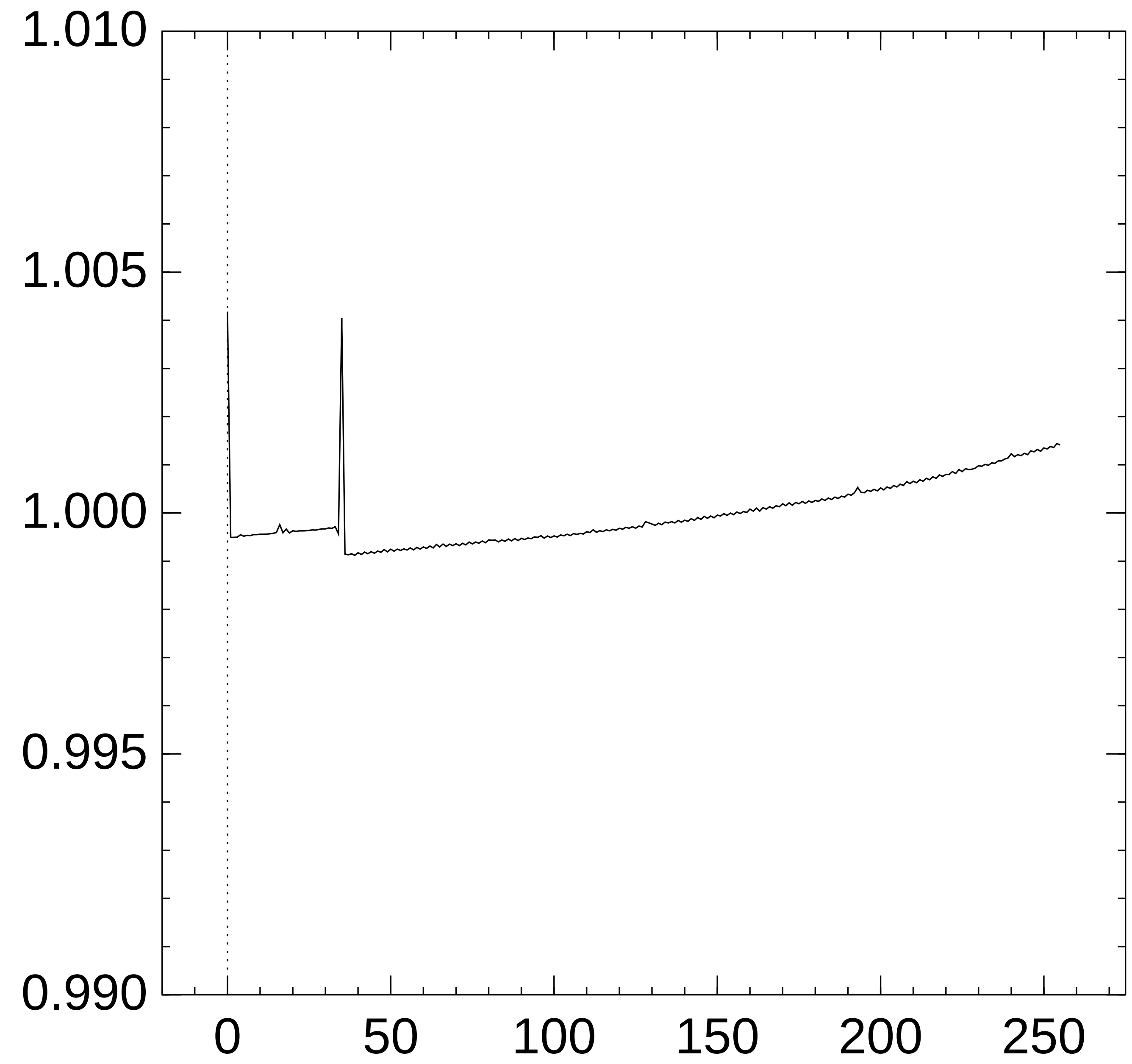
Graph of $256 \Pr[z_{34} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

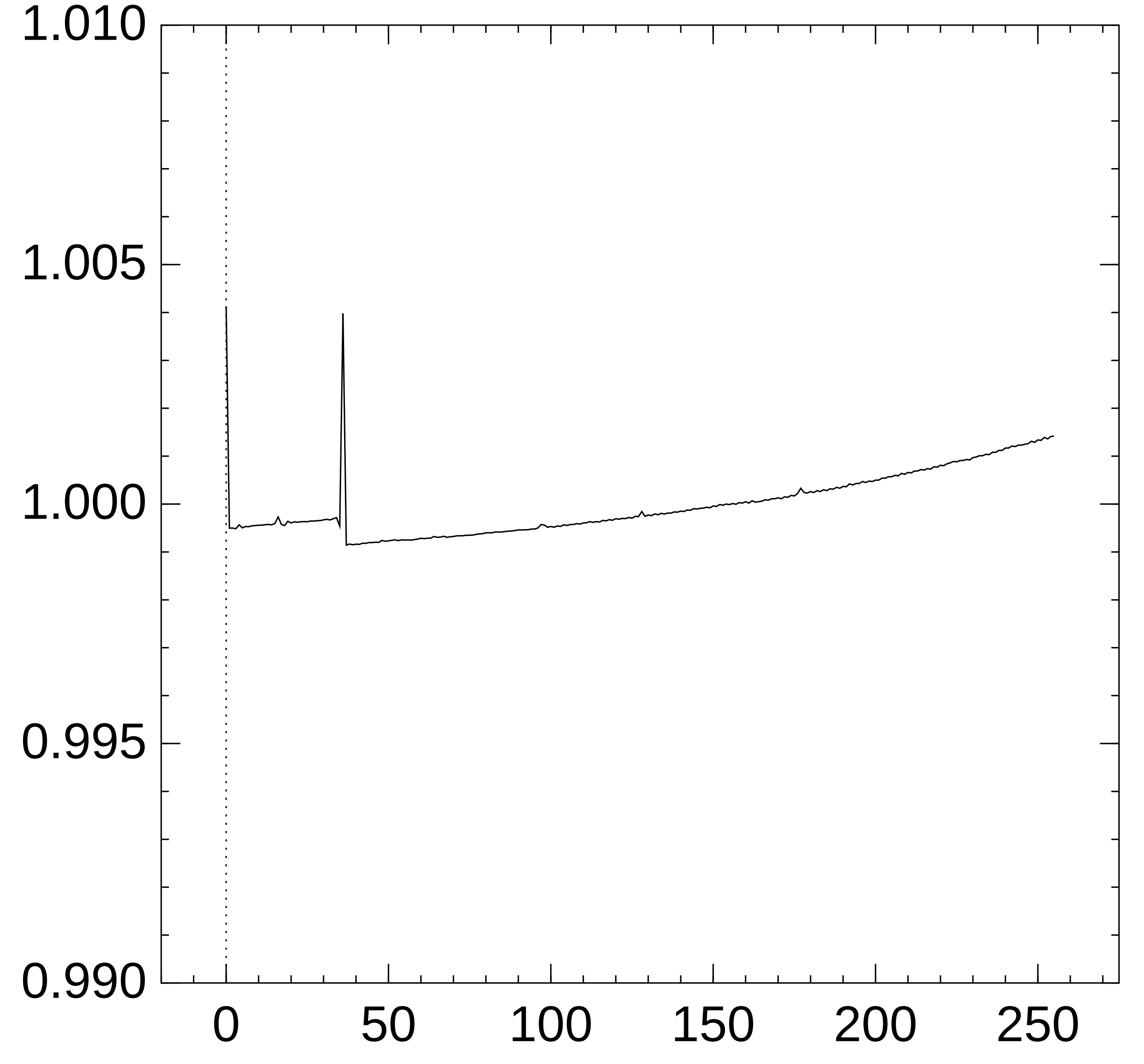
Graph of $256 \Pr[z_{35} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

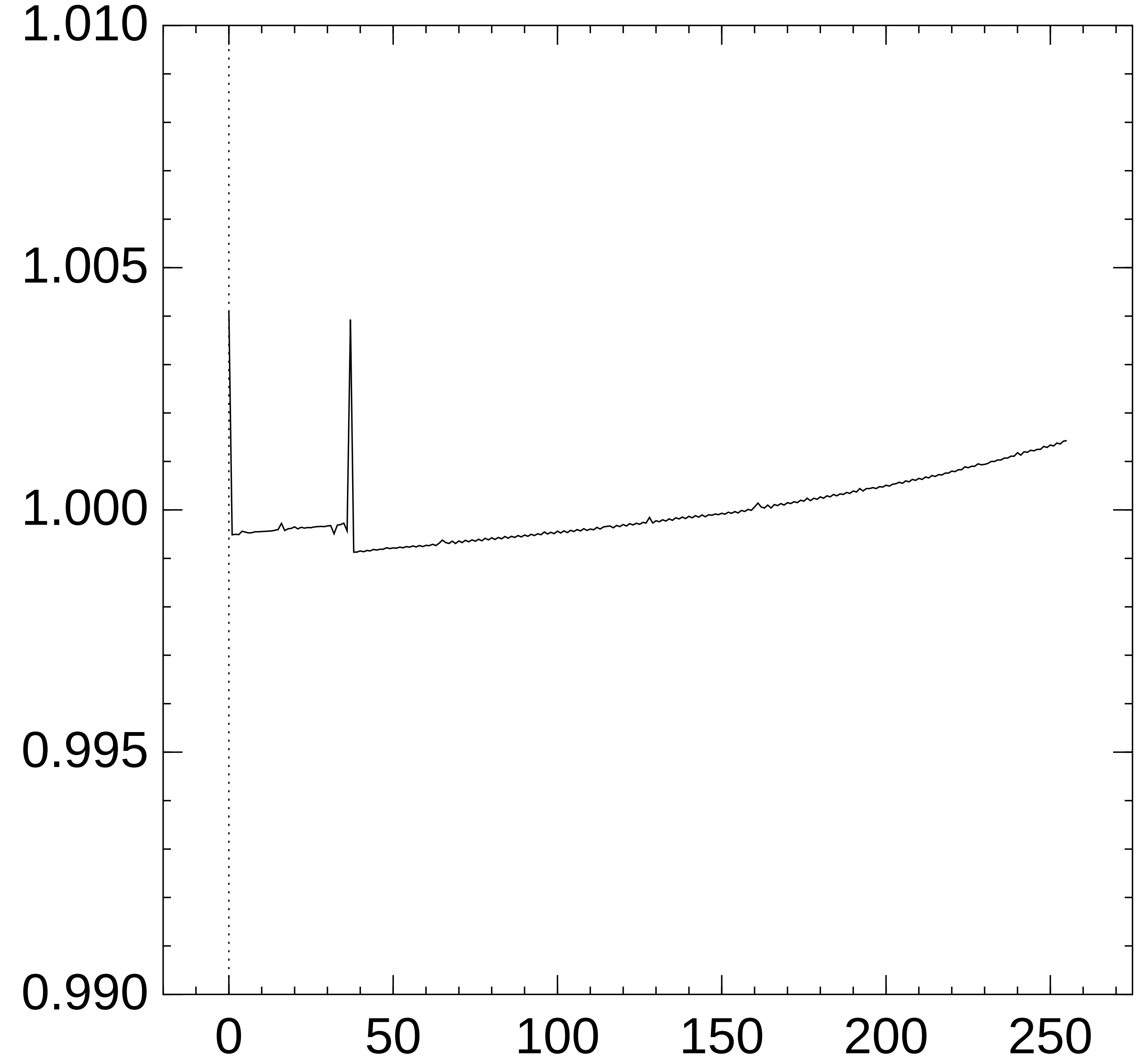
Graph of $256 \Pr[z_{36} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

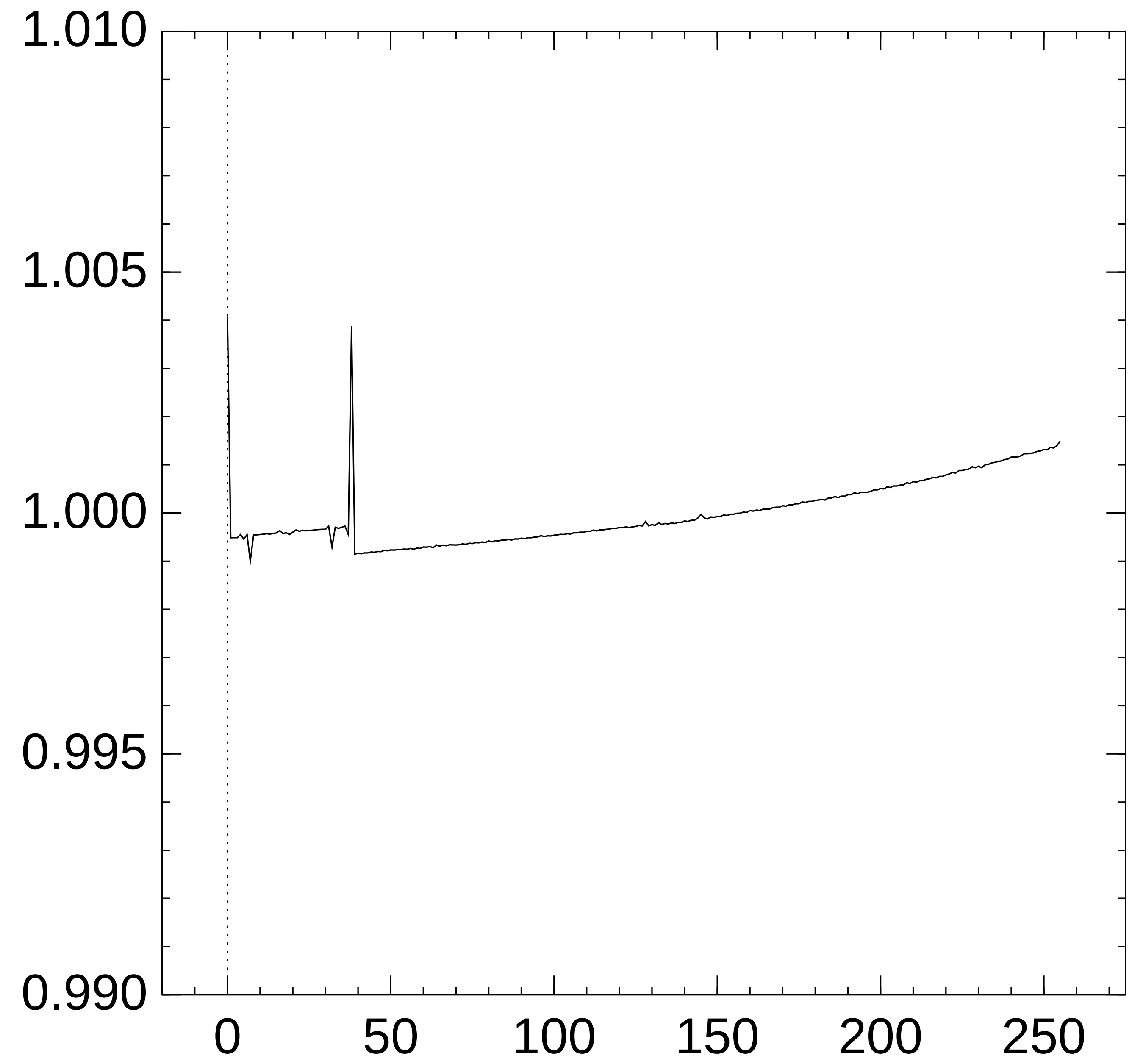
Graph of $256 \Pr[z_{37} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

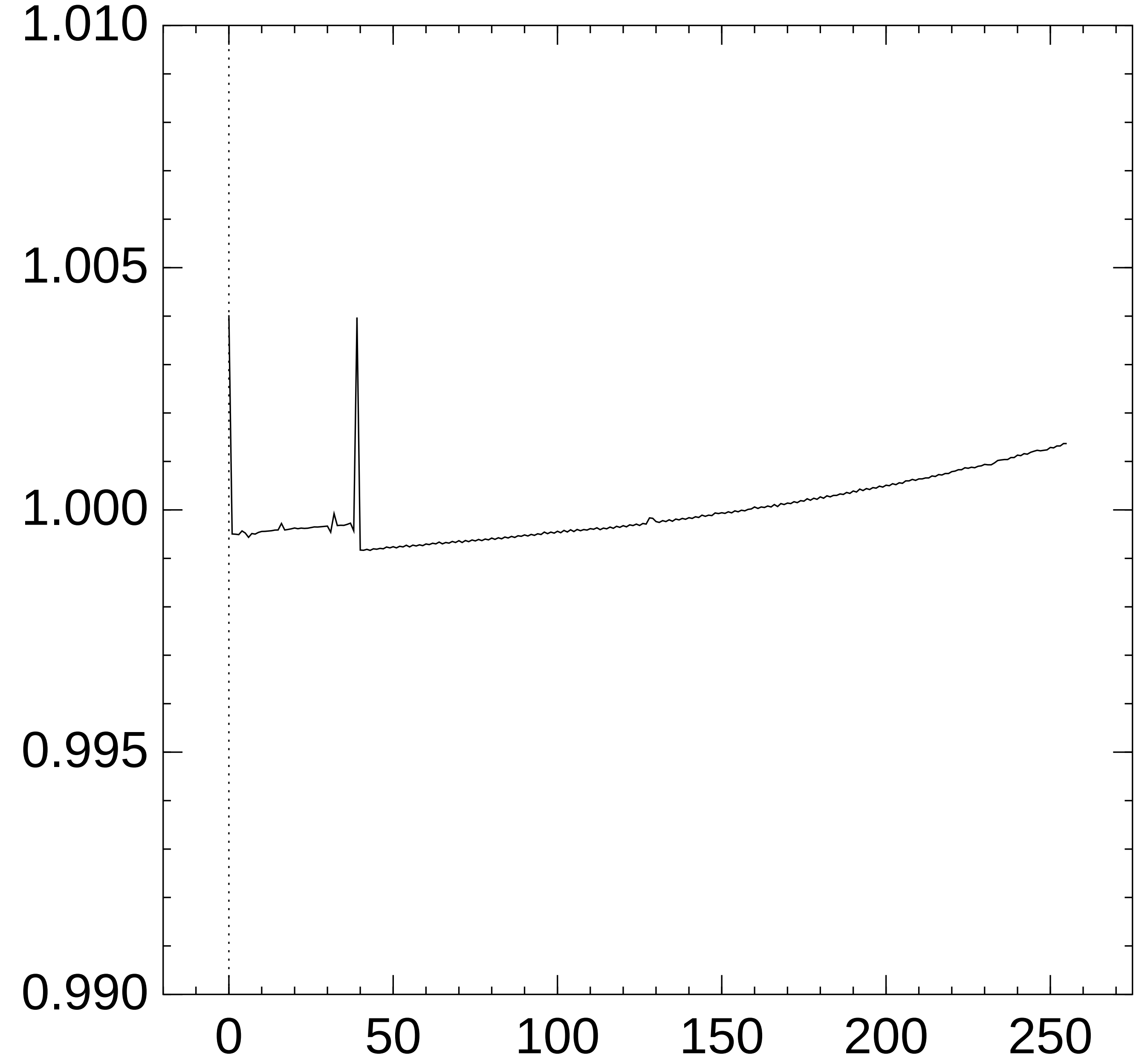
Graph of $256 \Pr[z_{38} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

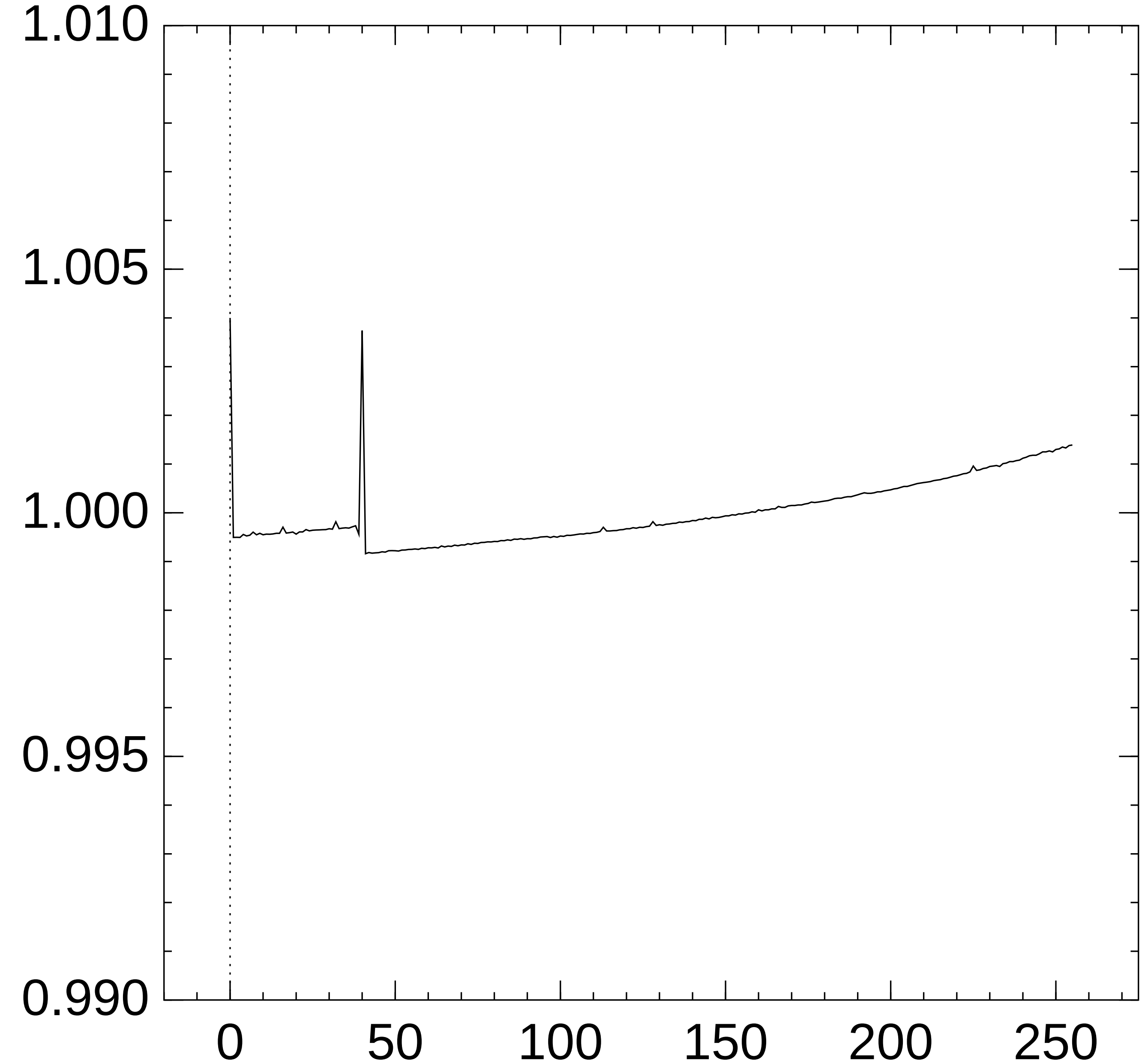
Graph of $256 \Pr[z_{39} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

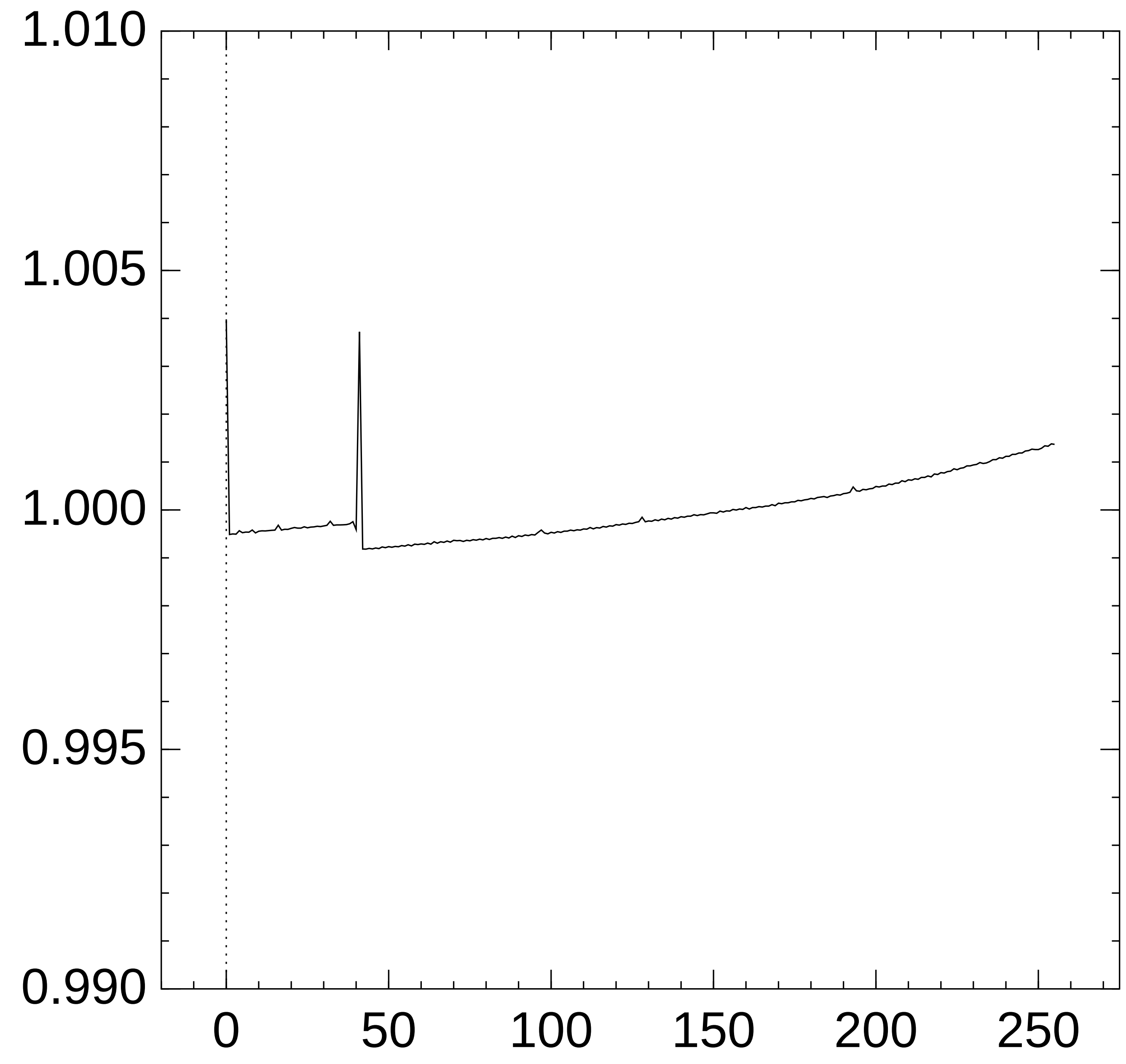
Graph of $256 \Pr[z_{40} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

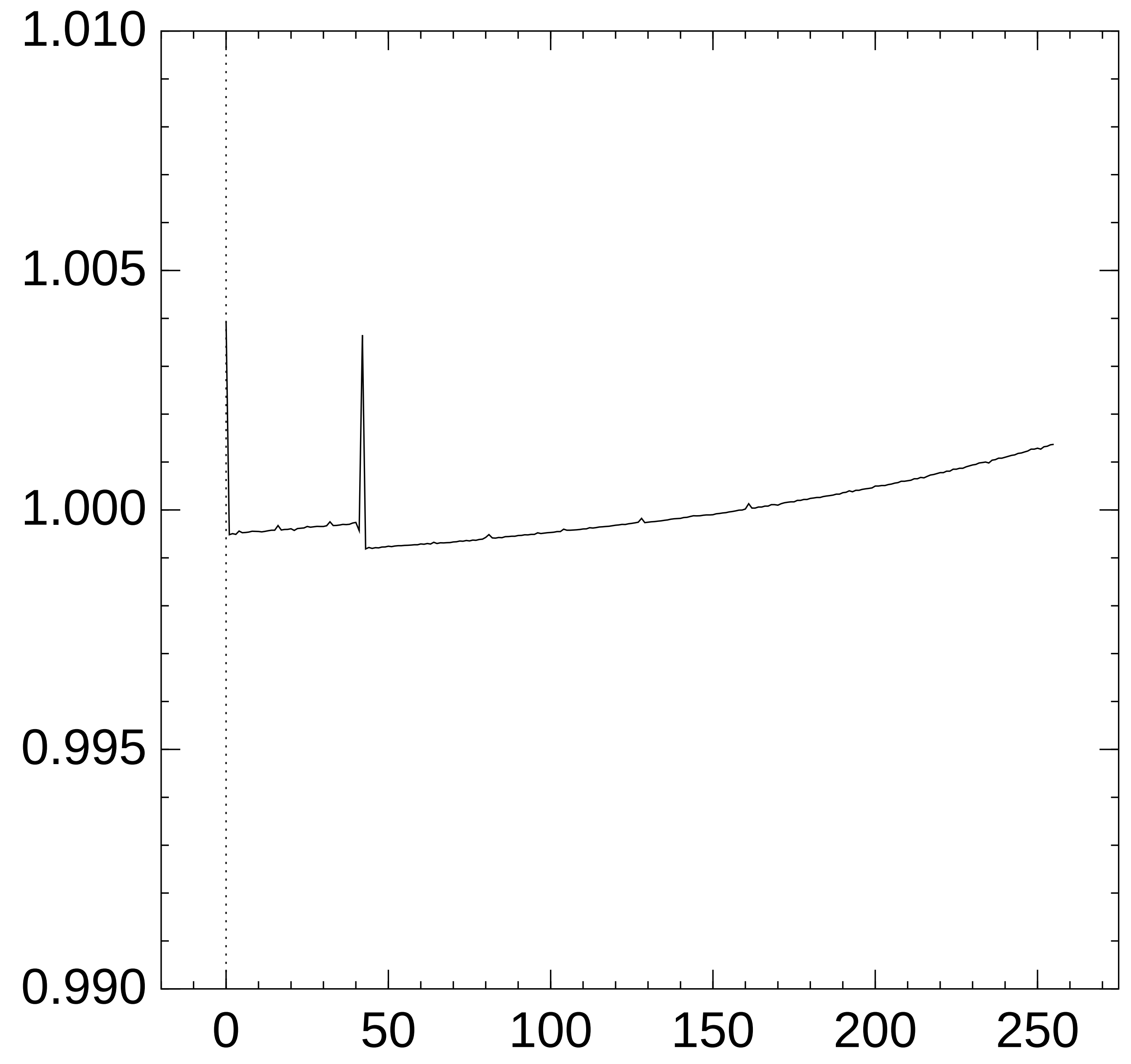
Graph of $256 \Pr[z_{41} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

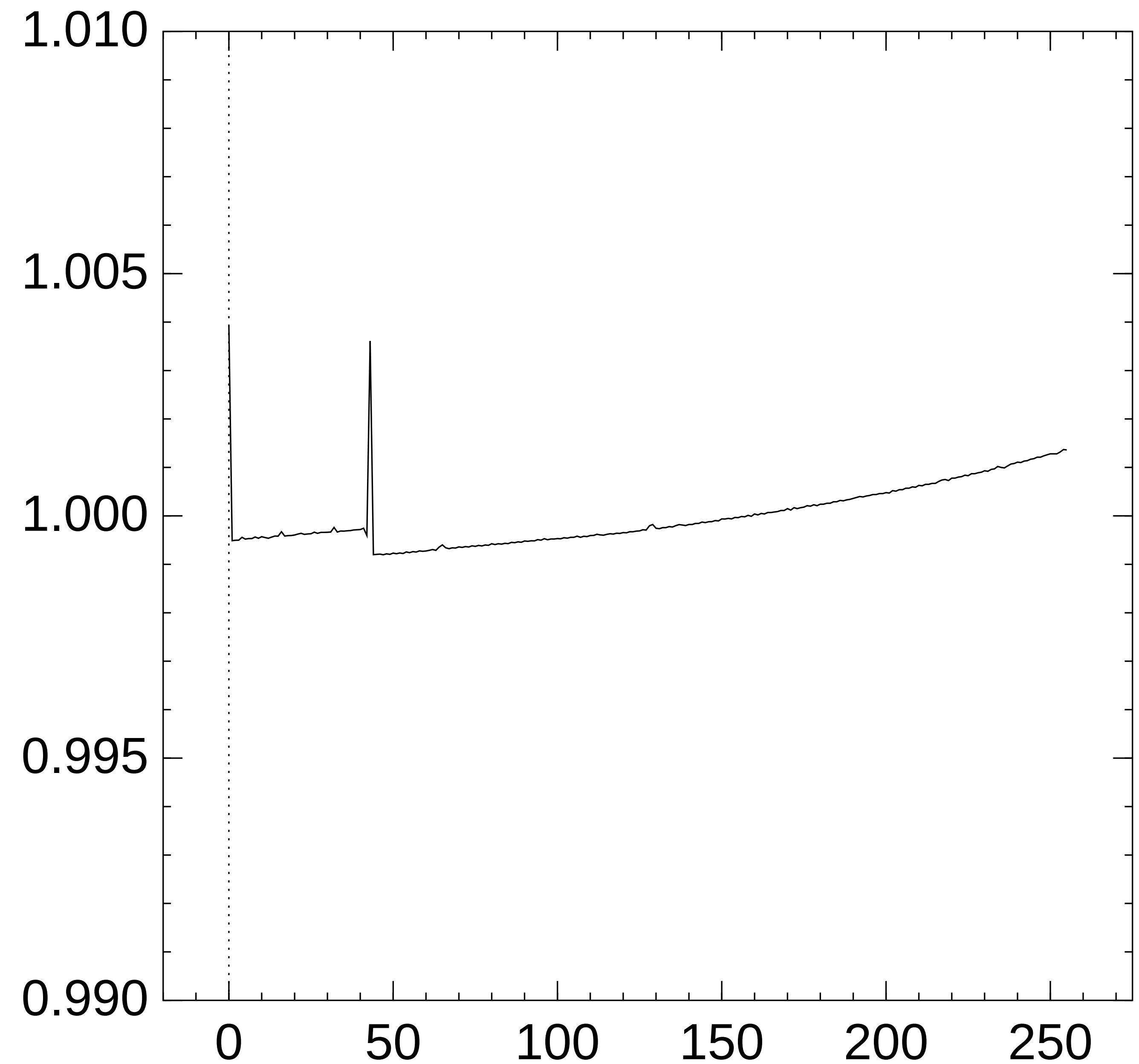
Graph of $256 \Pr[z_{42} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

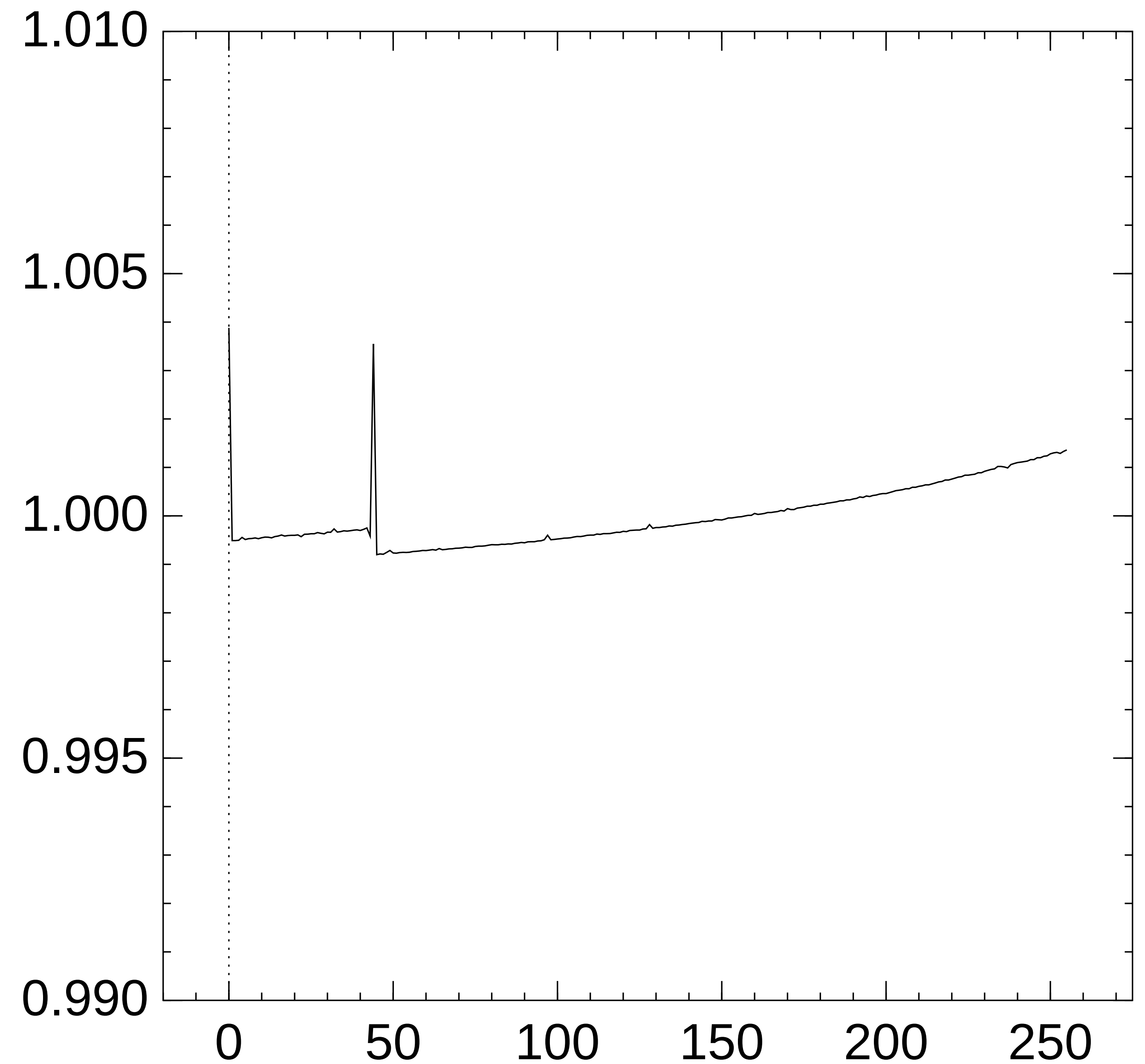
Graph of $256 \Pr[z_{43} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

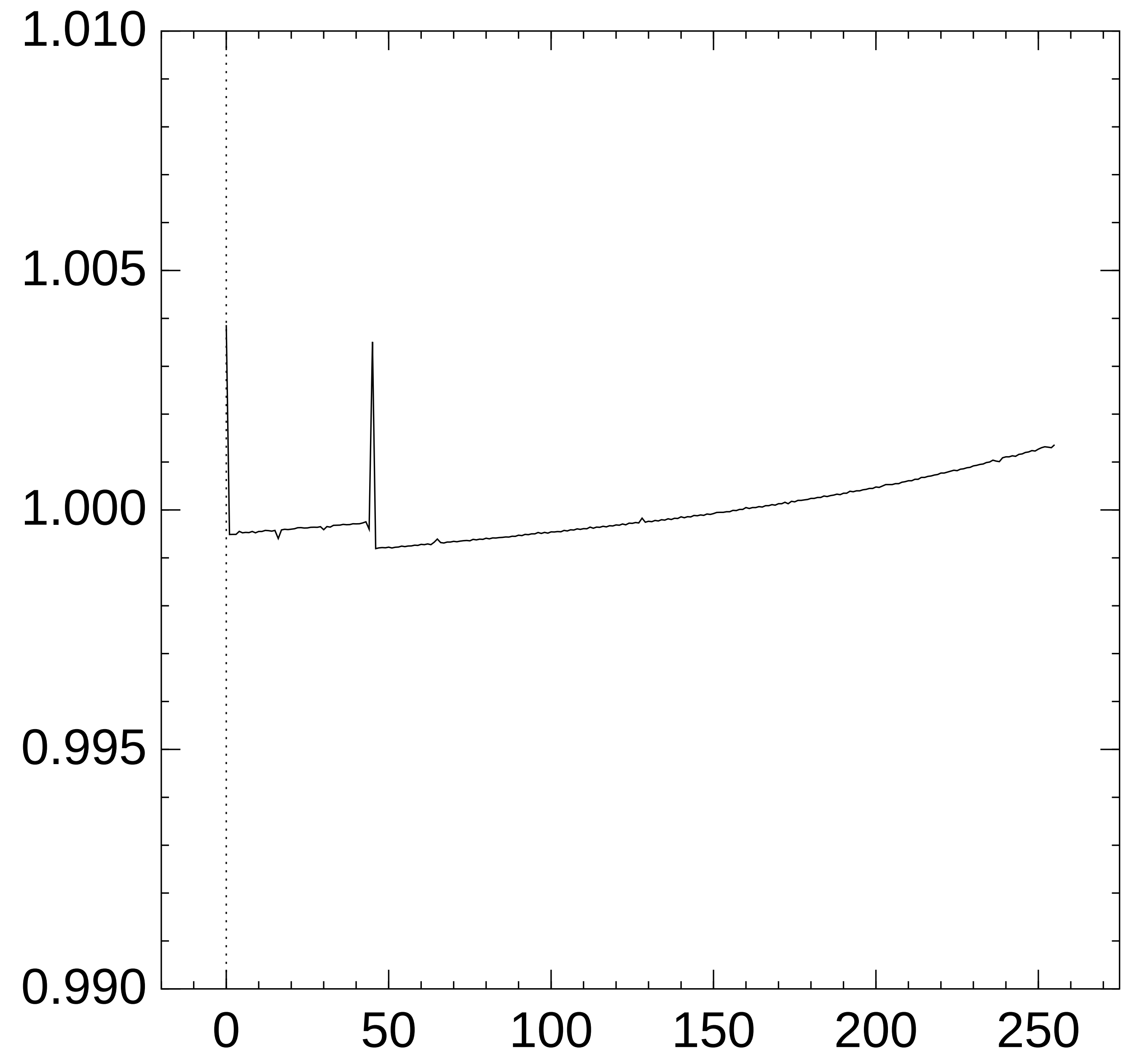
Graph of $256 \Pr[z_{44} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

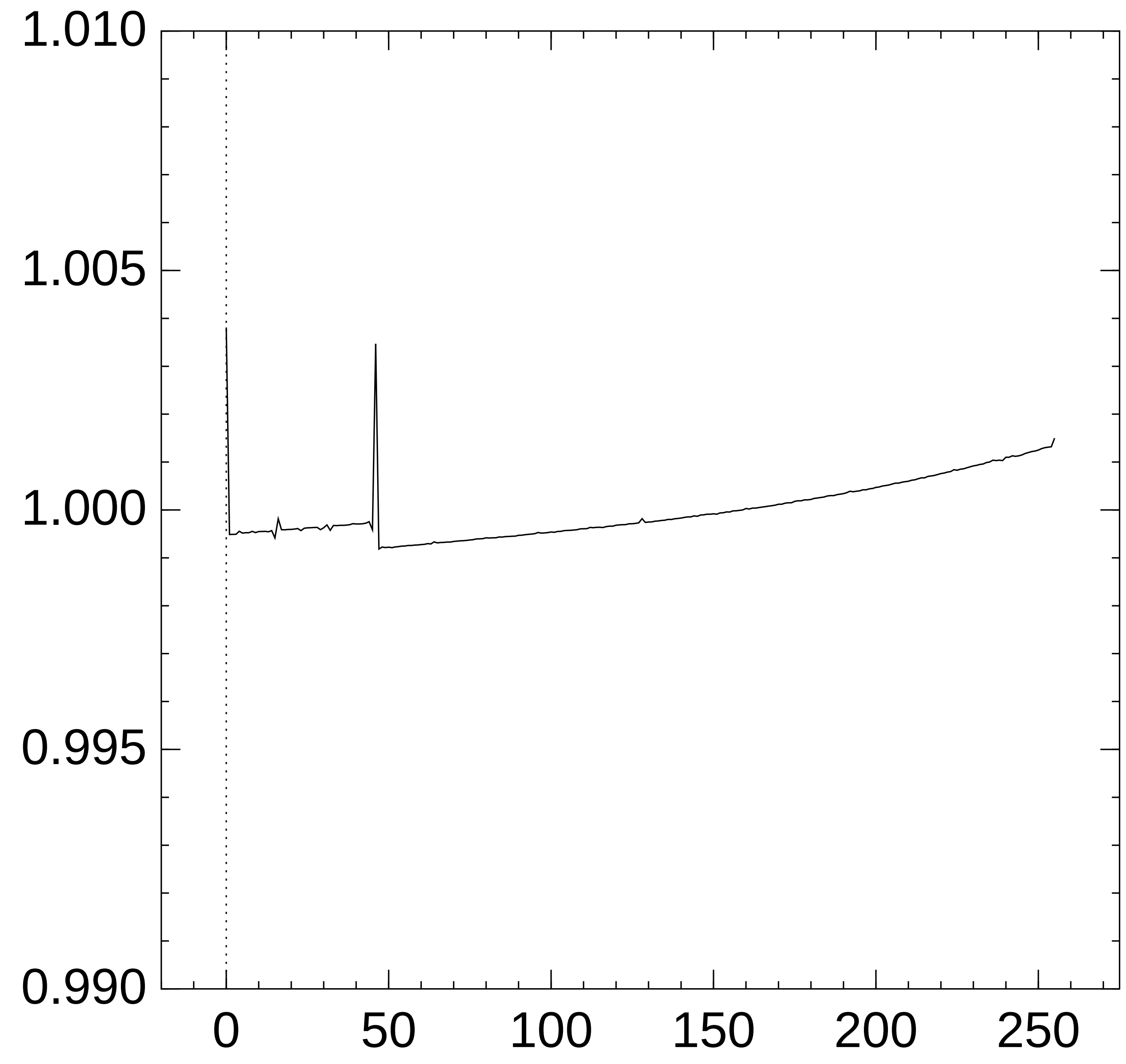
Graph of $256 \Pr[z_{45} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

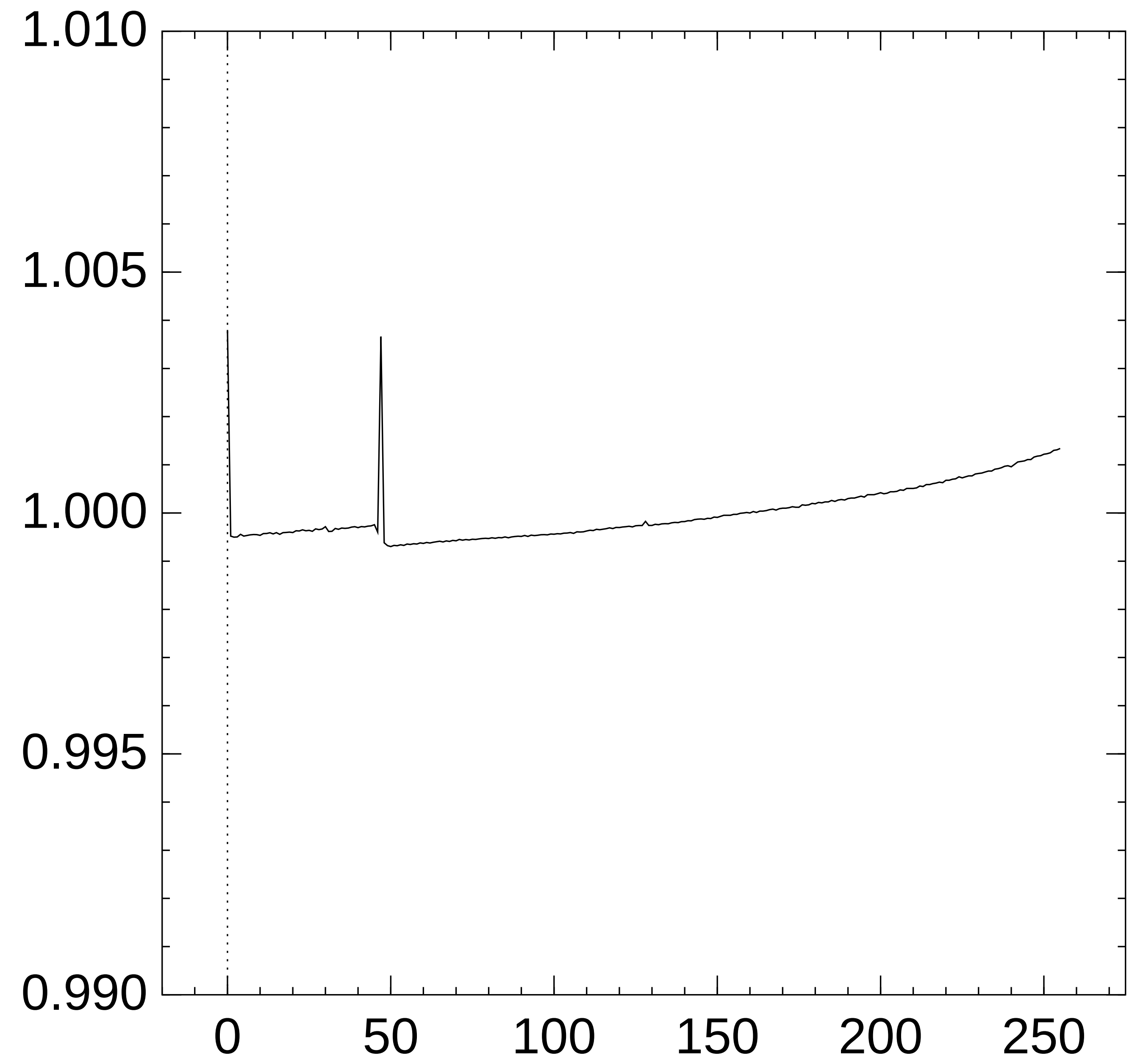
Graph of $256 \Pr[z_{46} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

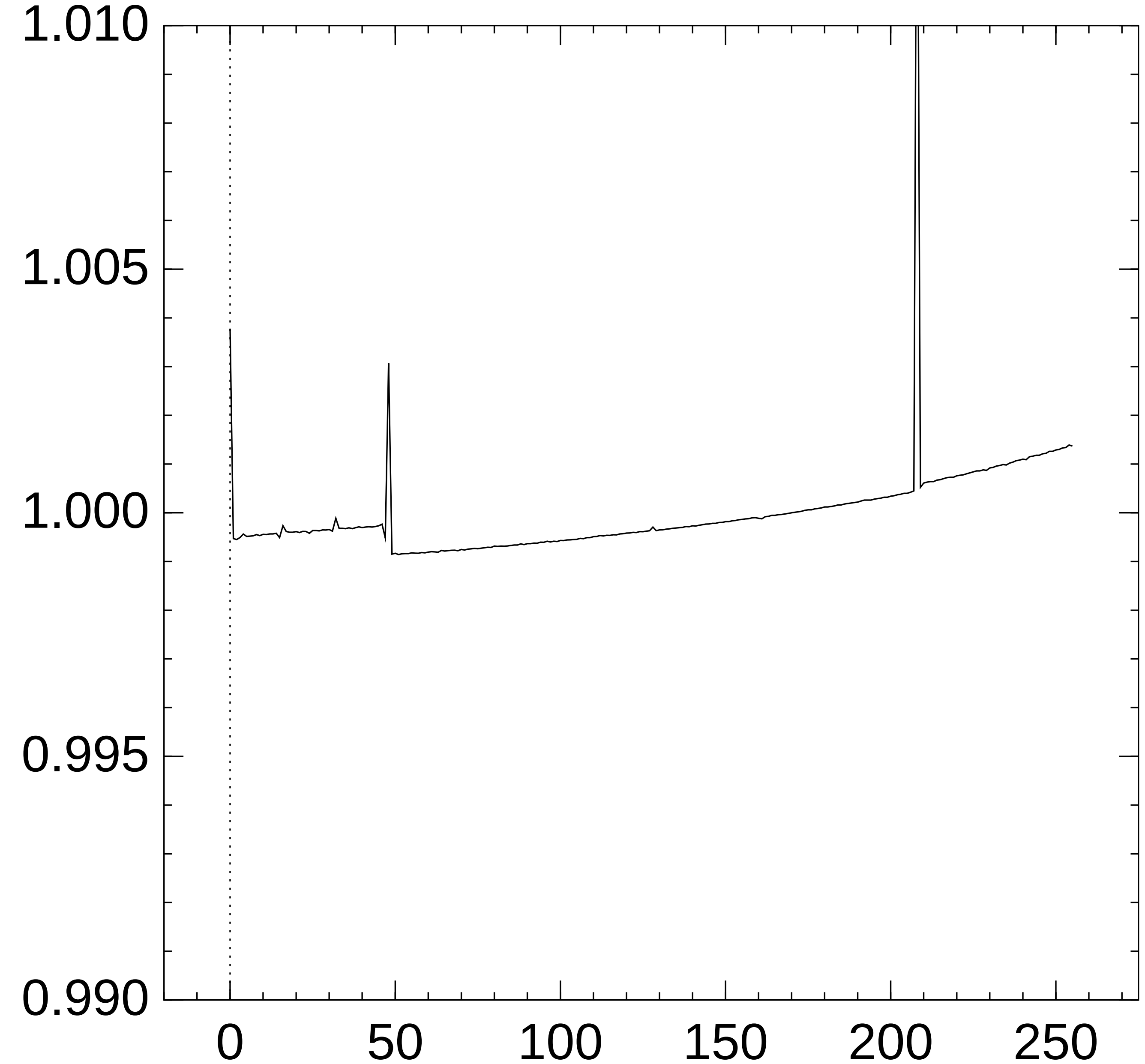
Graph of $256 \Pr[z_{47} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

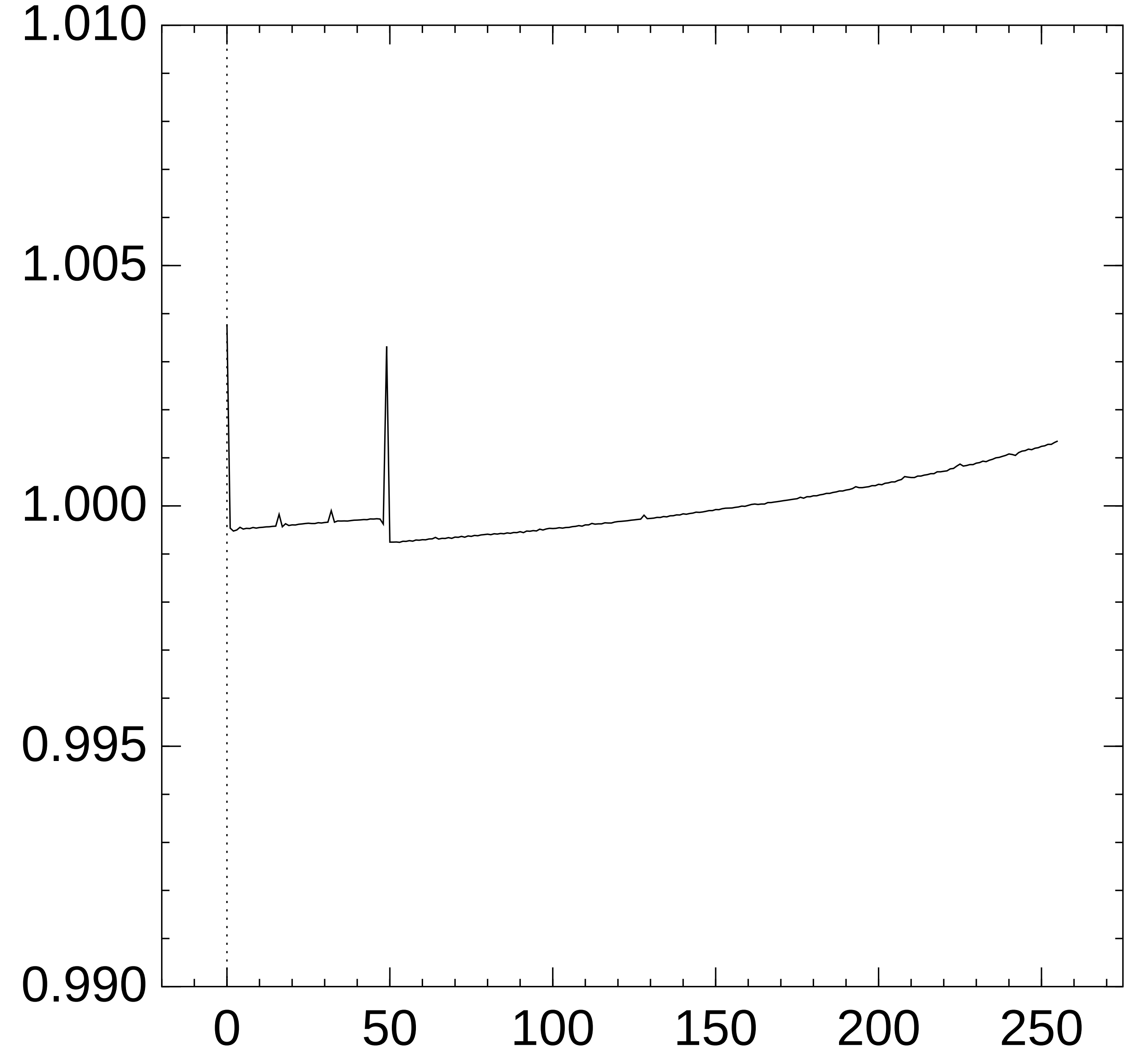
Graph of $256 \Pr[z_{48} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

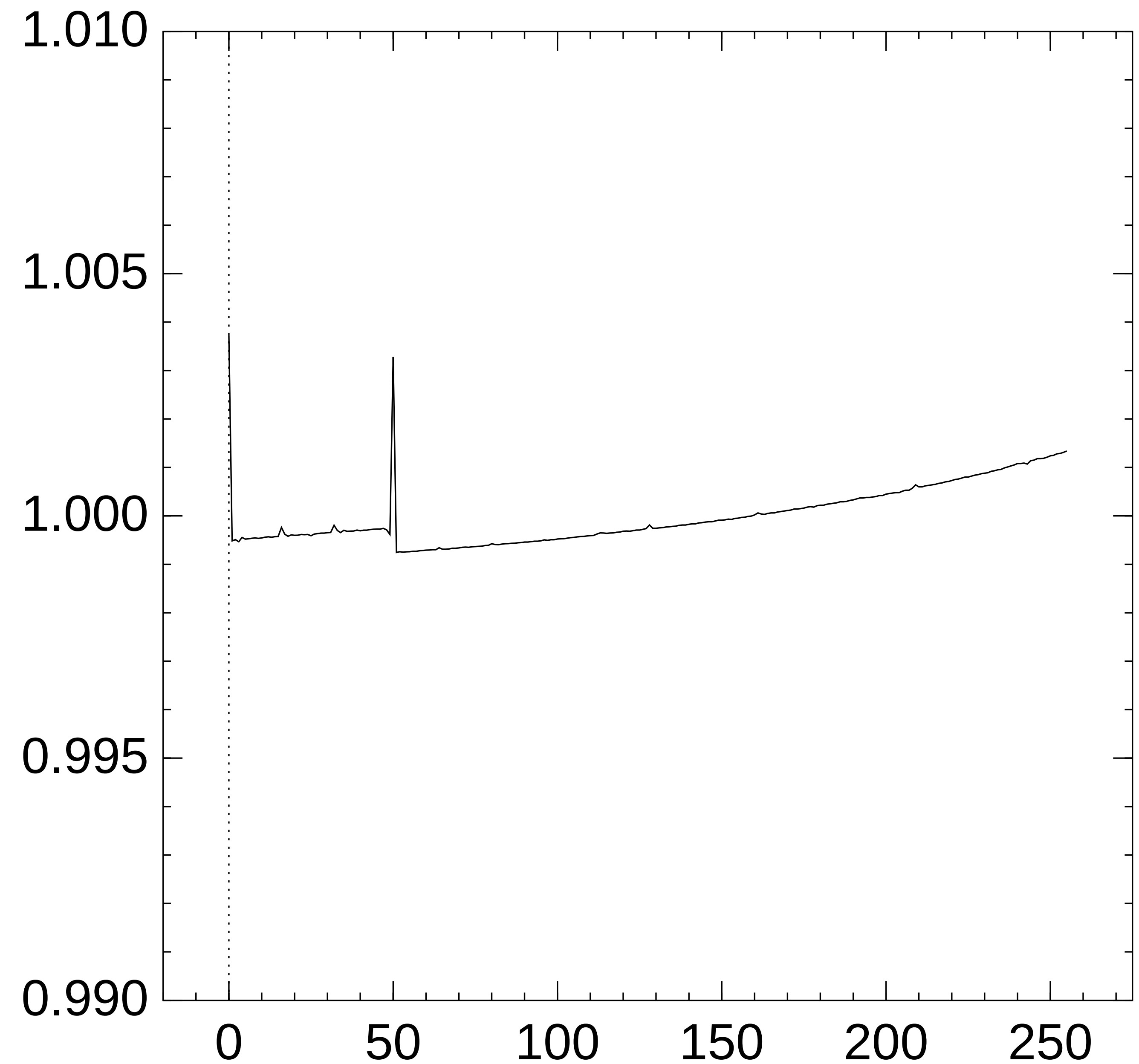
Graph of $256 \Pr[z_{49} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

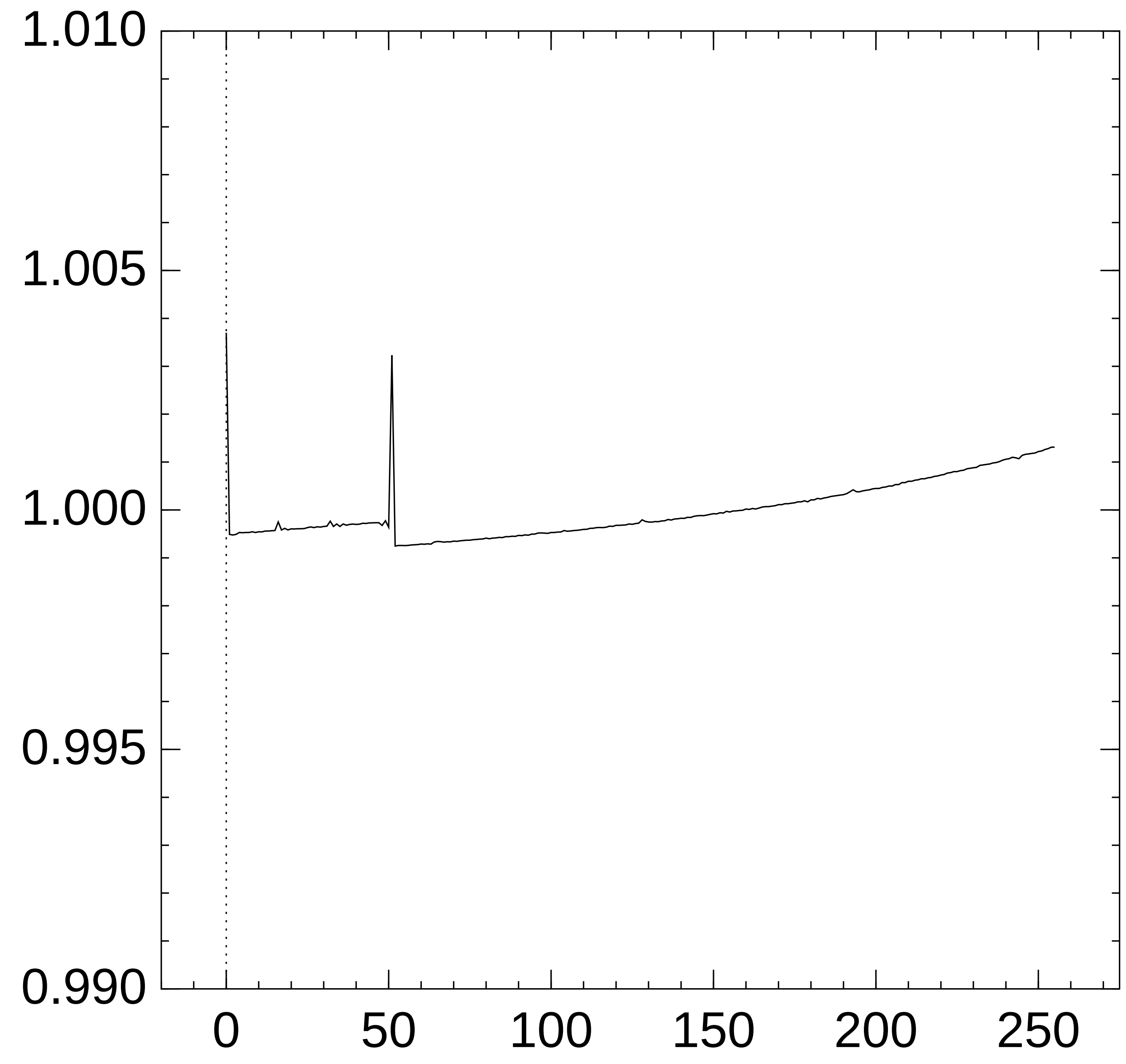
Graph of $256 \Pr[z_{50} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

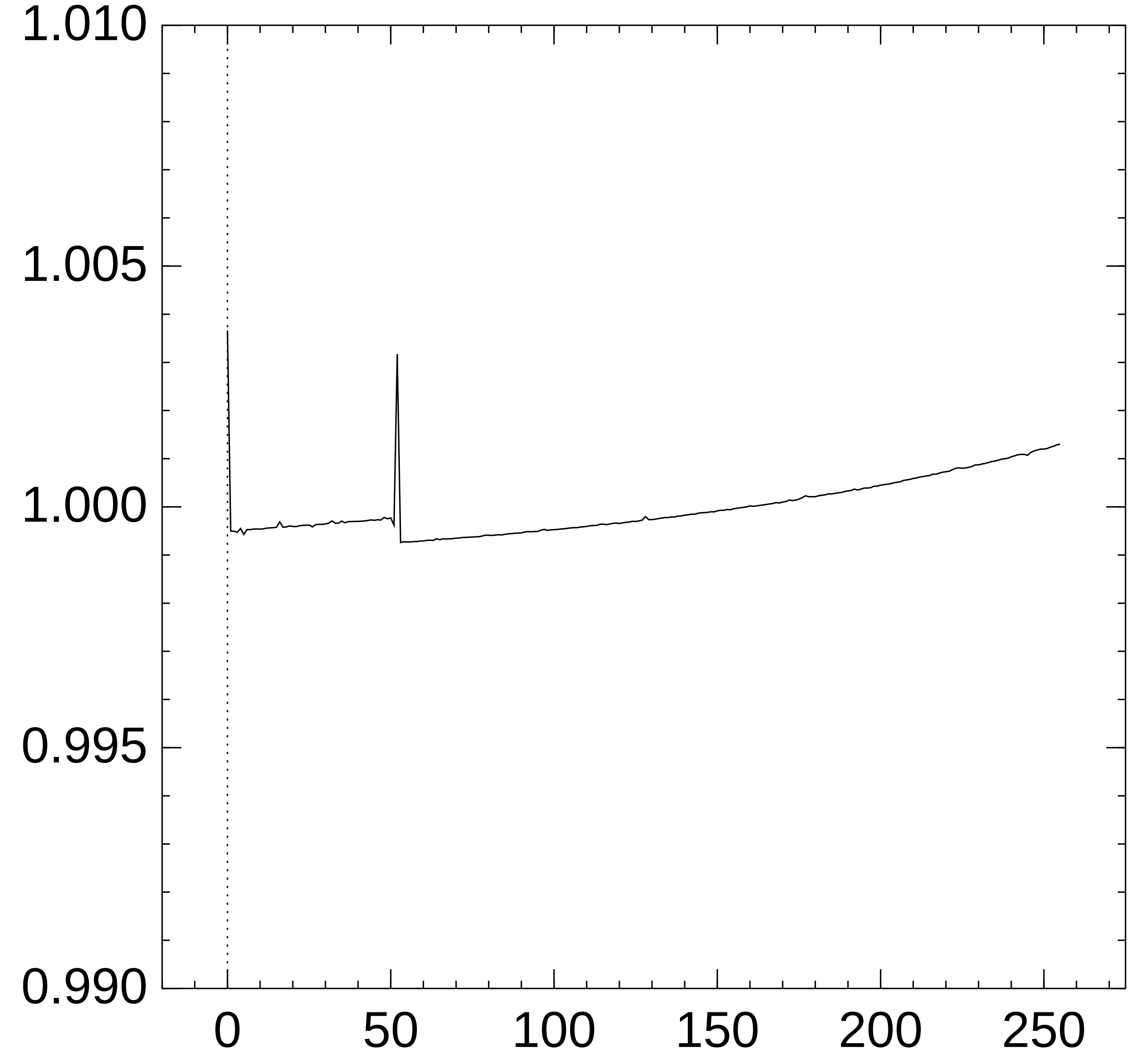
Graph of $256 \Pr[z_{51} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

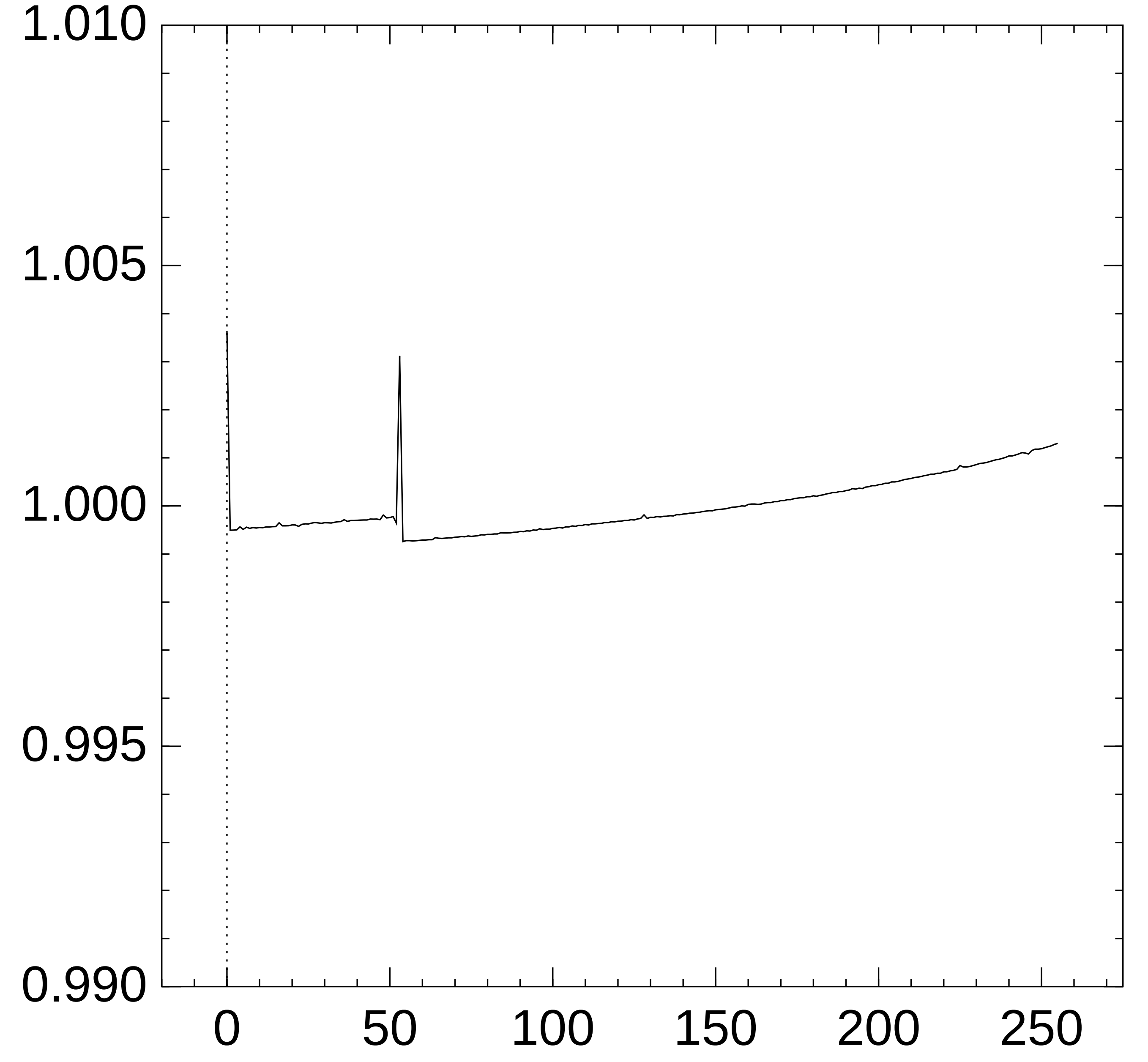
Graph of $256 \Pr[z_{52} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

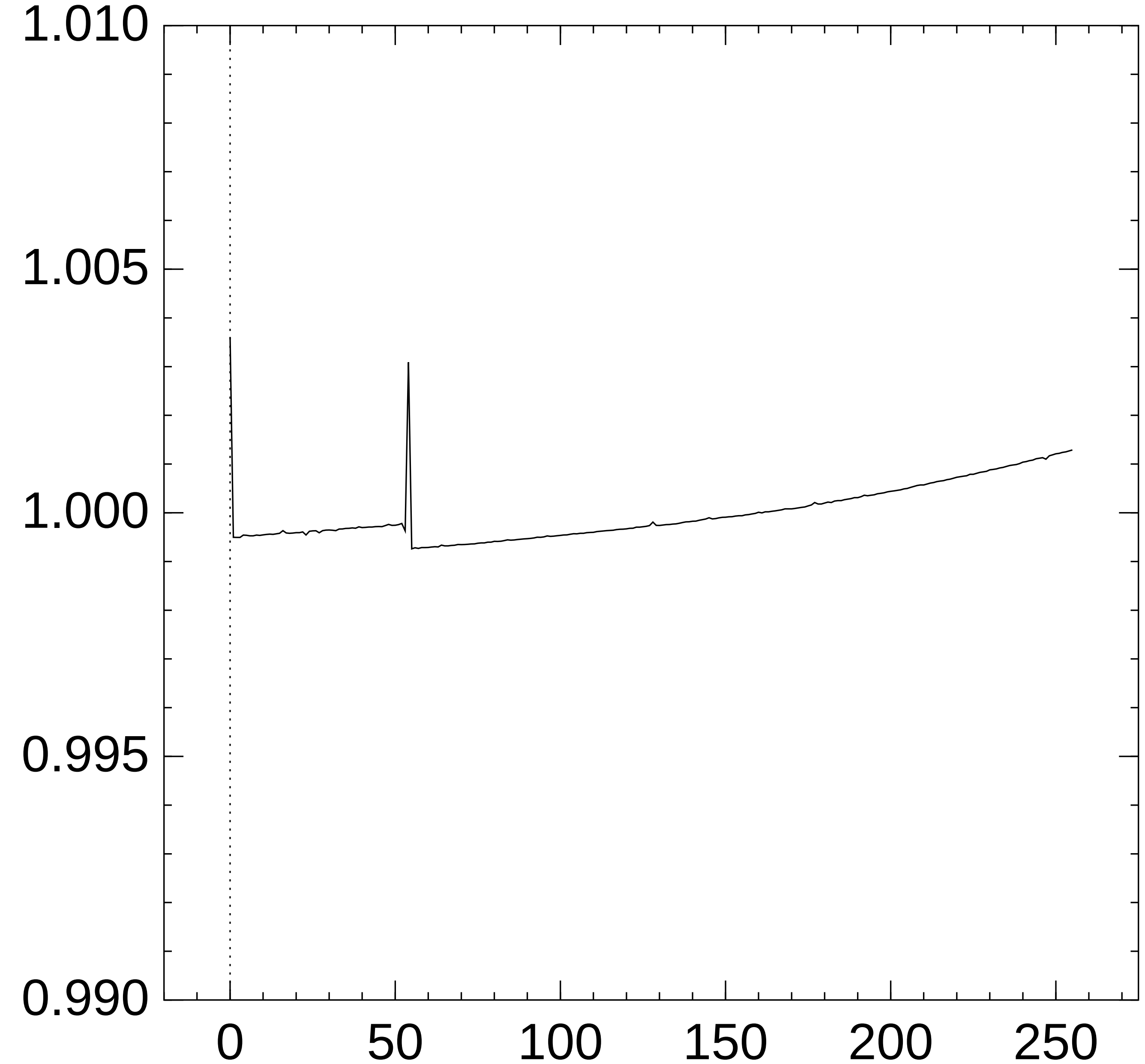
Graph of $256 \Pr[z_{53} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

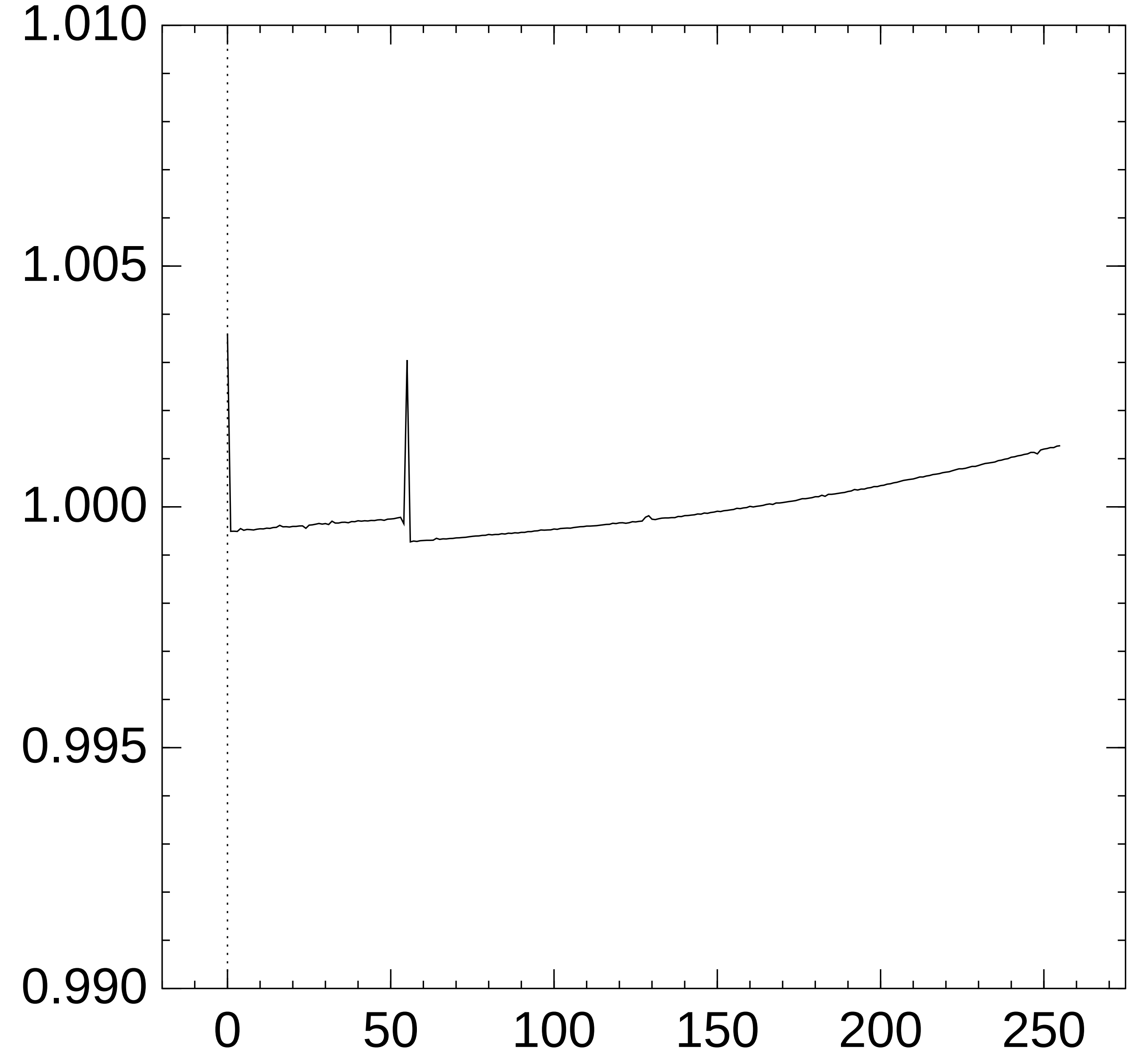
Graph of $256 \Pr[z_{54} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

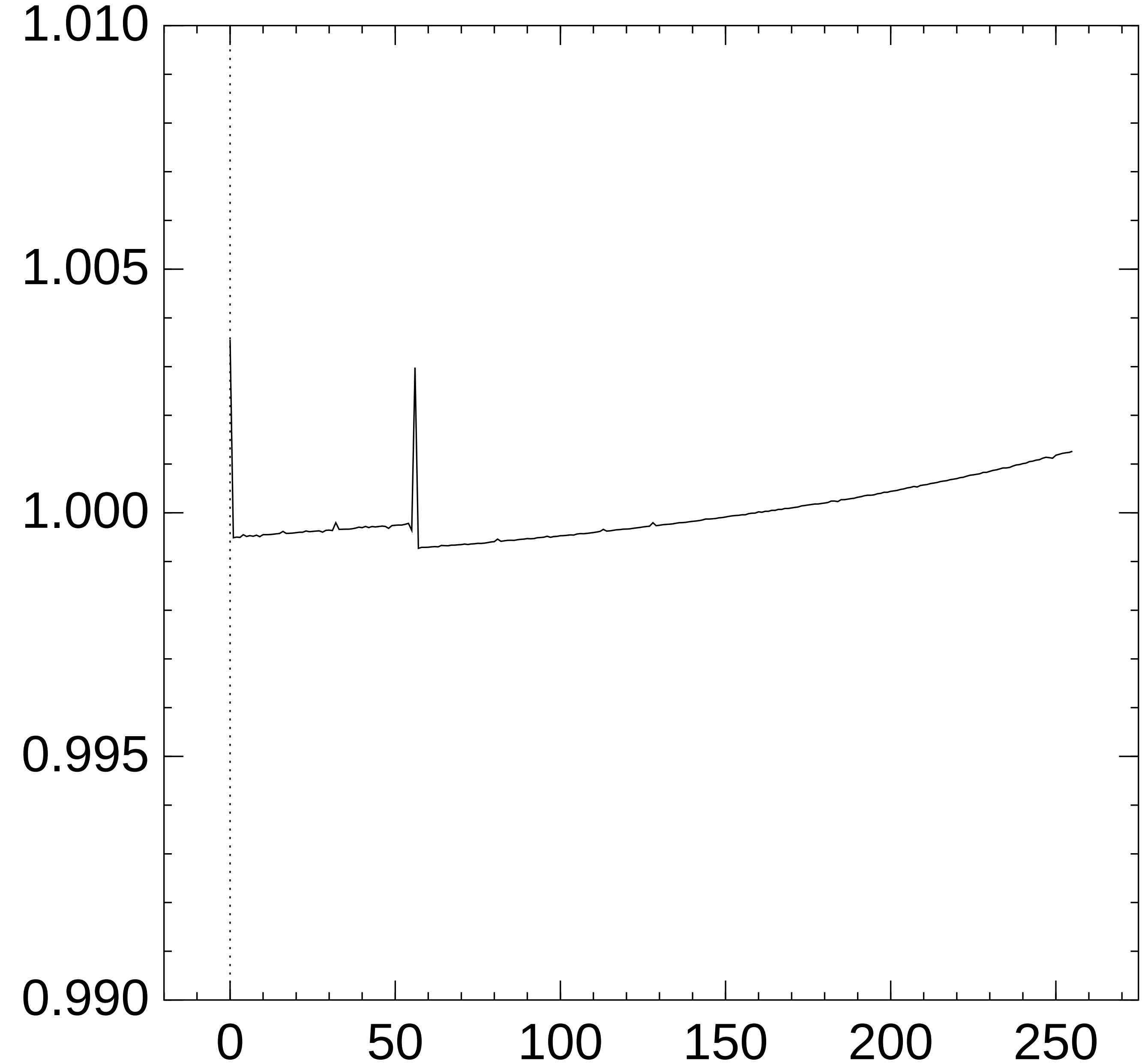
Graph of $256 \Pr[z_{55} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

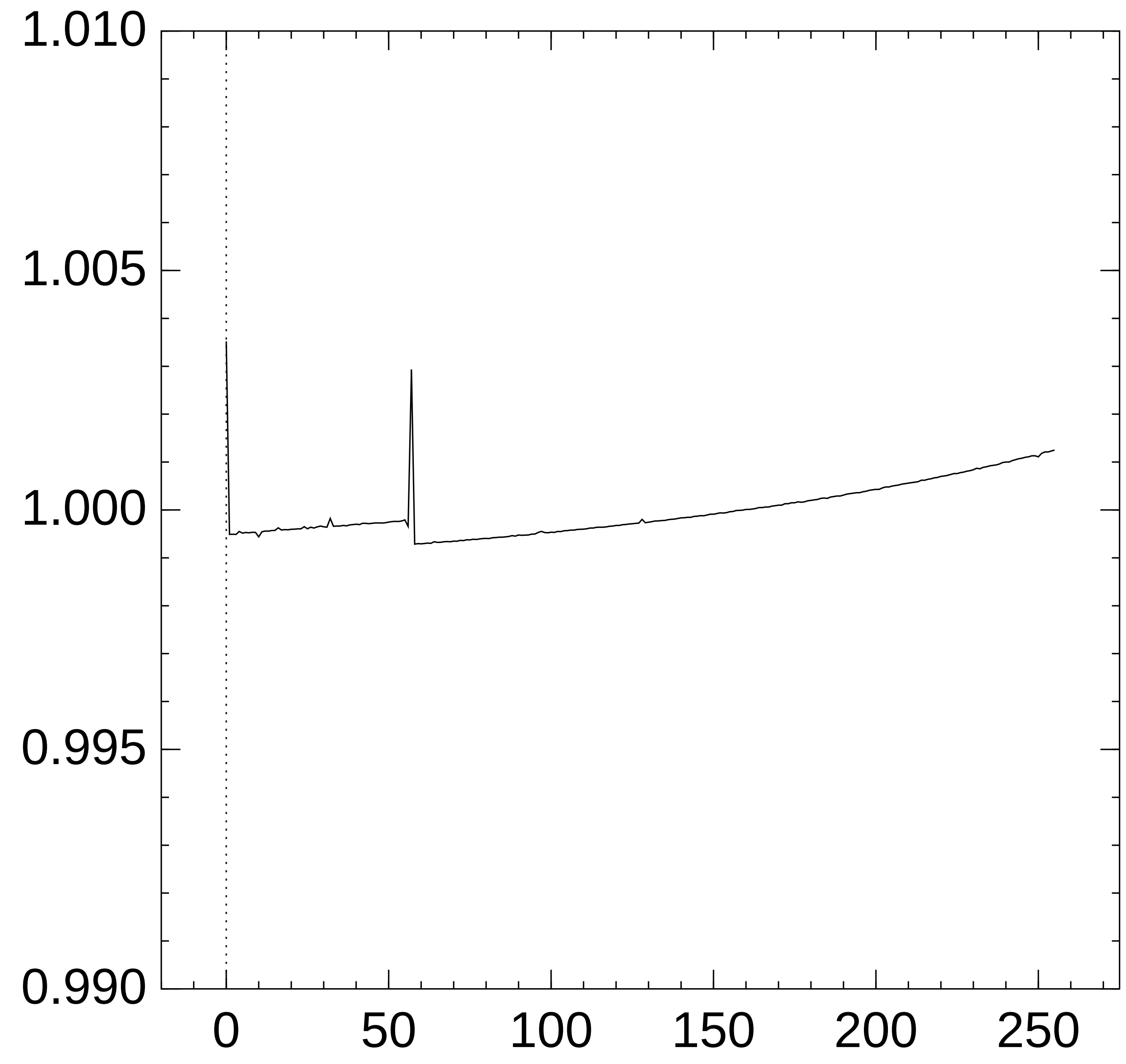
Graph of $256 \Pr[z_{56} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

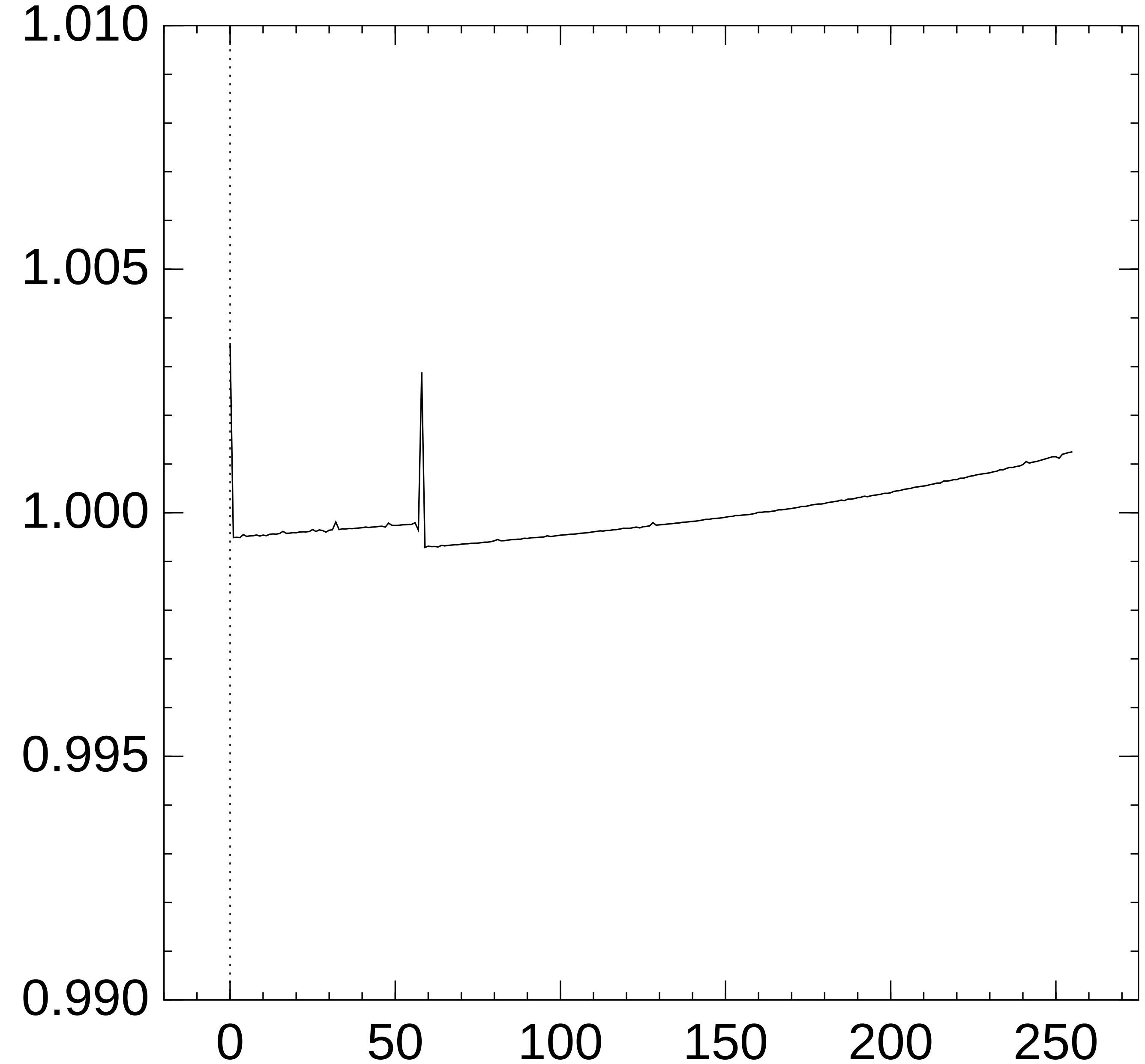
Graph of $256 \Pr[z_{57} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

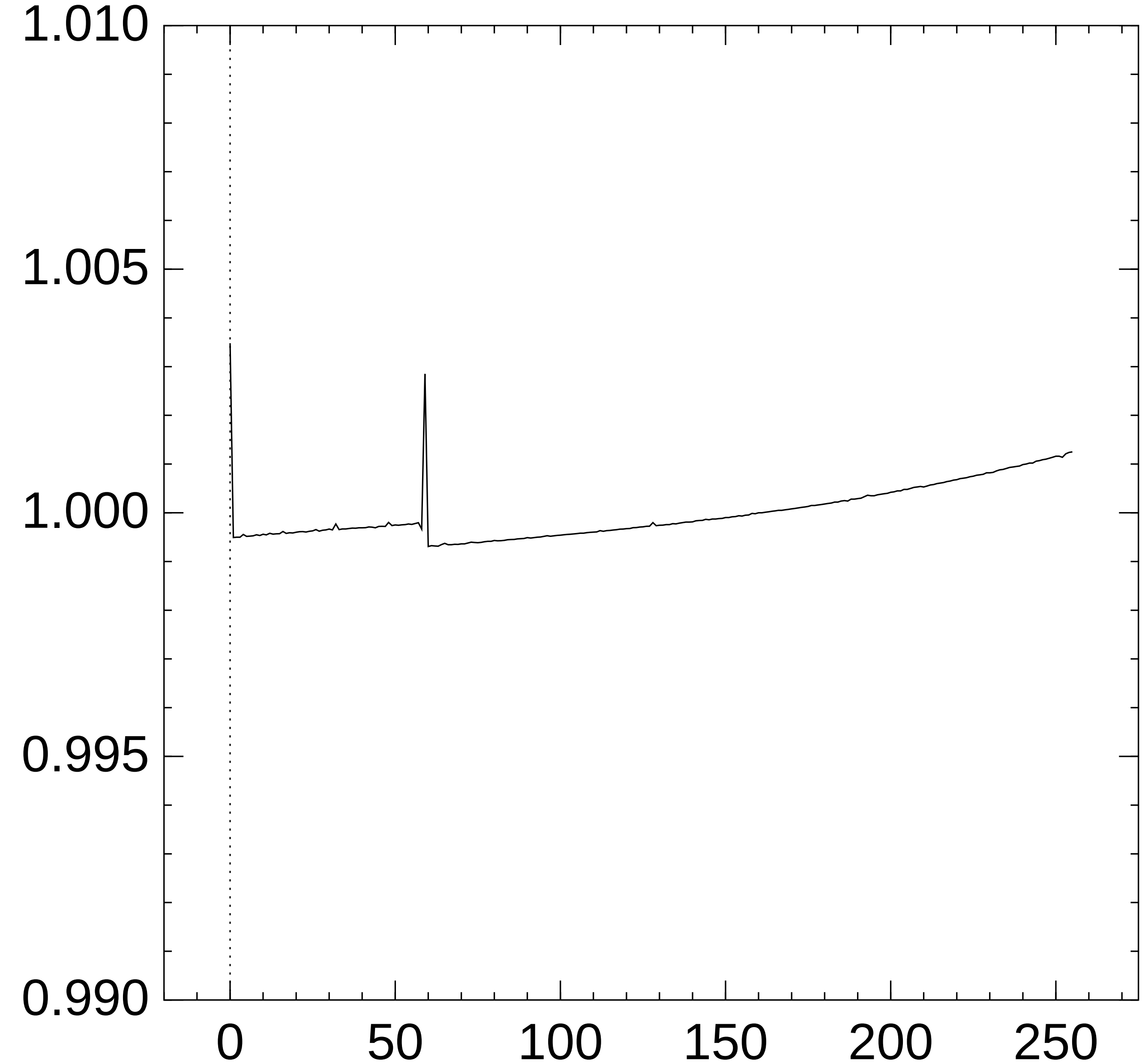
Graph of $256 \Pr[z_{58} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

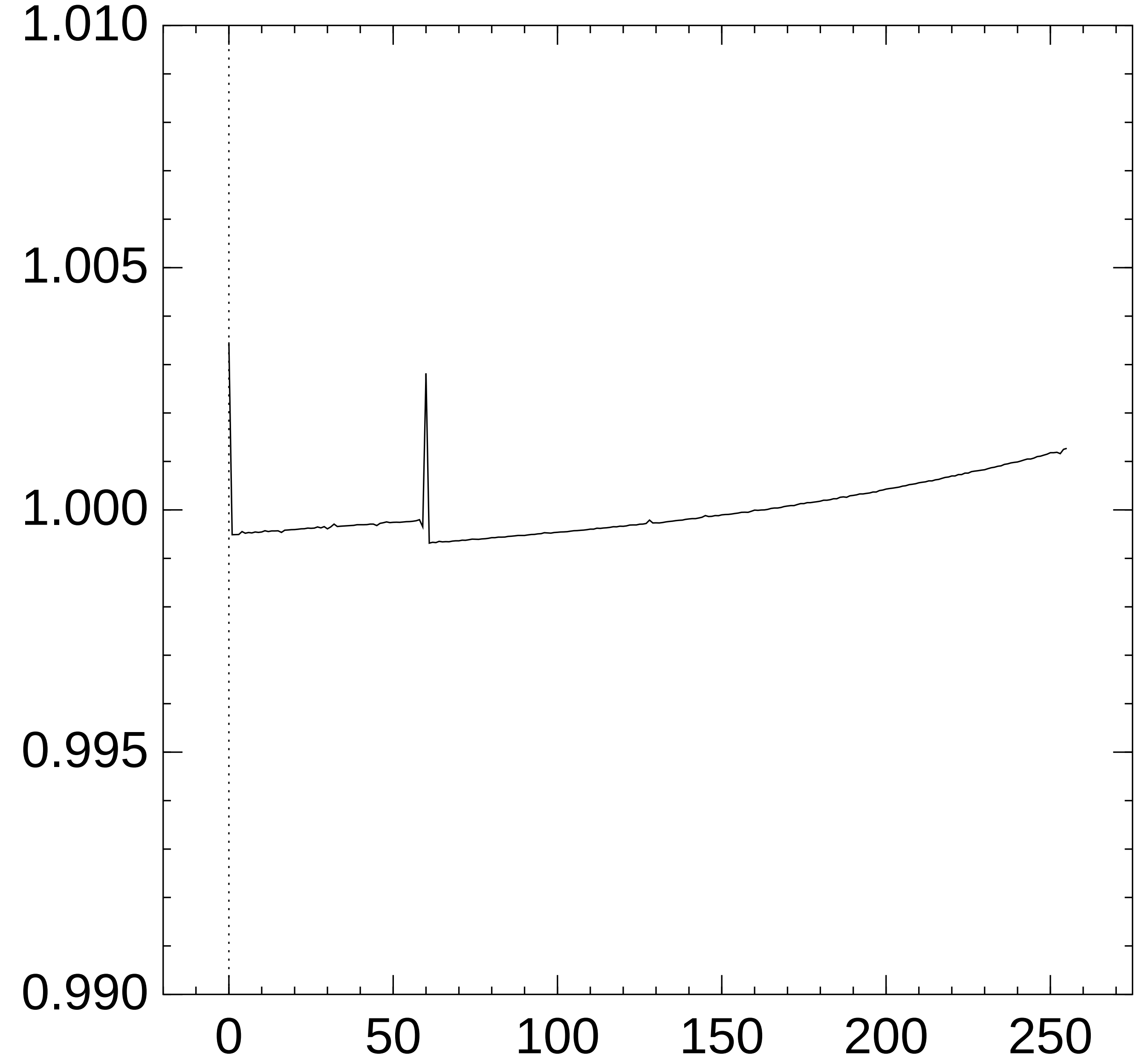
Graph of $256 \Pr[z_{59} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

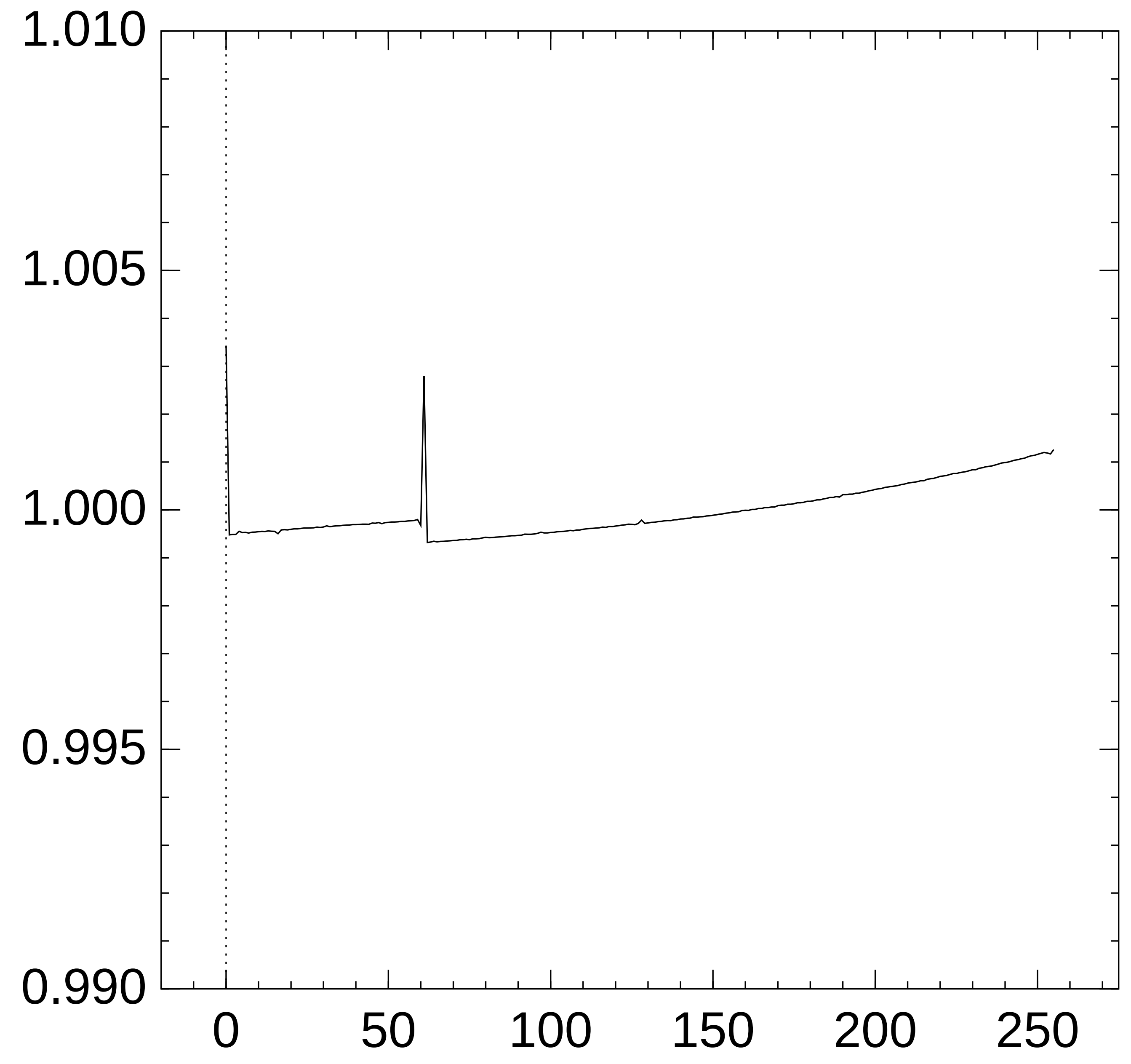
Graph of $256 \Pr[z_{60} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

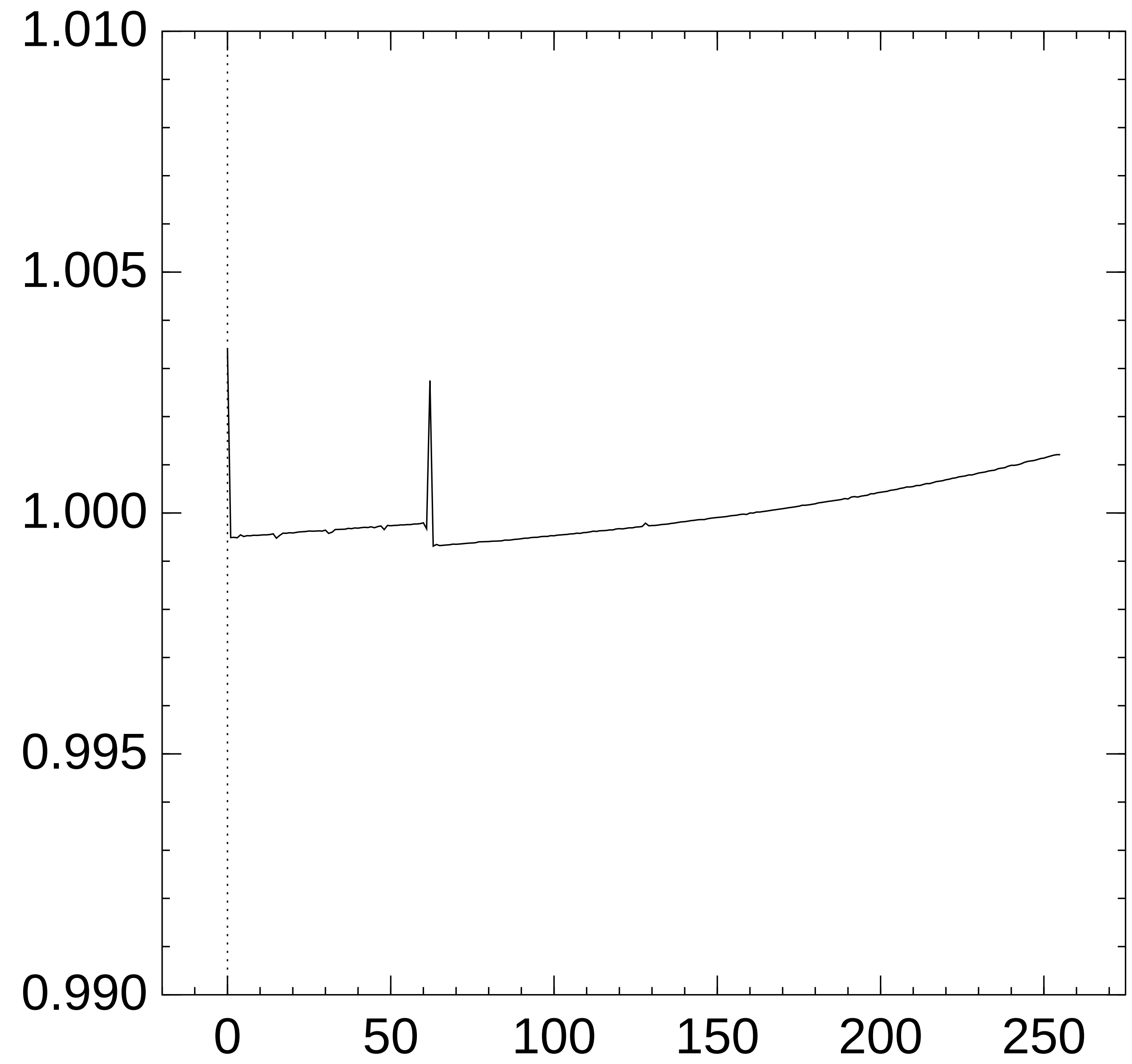
Graph of $256 \Pr[z_{61} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

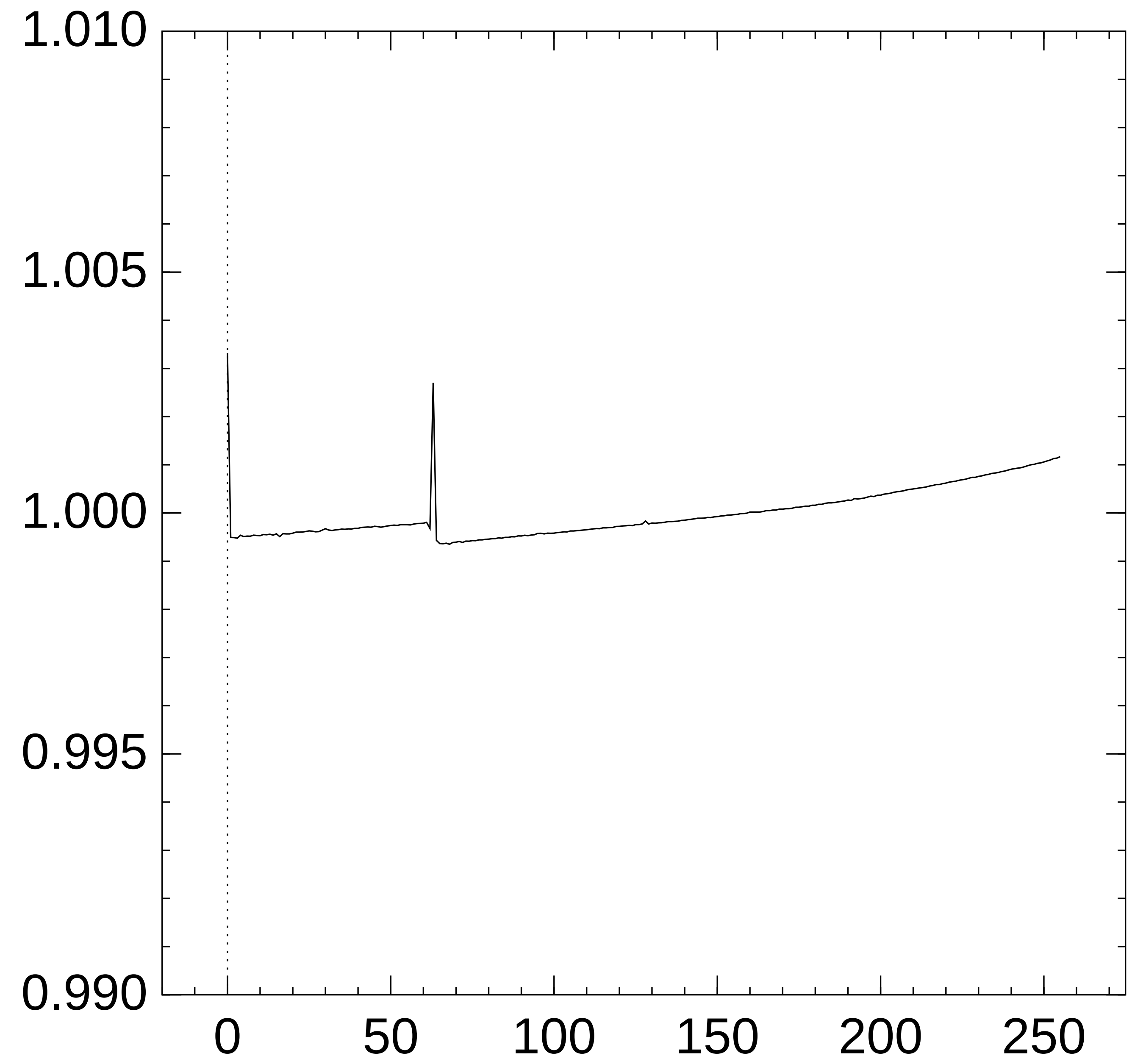
Graph of $256 \Pr[z_{62} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

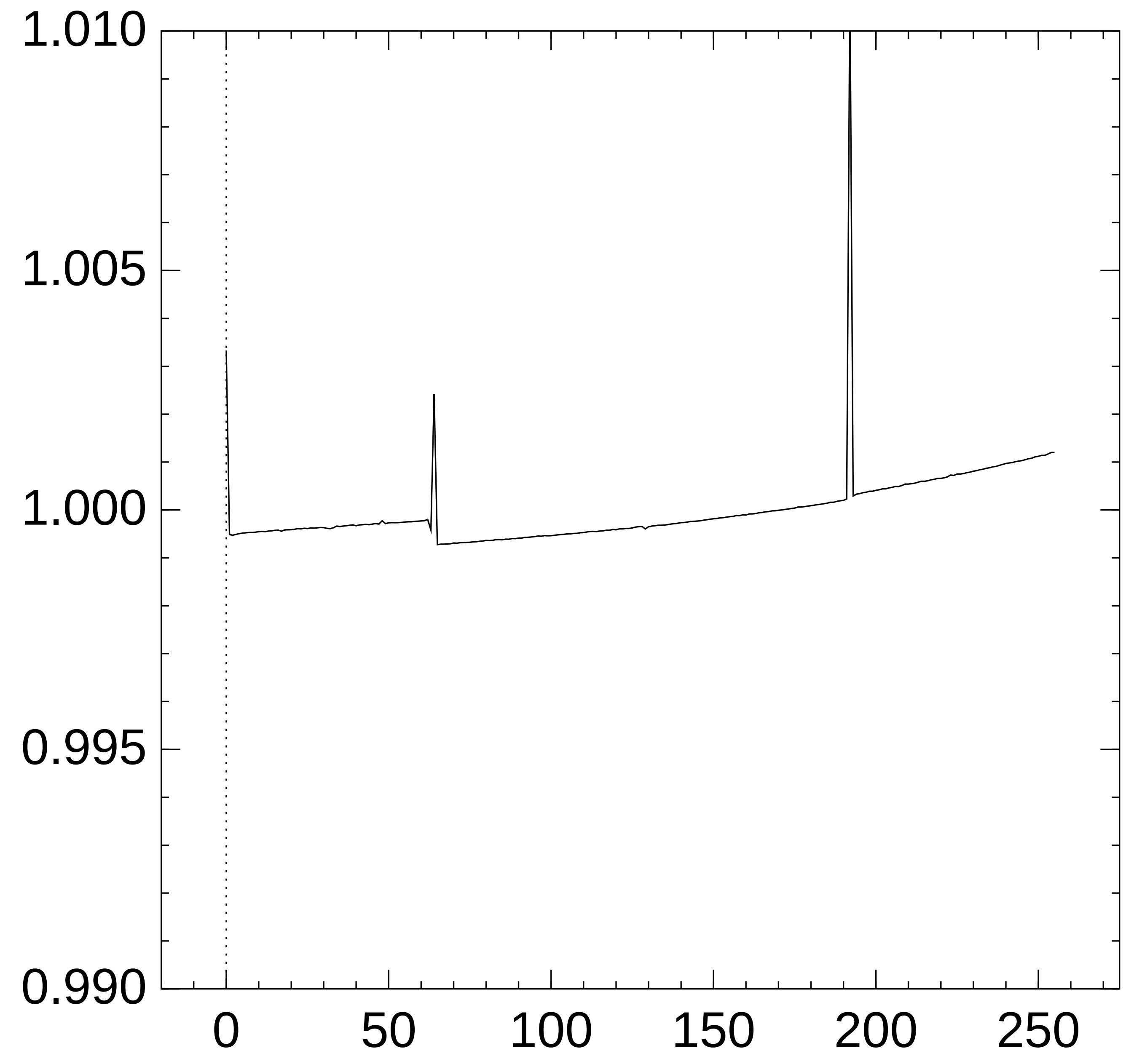
Graph of $256 \Pr[z_{63} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

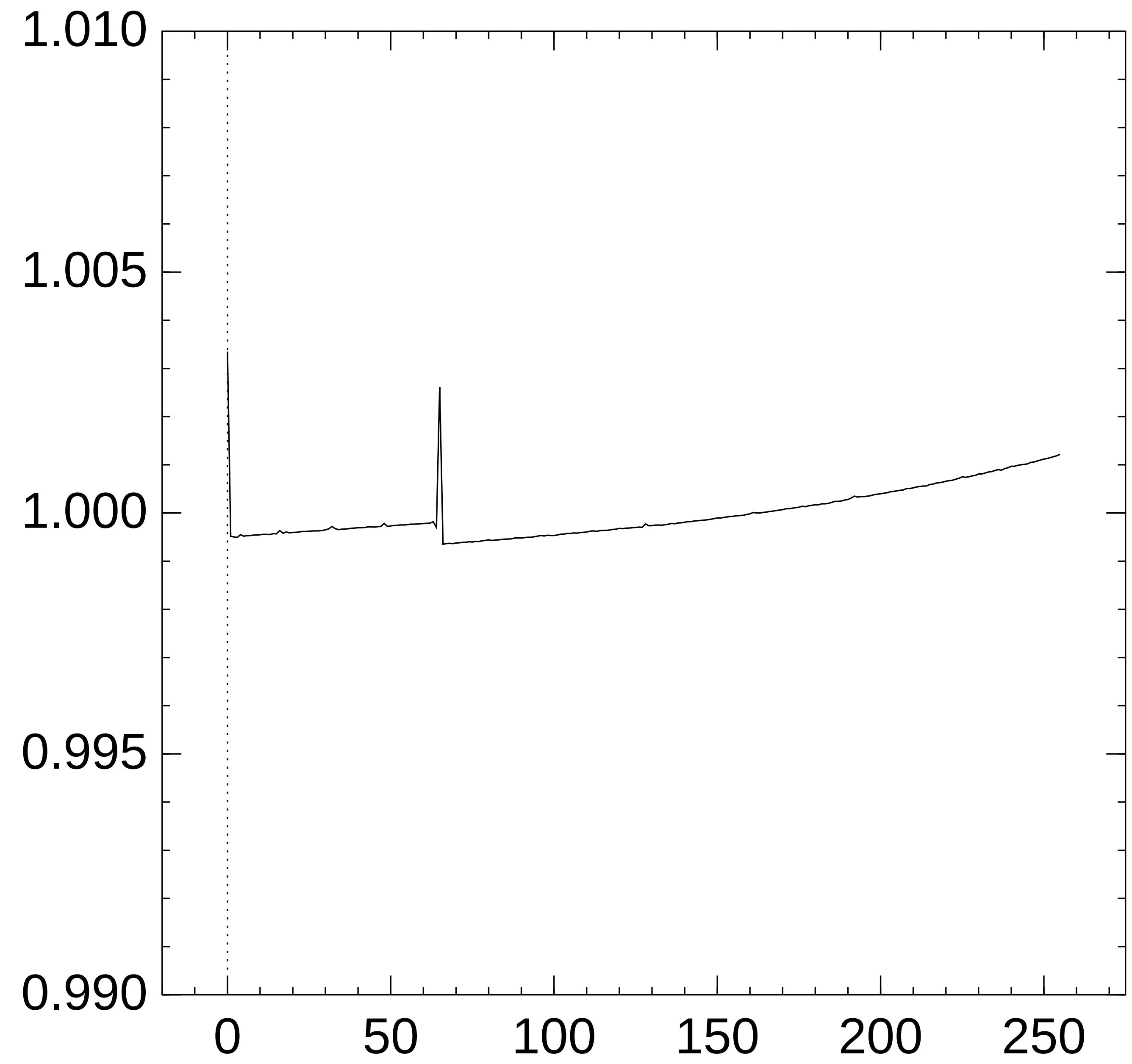
Graph of $256 \Pr[z_{64} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

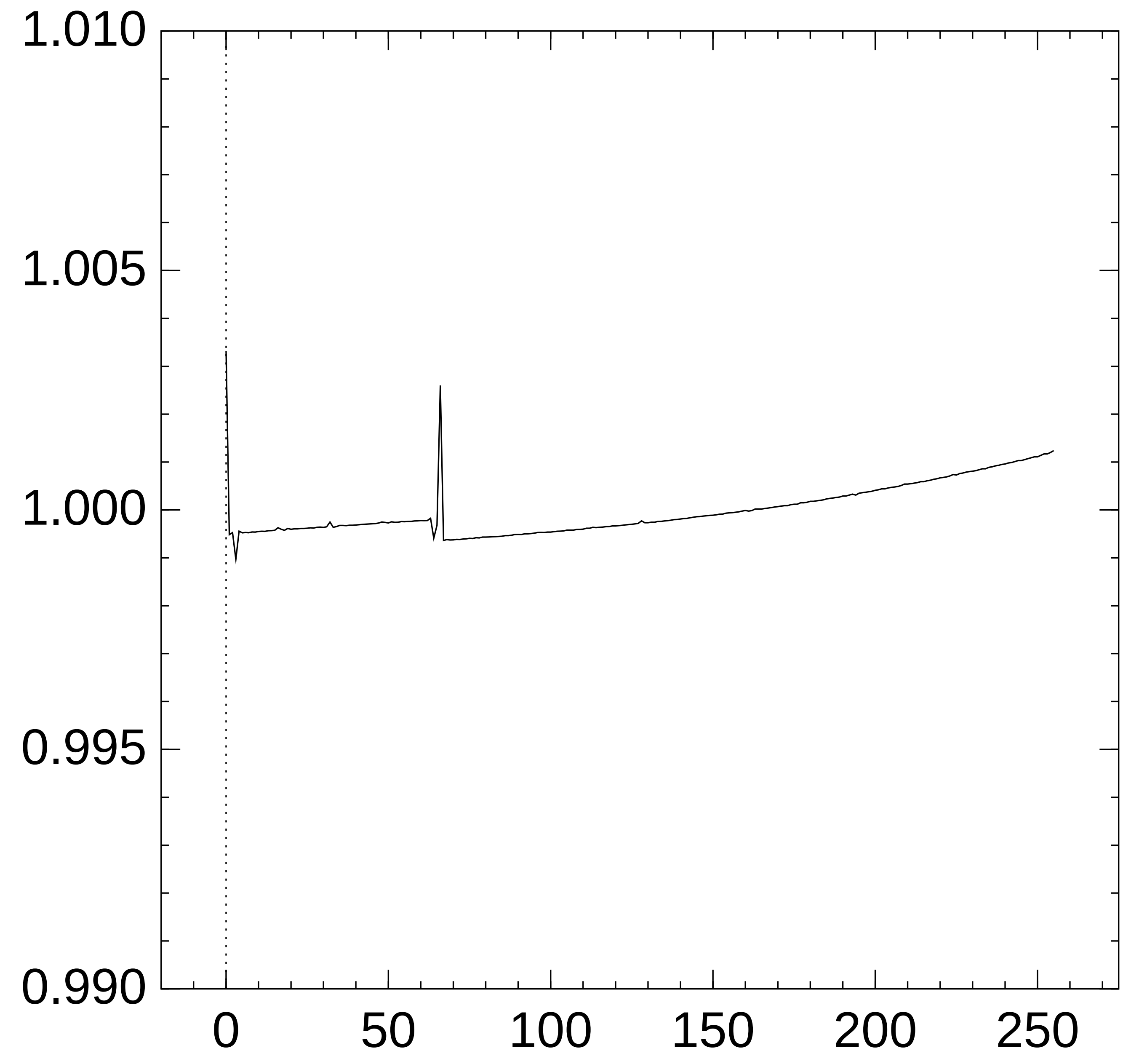
Graph of $256 \Pr[z_{65} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

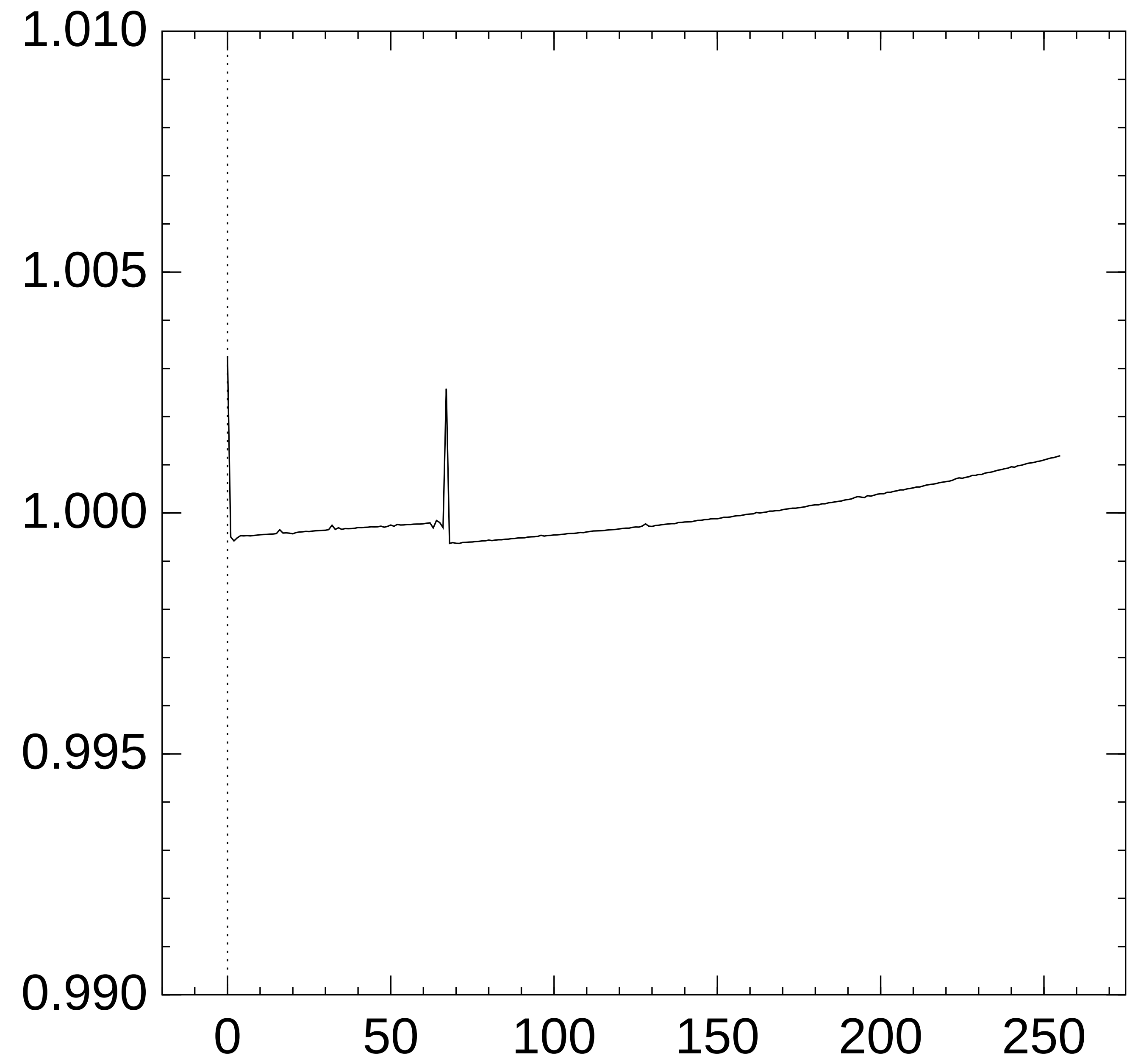
Graph of $256 \Pr[z_{66} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

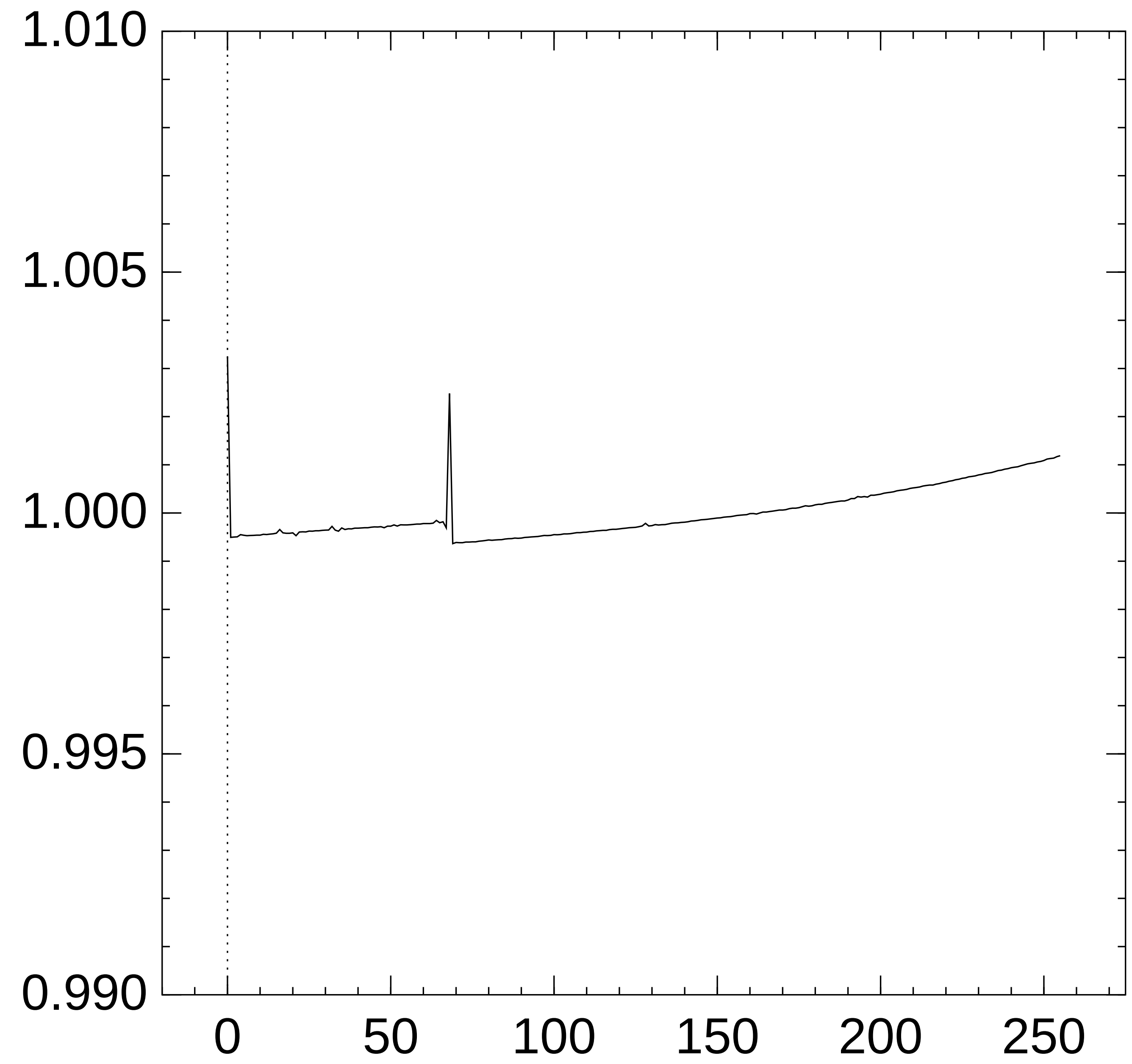
Graph of $256 \Pr[z_{67} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

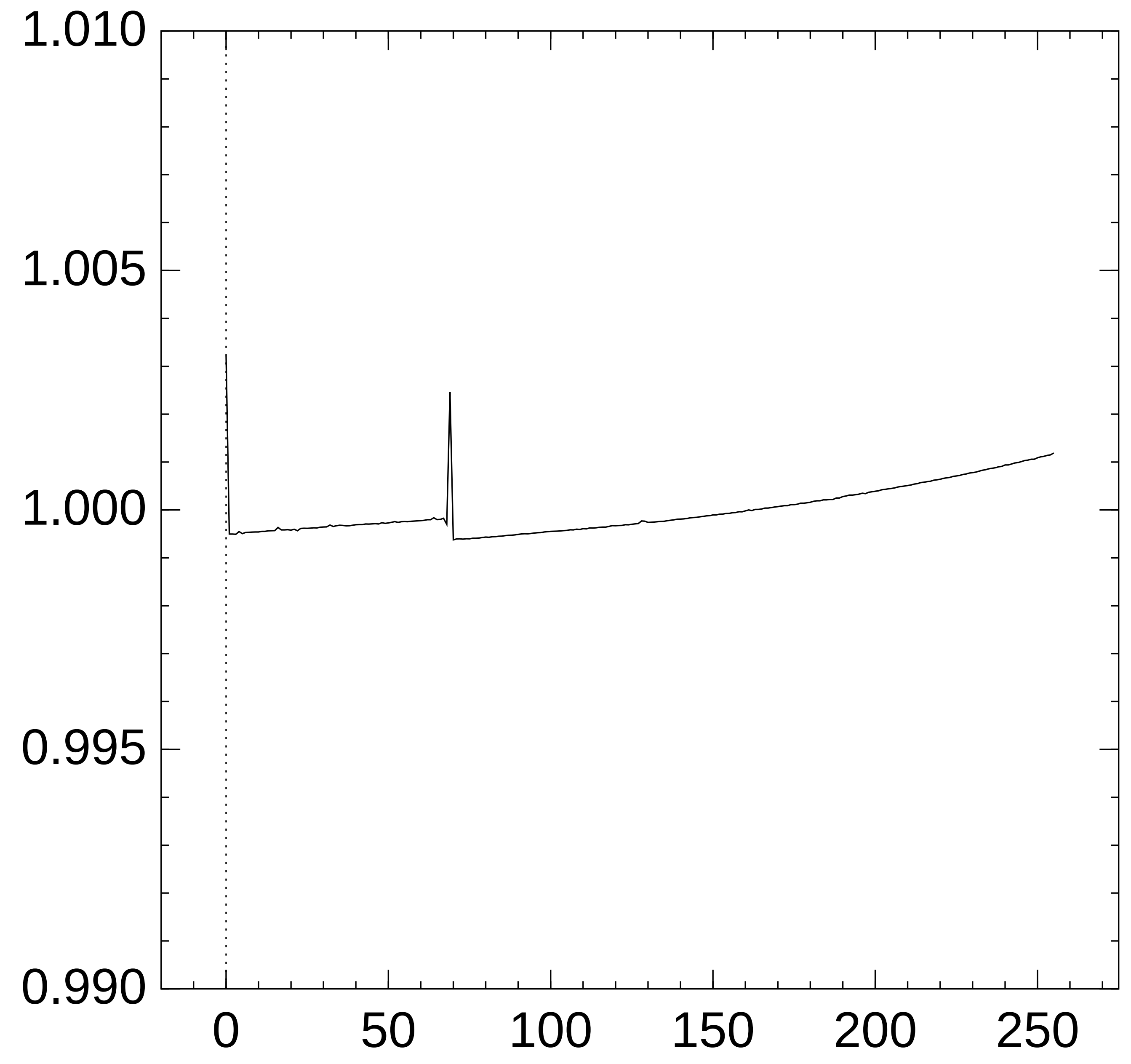
Graph of $256 \Pr[z_{68} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

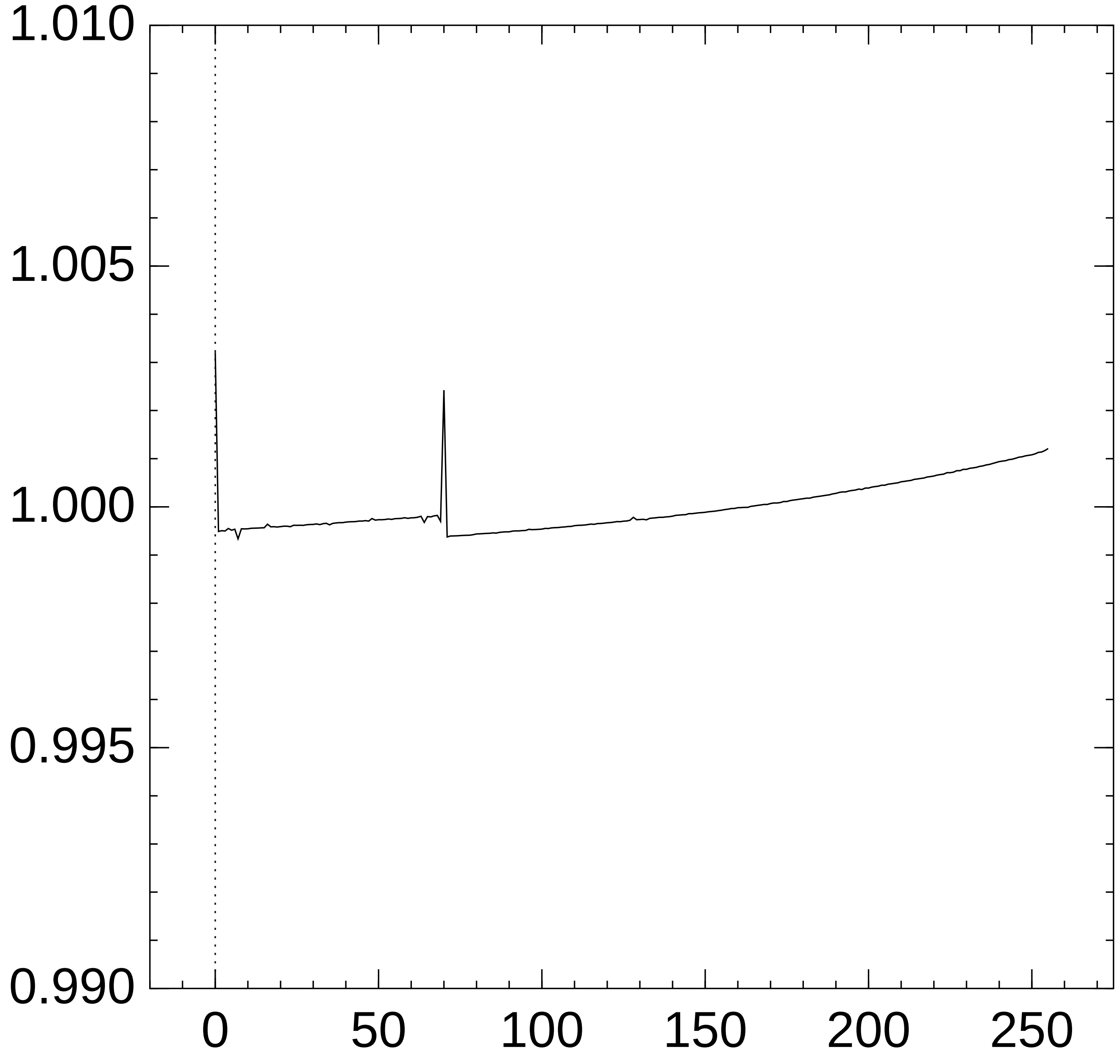
Graph of $256 \Pr[z_{69} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

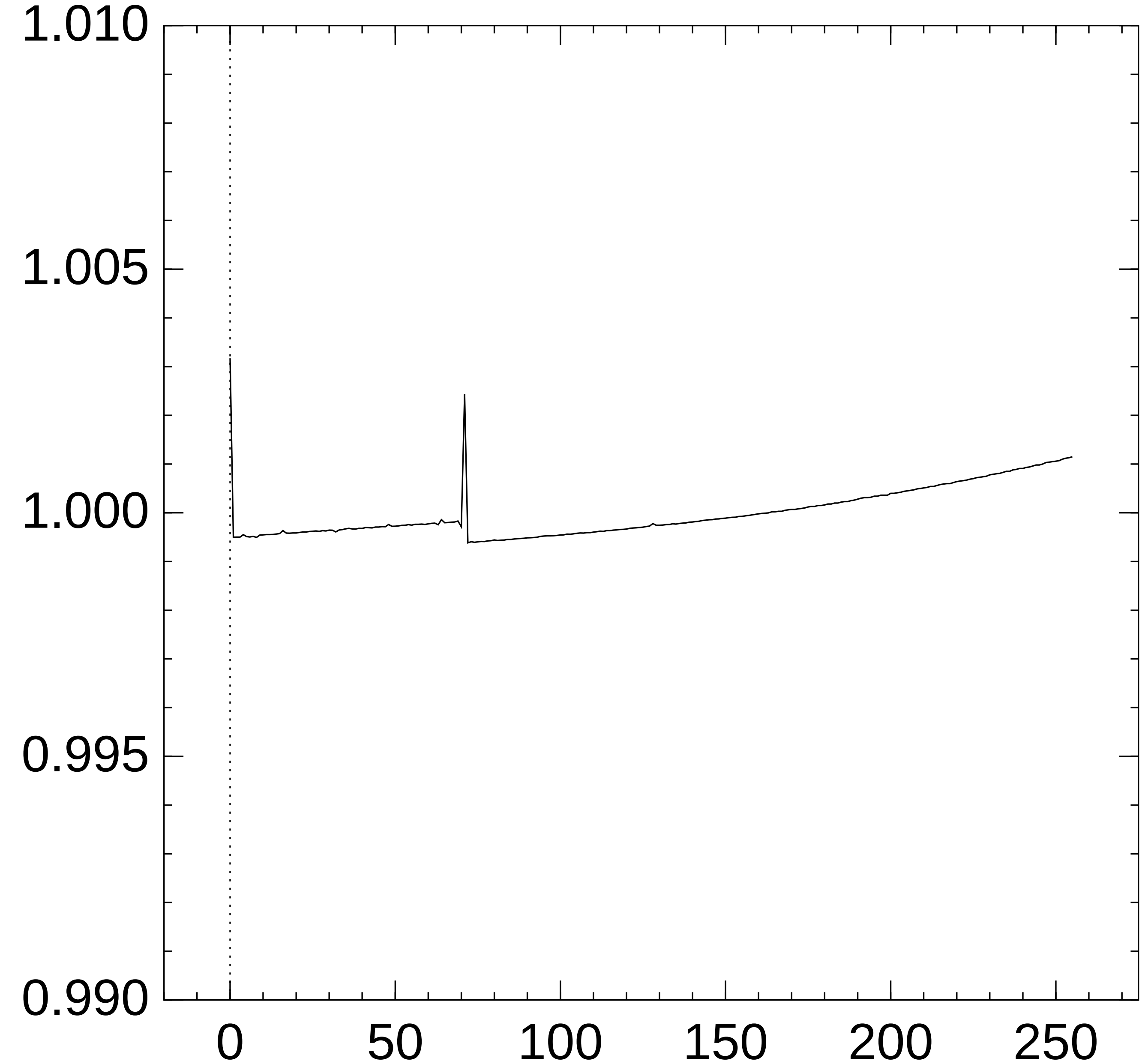
Graph of $256 \Pr[z_{70} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

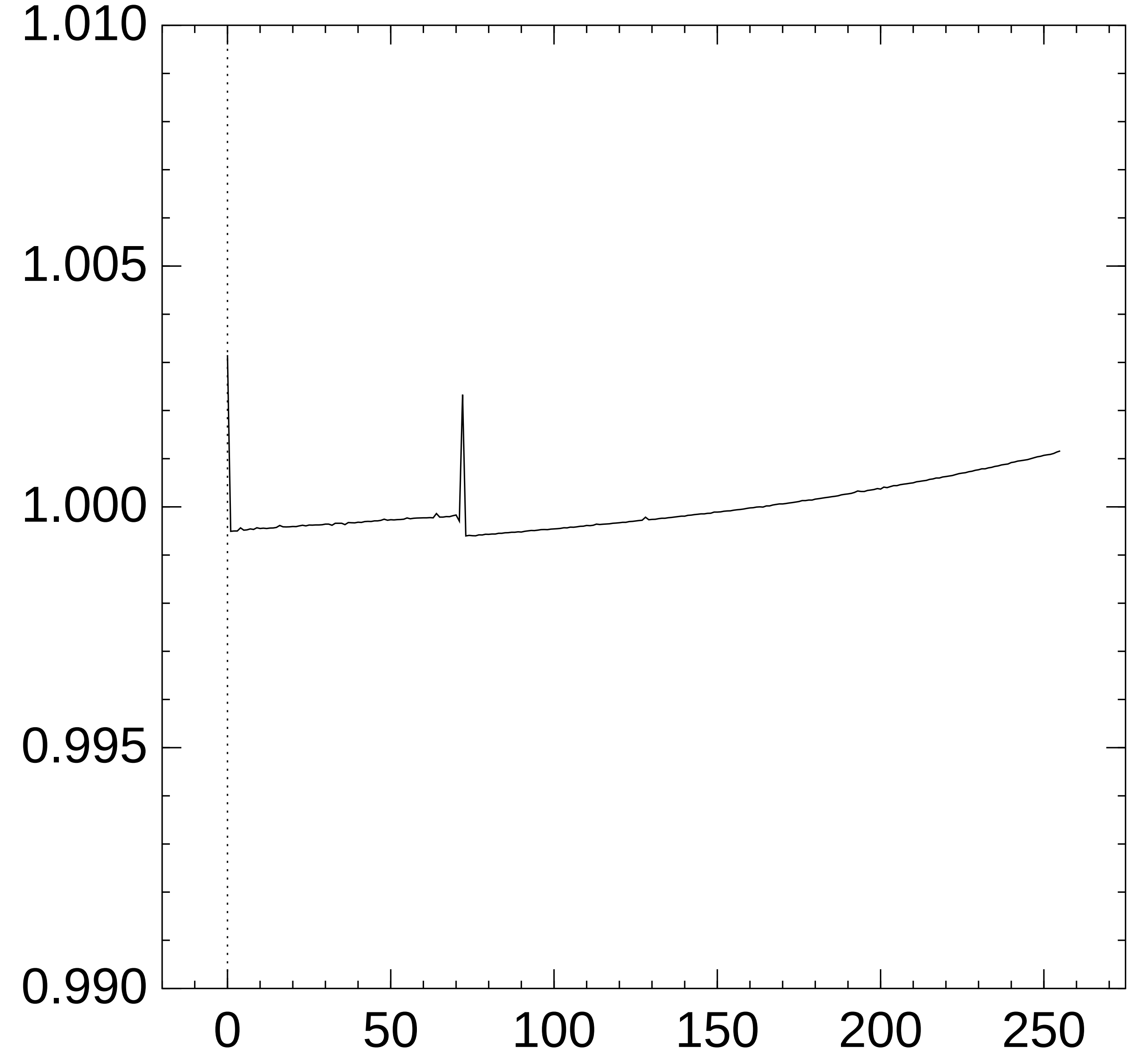
Graph of $256 \Pr[z_{71} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

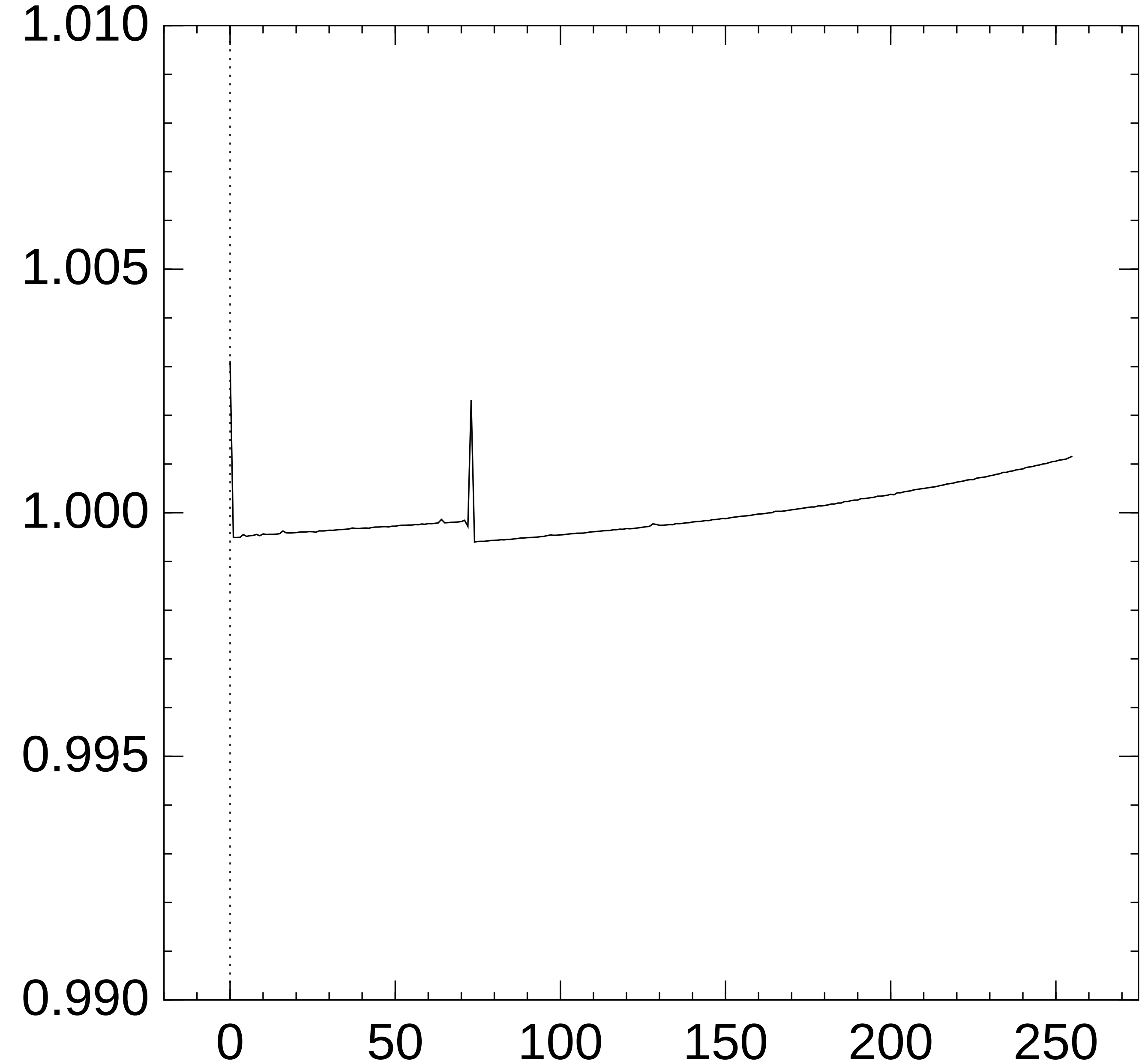
Graph of $256 \Pr[z_{72} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

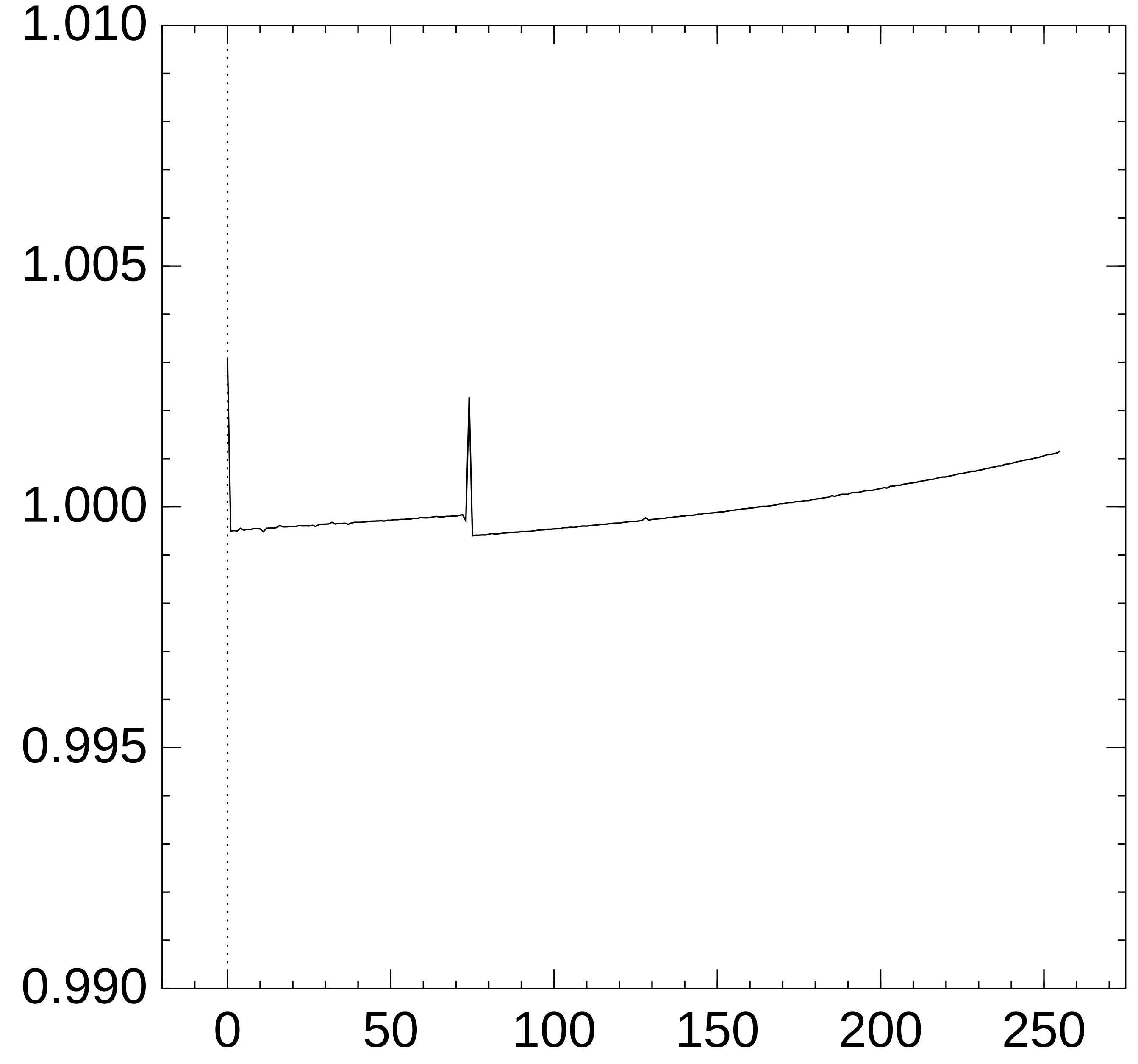
Graph of $256 \Pr[z_{73} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

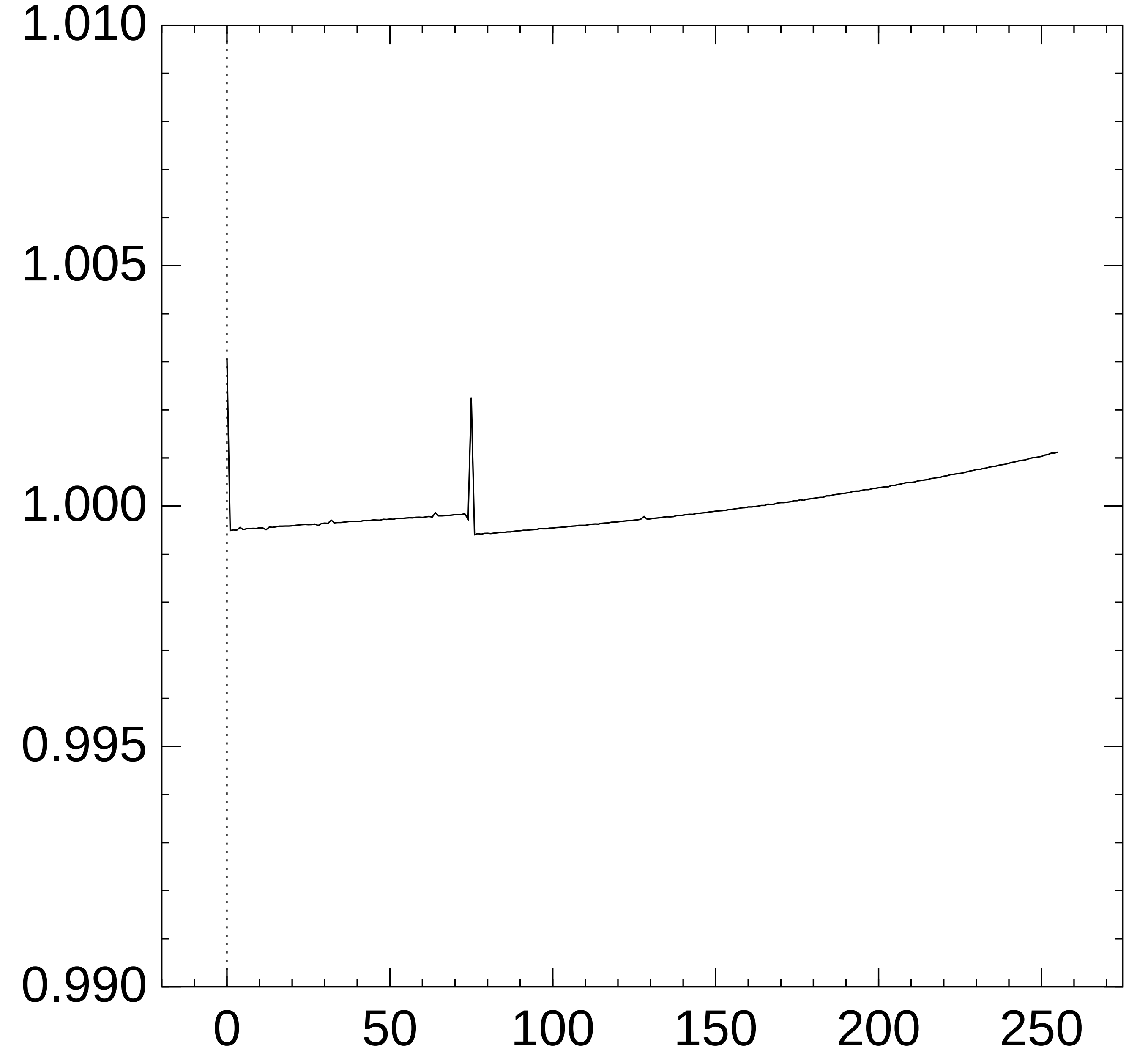
Graph of $256 \Pr[z_{74} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

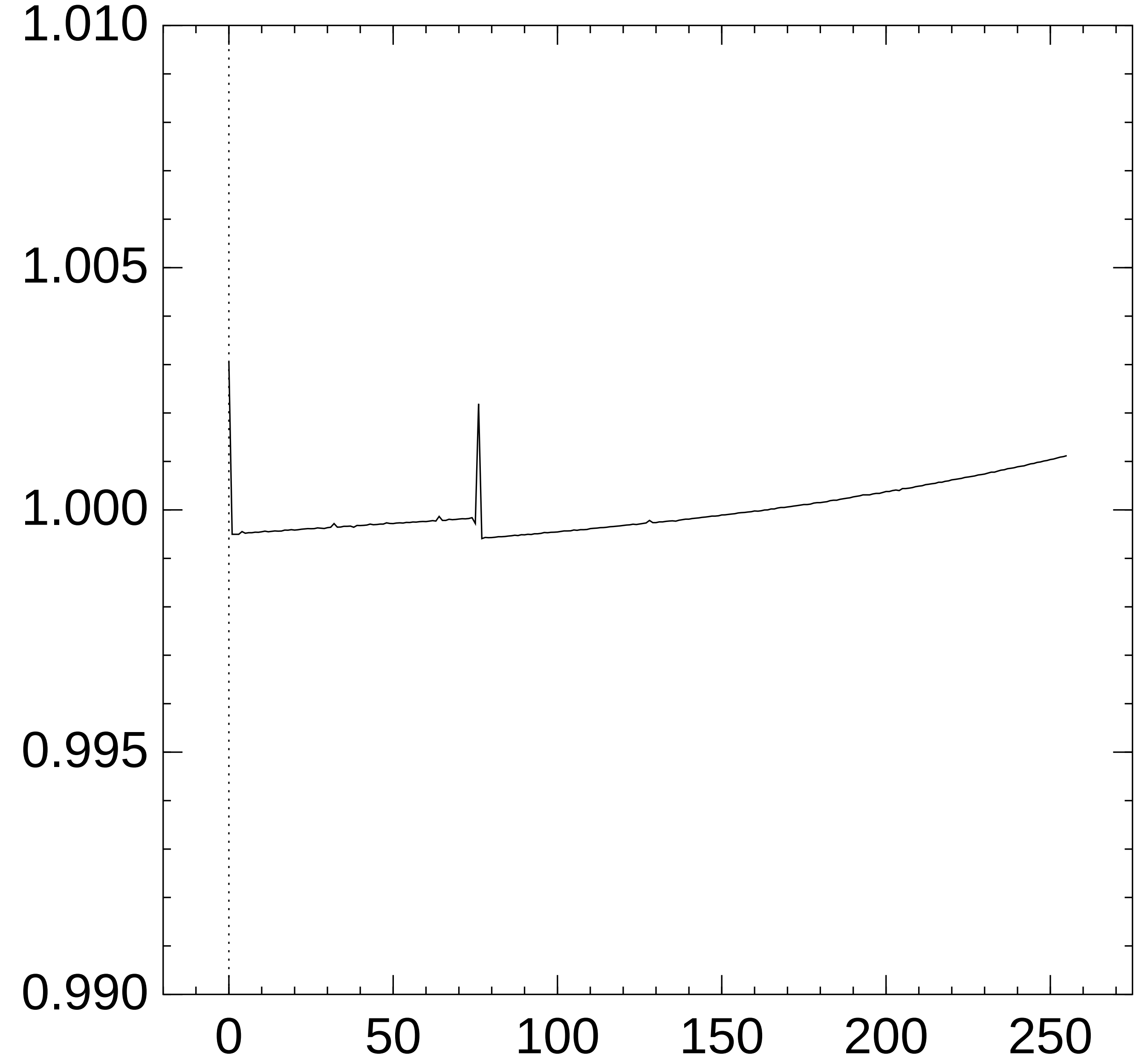
Graph of $256 \Pr[z_{75} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

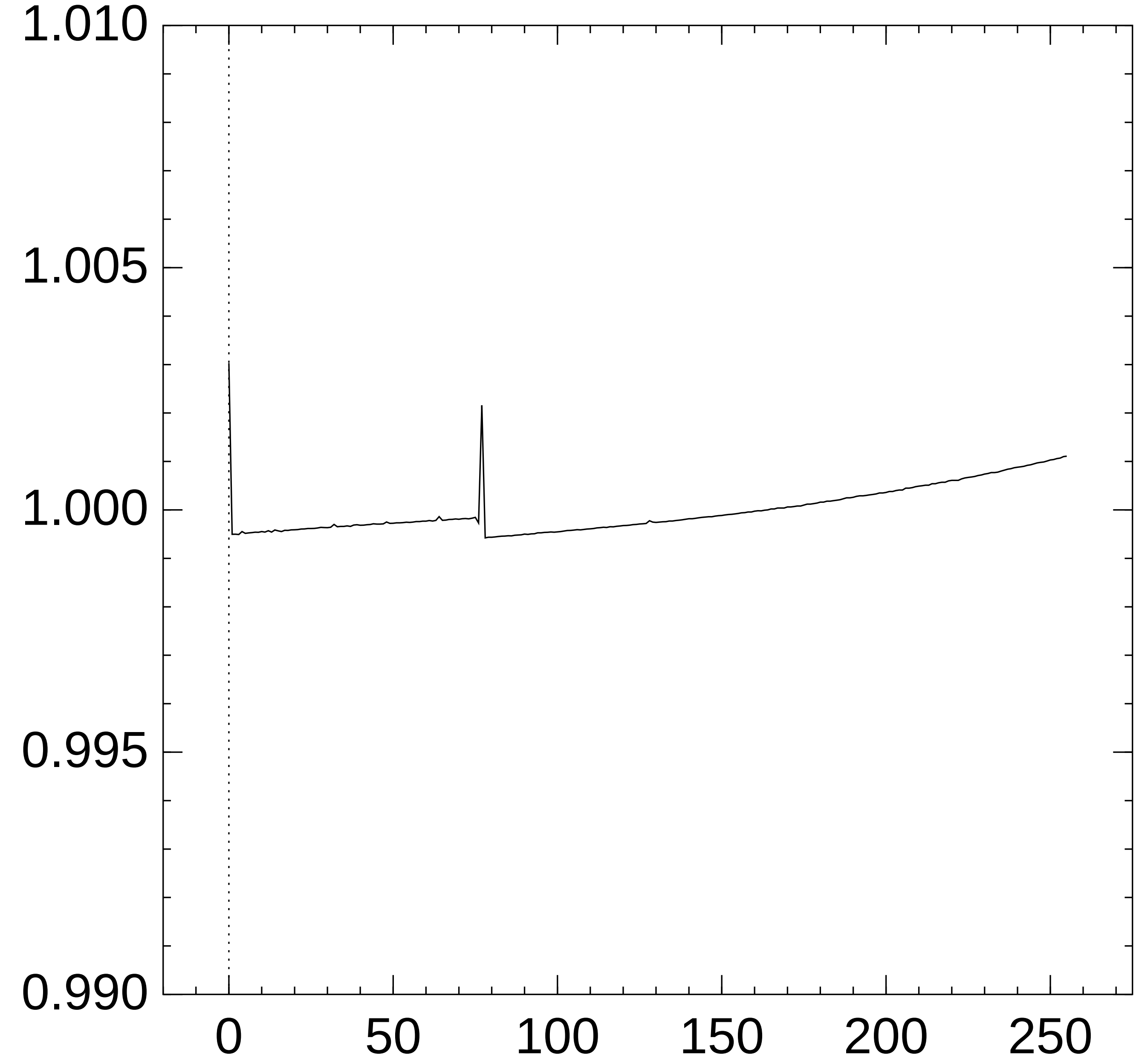
Graph of $256 \Pr[z_{76} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

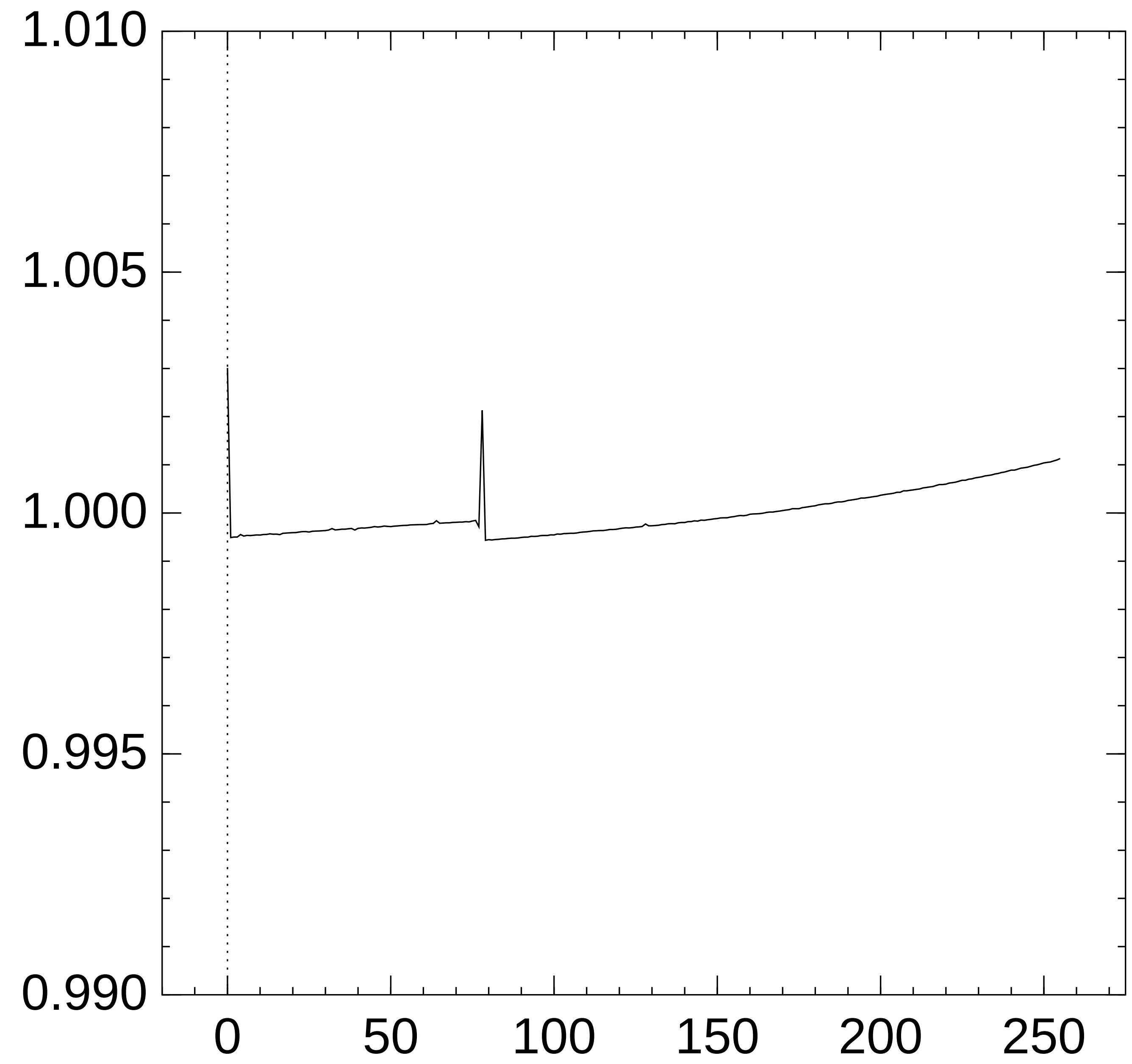
Graph of $256 \Pr[z_{77} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

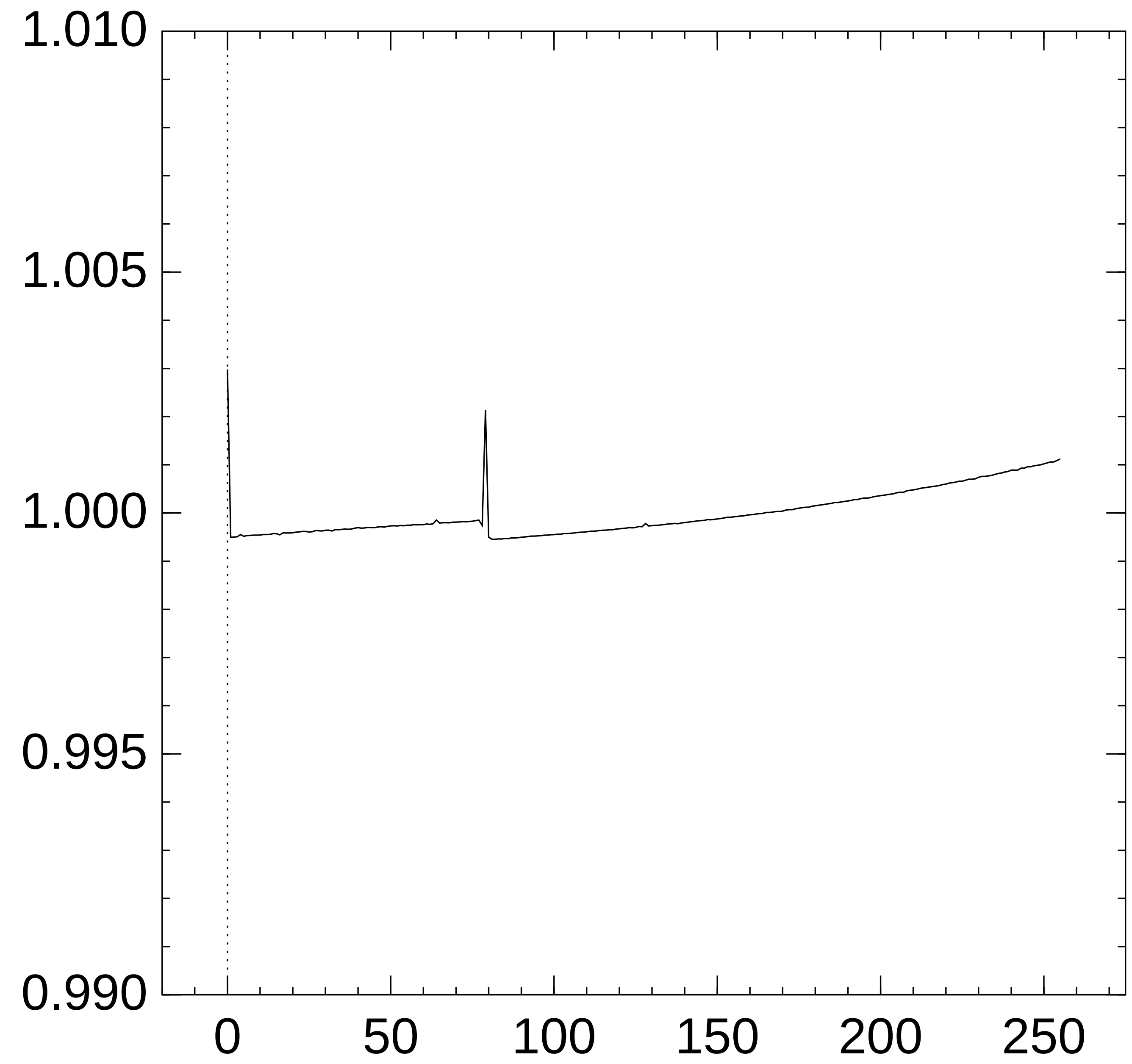
Graph of $256 \Pr[z_{78} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

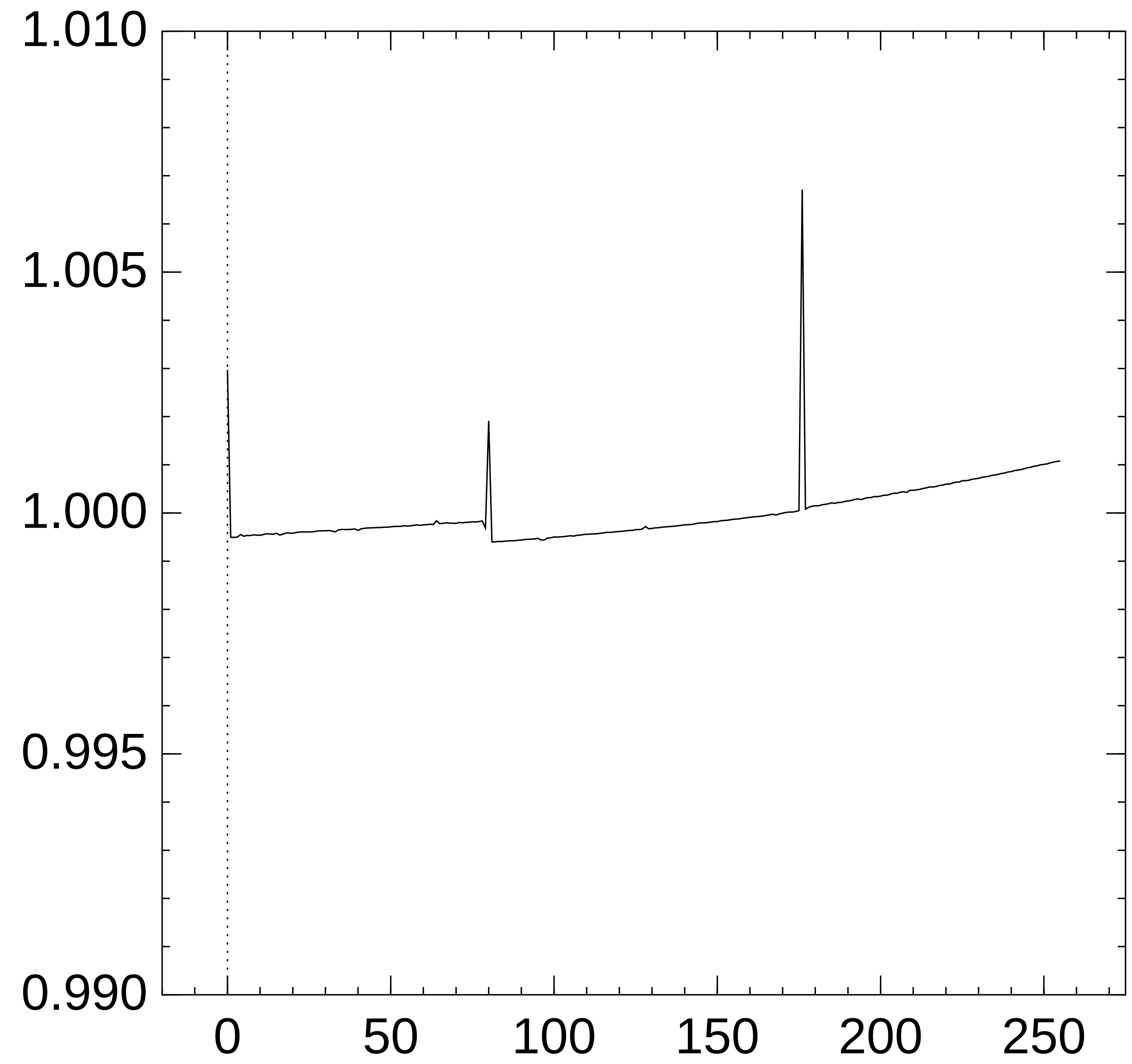
Graph of $256 \Pr[z_{79} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

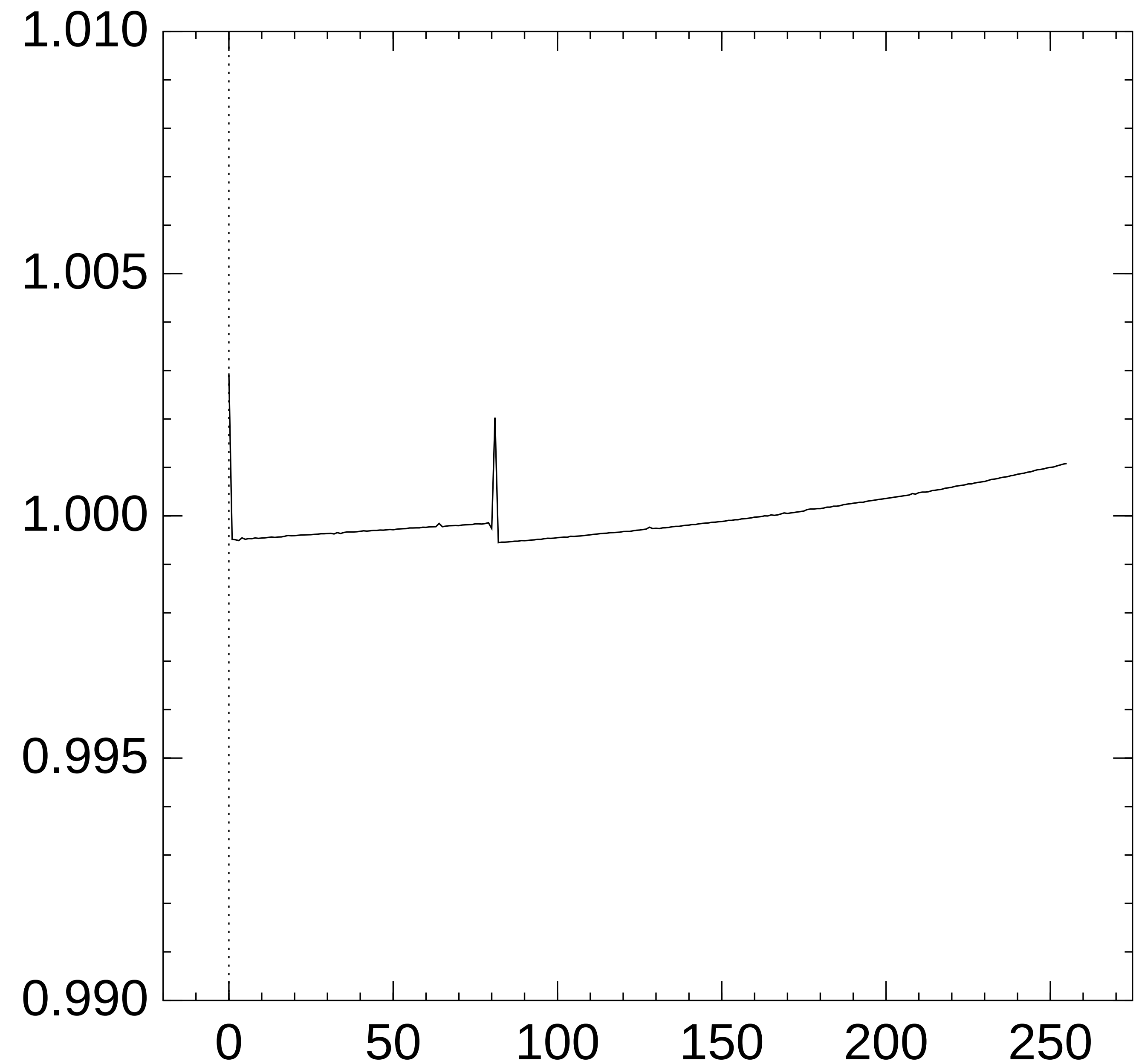
Graph of $256 \Pr[z_{80} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

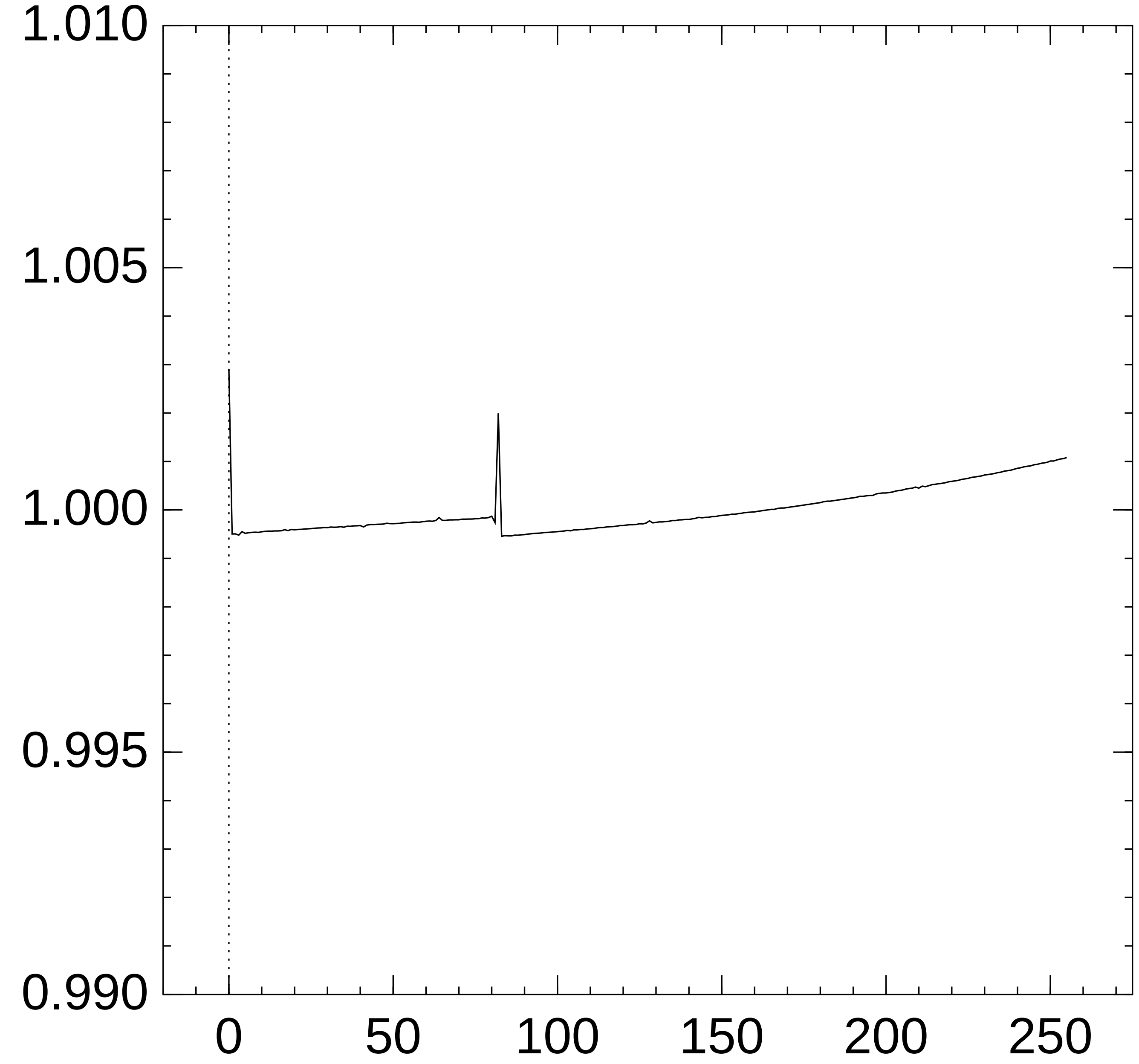
Graph of $256 \Pr[z_{81} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

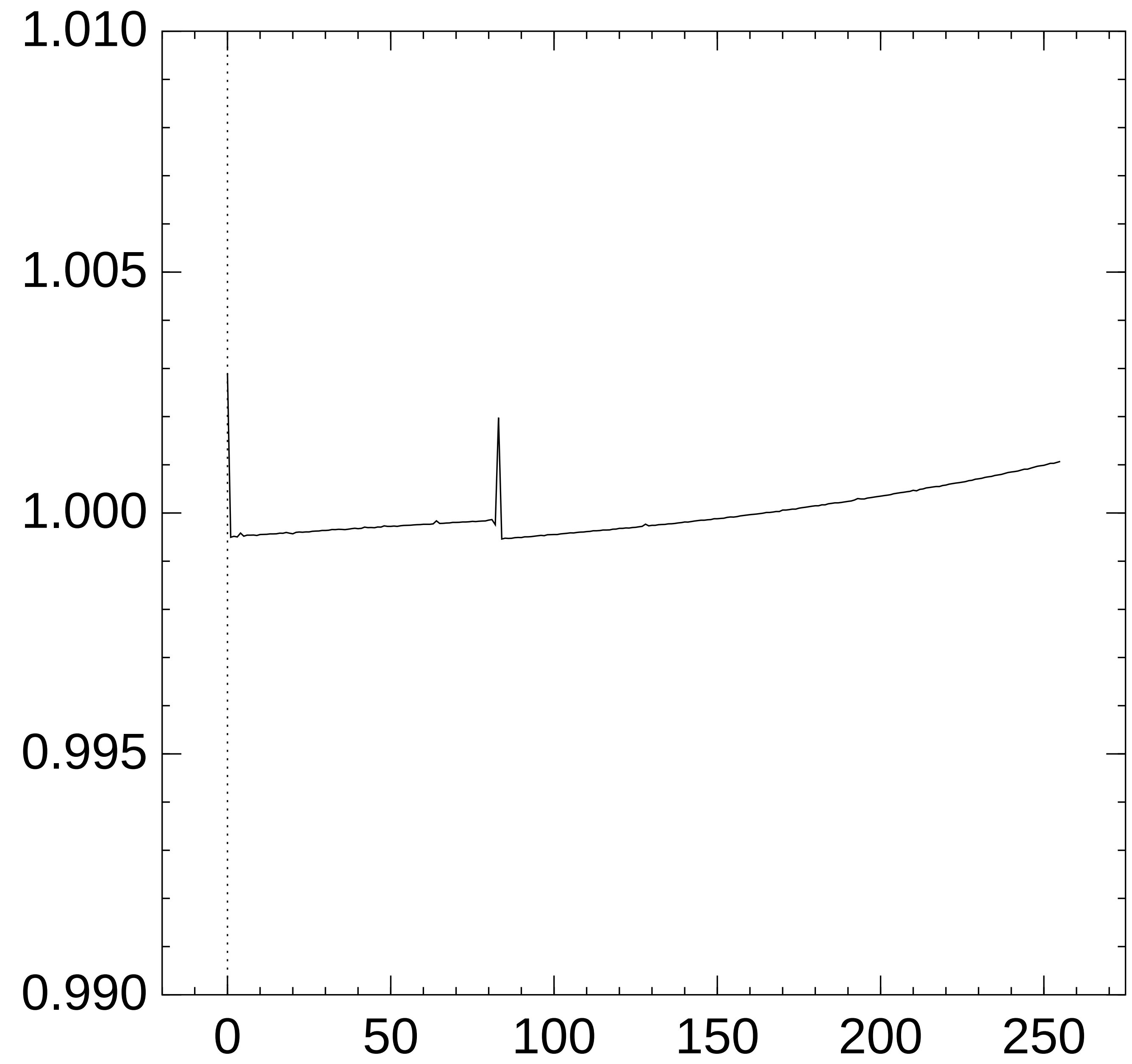
Graph of $256 \Pr[z_{82} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

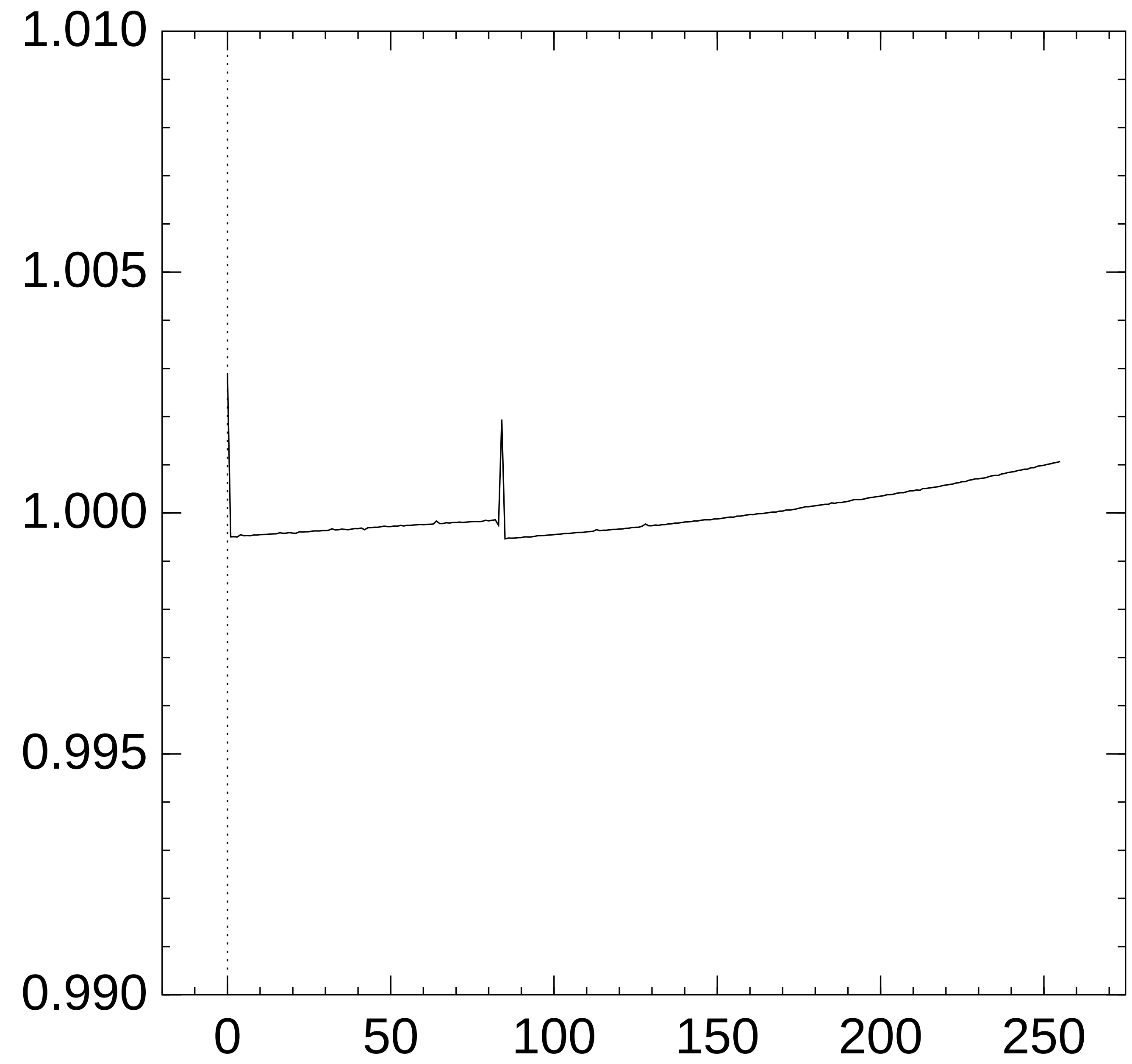
Graph of $256 \Pr[z_{83} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

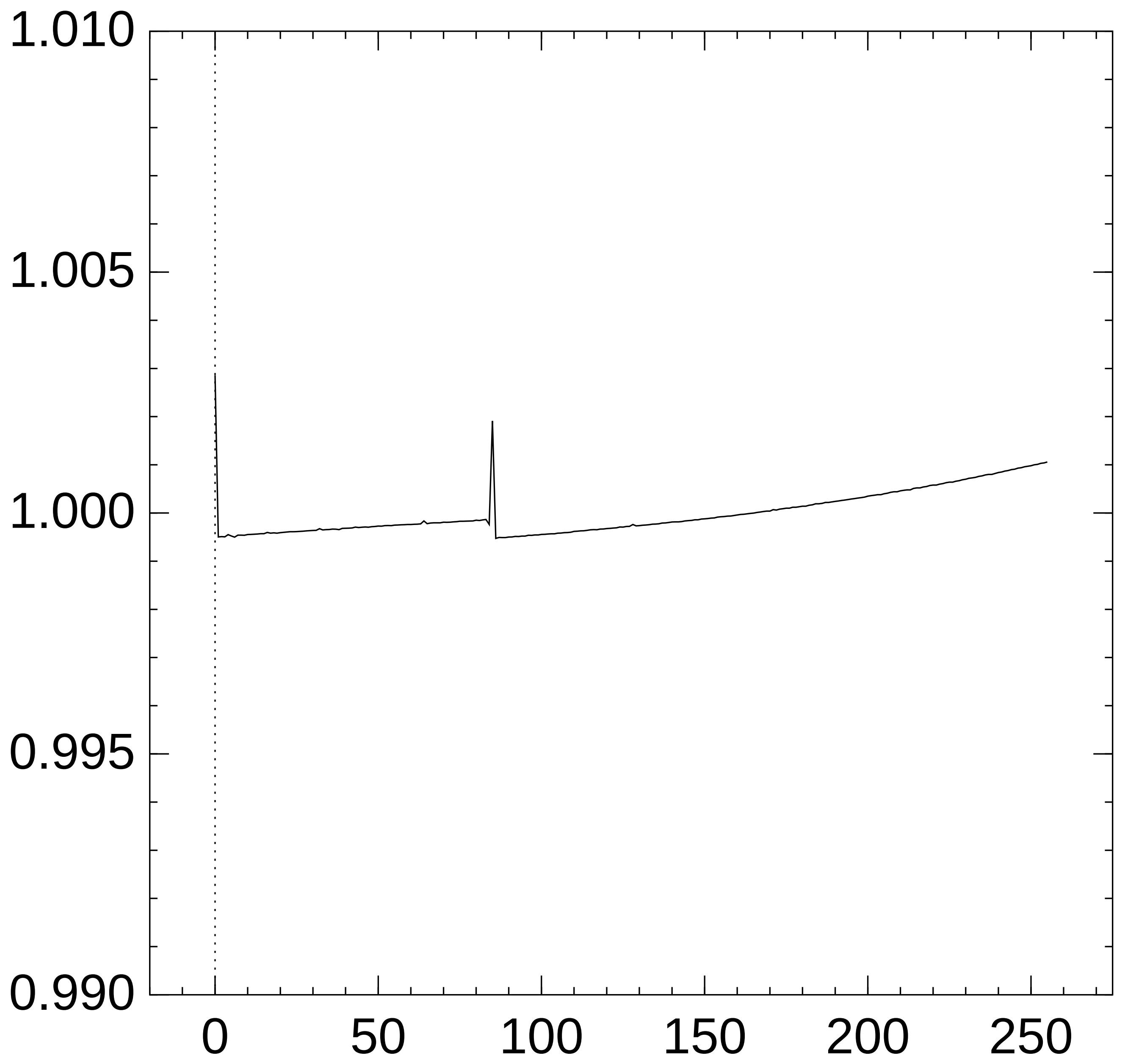
Graph of $256 \Pr[z_{84} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

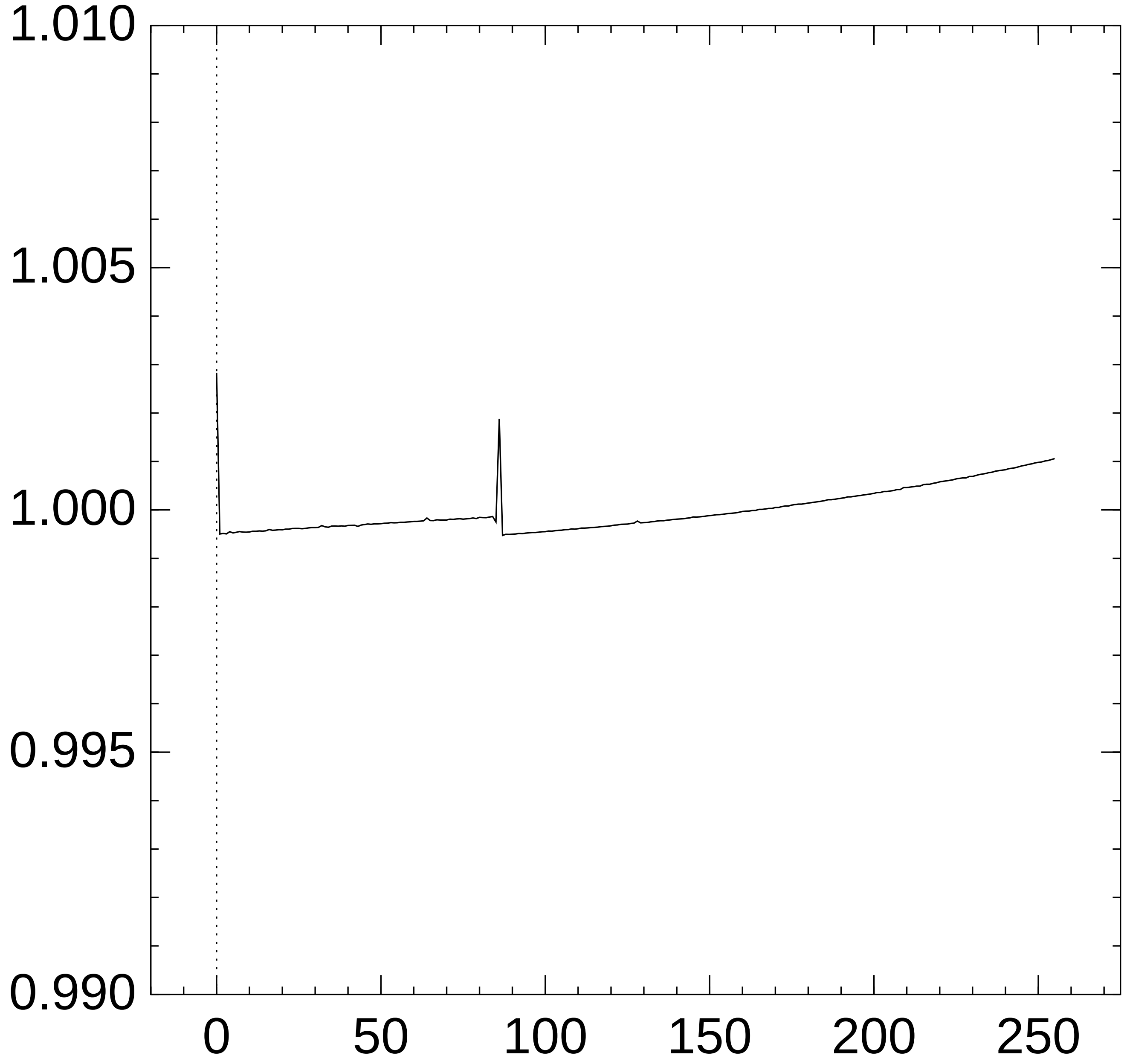
Graph of $256 \Pr[z_{85} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

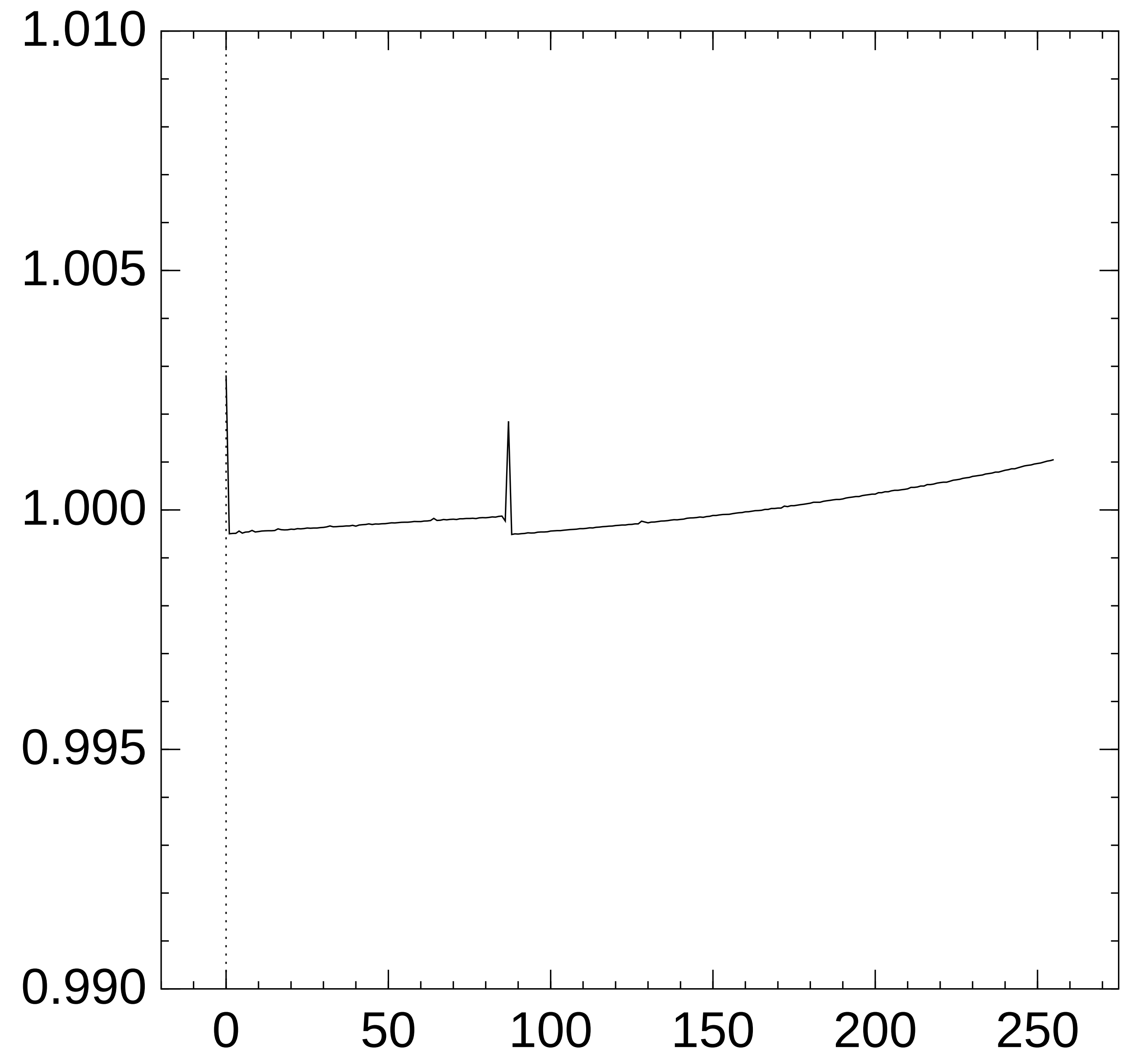
Graph of $256 \Pr[z_{86} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

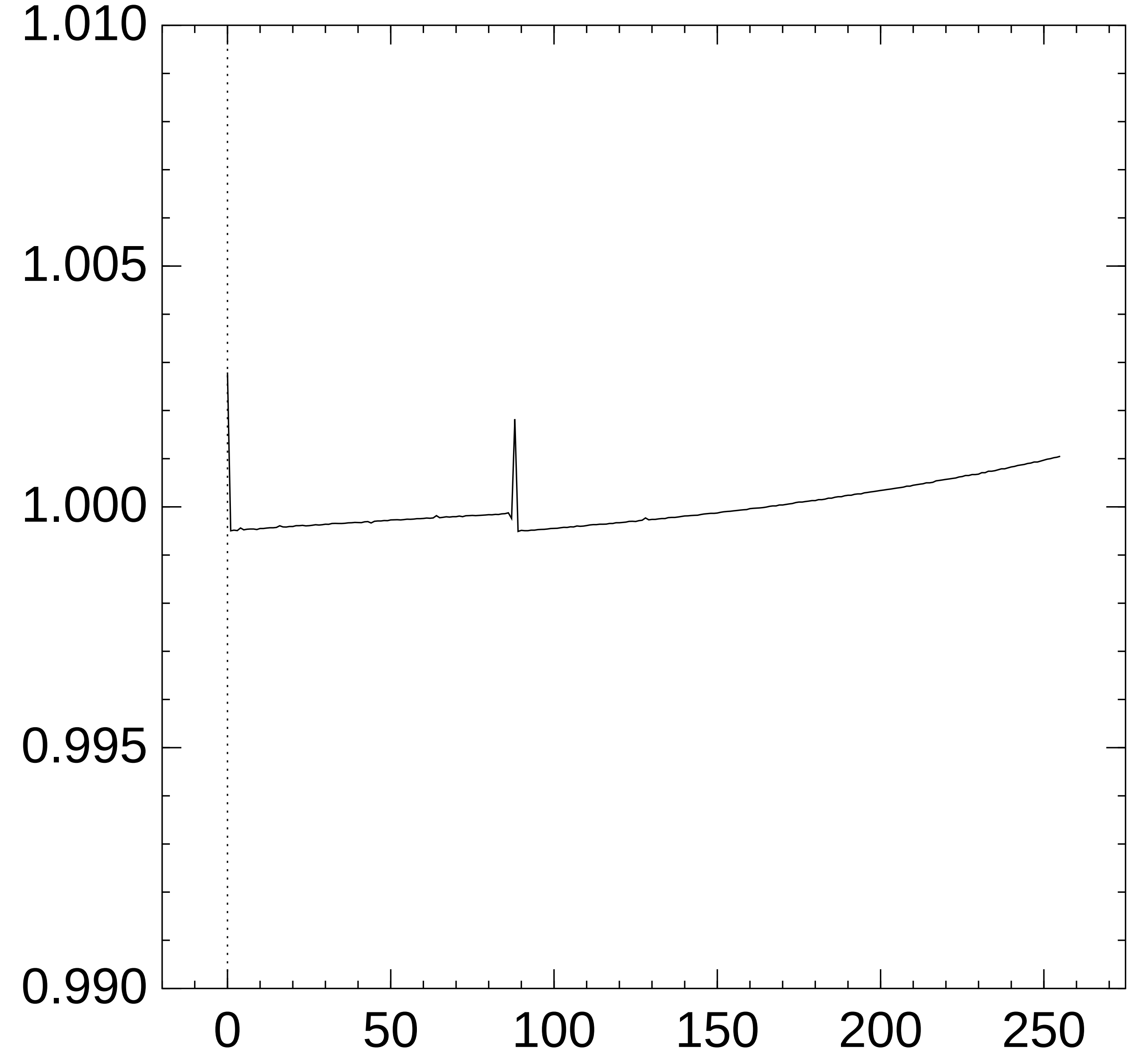
Graph of $256 \Pr[z_{87} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

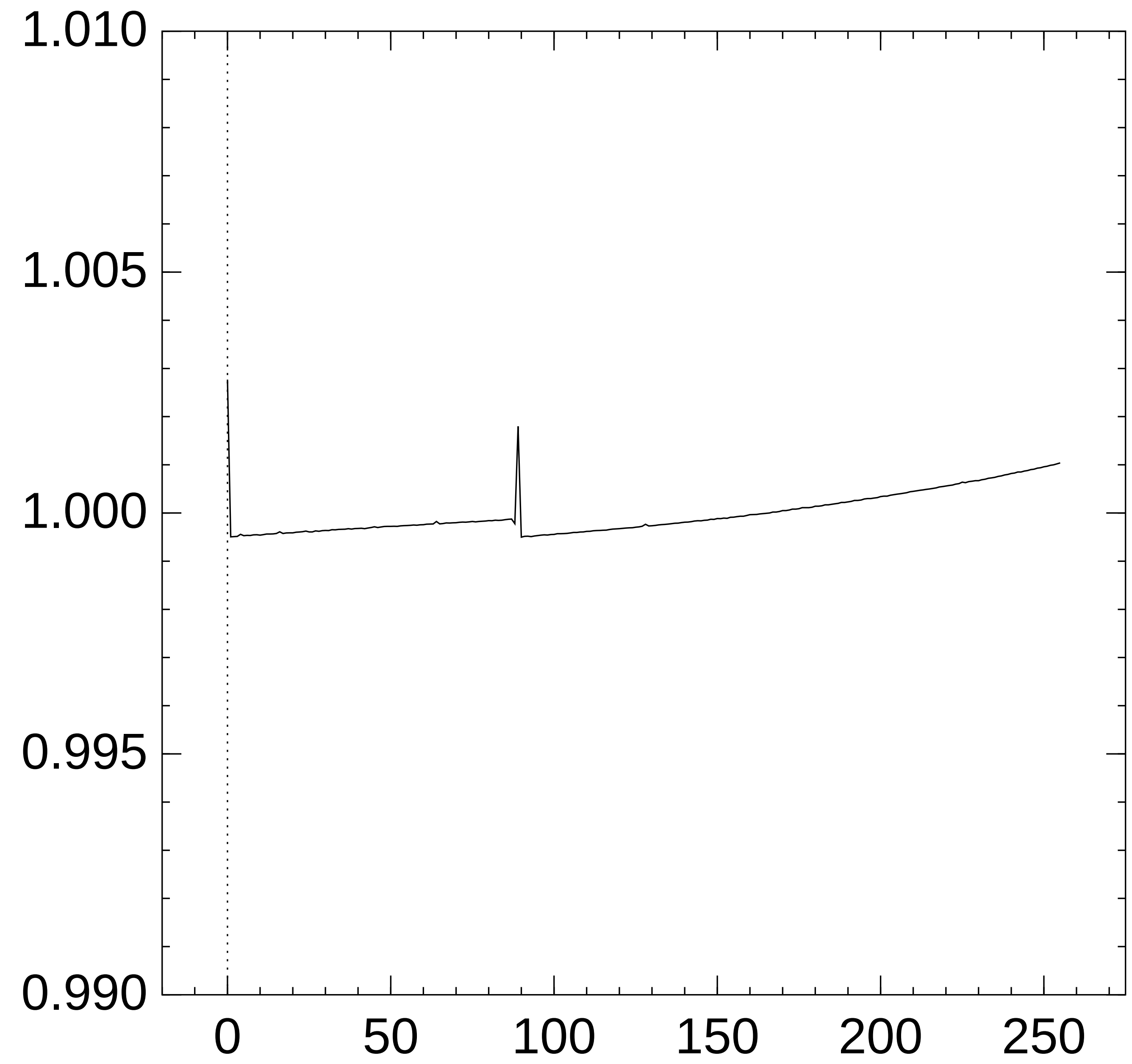
Graph of $256 \Pr[z_{88} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

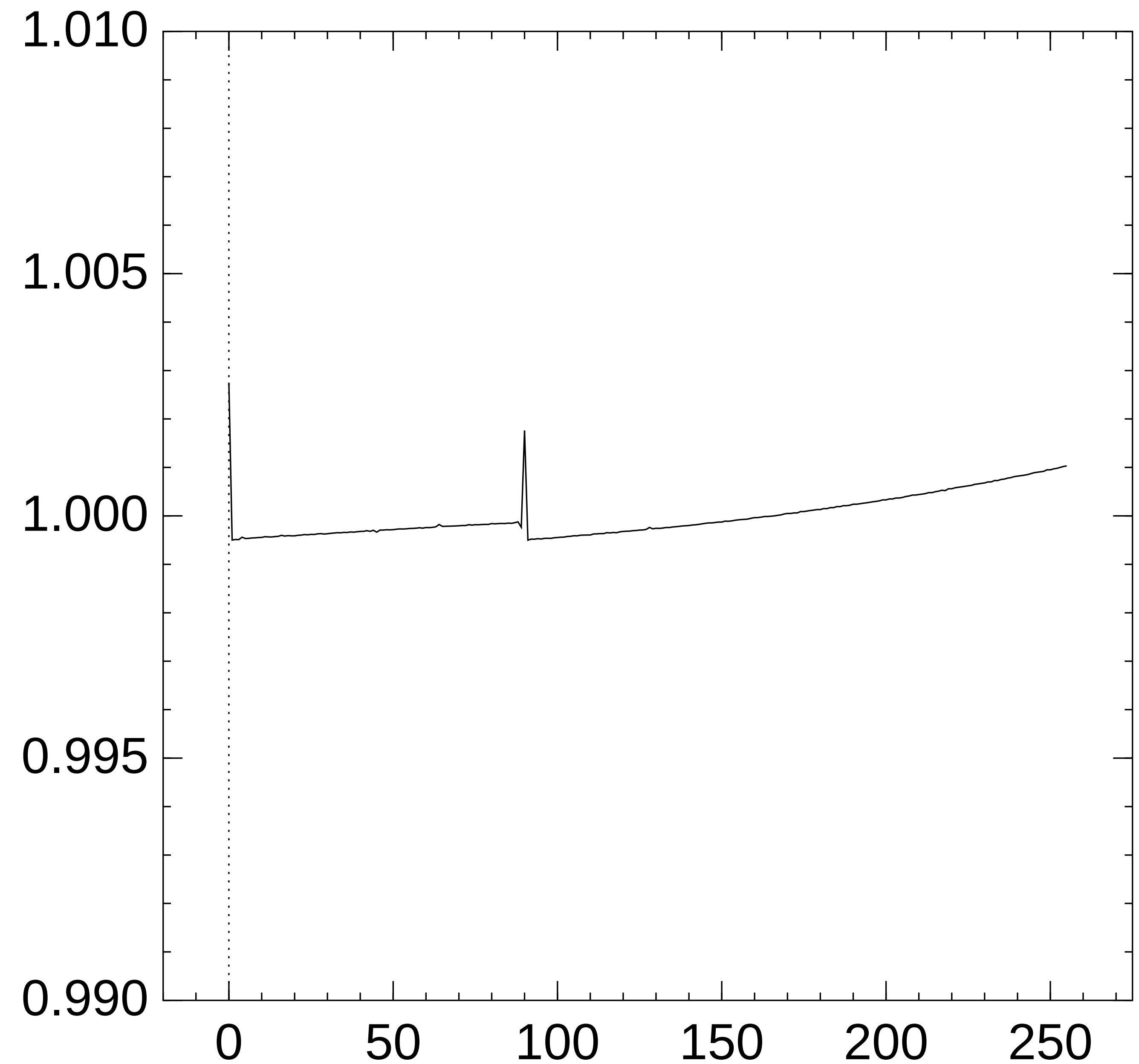
Graph of $256 \Pr[z_{89} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

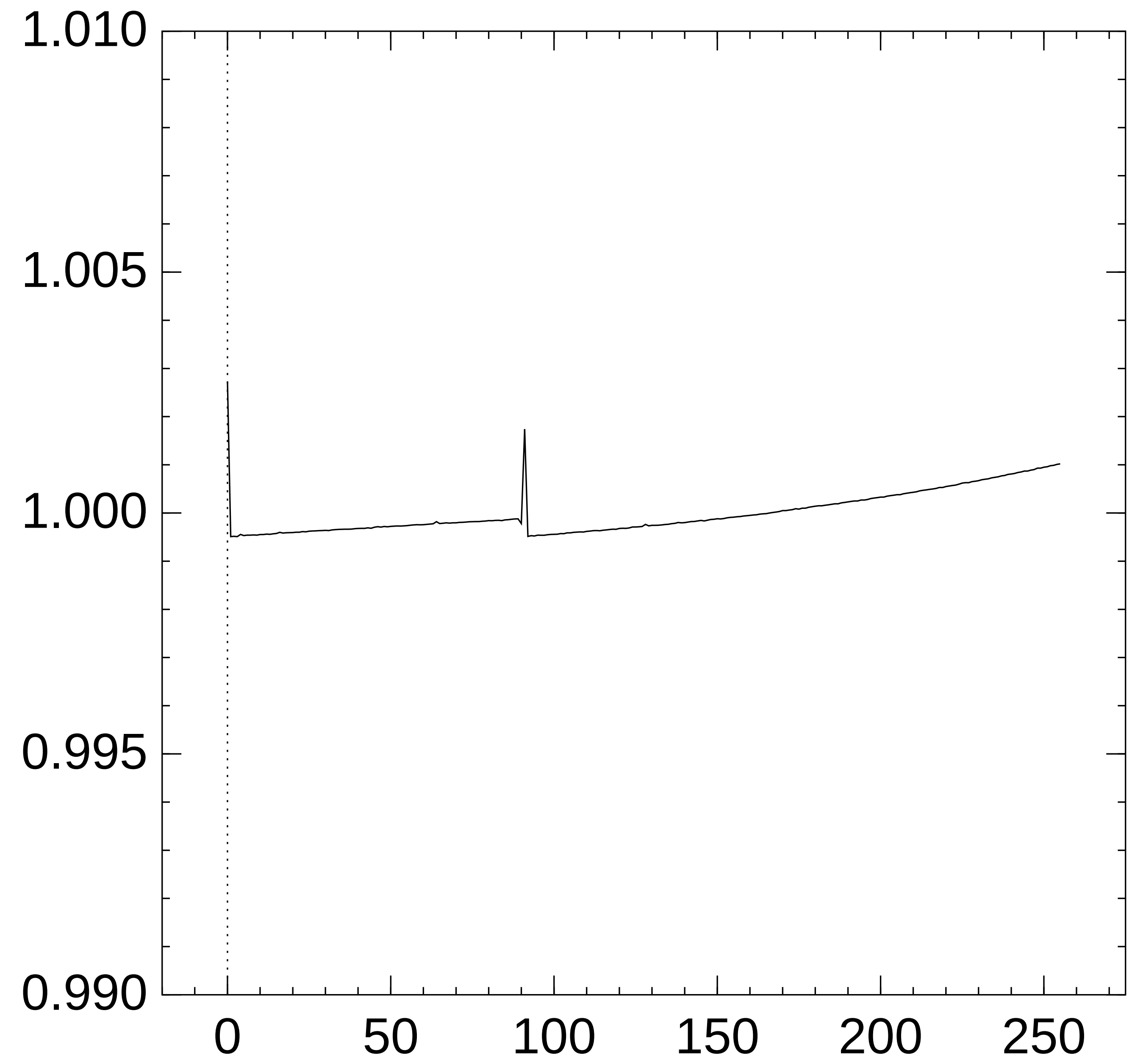
Graph of $256\,\Pr[z_{90} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

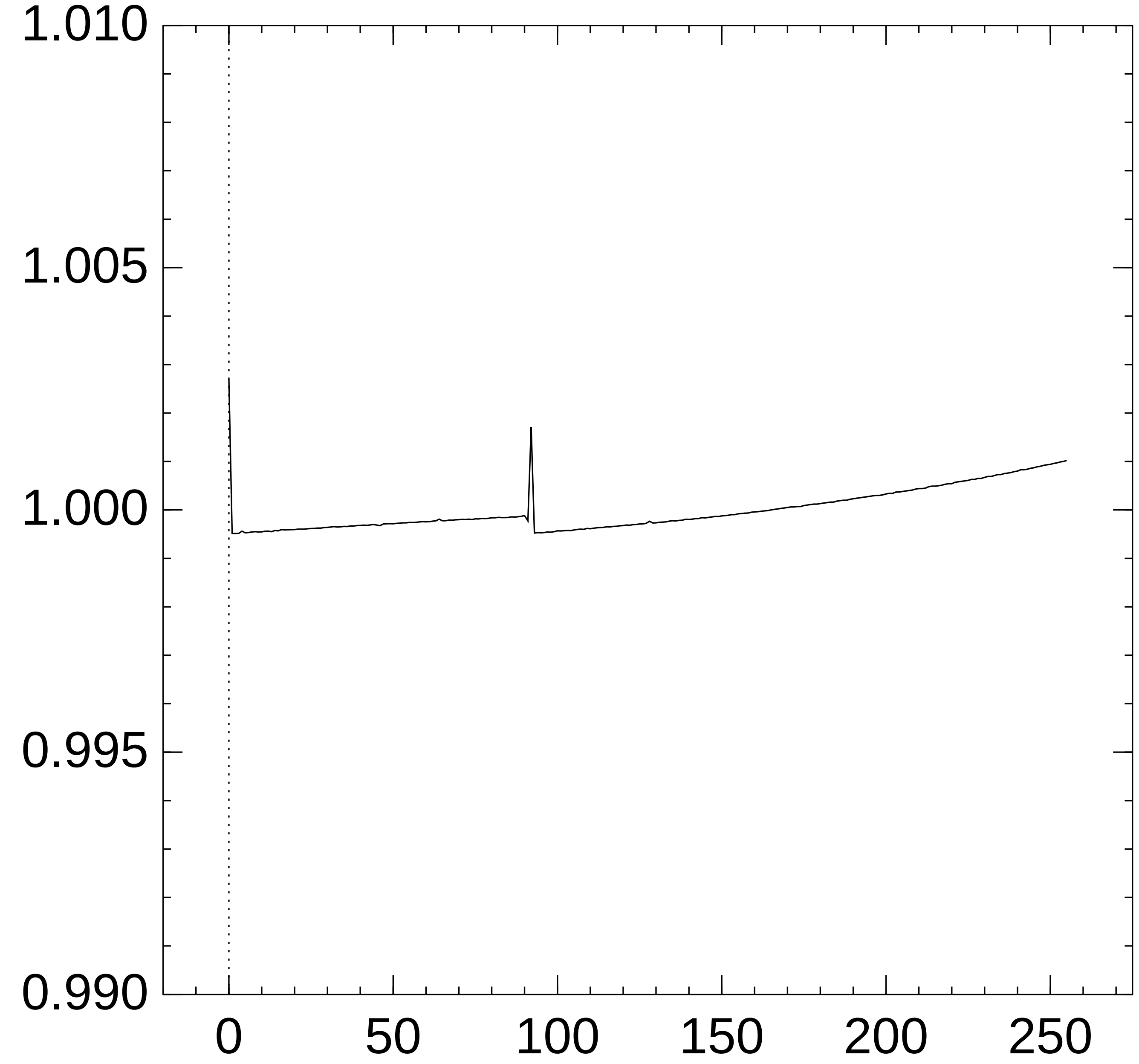
Graph of $256 \Pr[z_{91} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

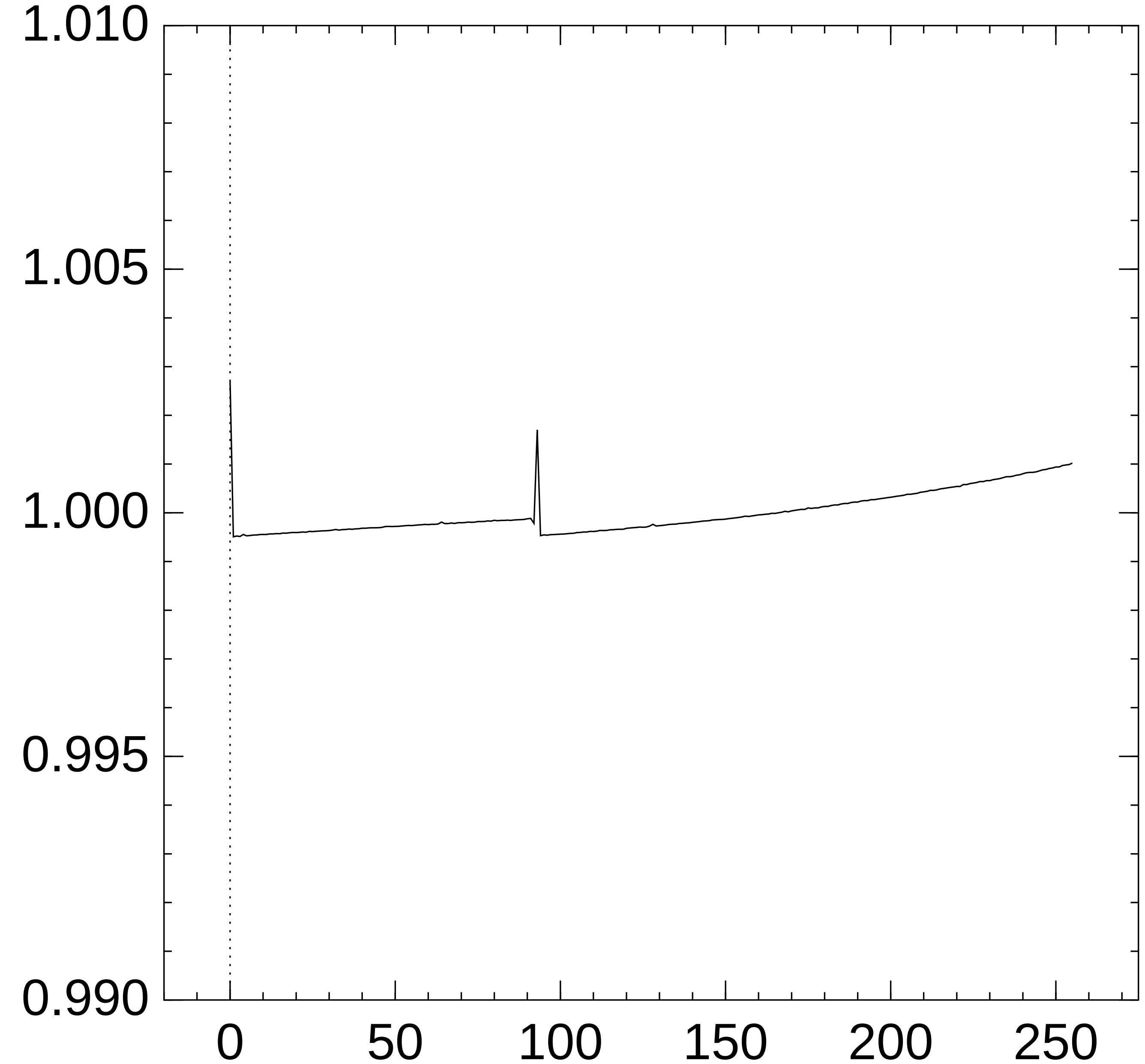
Graph of $256\,\Pr[z_{92} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

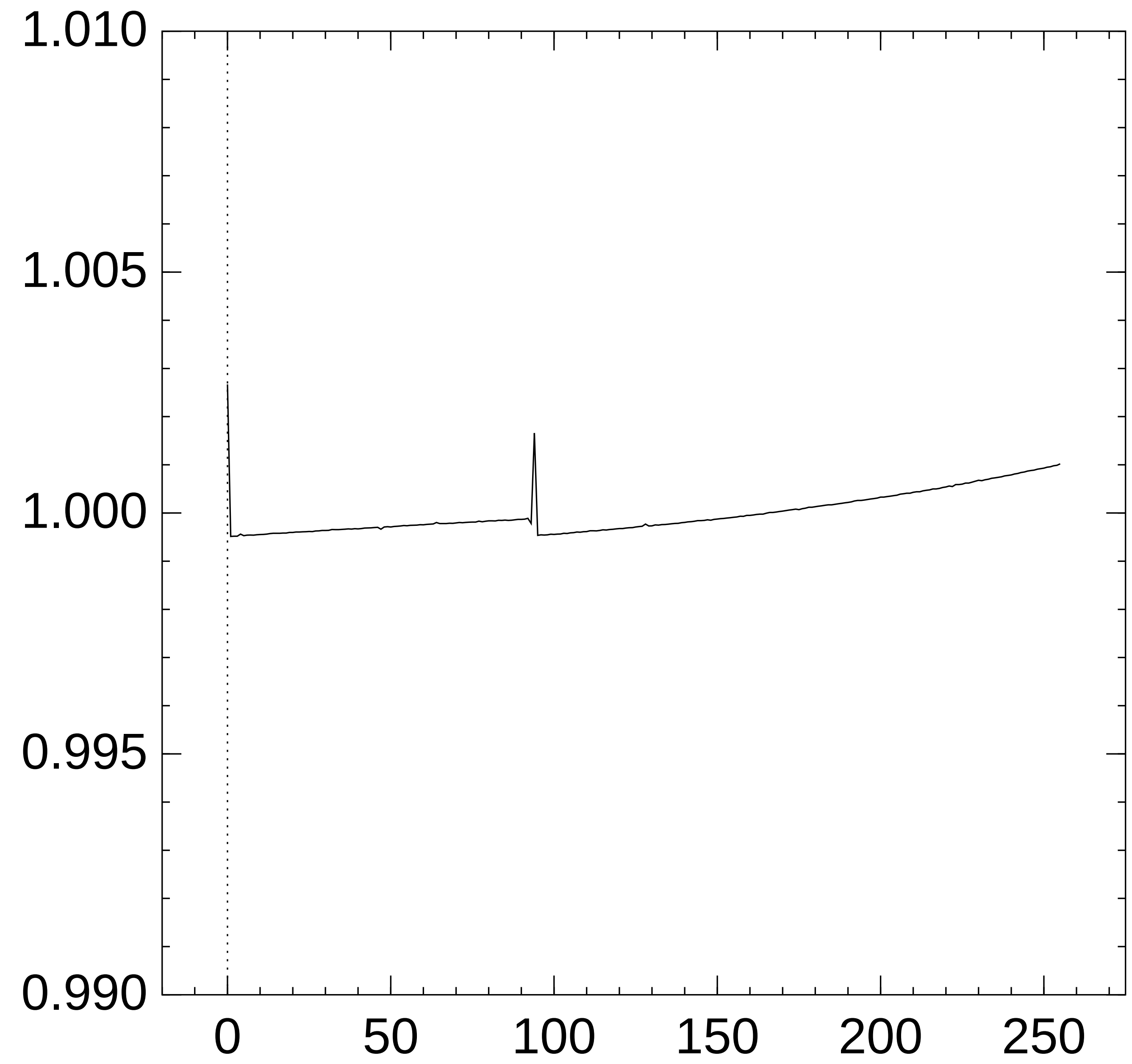
Graph of $256 \Pr[z_{93} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

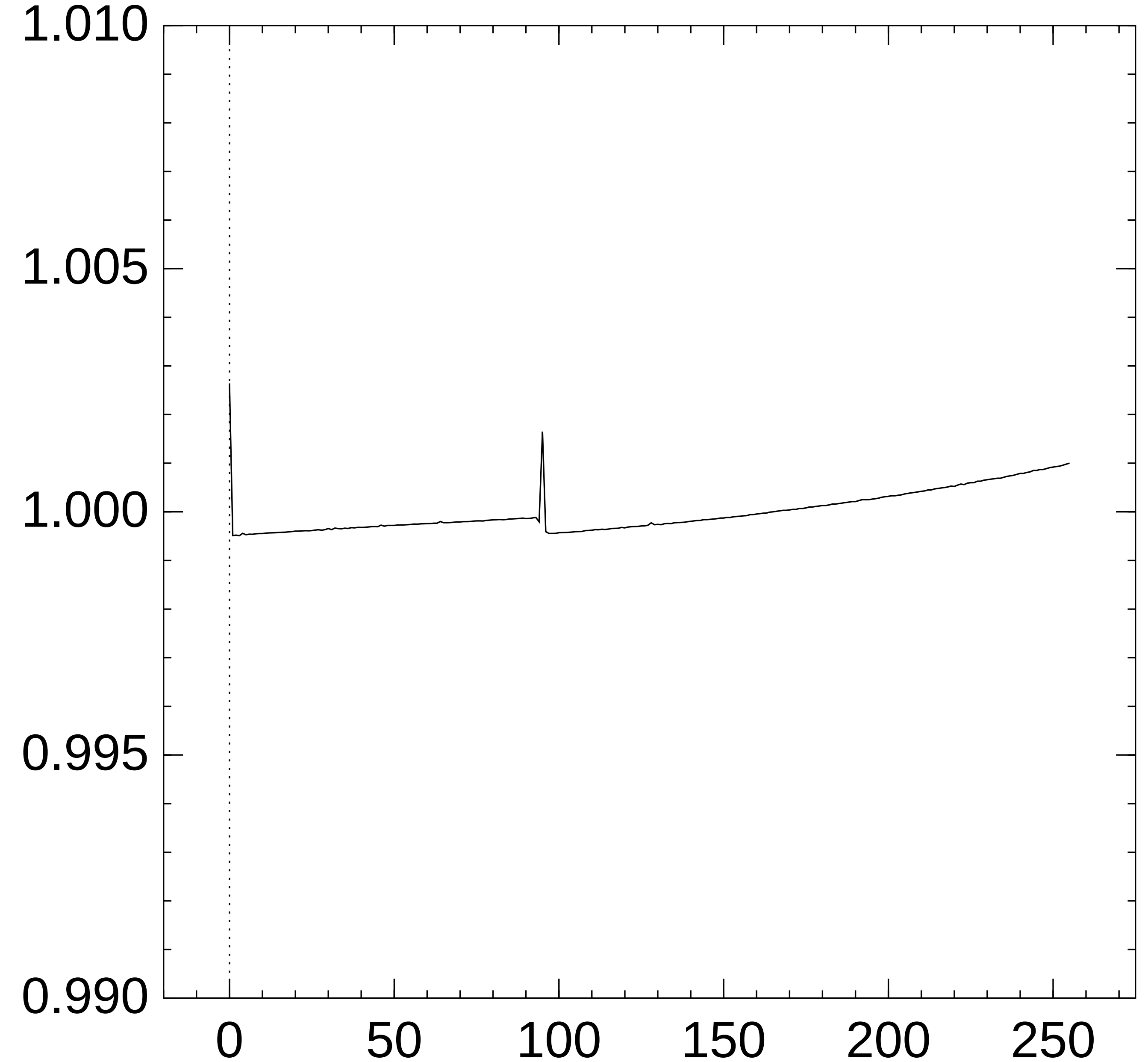
Graph of $256\,\Pr[z_{94} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

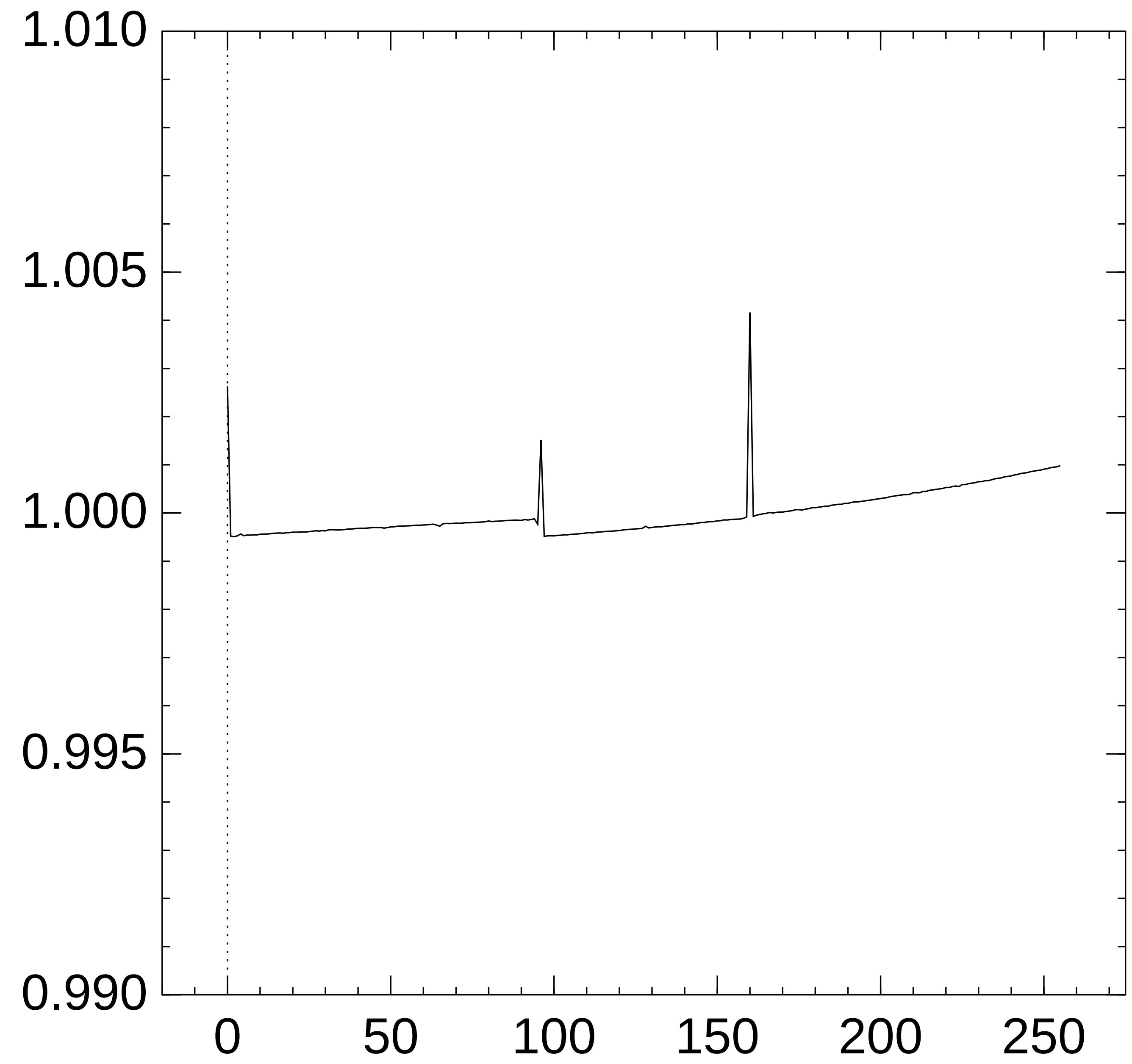
Graph of $256 \Pr[z_{95} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

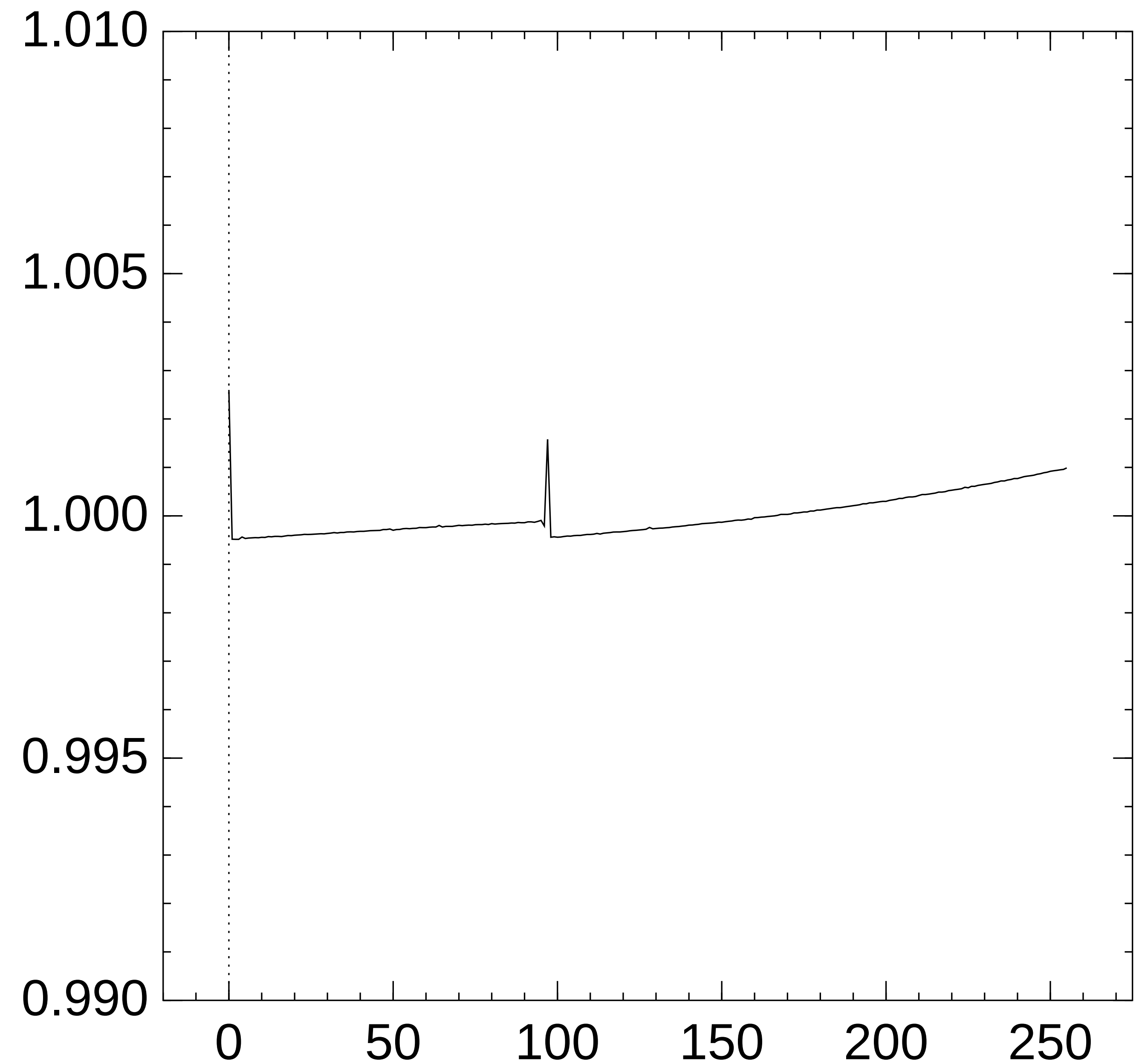
Graph of $256 \Pr[z_{96} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

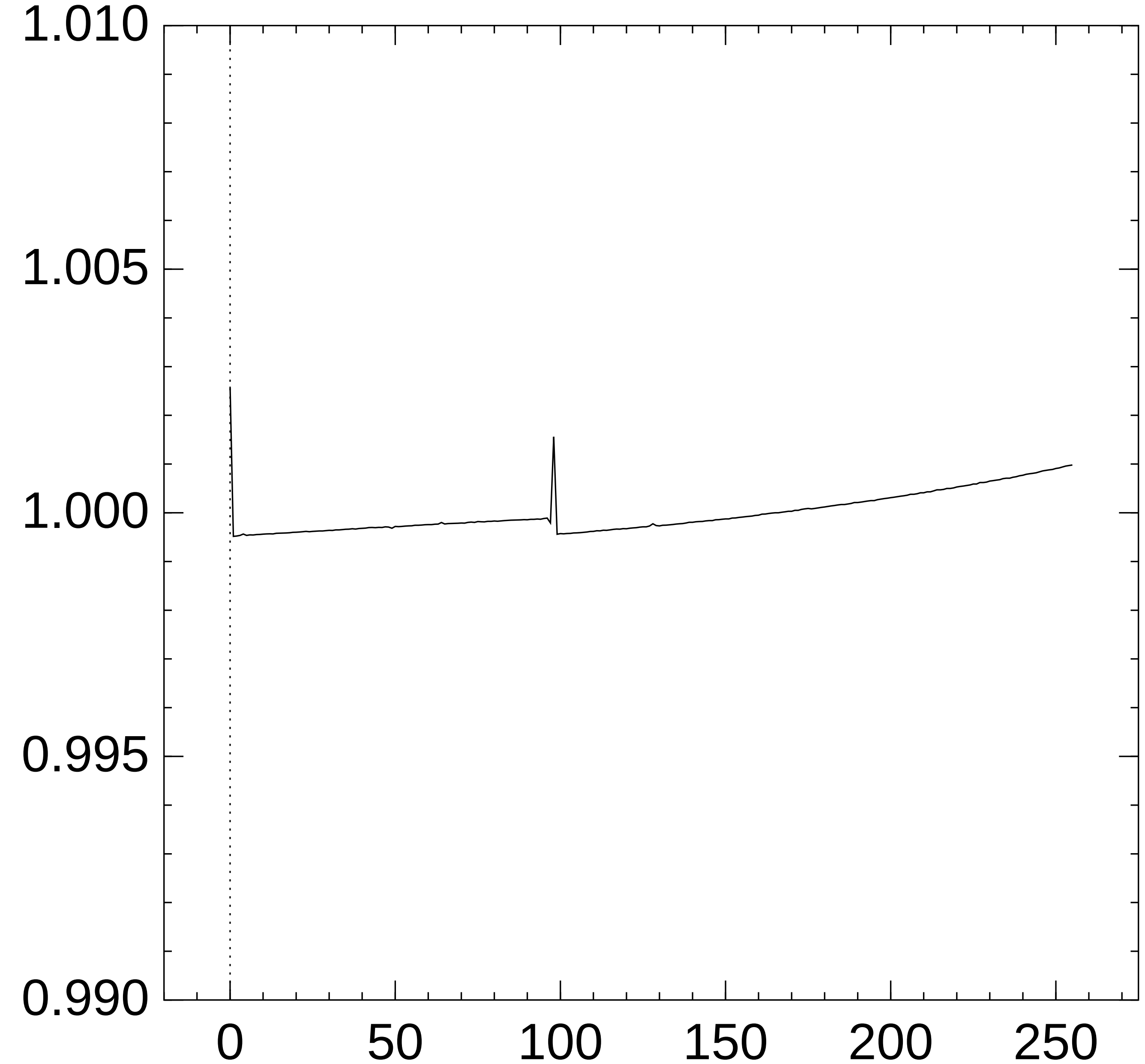
Graph of $256 \Pr[z_{97} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

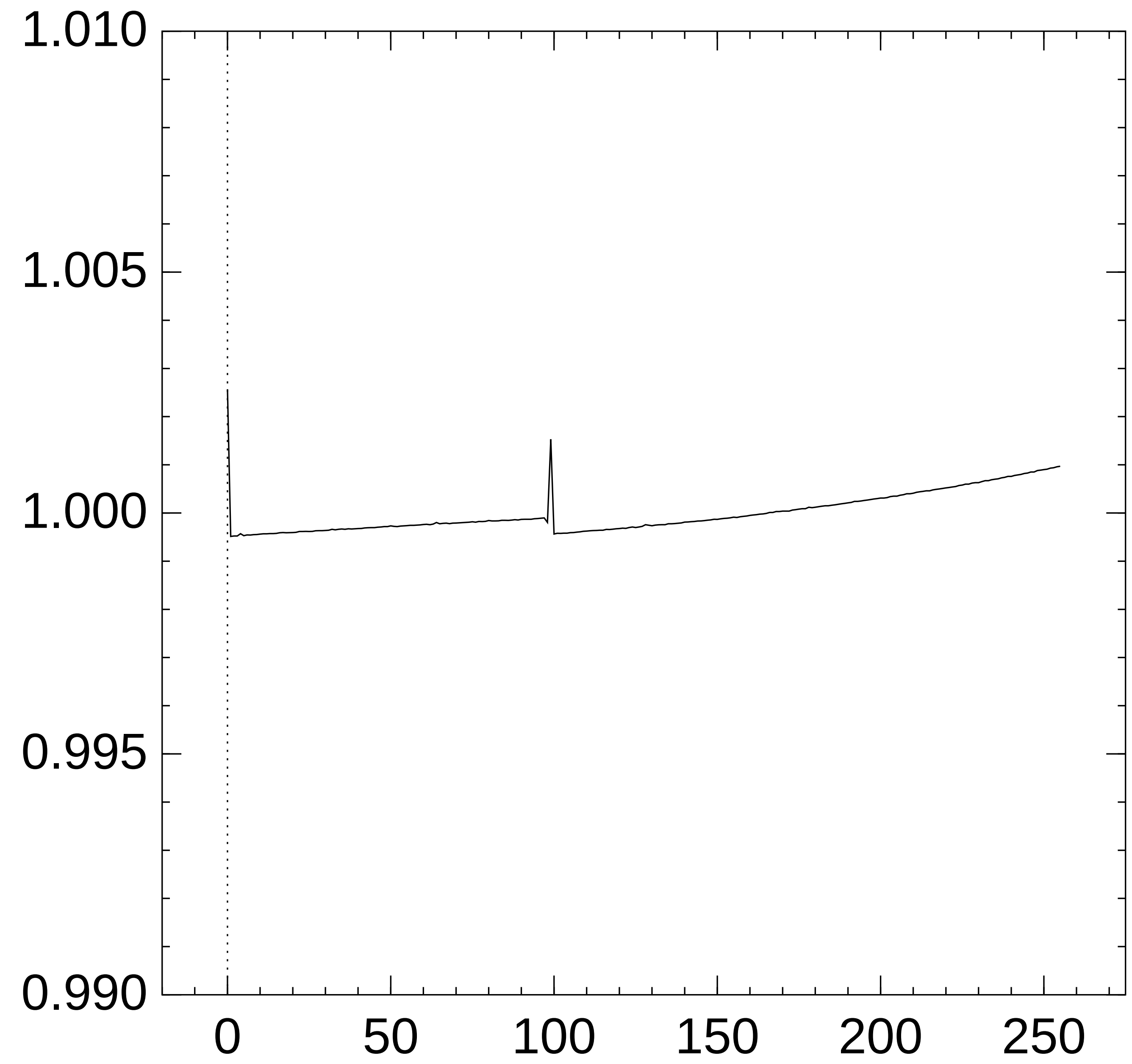
Graph of $256 \Pr[z_{98} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

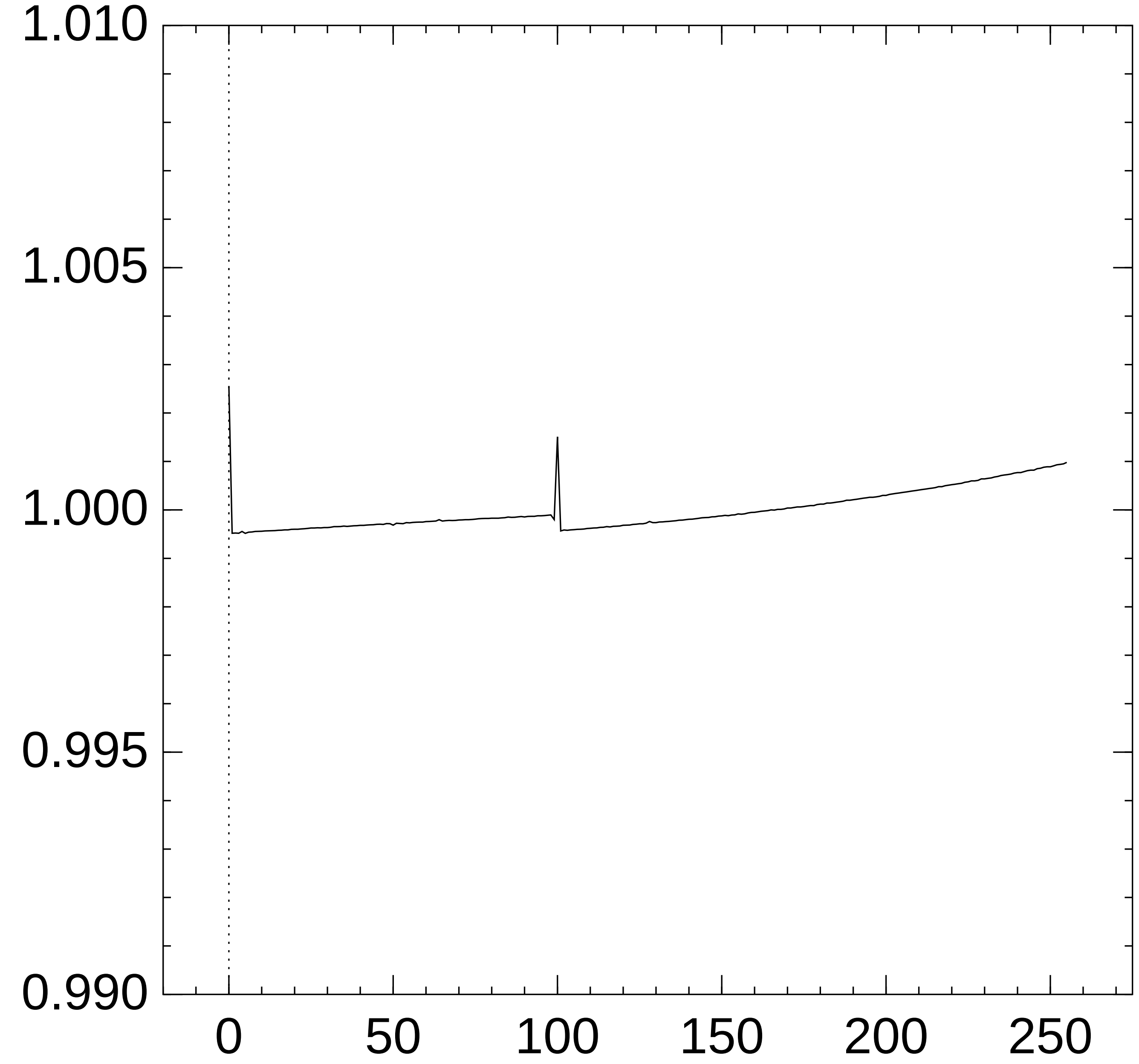
Graph of $256 \Pr[z_{99} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

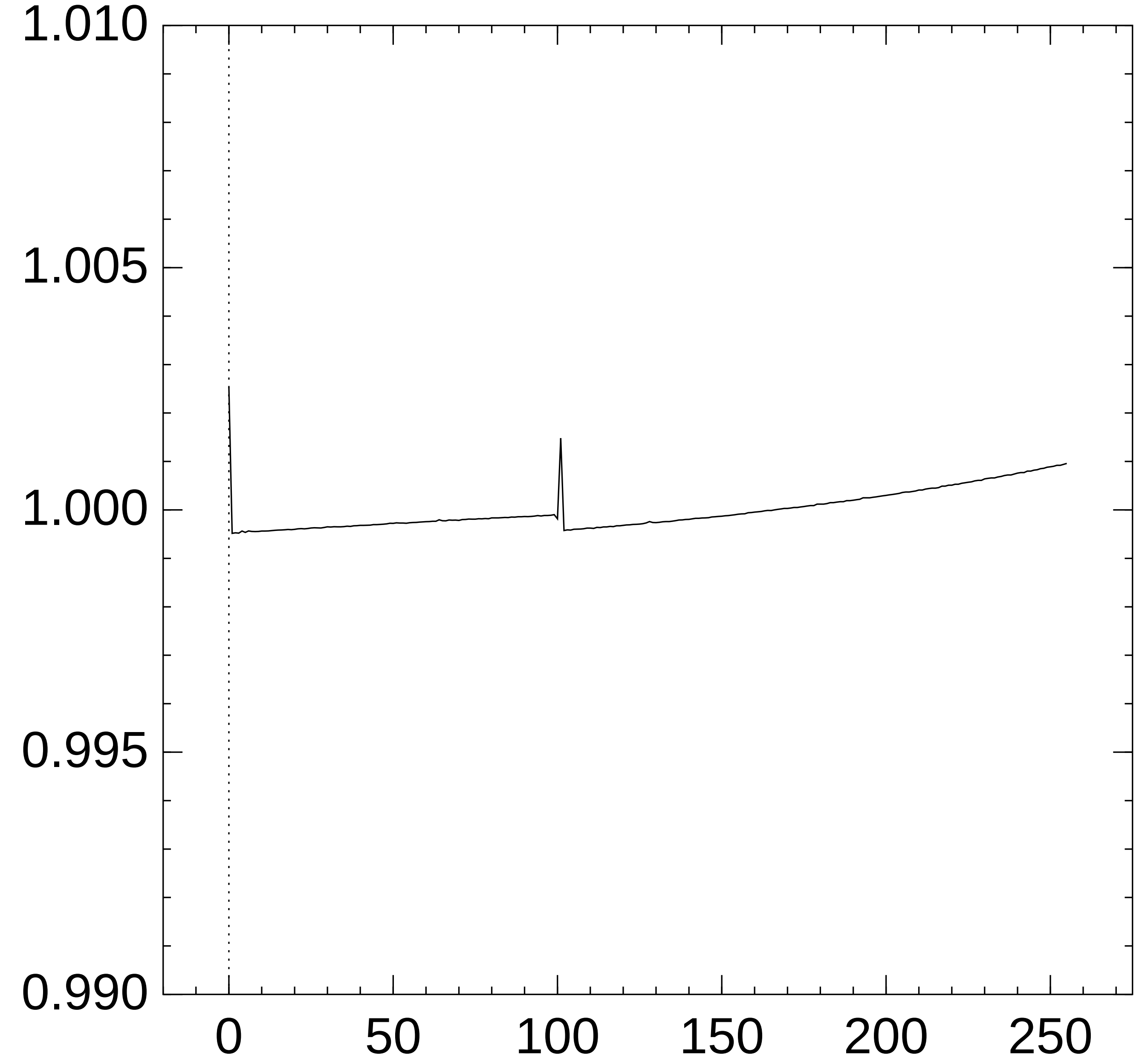
Graph of $256 \Pr[z_{100} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

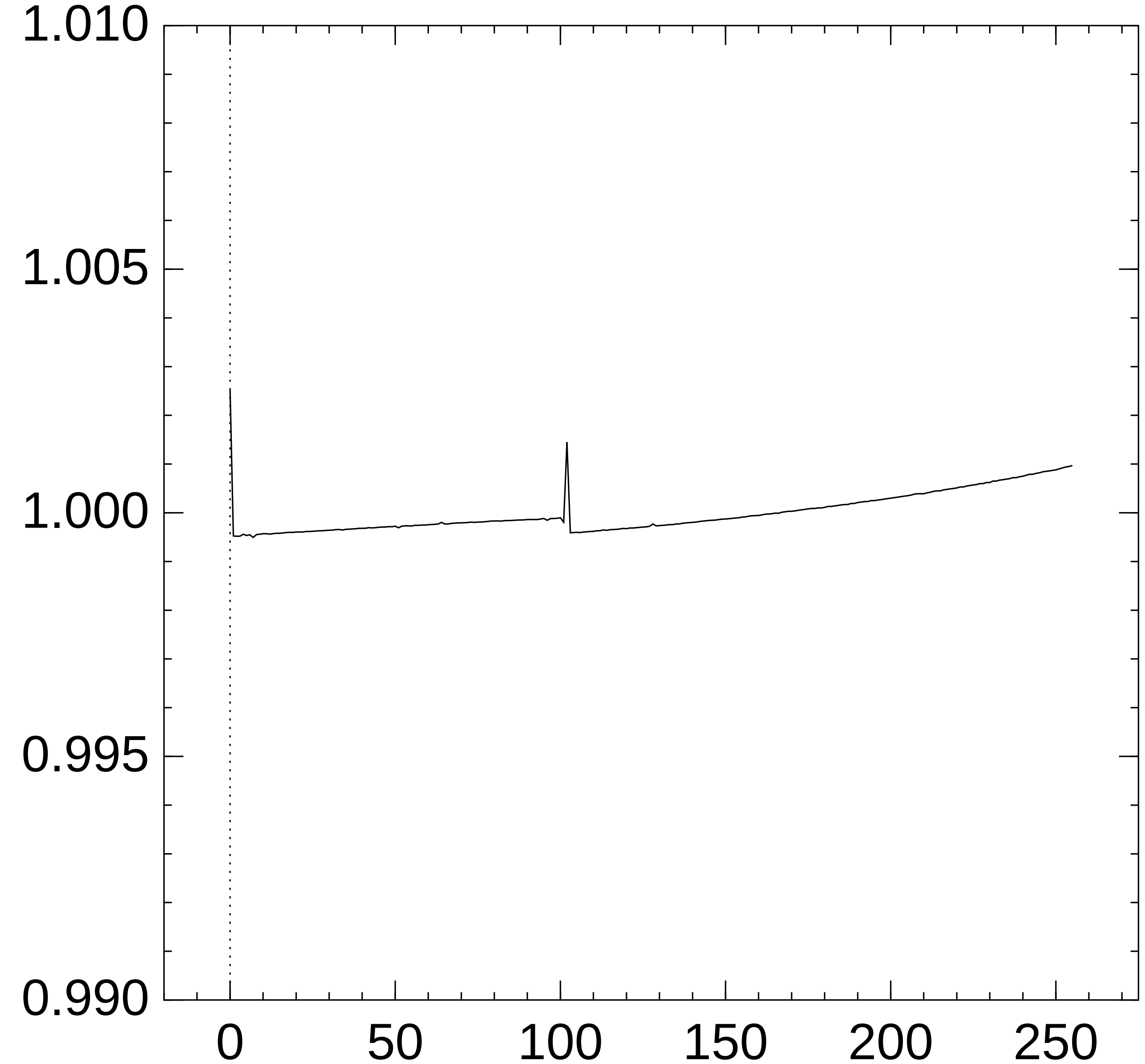
Graph of $256 \Pr[z_{101} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

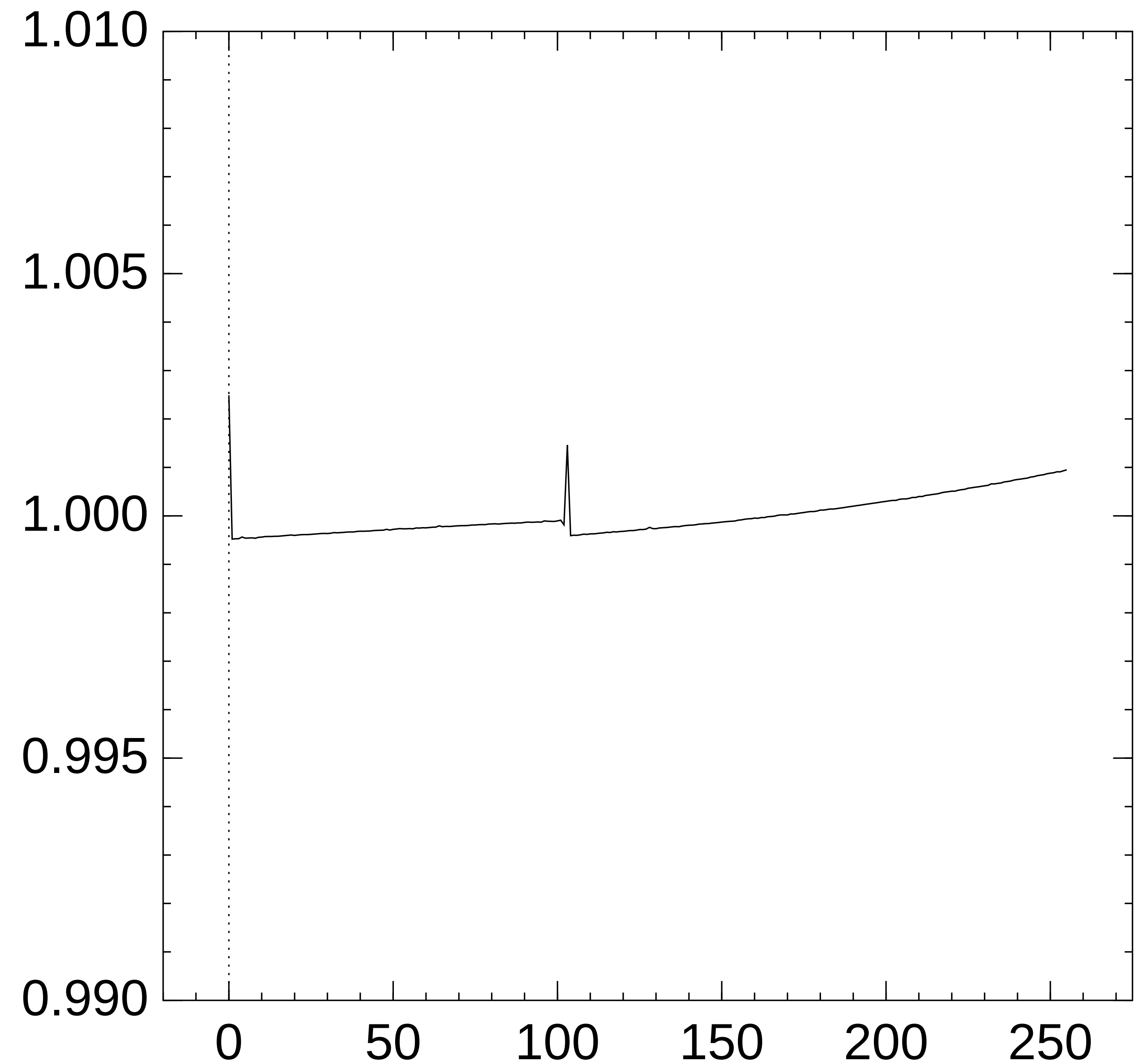
Graph of $256 \Pr[z_{102} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

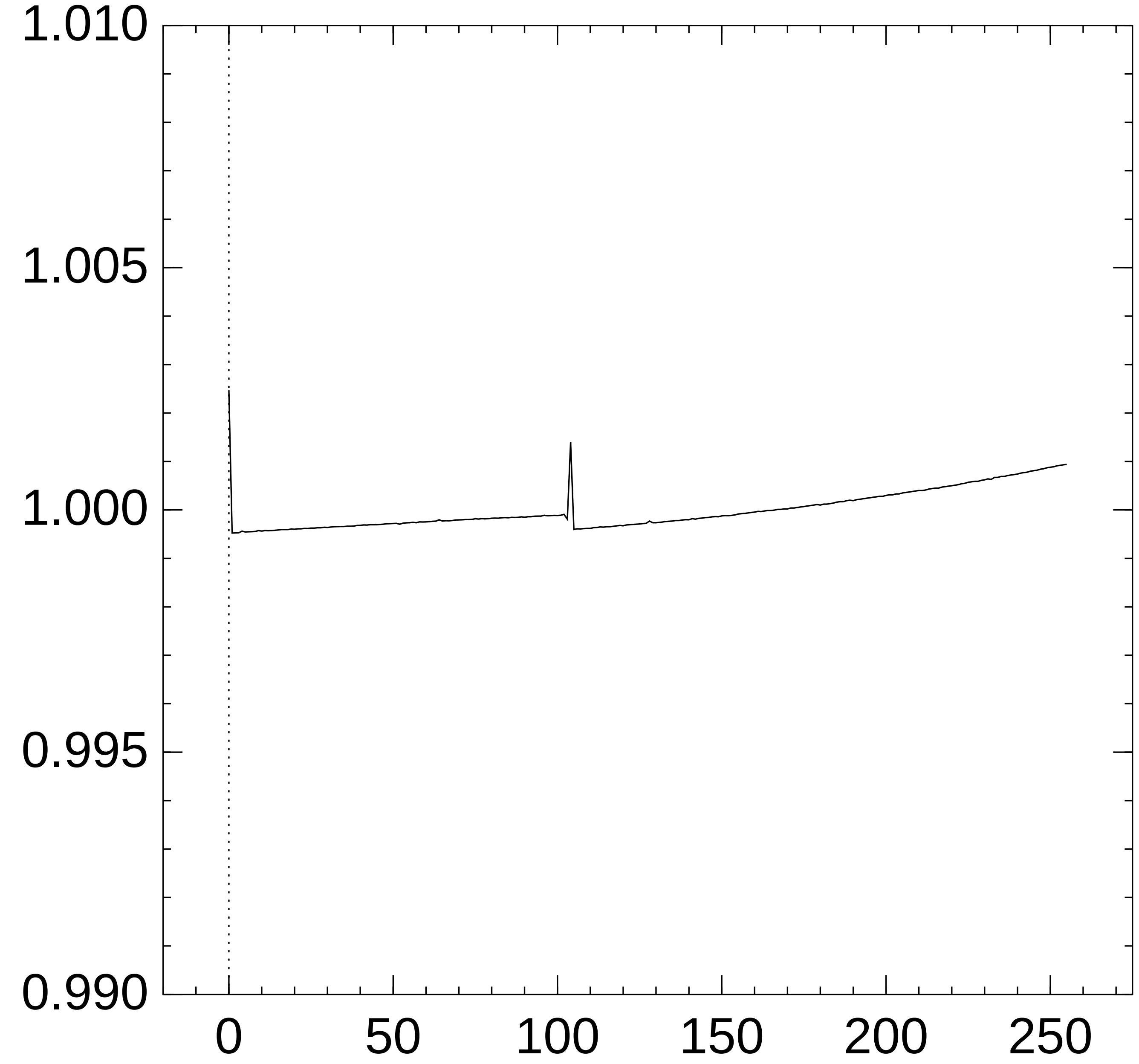
Graph of $256 \Pr[z_{103} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

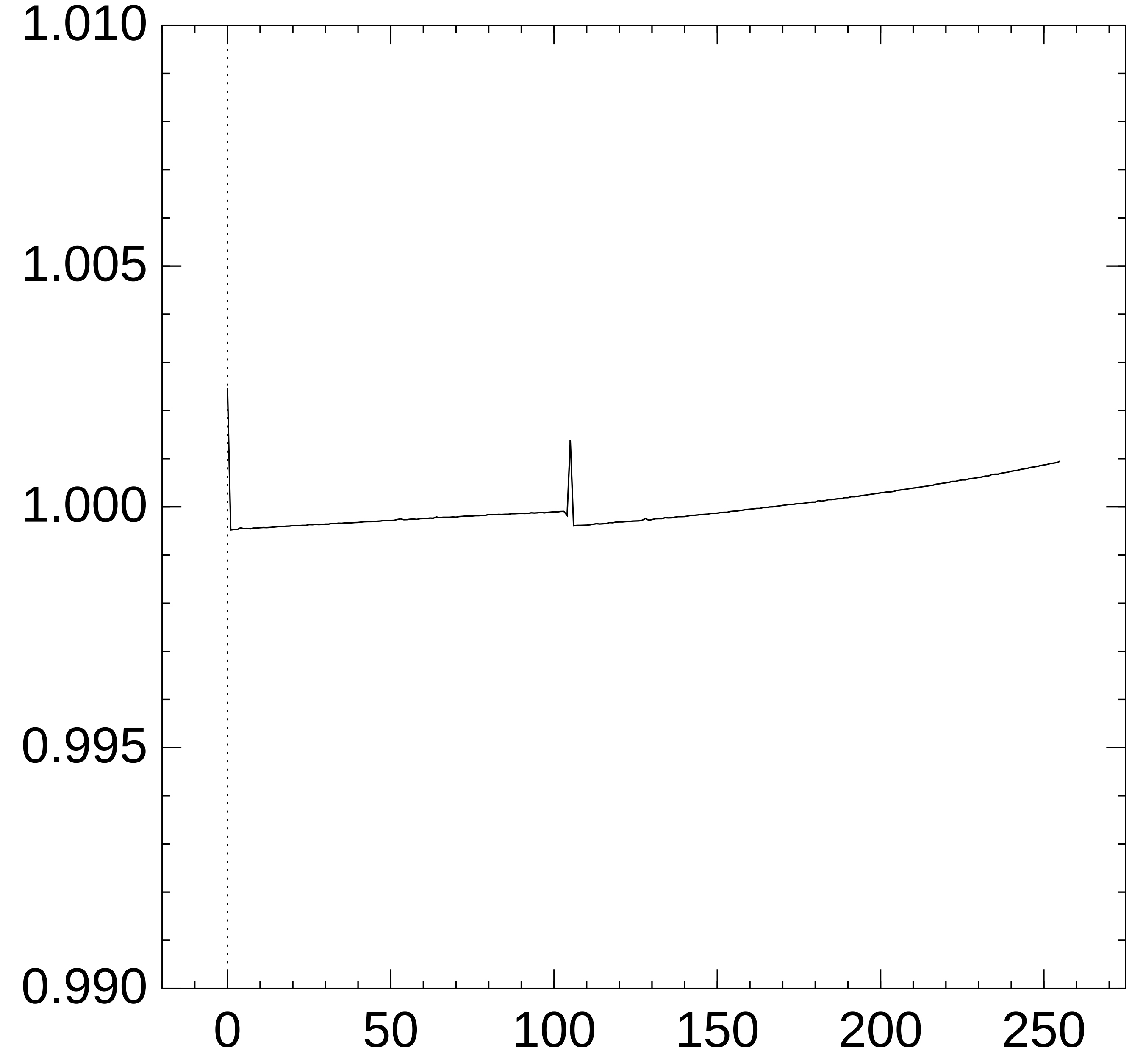
Graph of $256 \Pr[z_{104} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

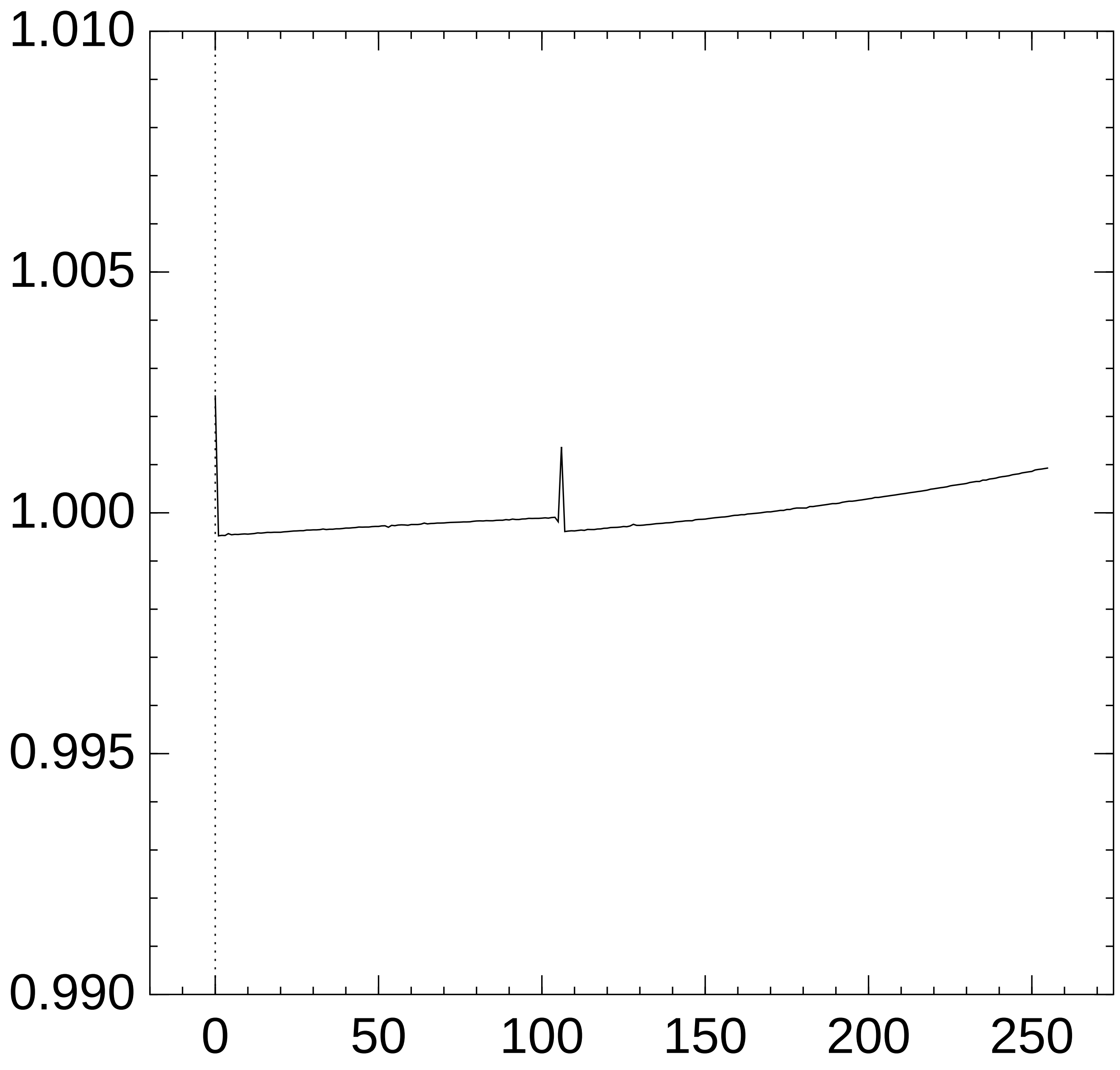
Graph of $256 \Pr[z_{105} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

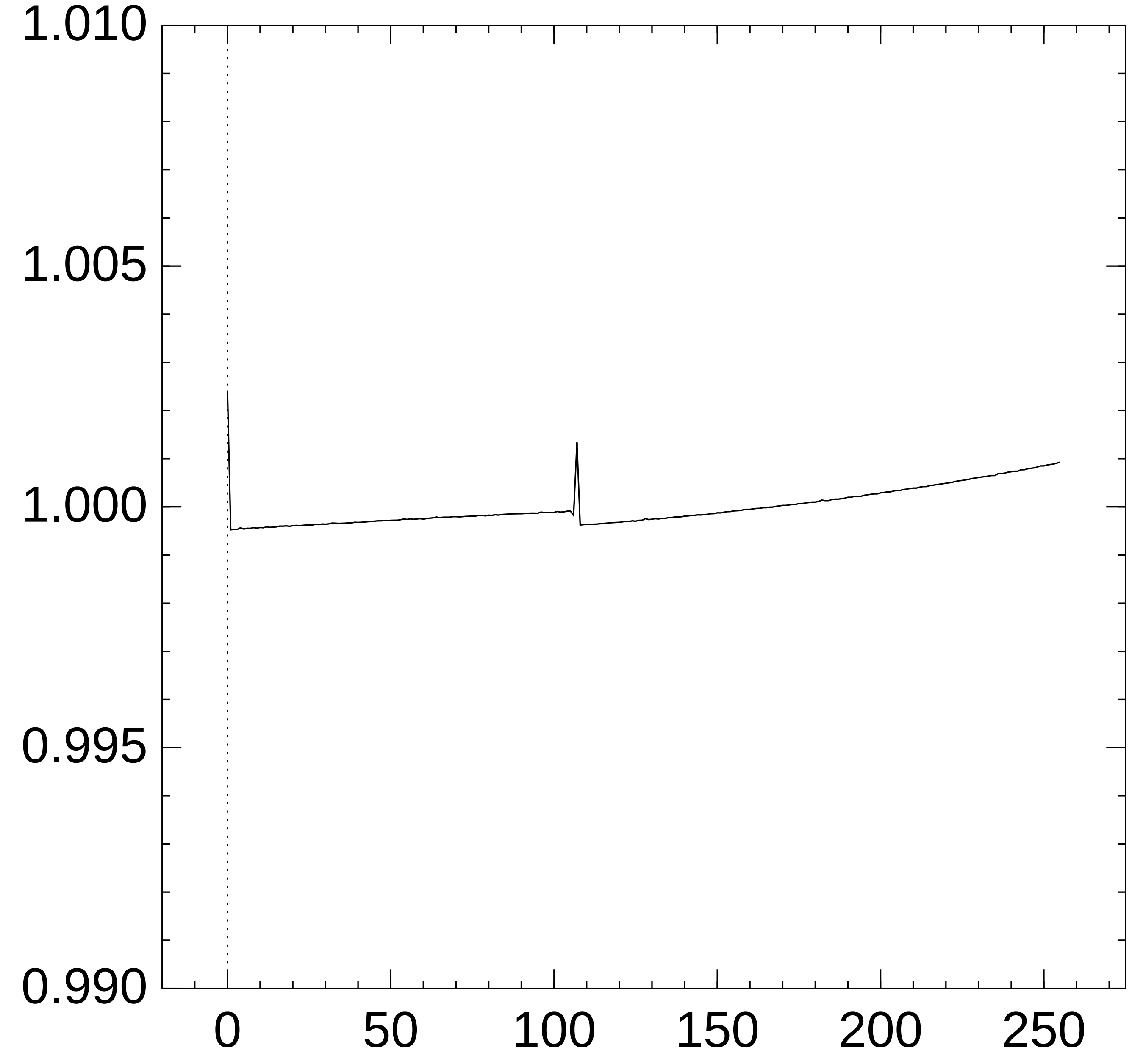
Graph of $256 \Pr[z_{106} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

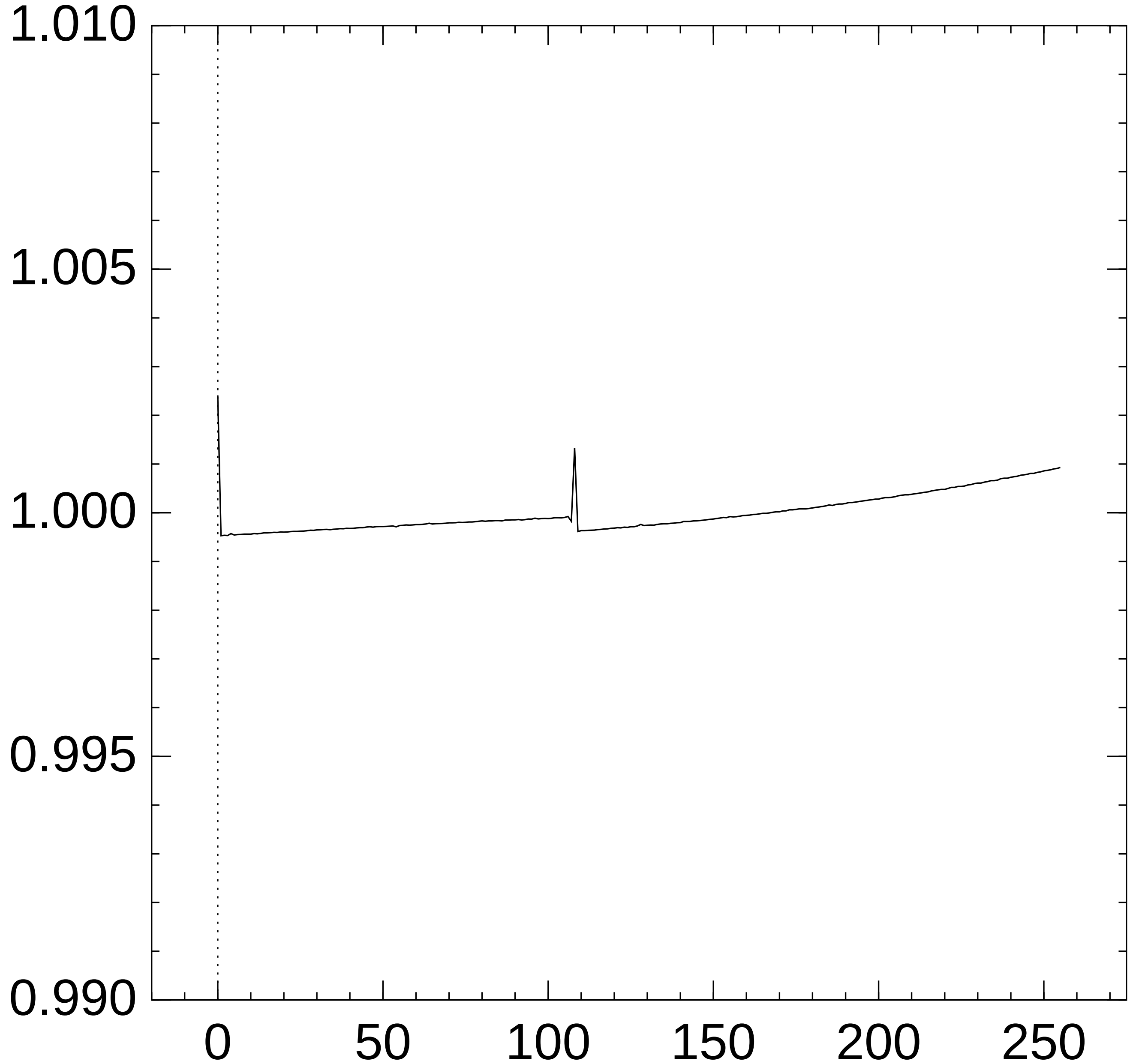
Graph of $256 \Pr[z_{107} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

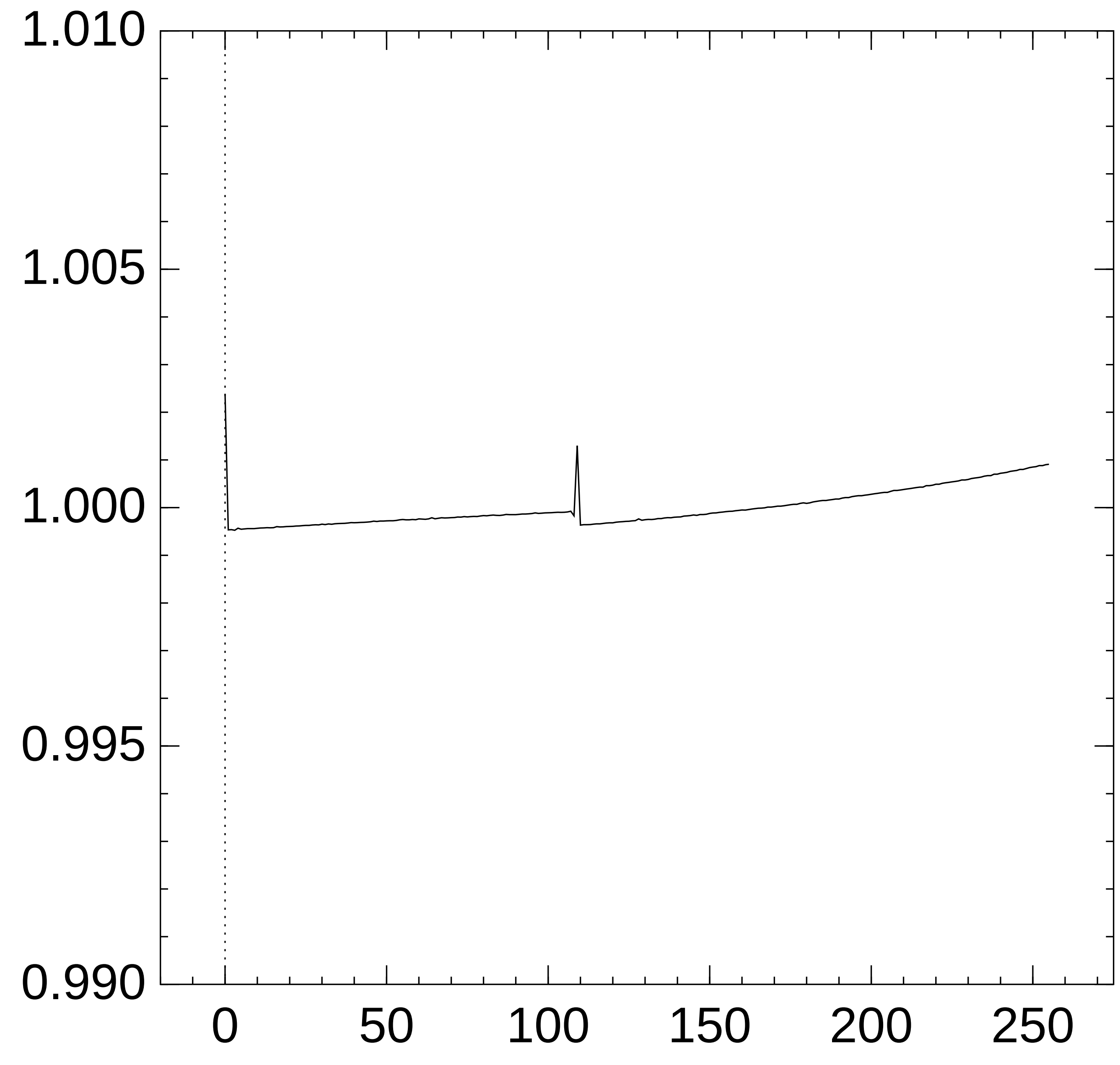
Graph of $256 \Pr[z_{108} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

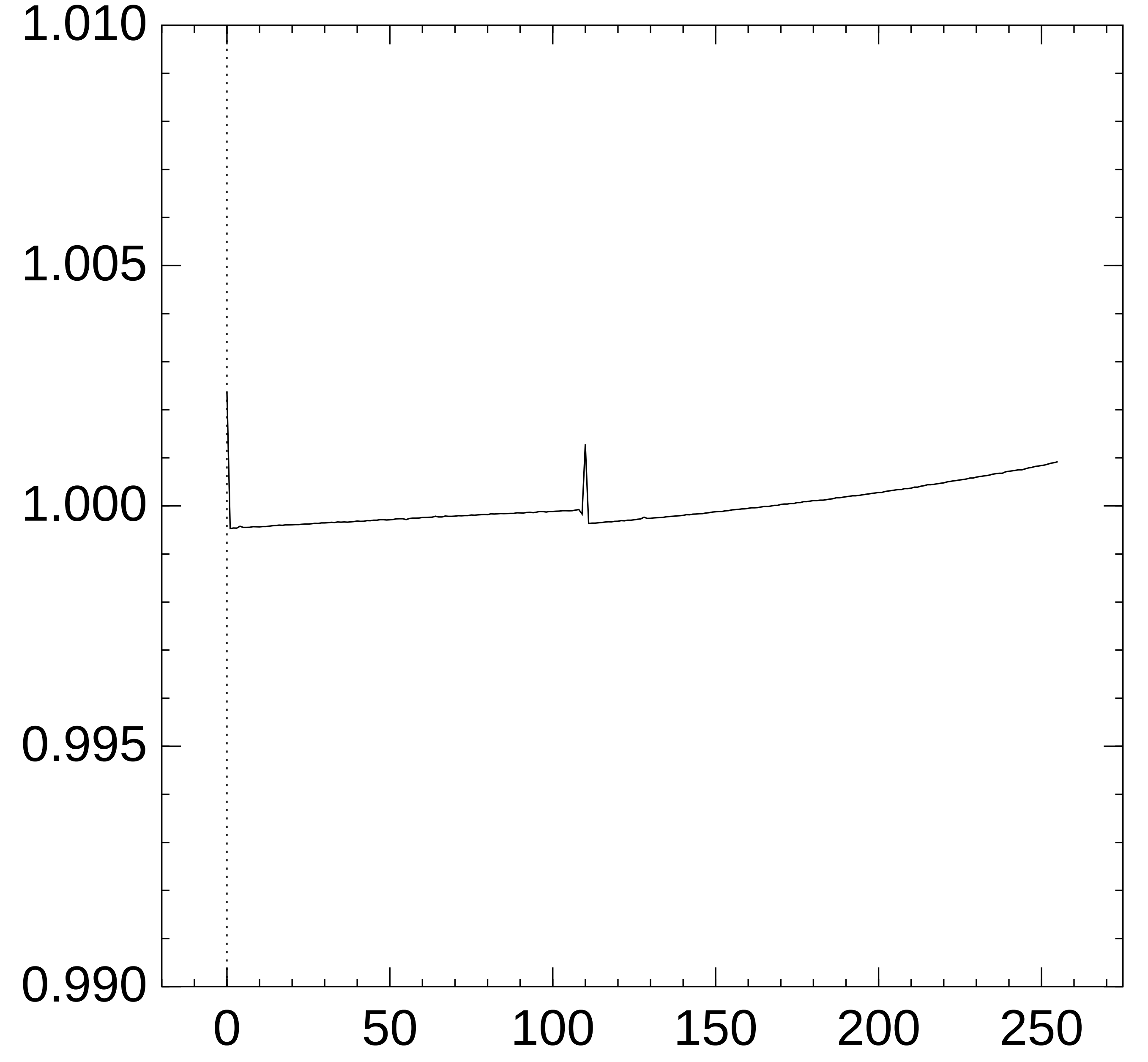
Graph of $256 \Pr[z_{109} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

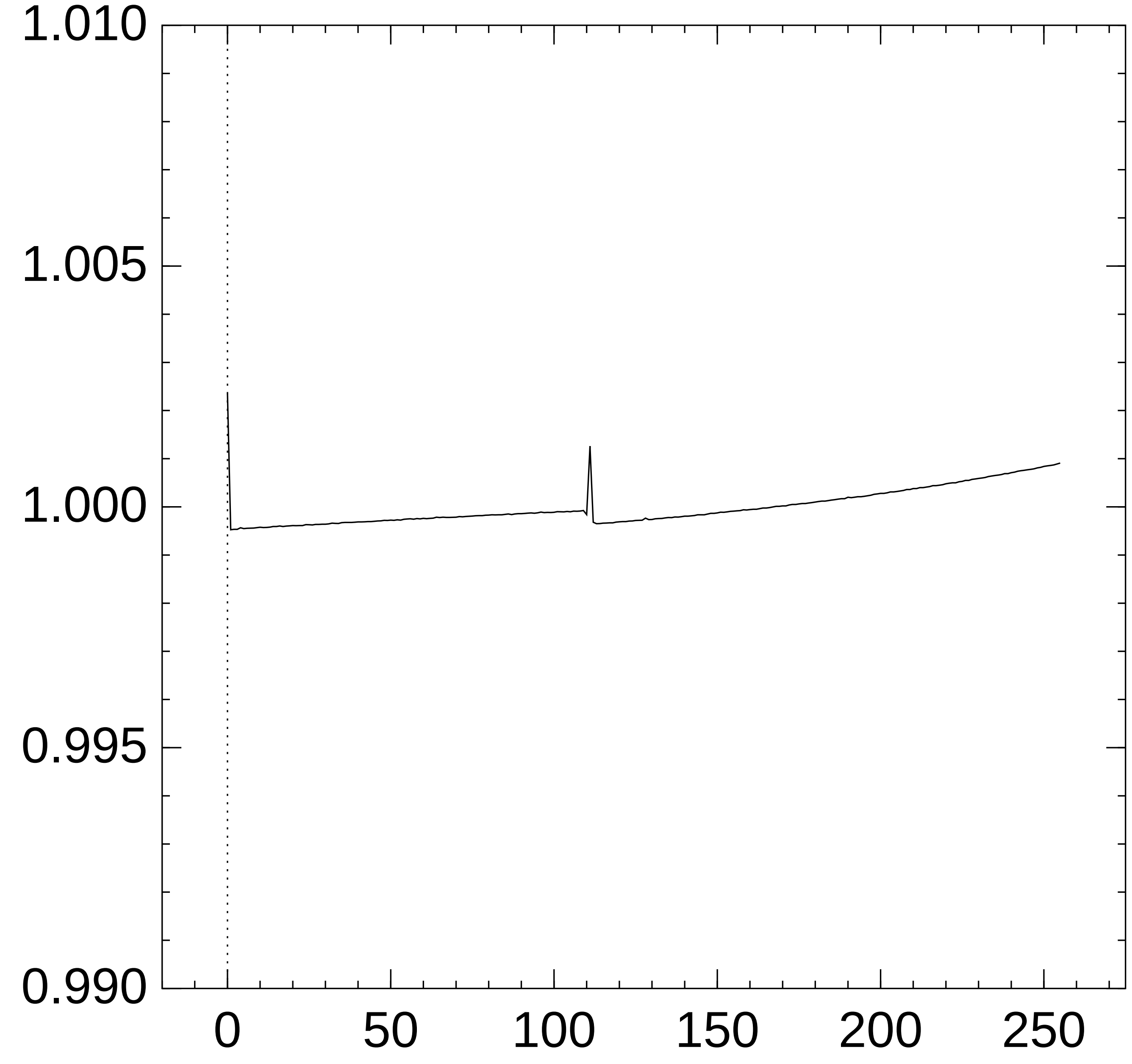
Graph of $256 \Pr[z_{110} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

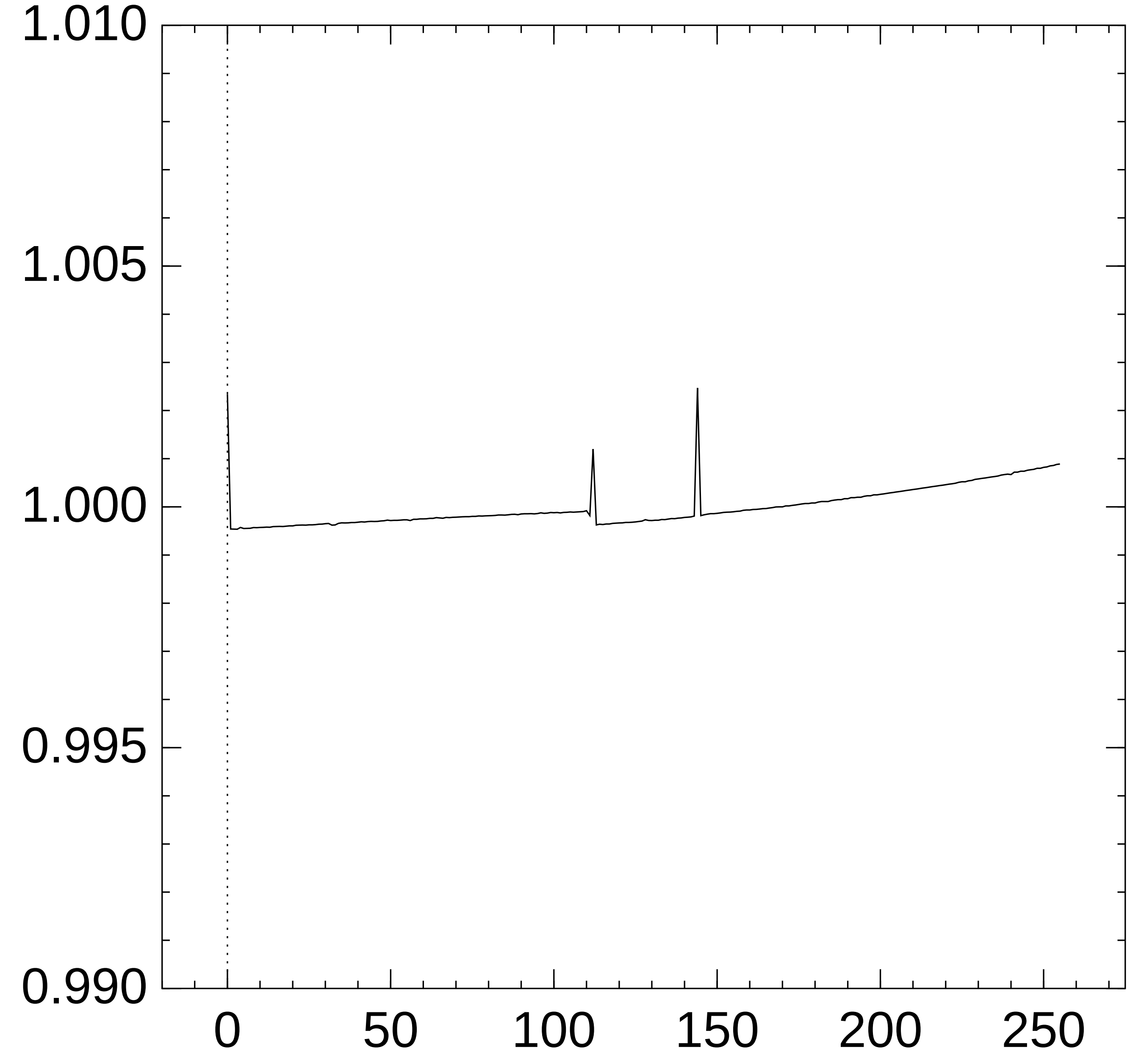
Graph of $256 \Pr[z_{111} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

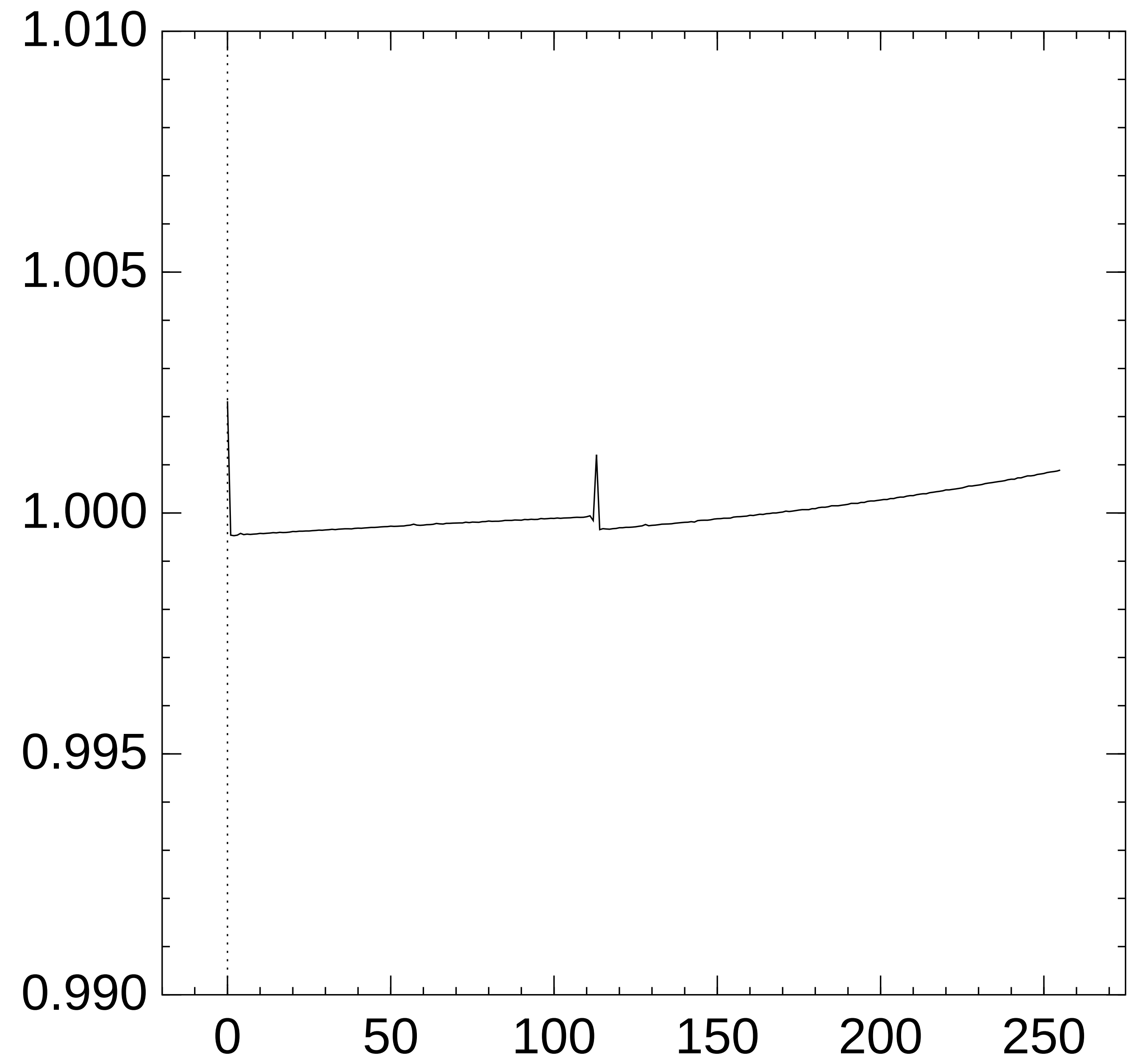
Graph of $256 \Pr[z_{112} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

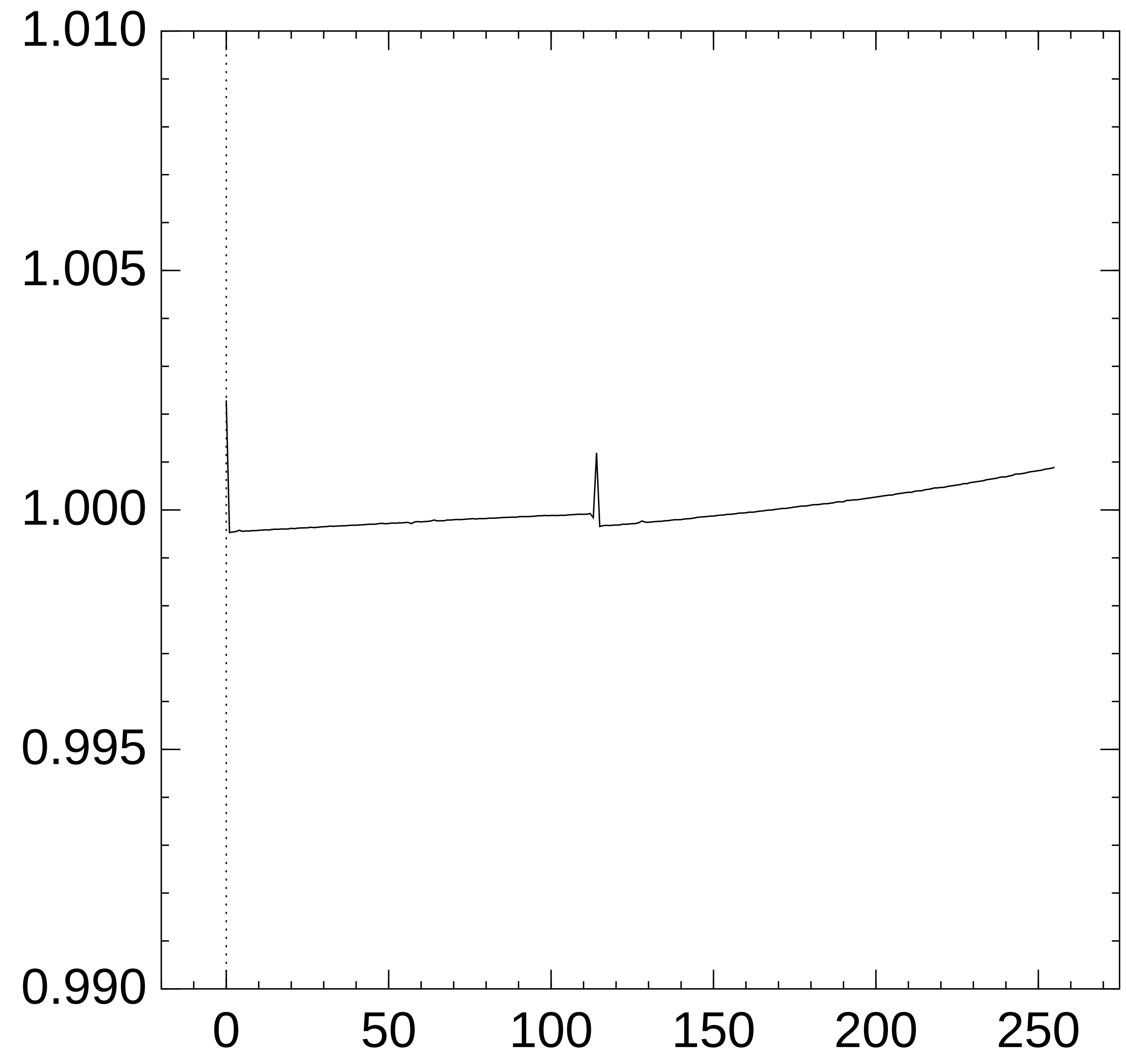
Graph of $256 \Pr[z_{113} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

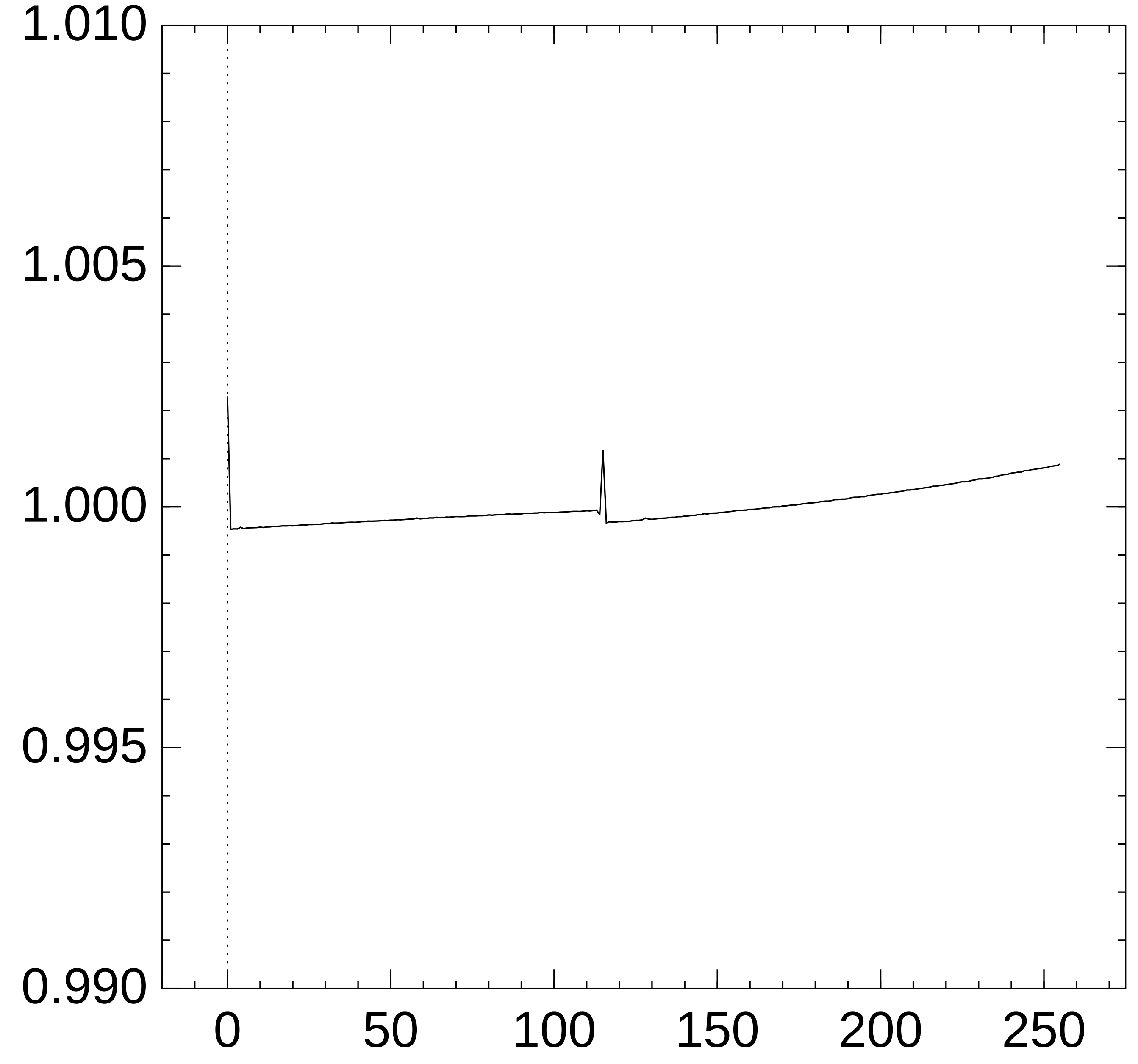
Graph of $256 \Pr[z_{114} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

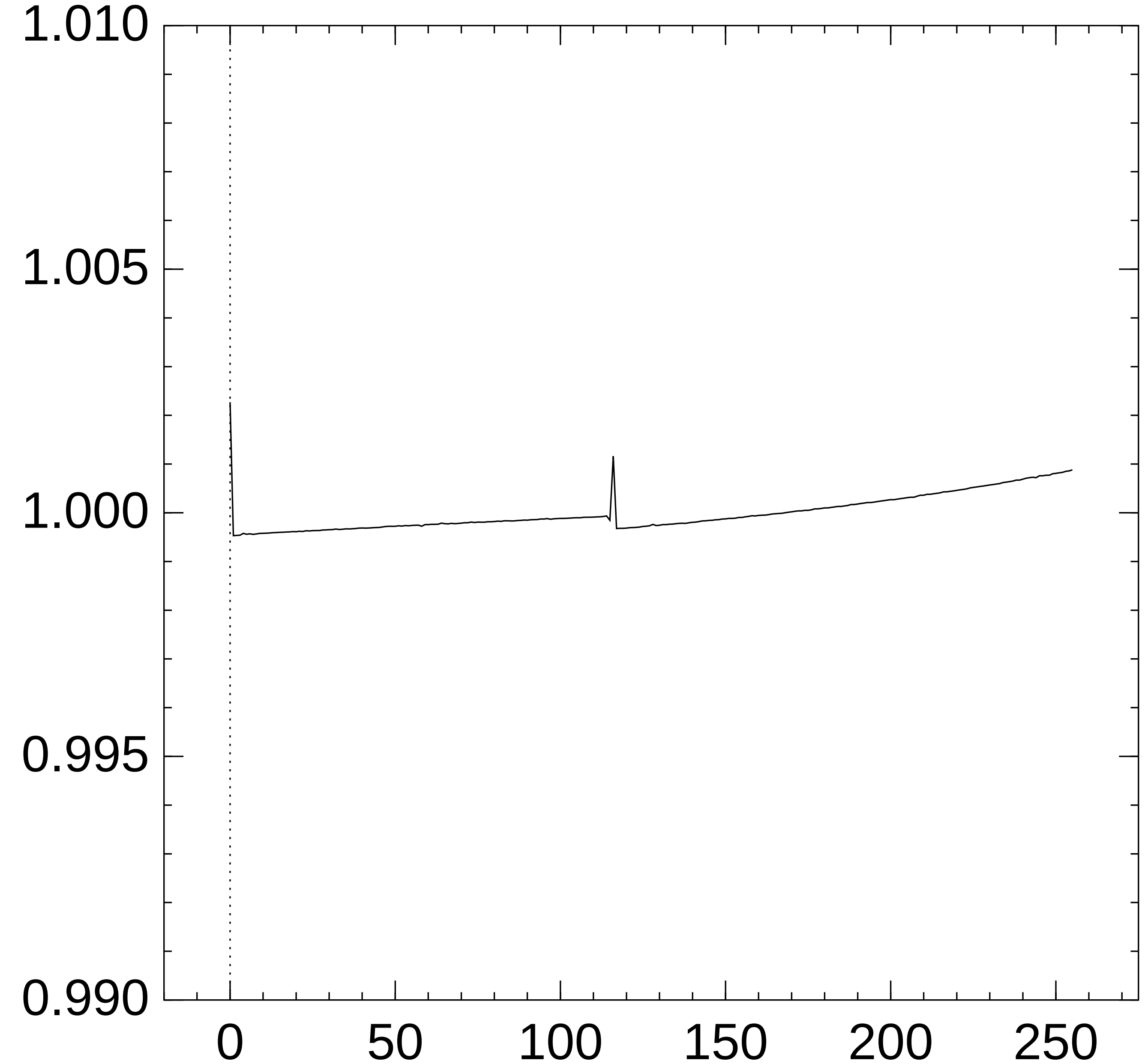
Graph of $256 \Pr[z_{115} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

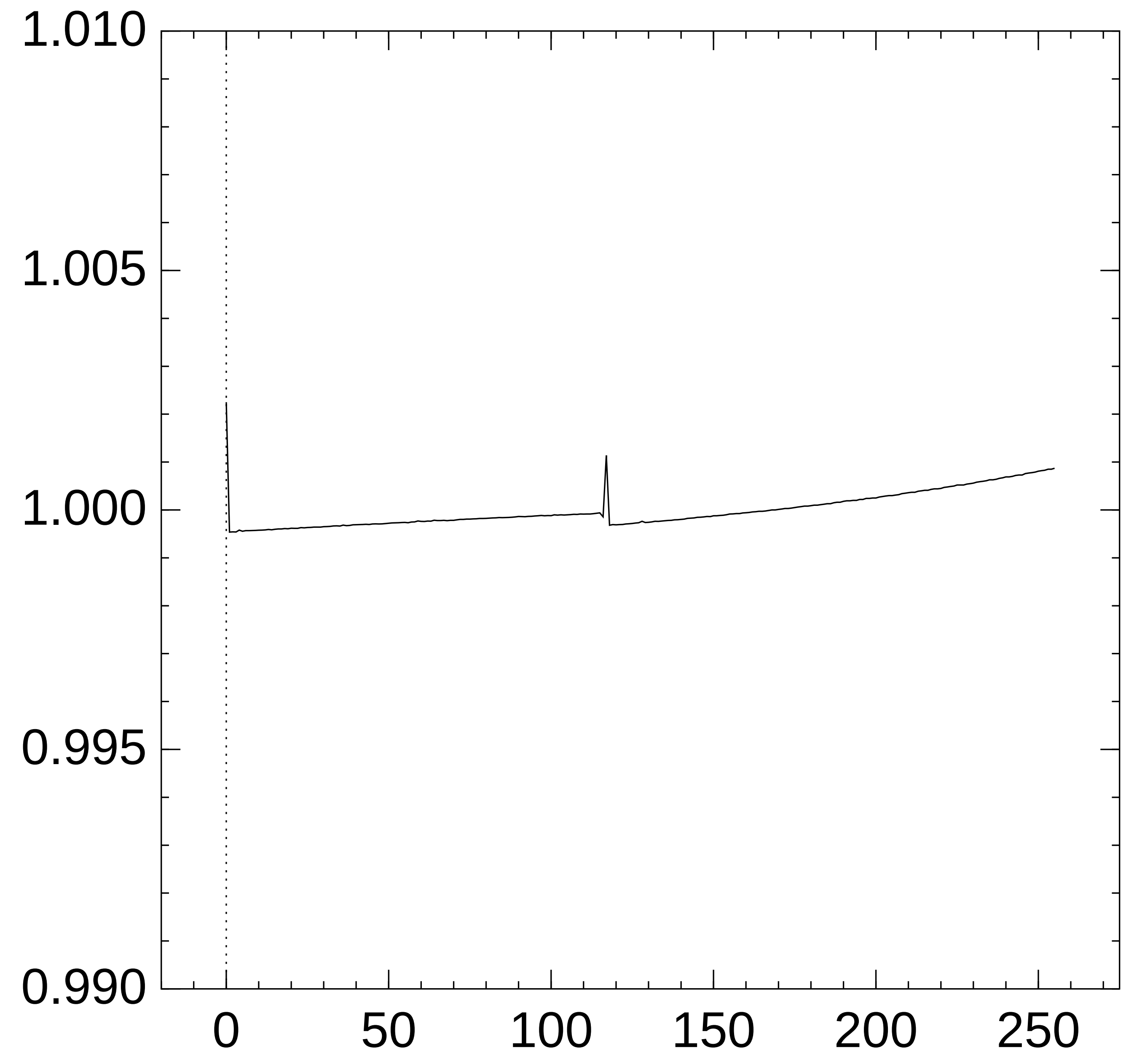
Graph of $256 \Pr[z_{116} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

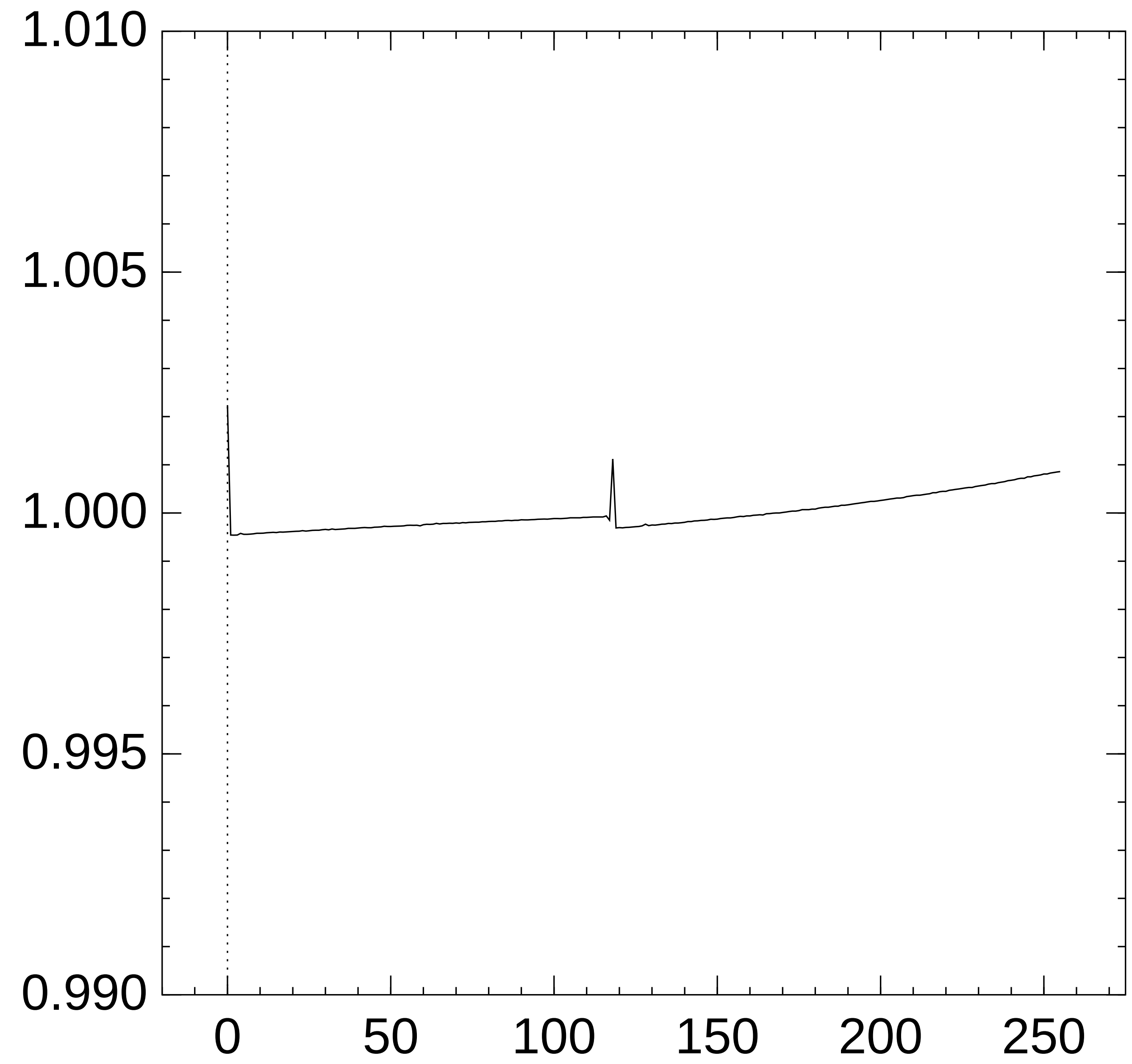
Graph of $256 \Pr[z_{117} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

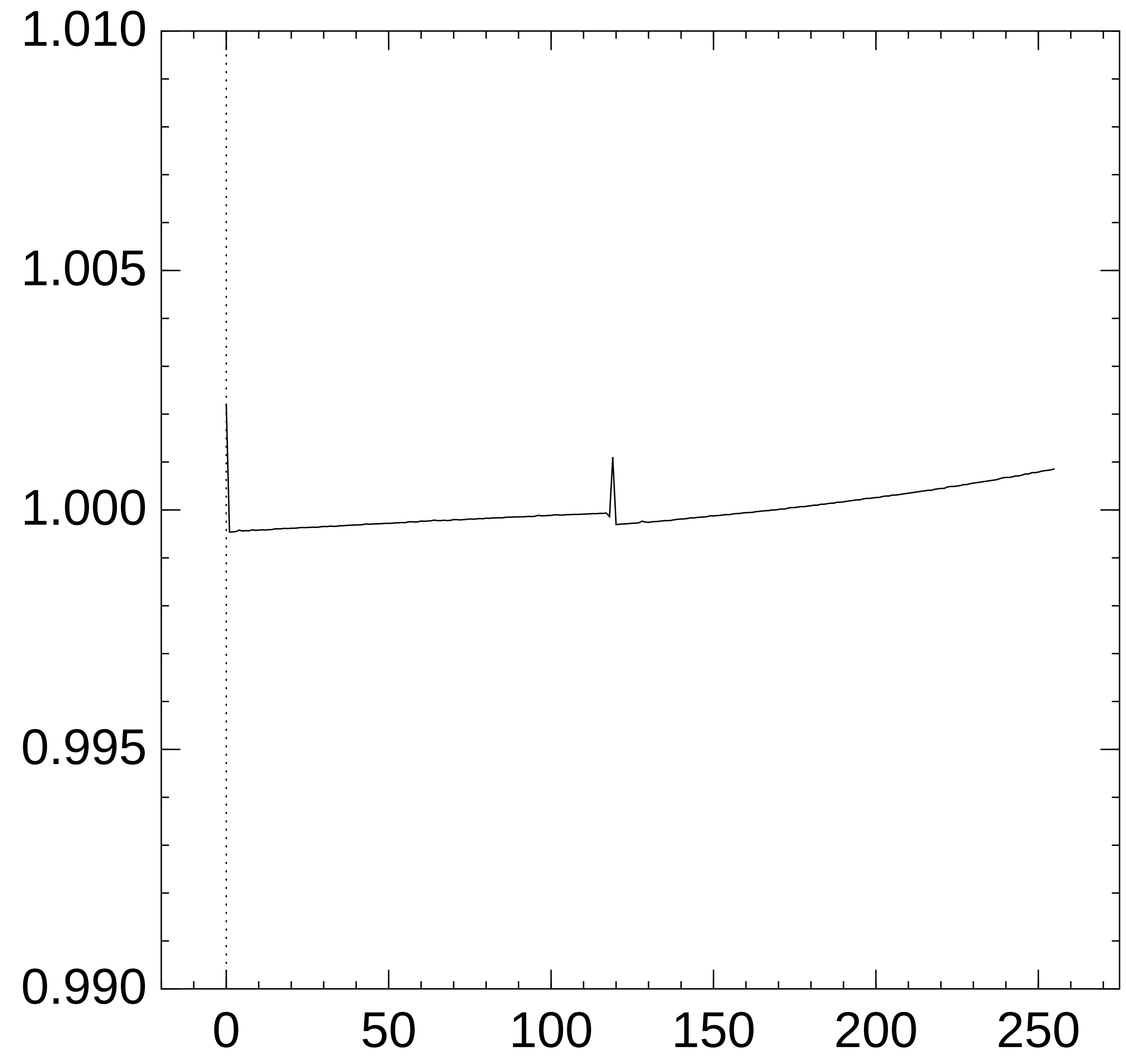
Graph of $256 \Pr[z_{118} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

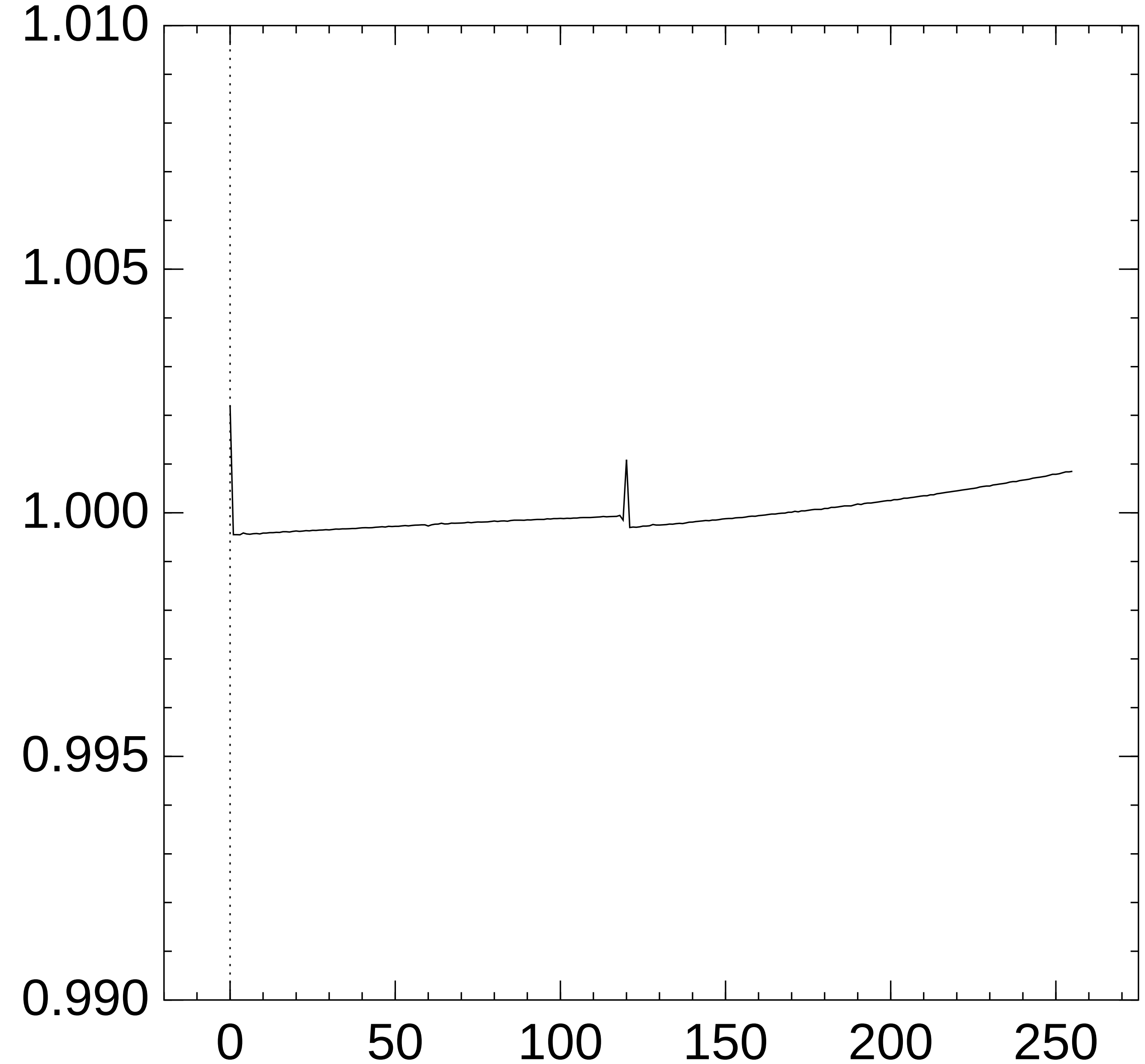
Graph of $256 \Pr[z_{119} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

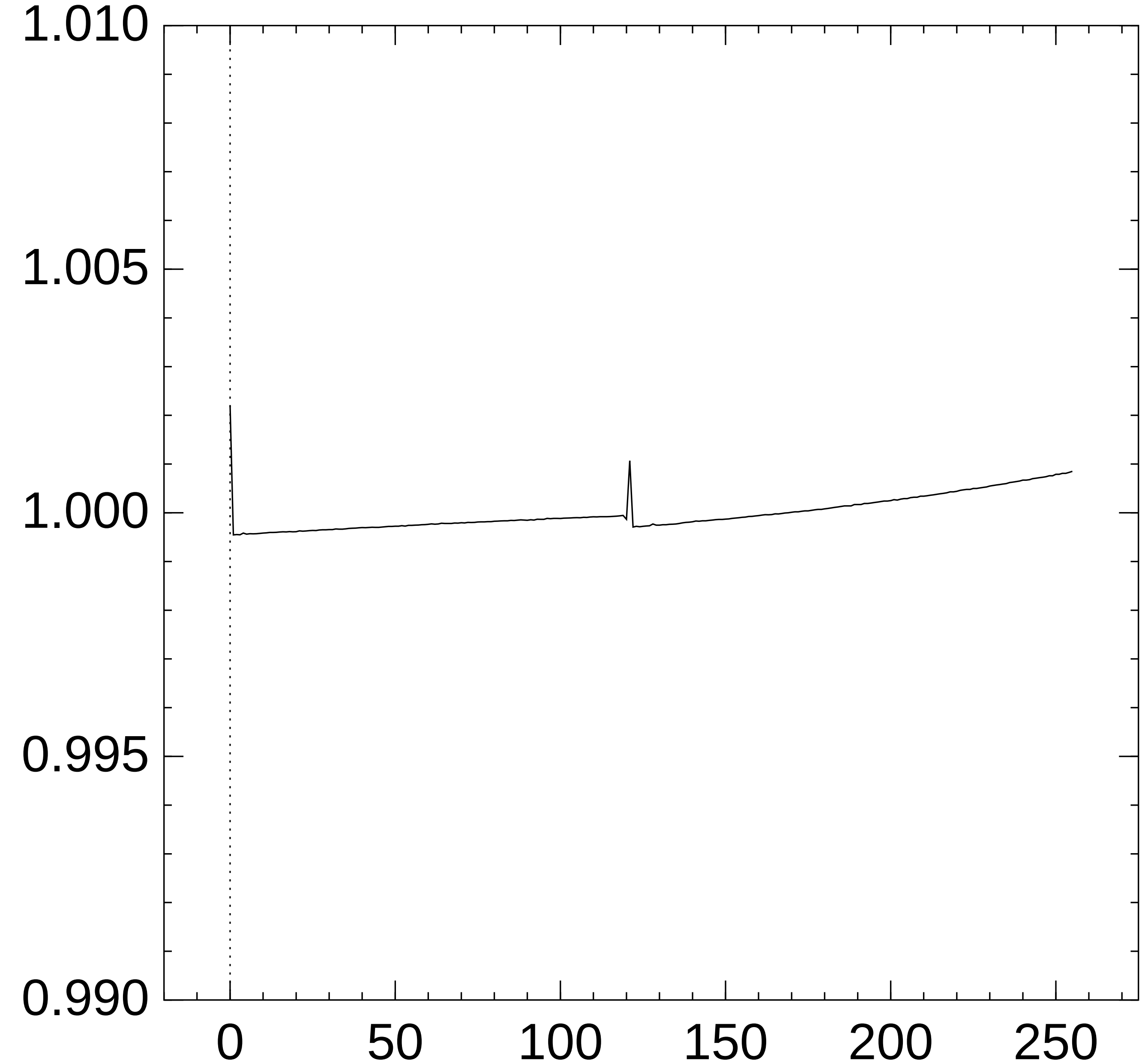
Graph of $256 \Pr[z_{120} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

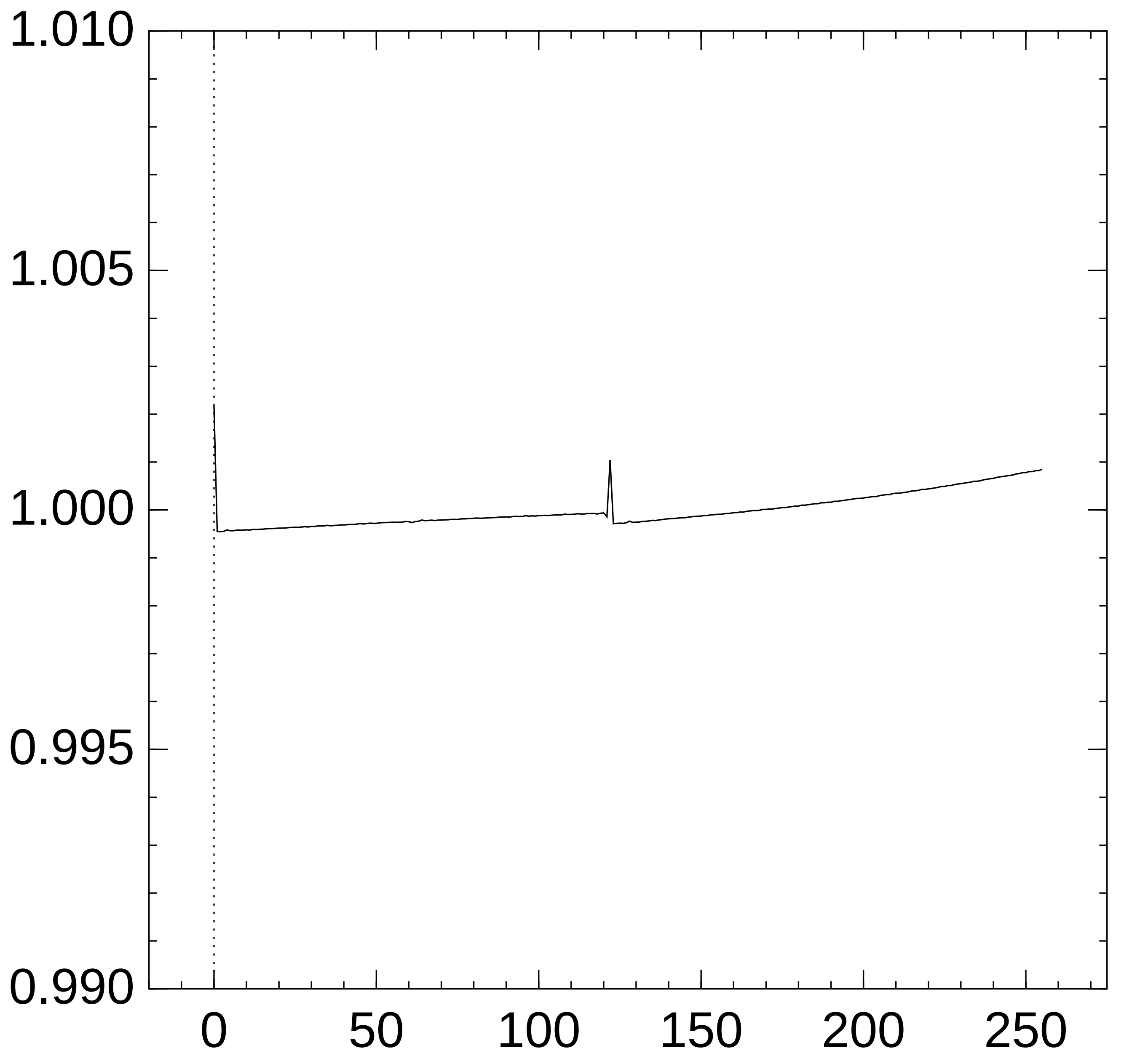
Graph of $256 \Pr[z_{121} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

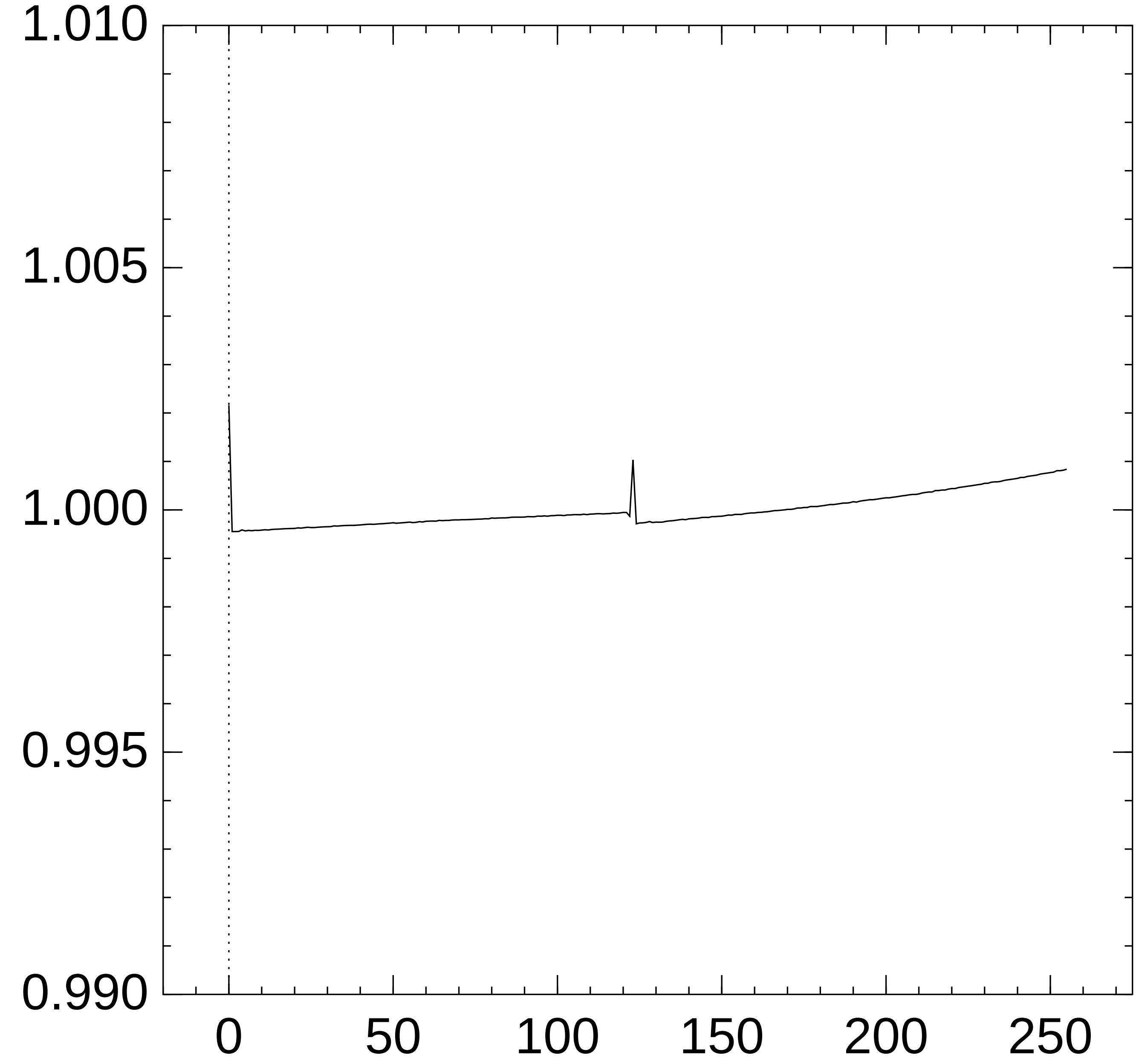
Graph of $256 \Pr[z_{122} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

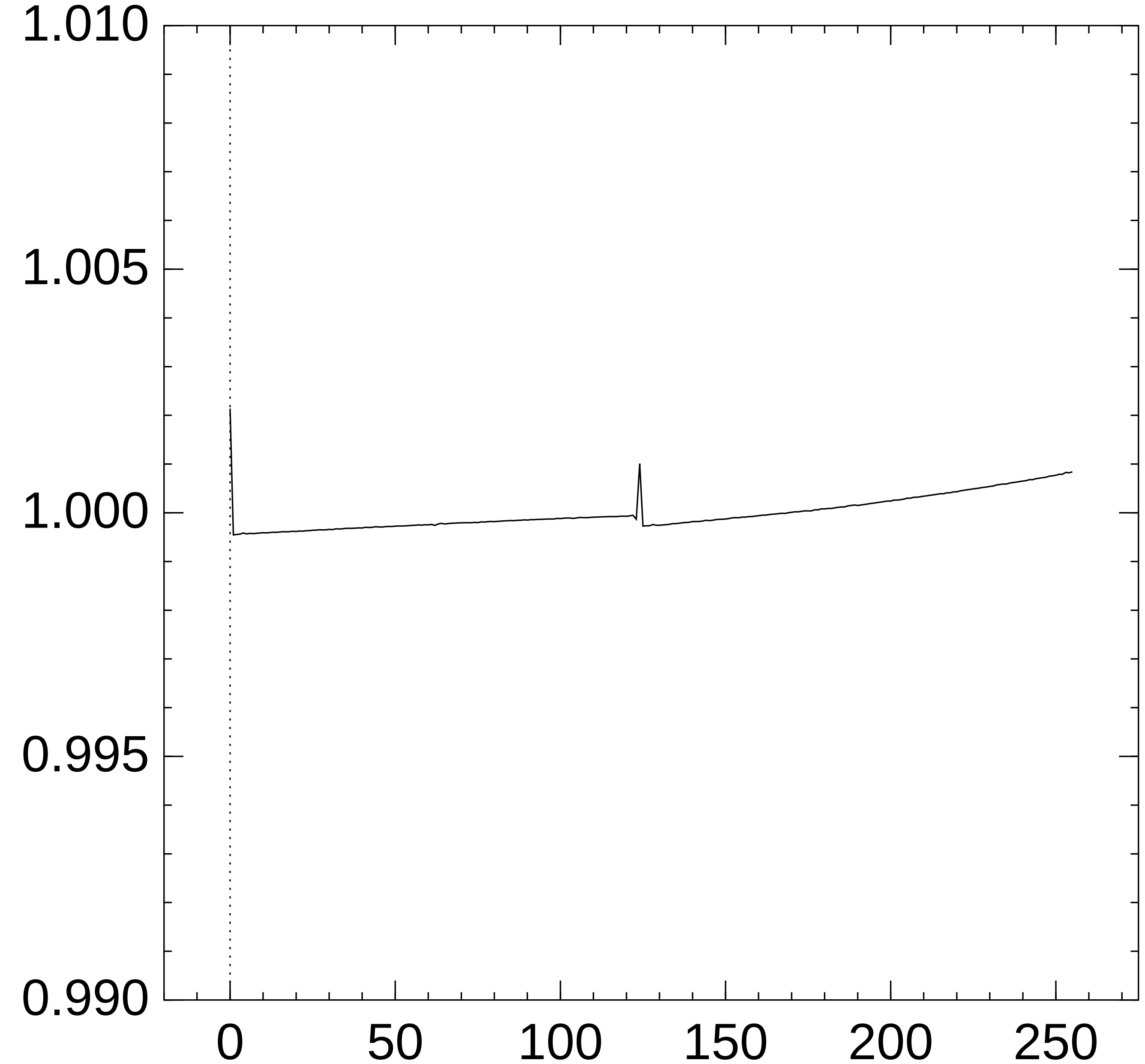
Graph of $256 \Pr[z_{123} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

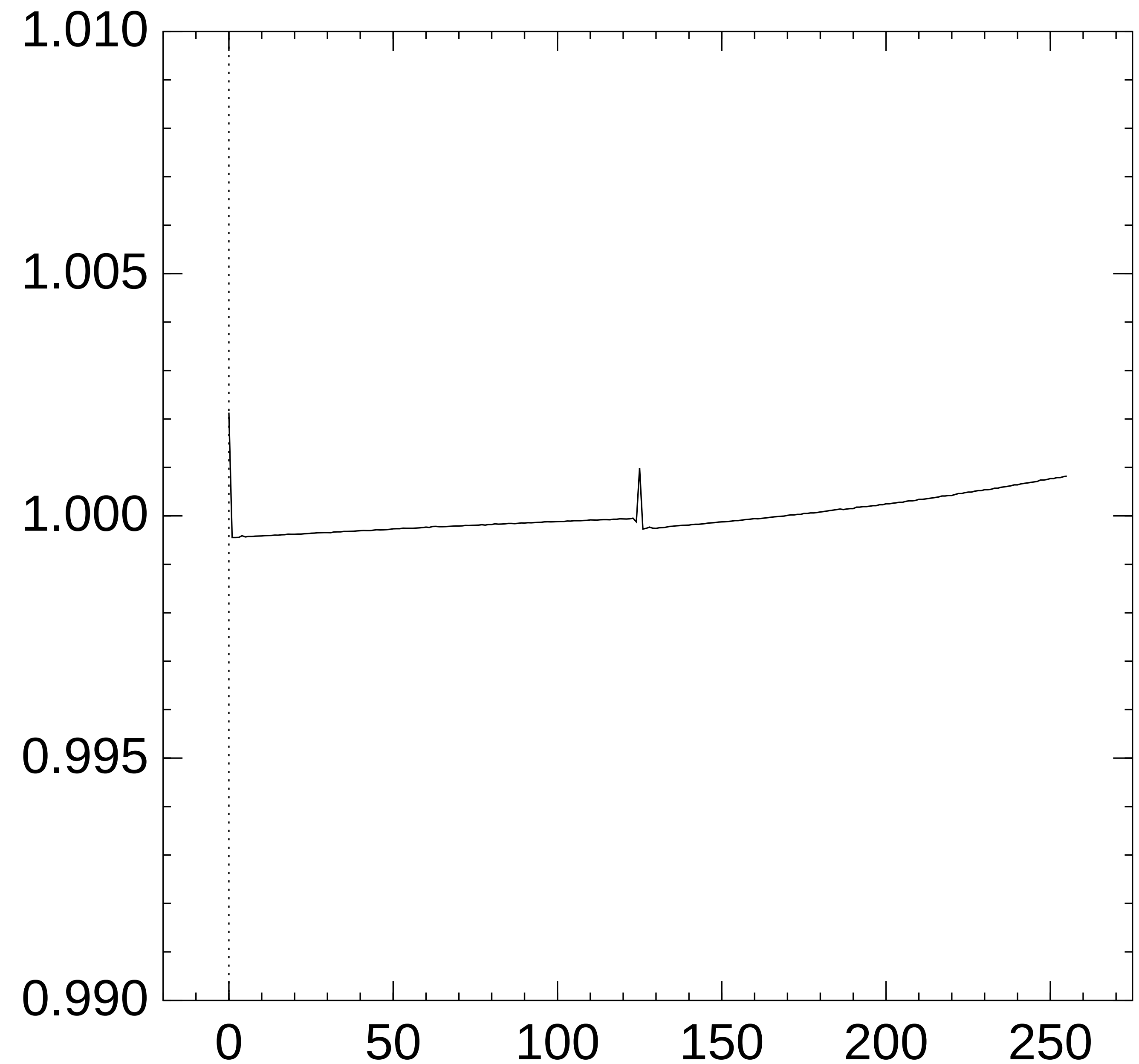
Graph of $256\Pr[z_{124} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

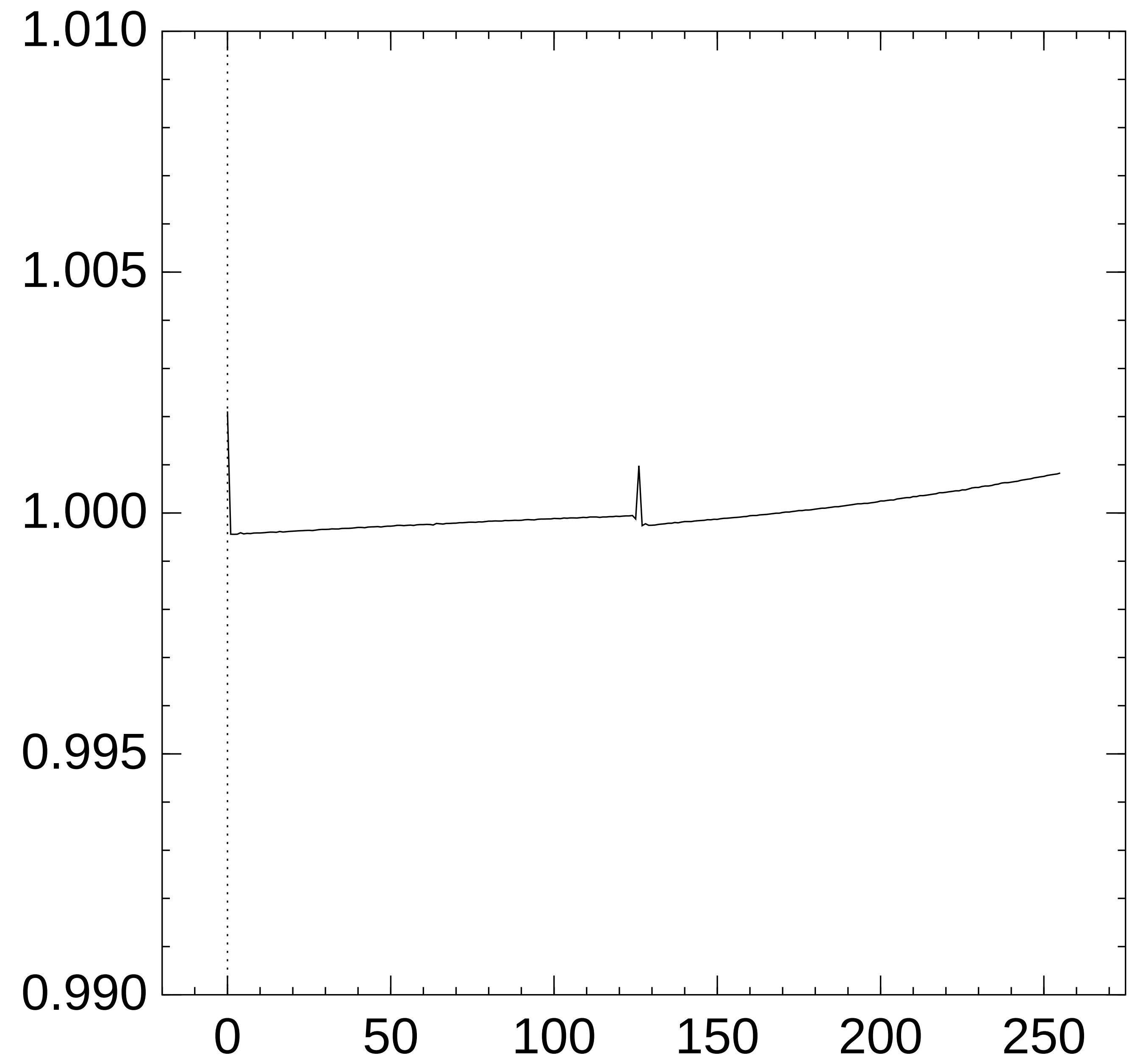
Graph of $256 \Pr[z_{125} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

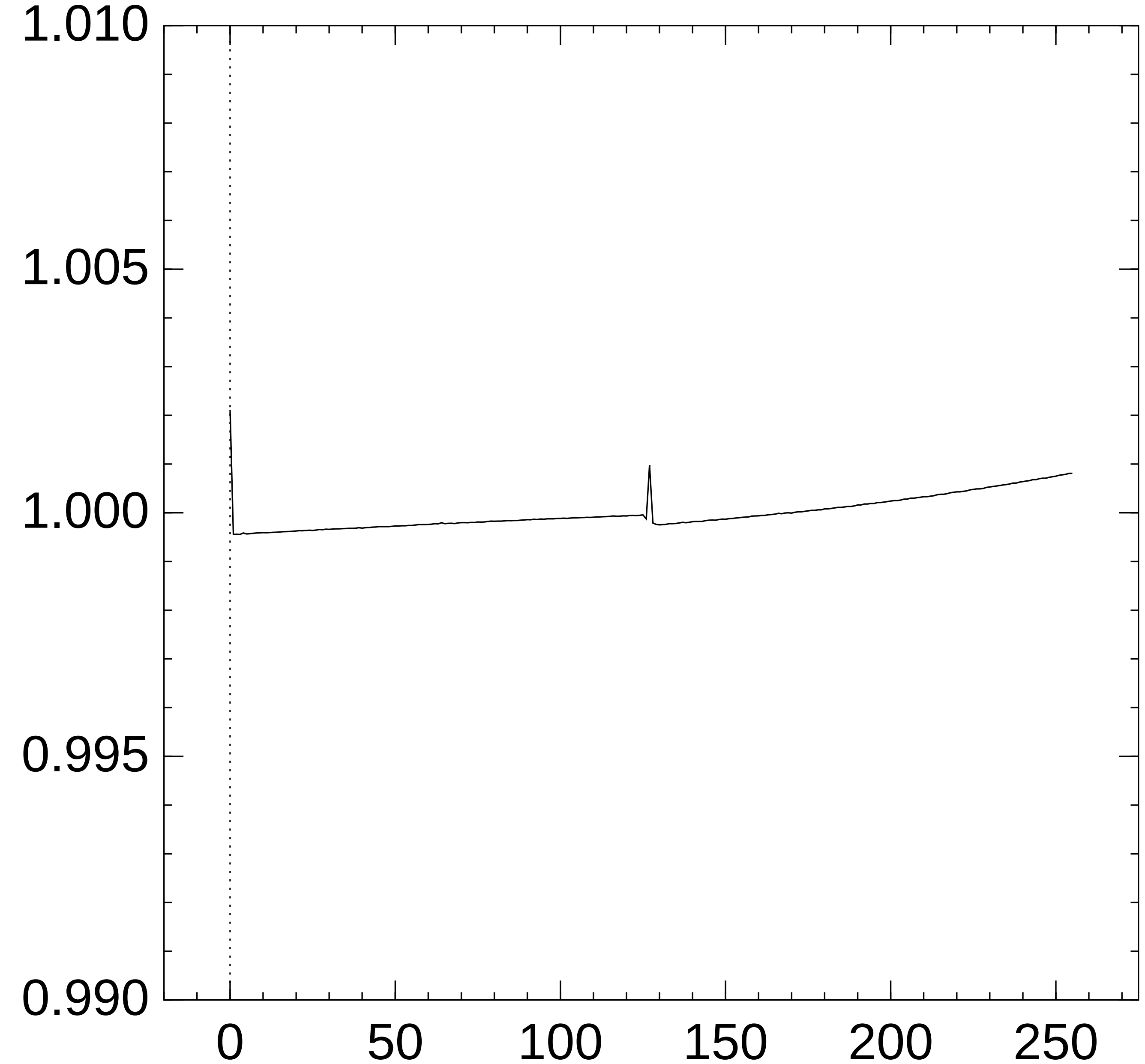
Graph of $256 \Pr[z_{126} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

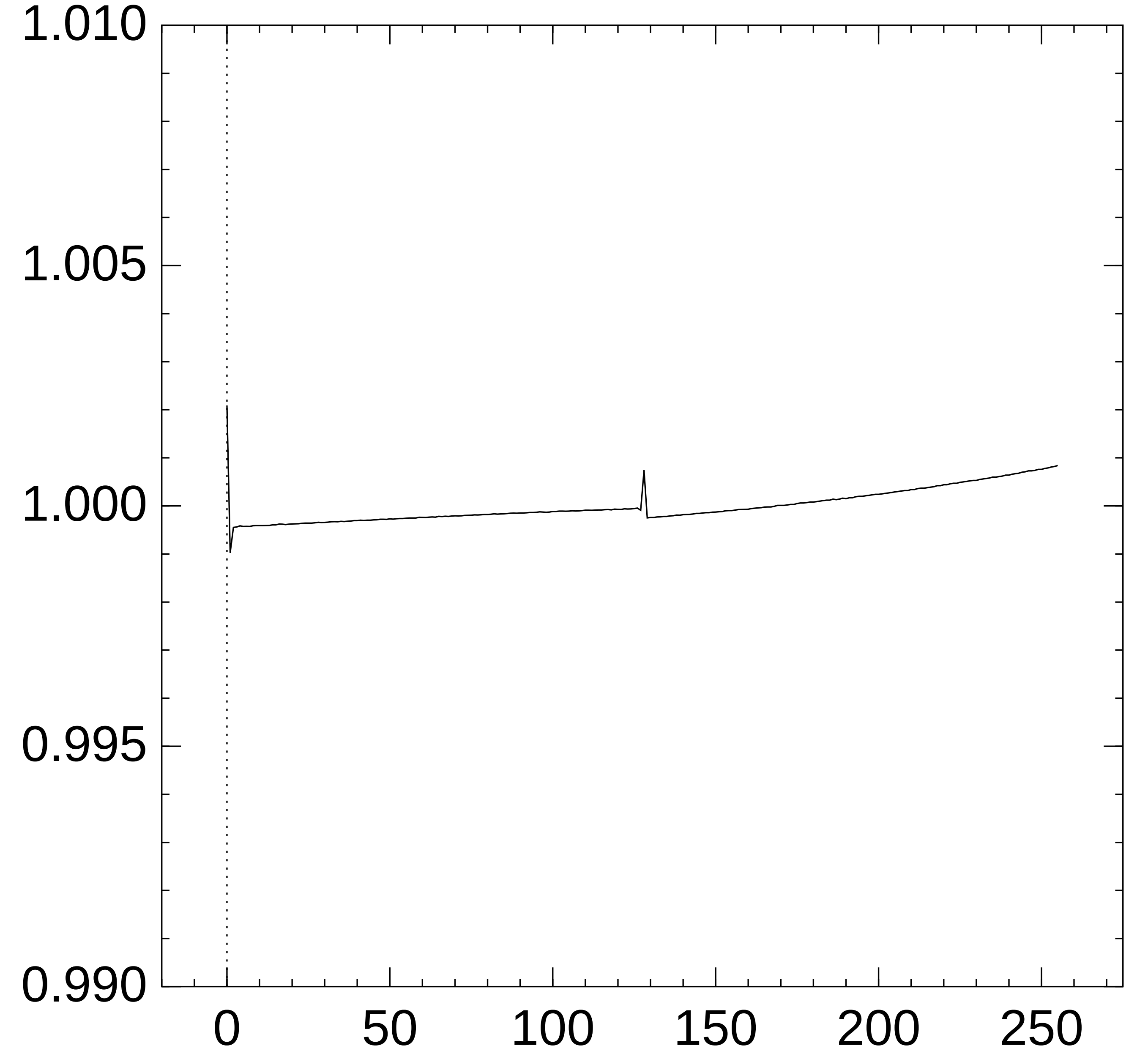
Graph of $256 \Pr[z_{127} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

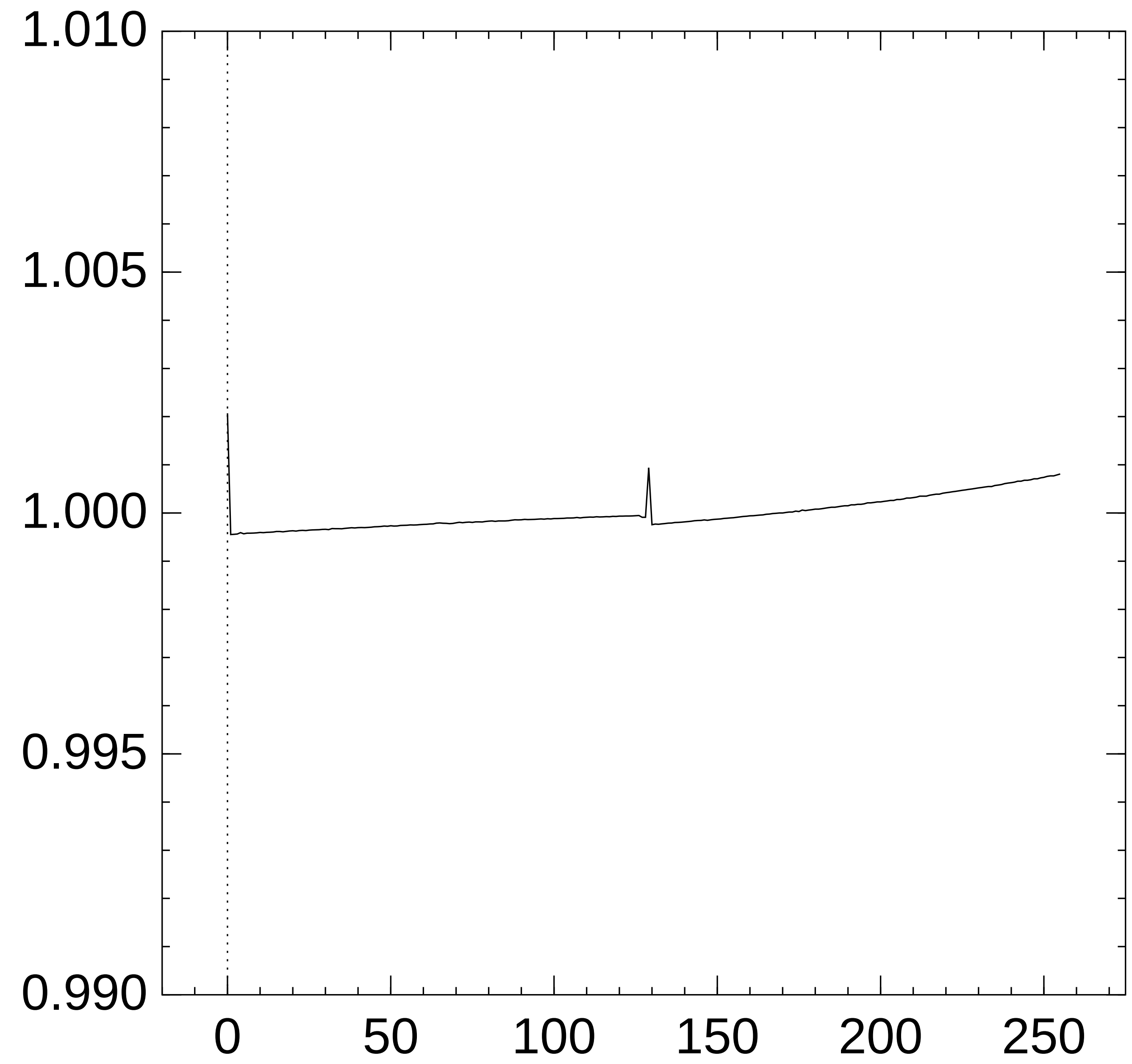
Graph of $256 \Pr[z_{128} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

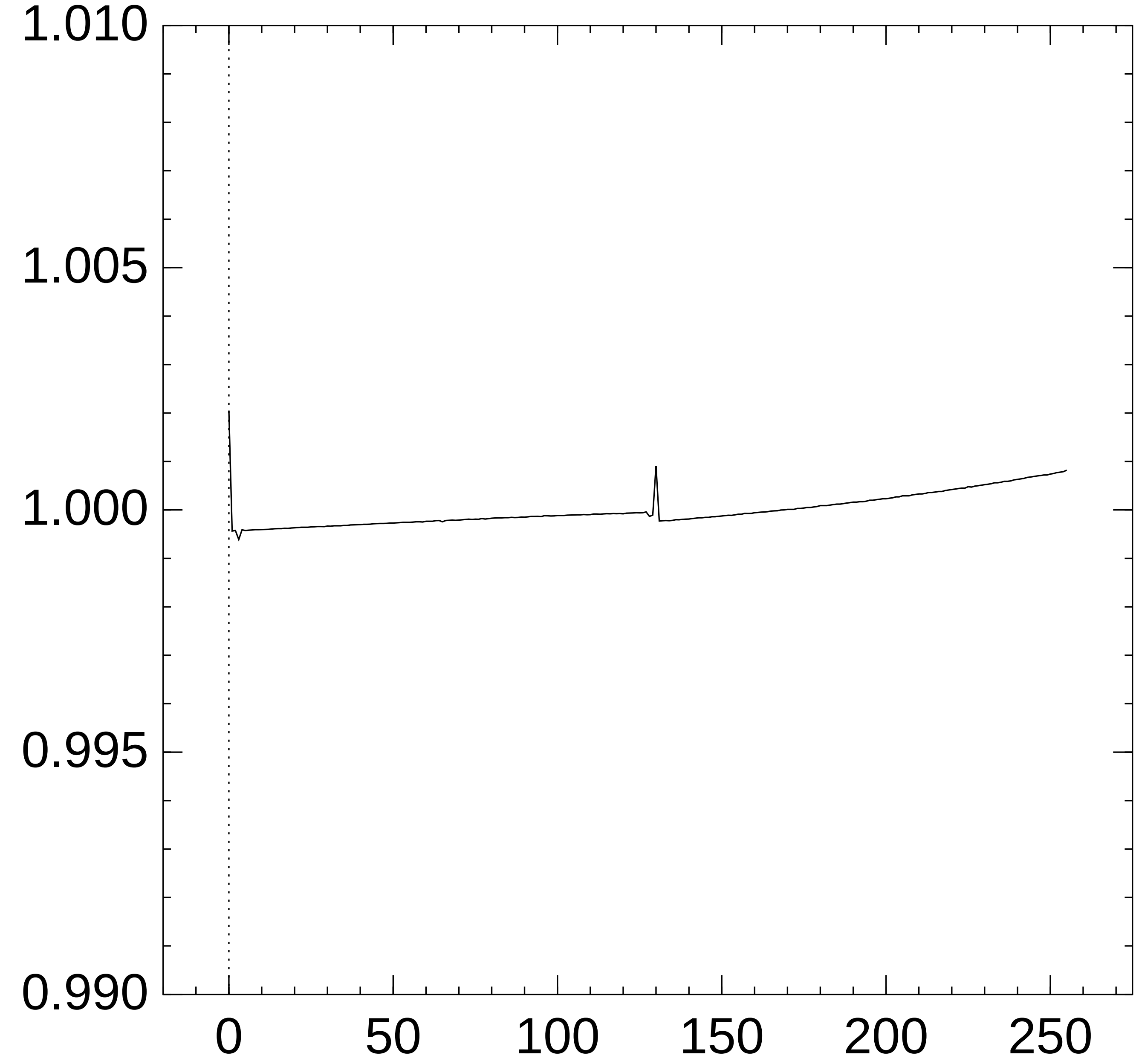
Graph of $256 \Pr[z_{129} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

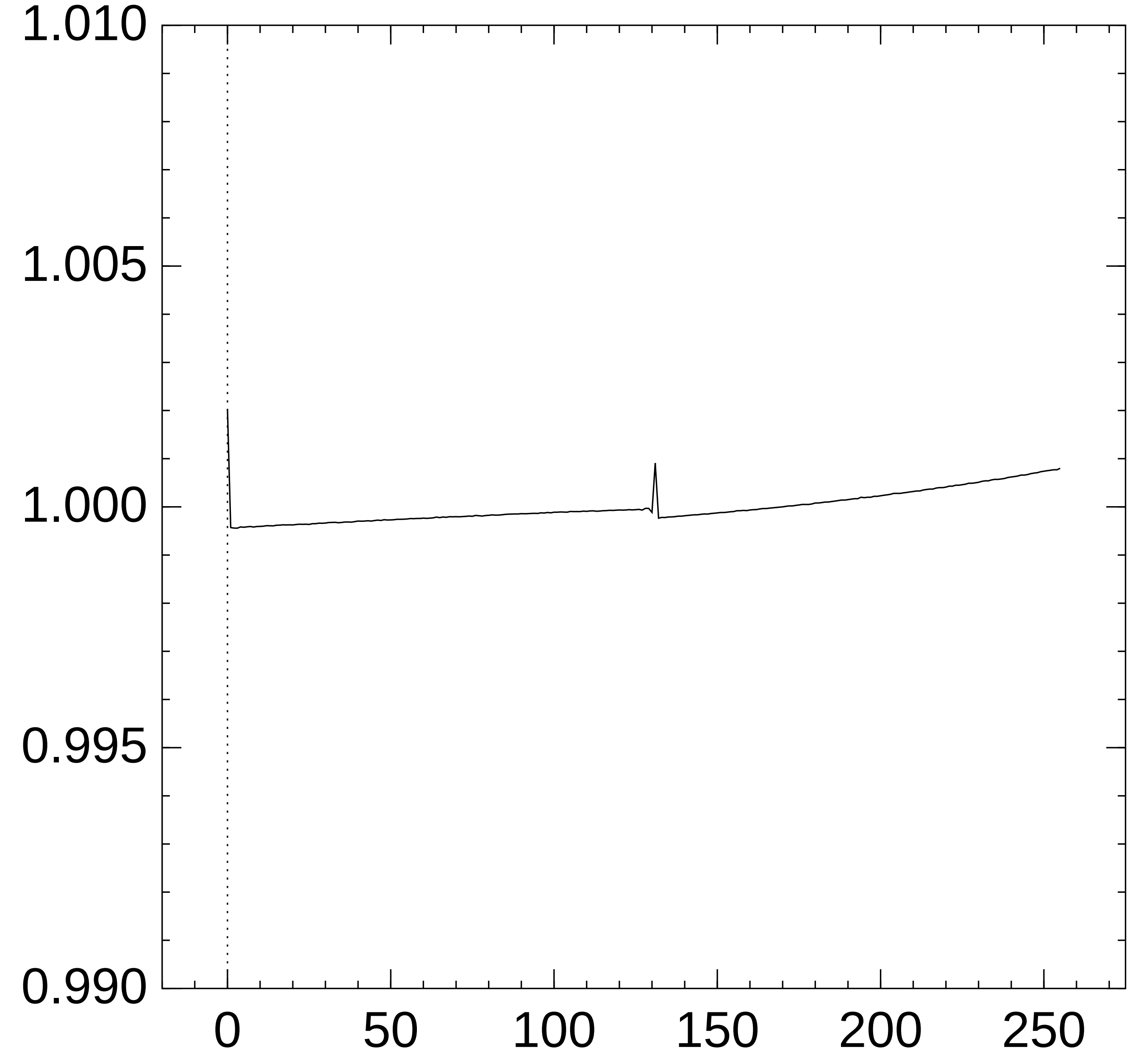
Graph of $256 \Pr[z_{130} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

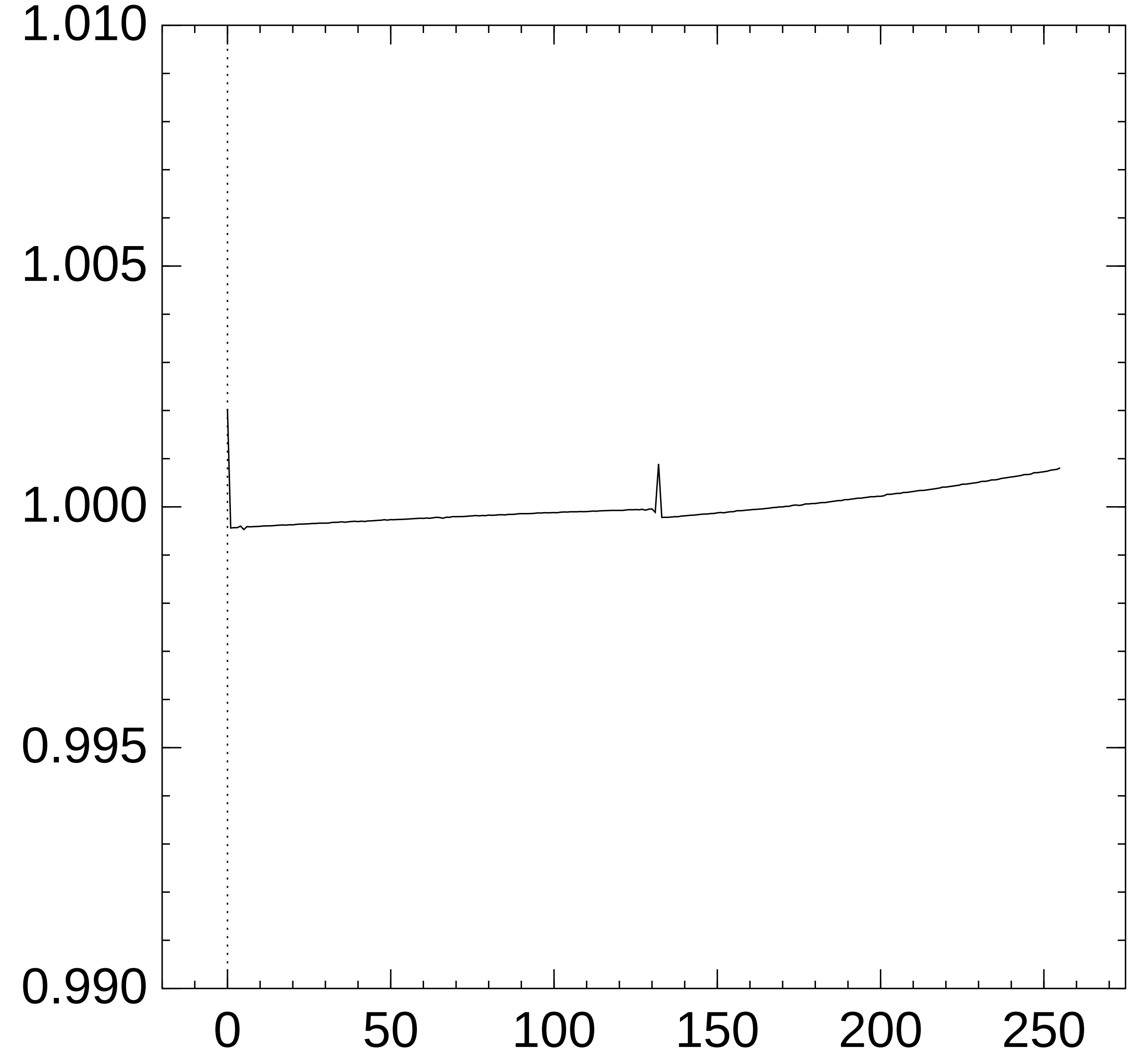
Graph of $256 \Pr[z_{131} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

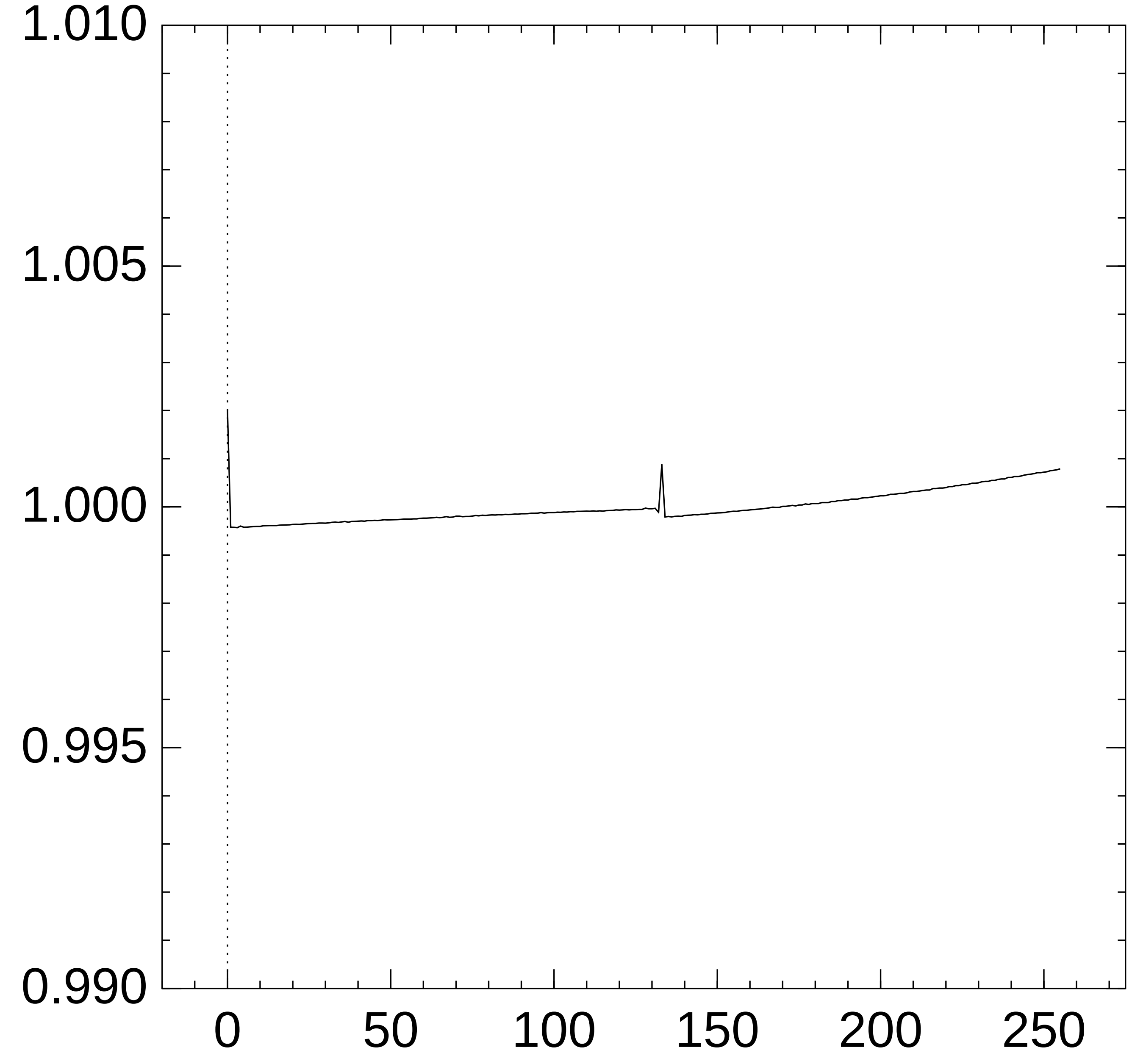
Graph of $256 \Pr[z_{132} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

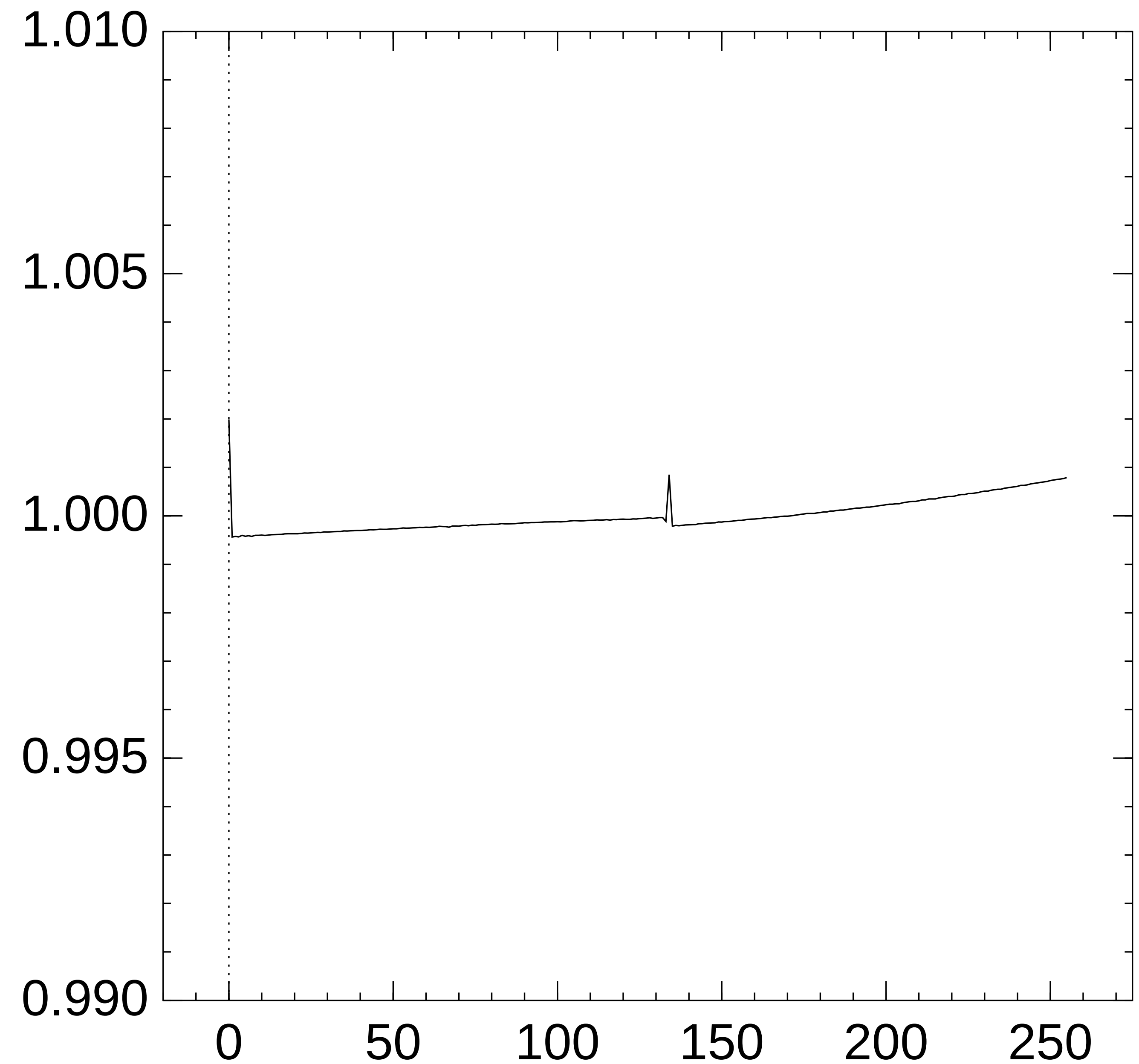
Graph of $256 \Pr[z_{133} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

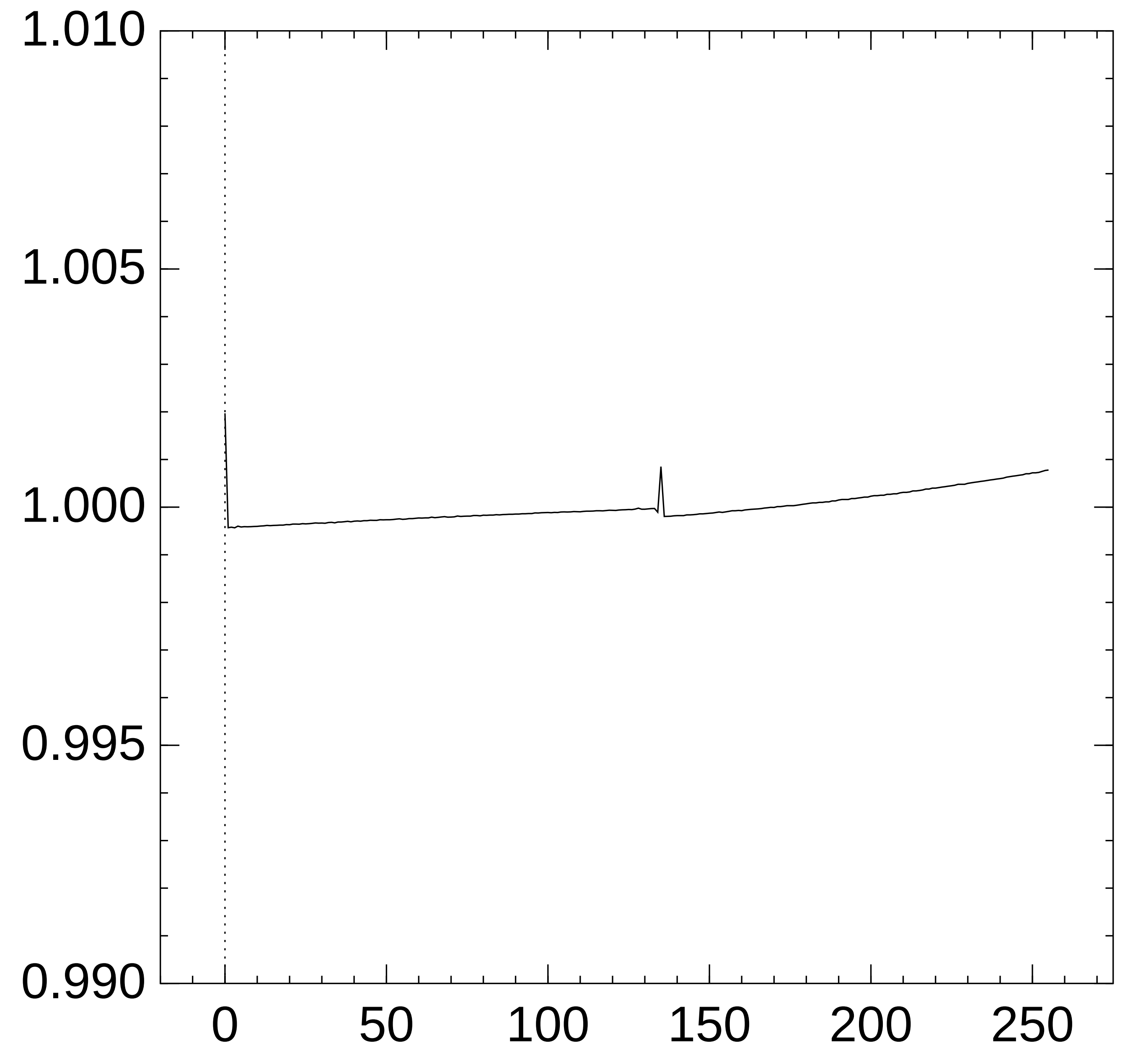
Graph of $256 \Pr[z_{134} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

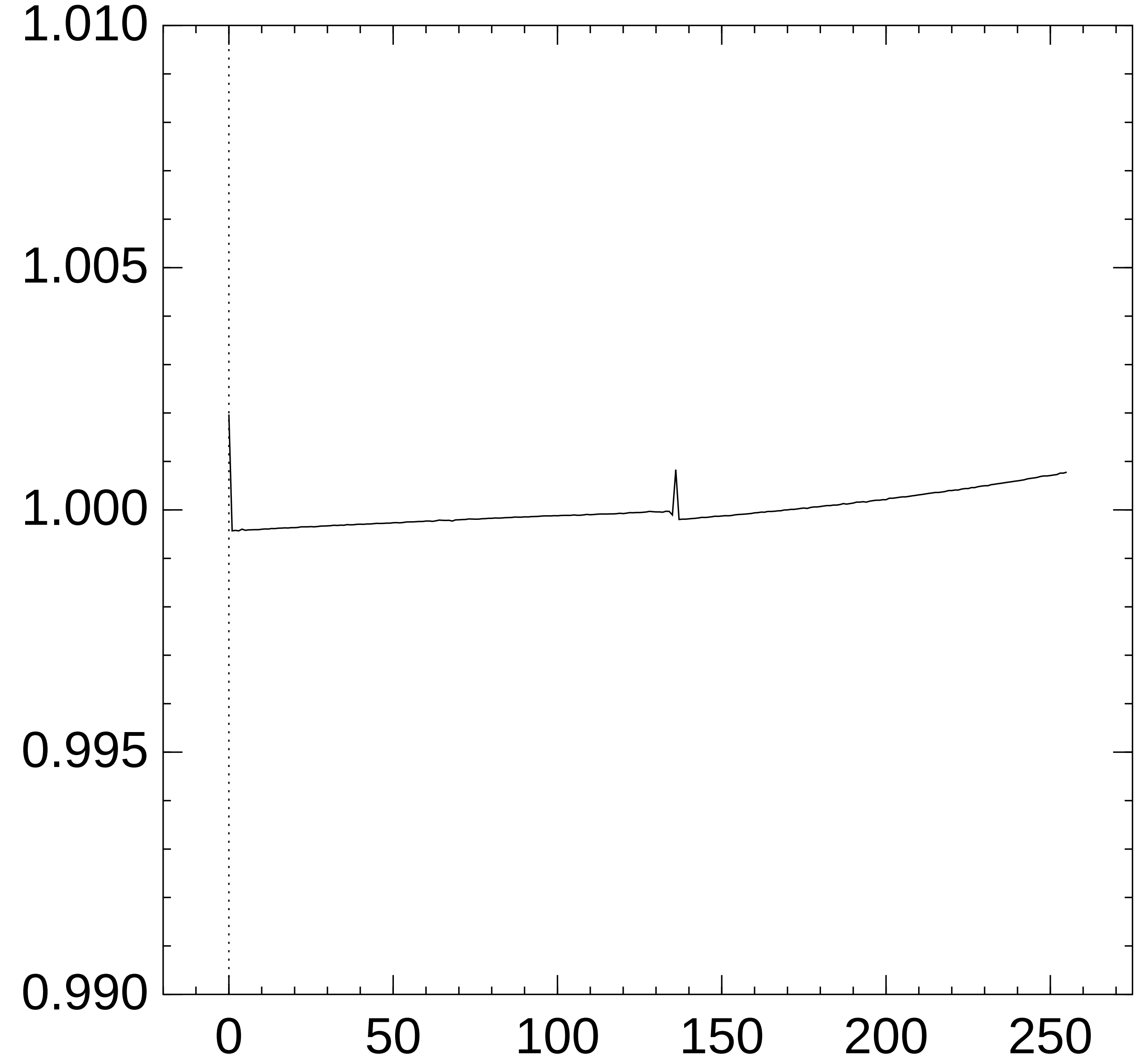
Graph of $256 \Pr[z_{135} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

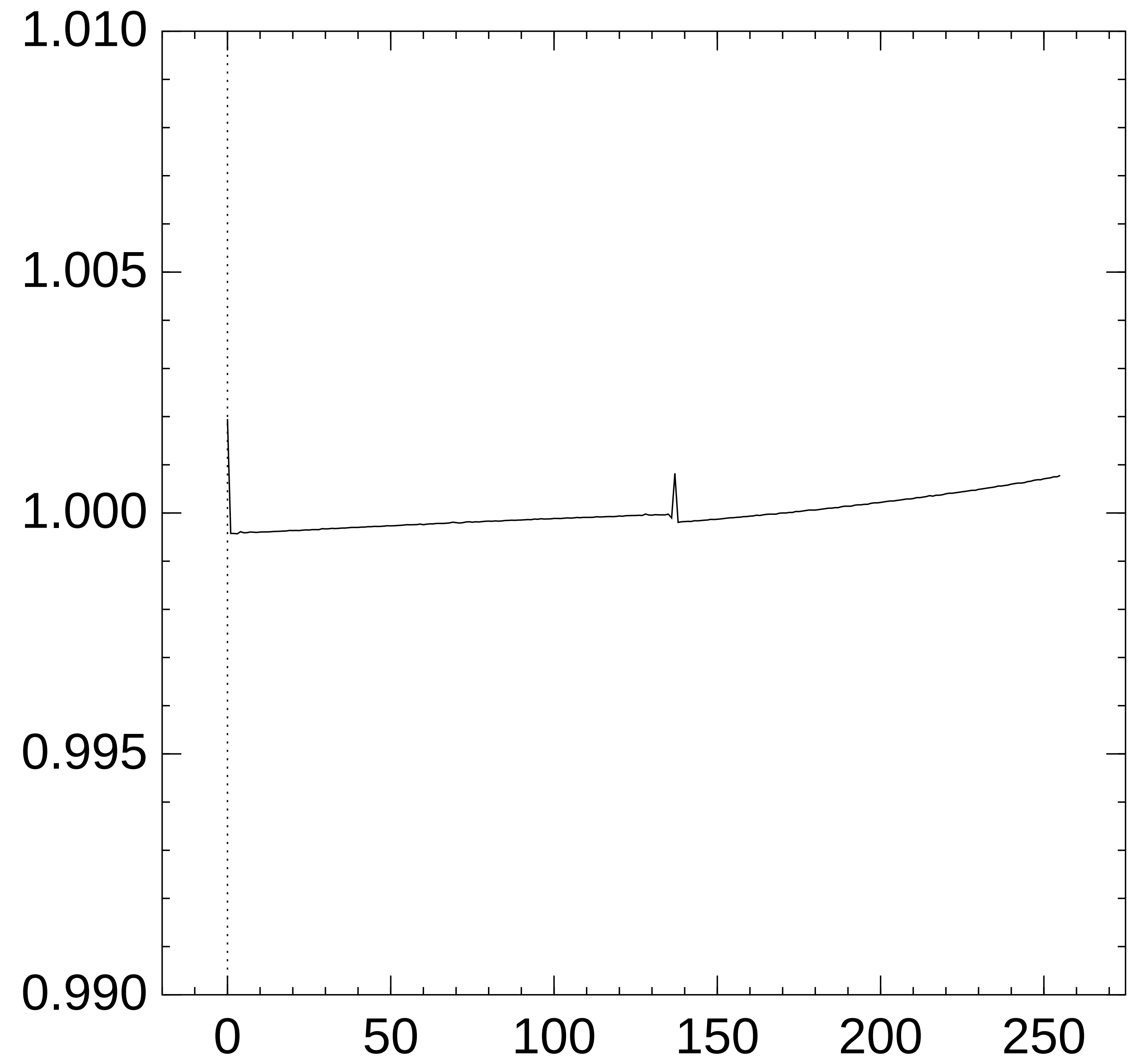
Graph of $256 \Pr[z_{136} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

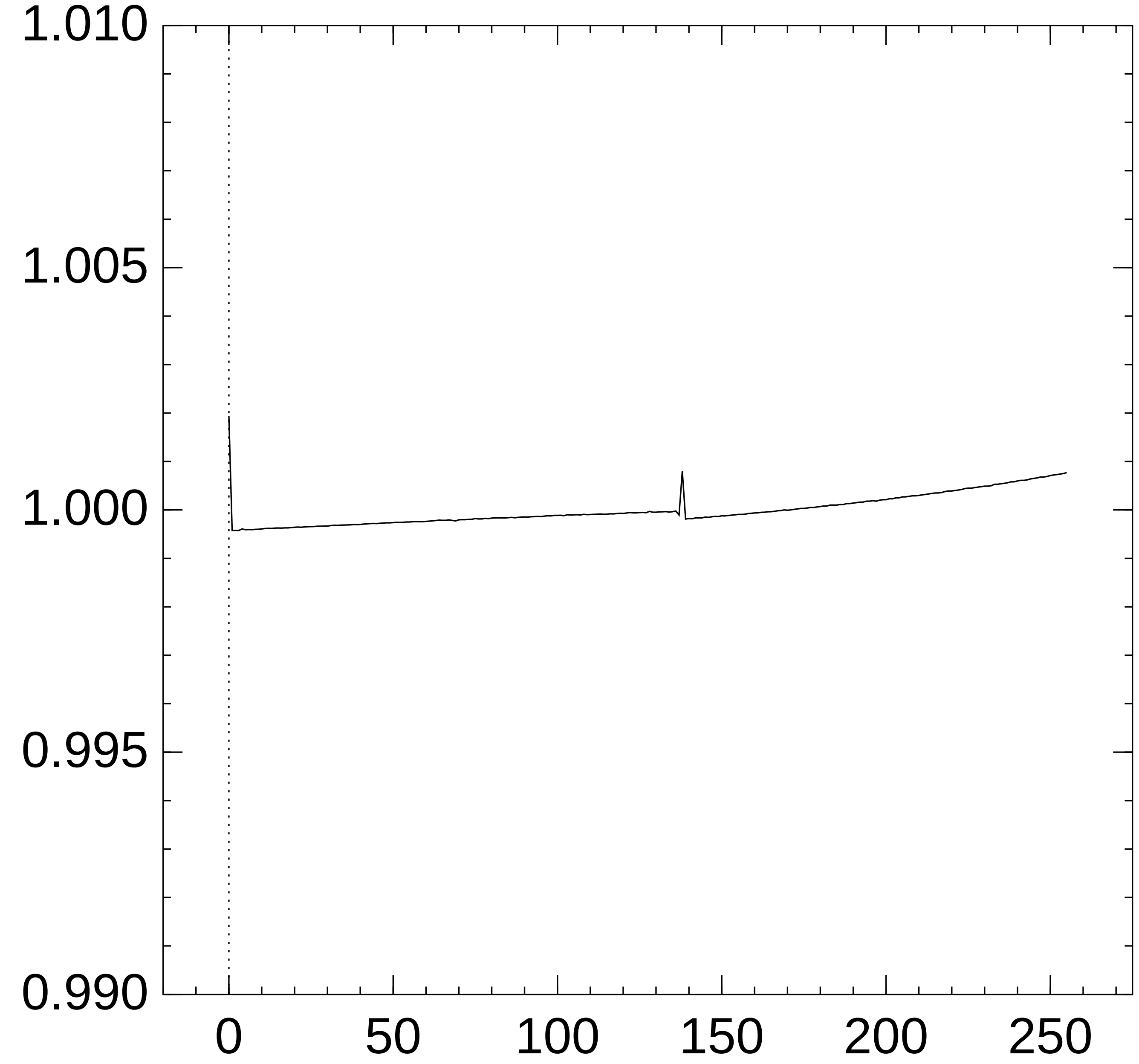
Graph of $256 \Pr[z_{137} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

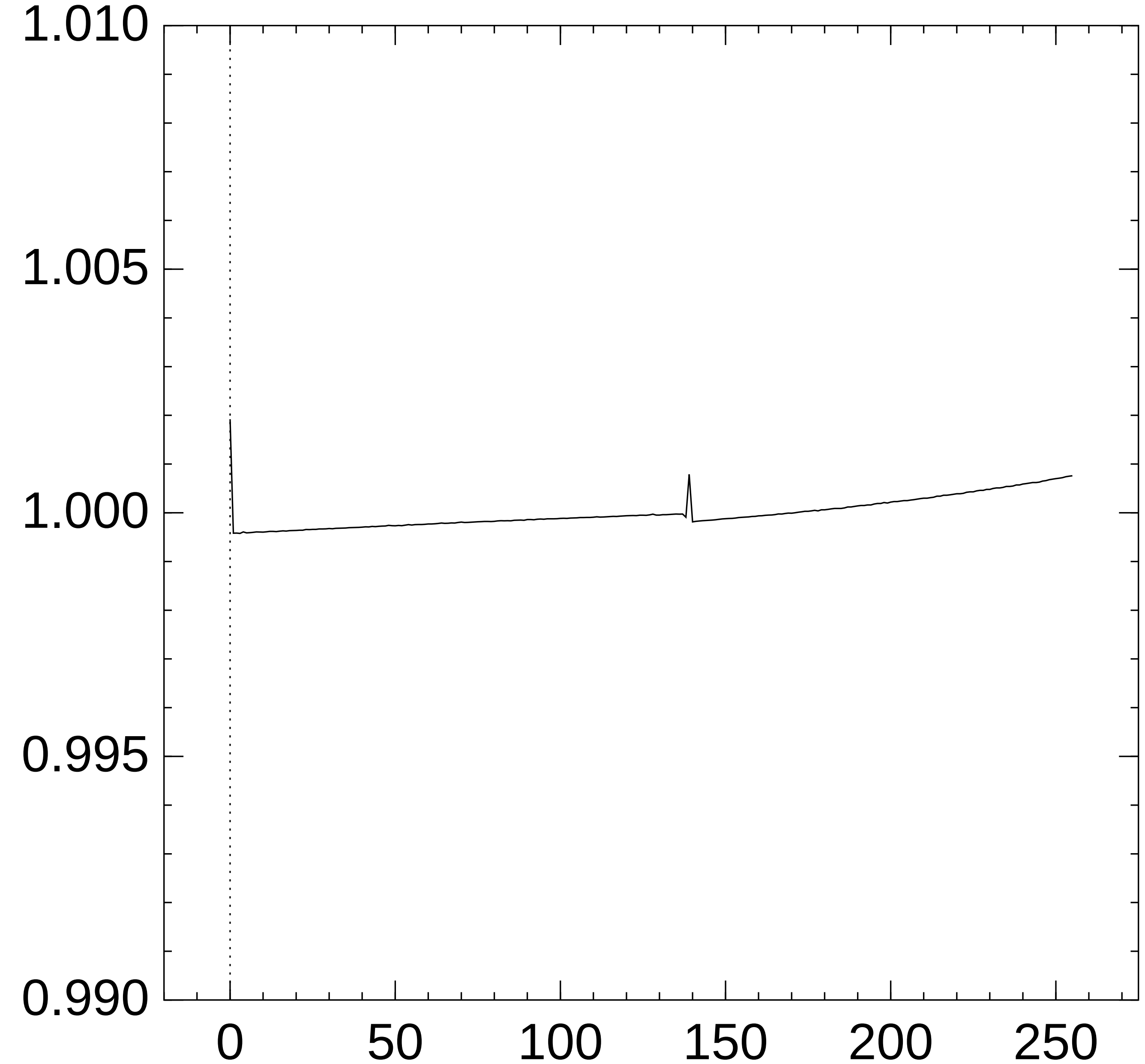
Graph of $256 \Pr[z_{138} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

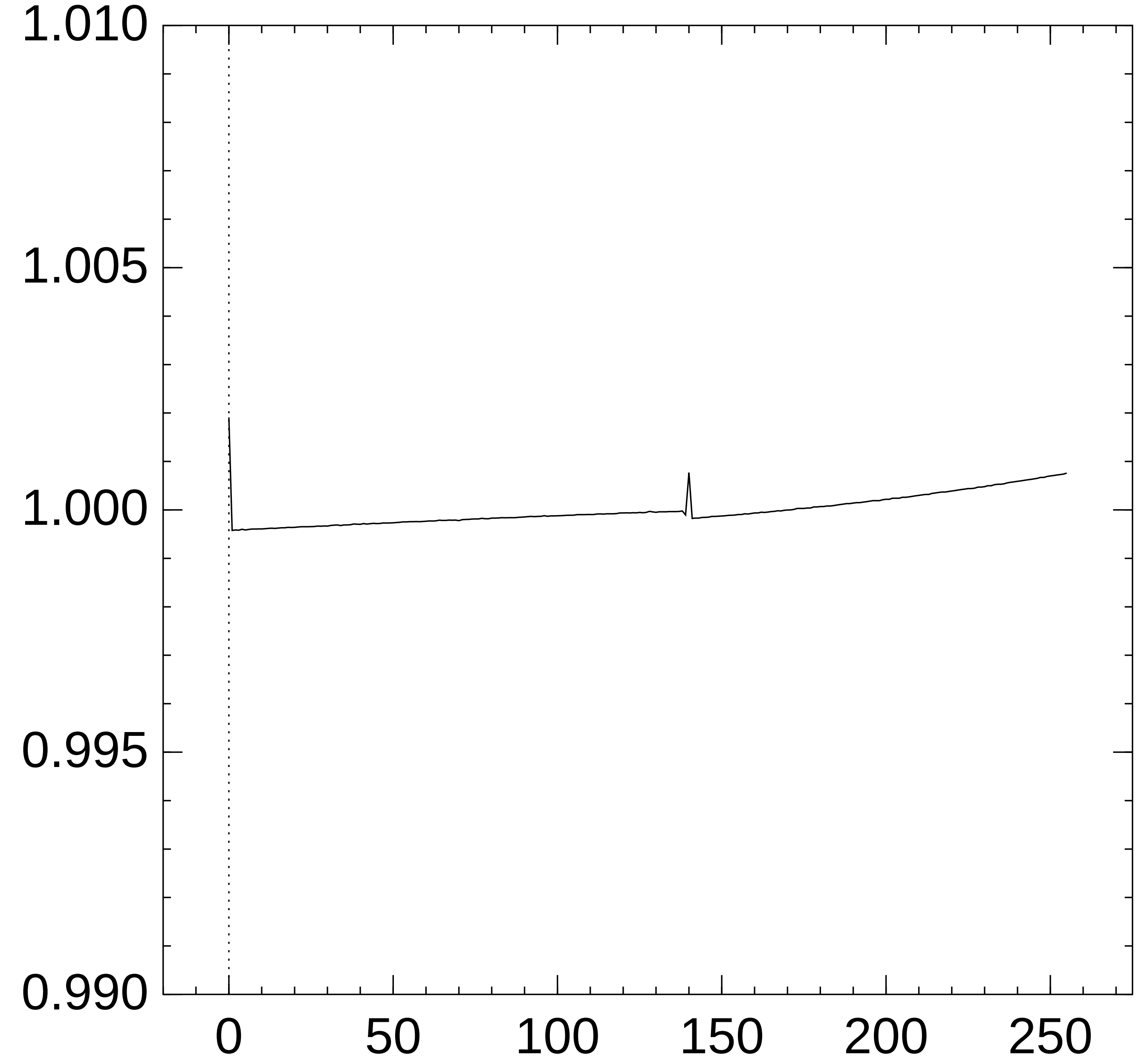
Graph of $256 \Pr[z_{139} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

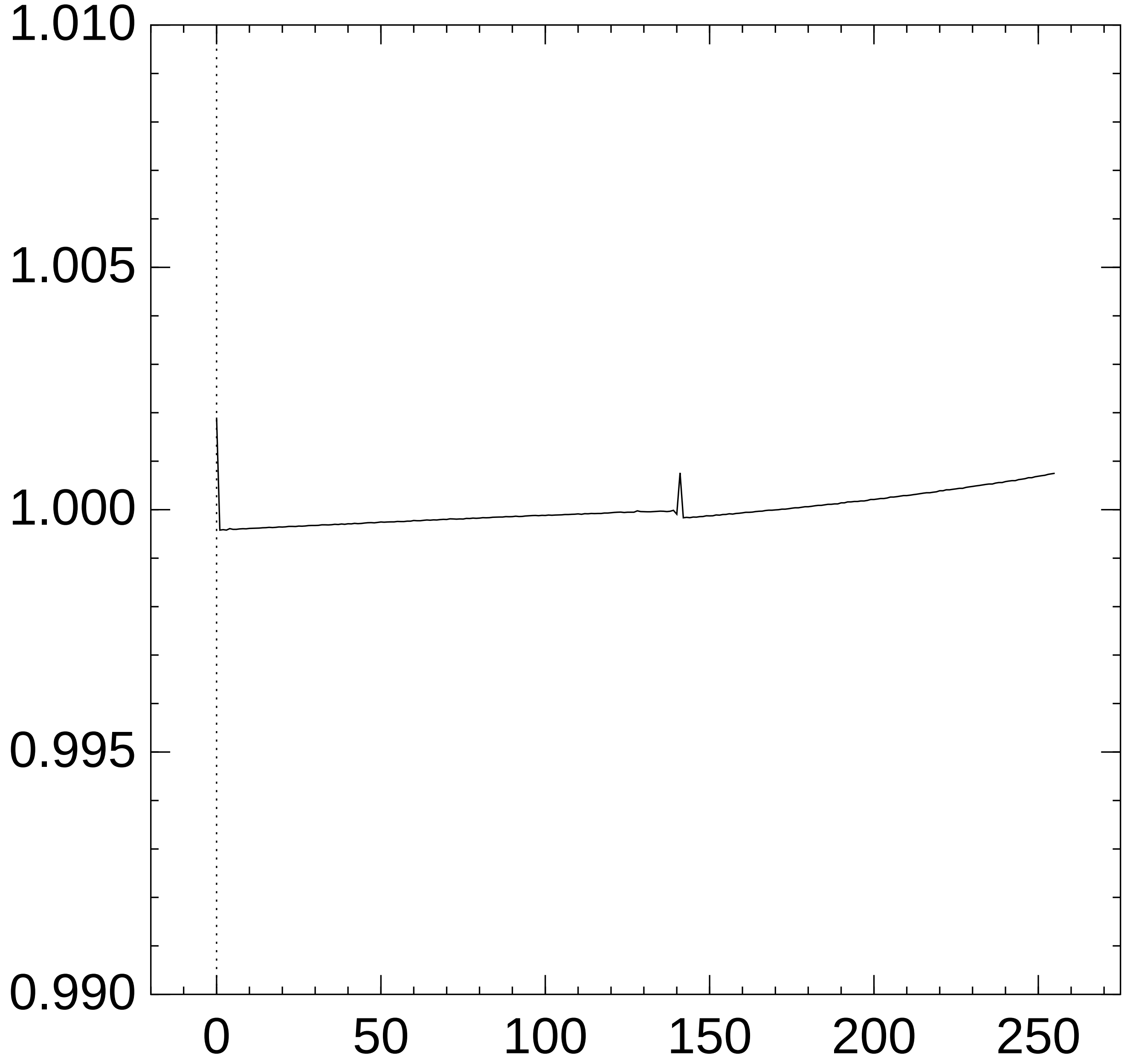
Graph of $256 \Pr[z_{140} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

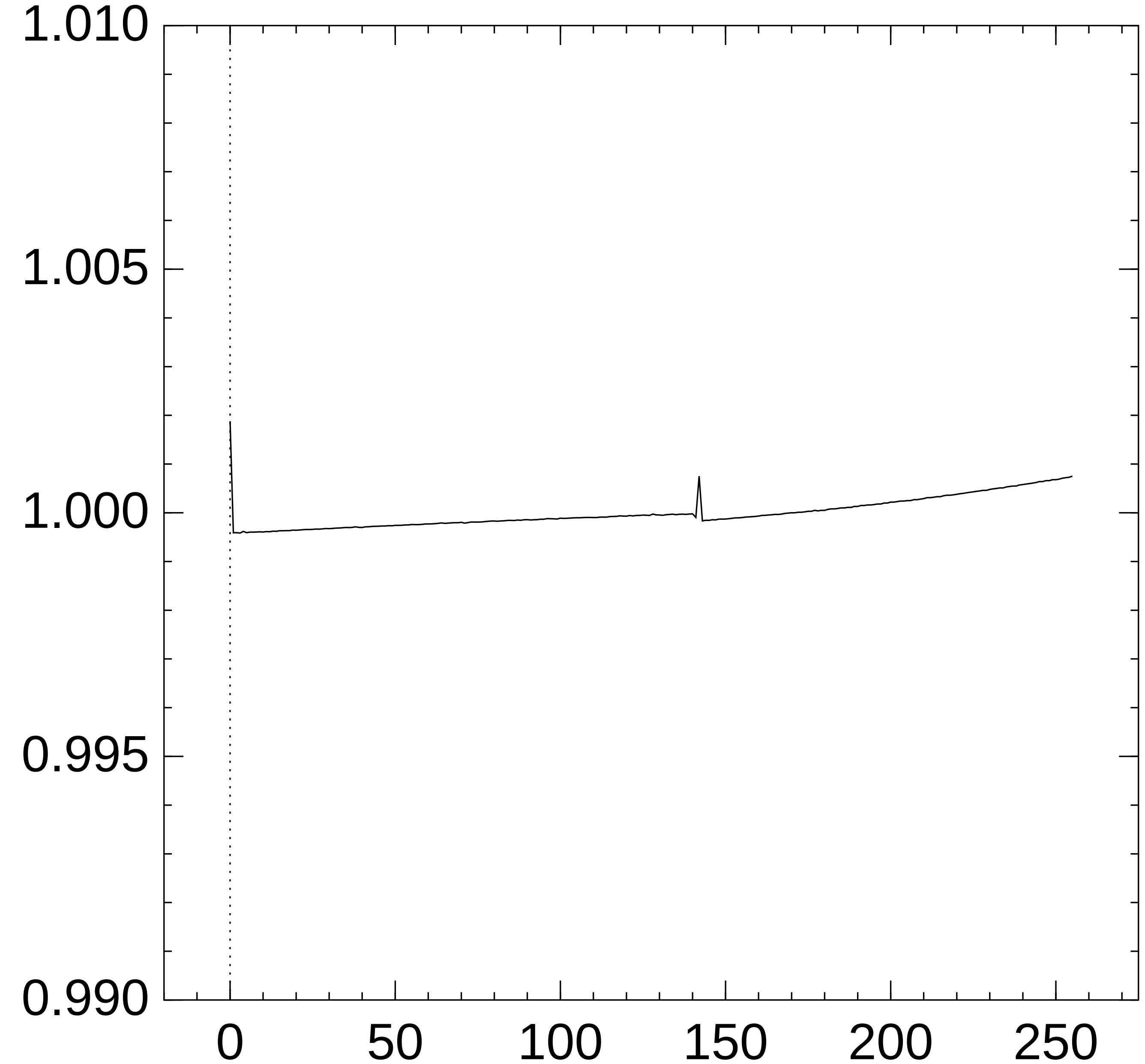
Graph of $256 \Pr[z_{141} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

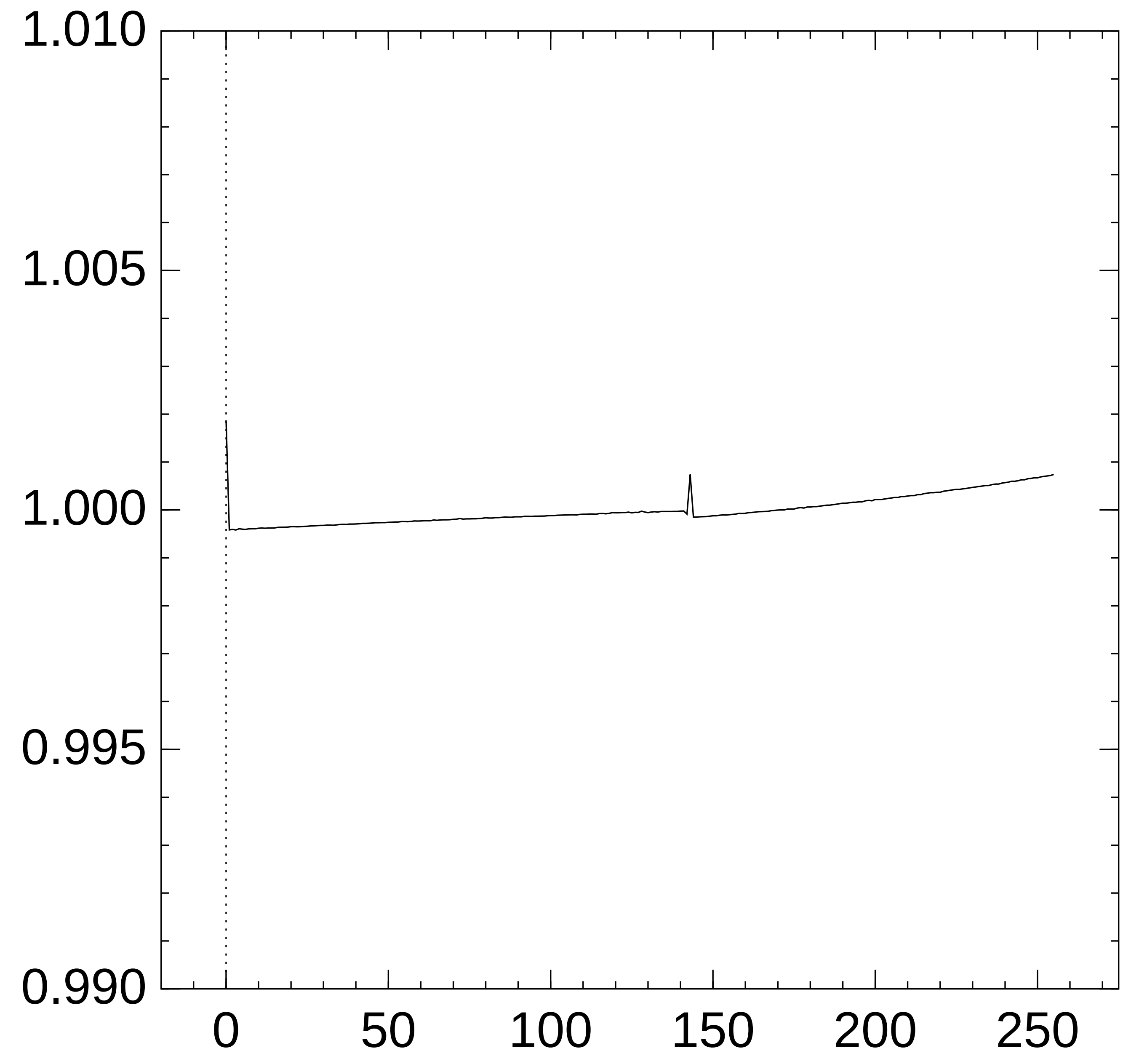
Graph of $256 \Pr[z_{142} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

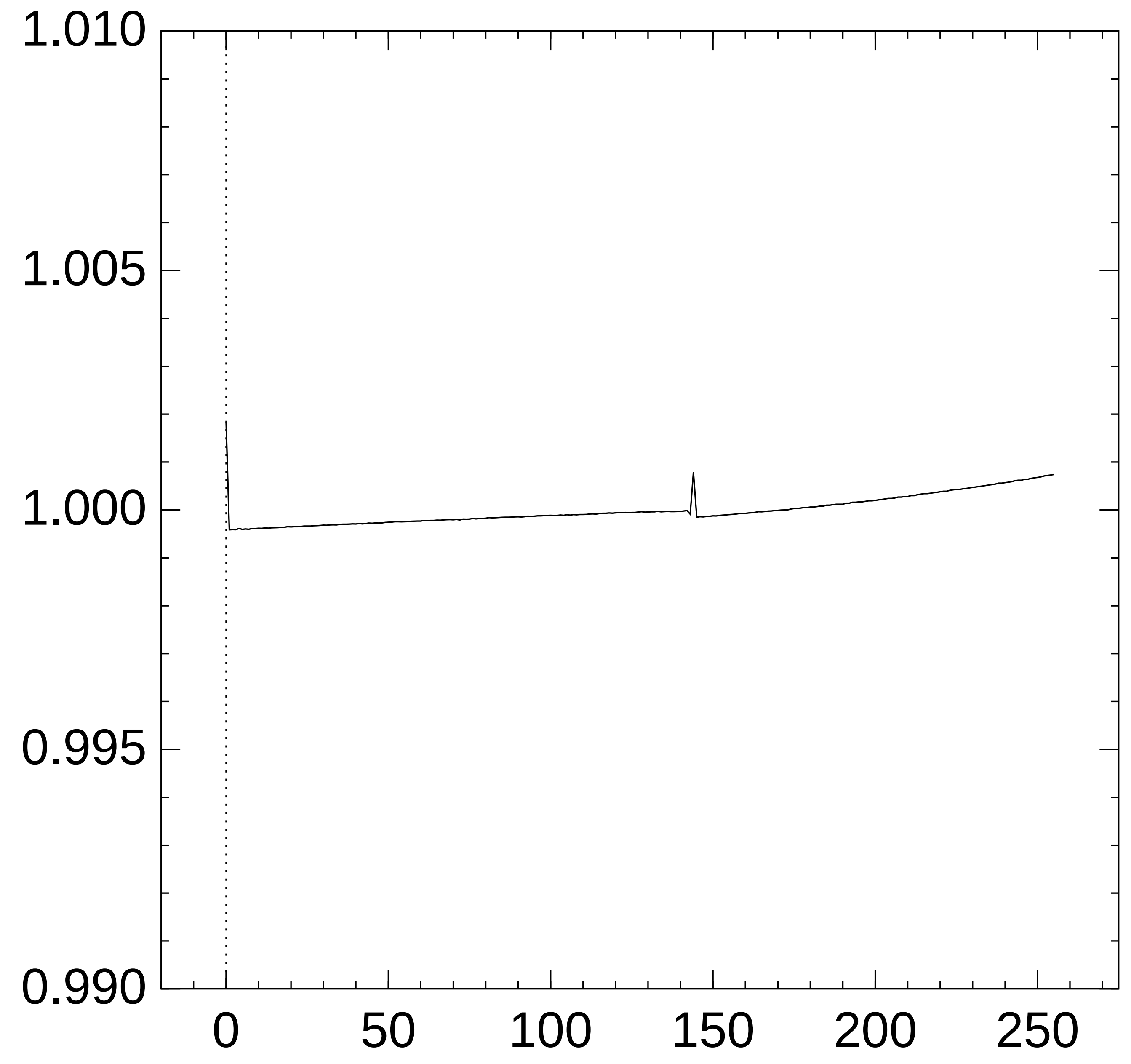
Graph of $256 \Pr[z_{143} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

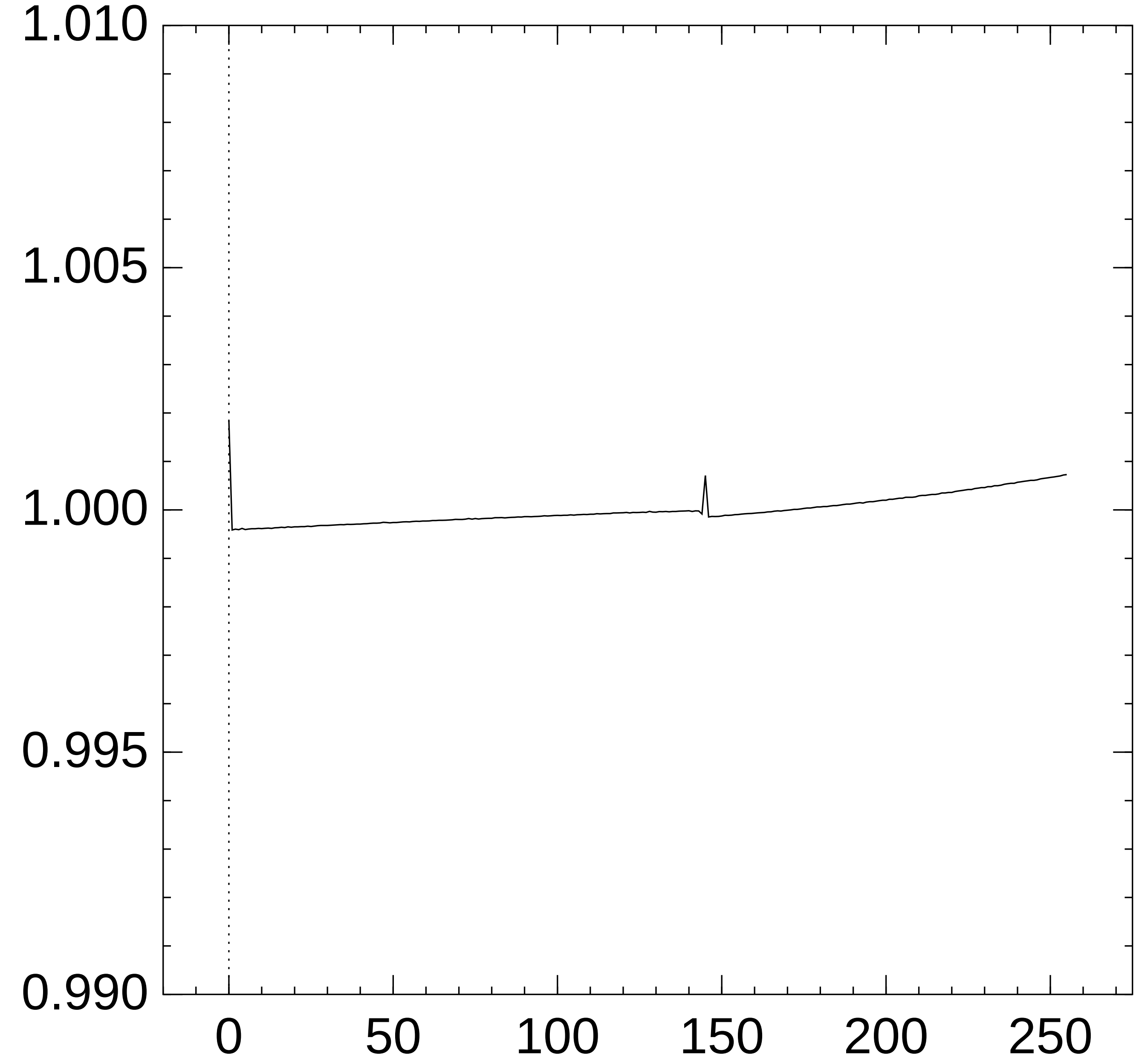
Graph of $256 \Pr[z_{144} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

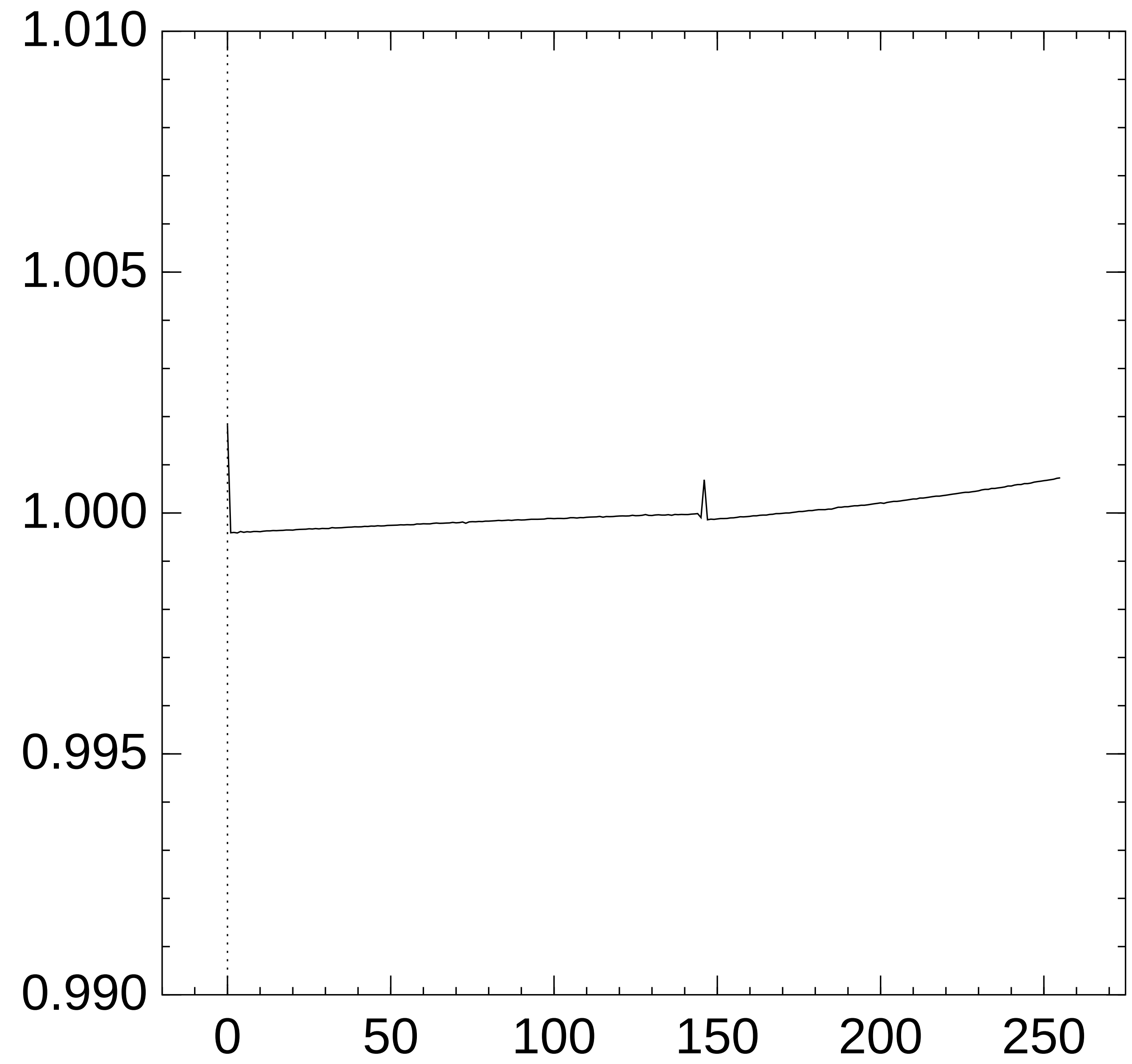
Graph of $256 \Pr[z_{145} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

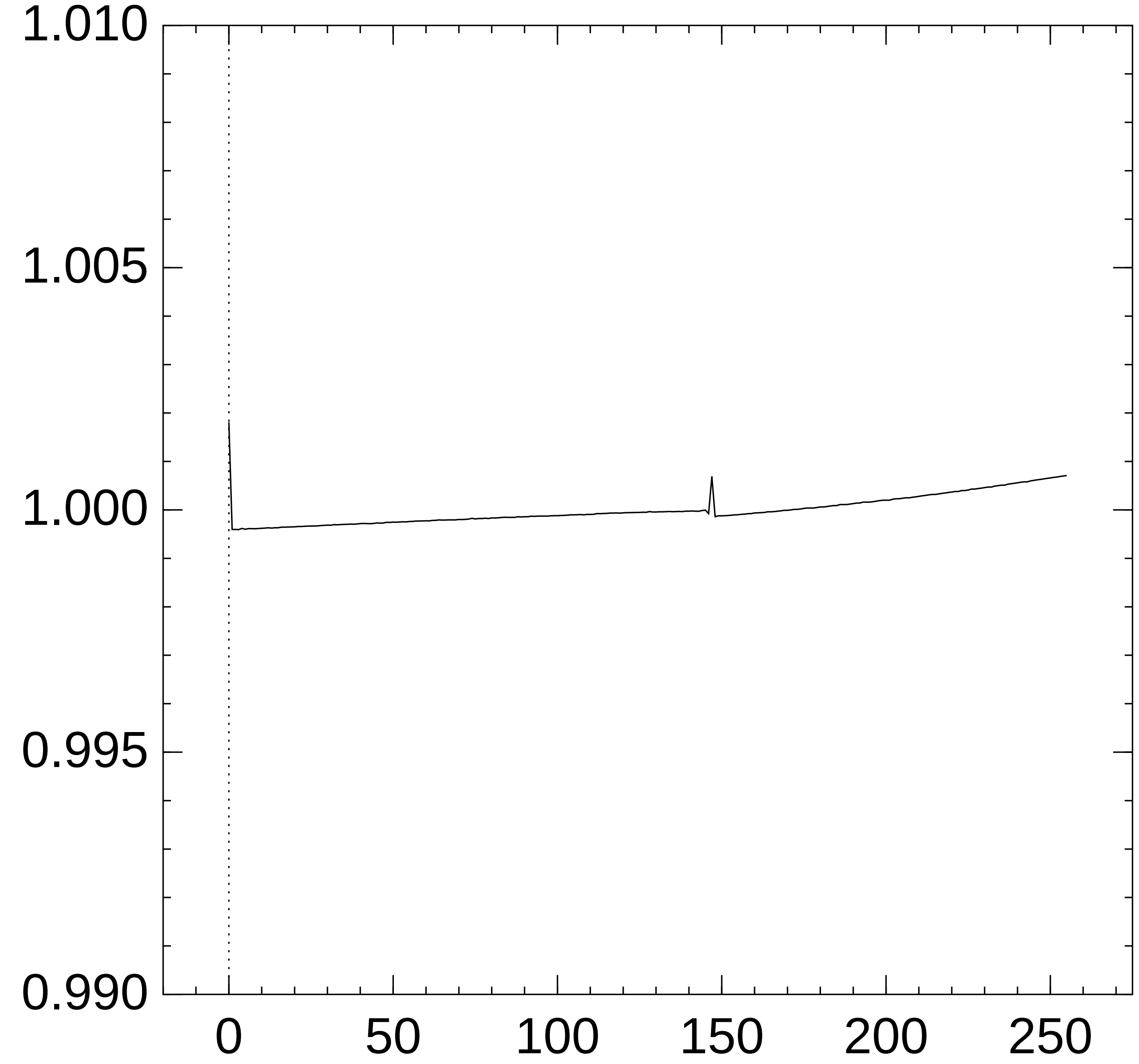
Graph of $256 \Pr[z_{146} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

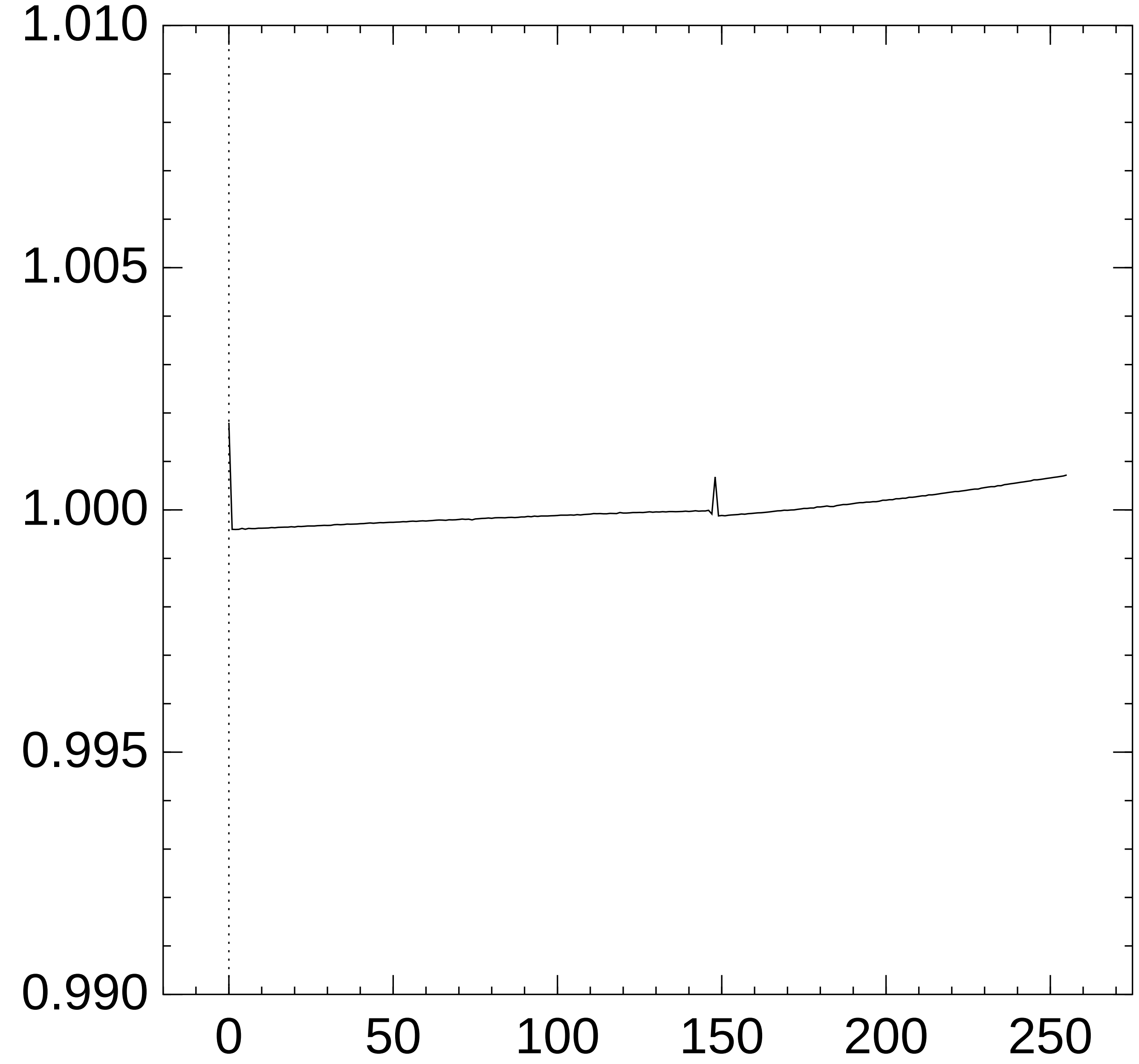
Graph of $256 \Pr[z_{147} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

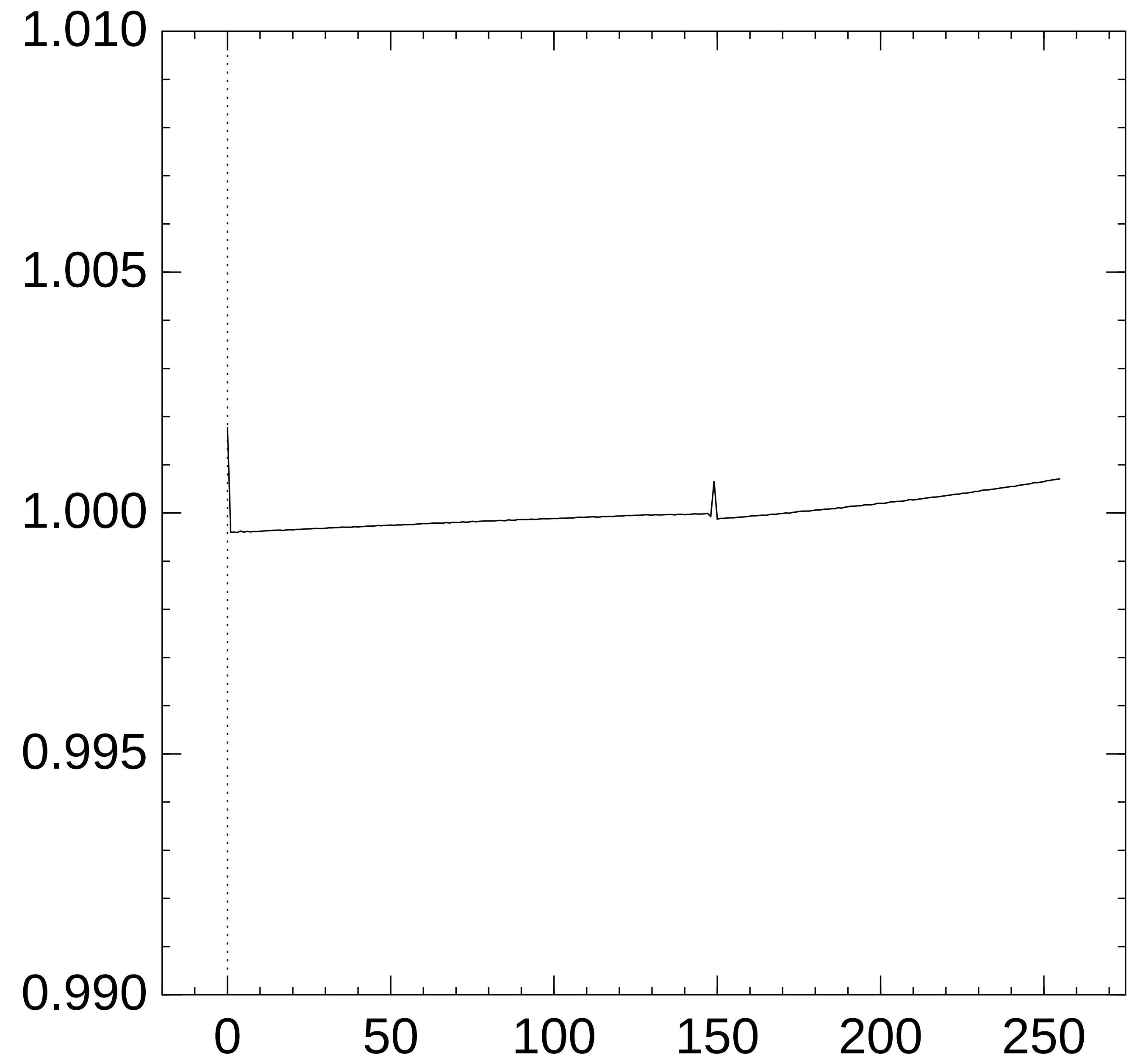
Graph of $256 \Pr[z_{148} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

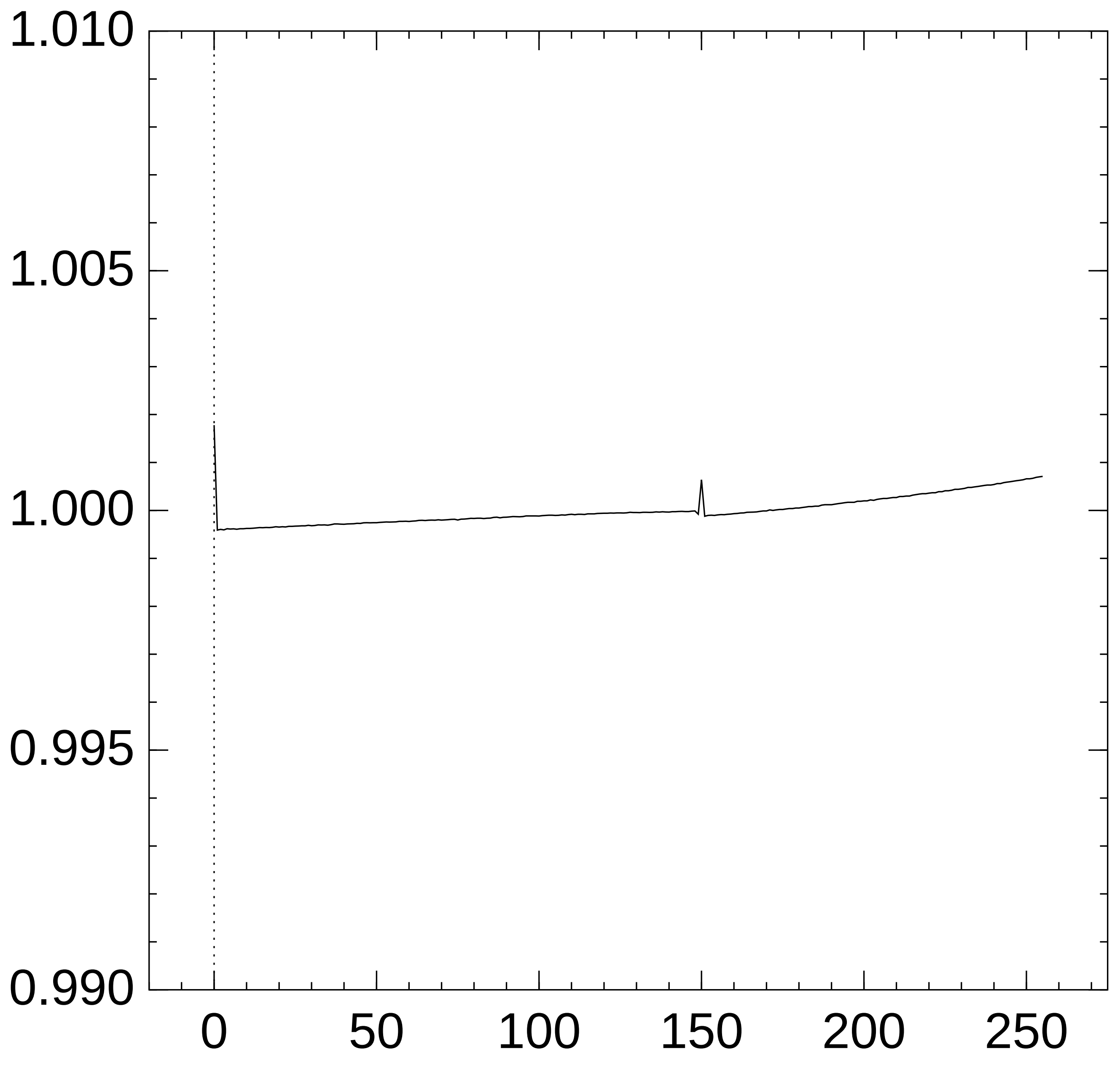
Graph of $256 \Pr[z_{149} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

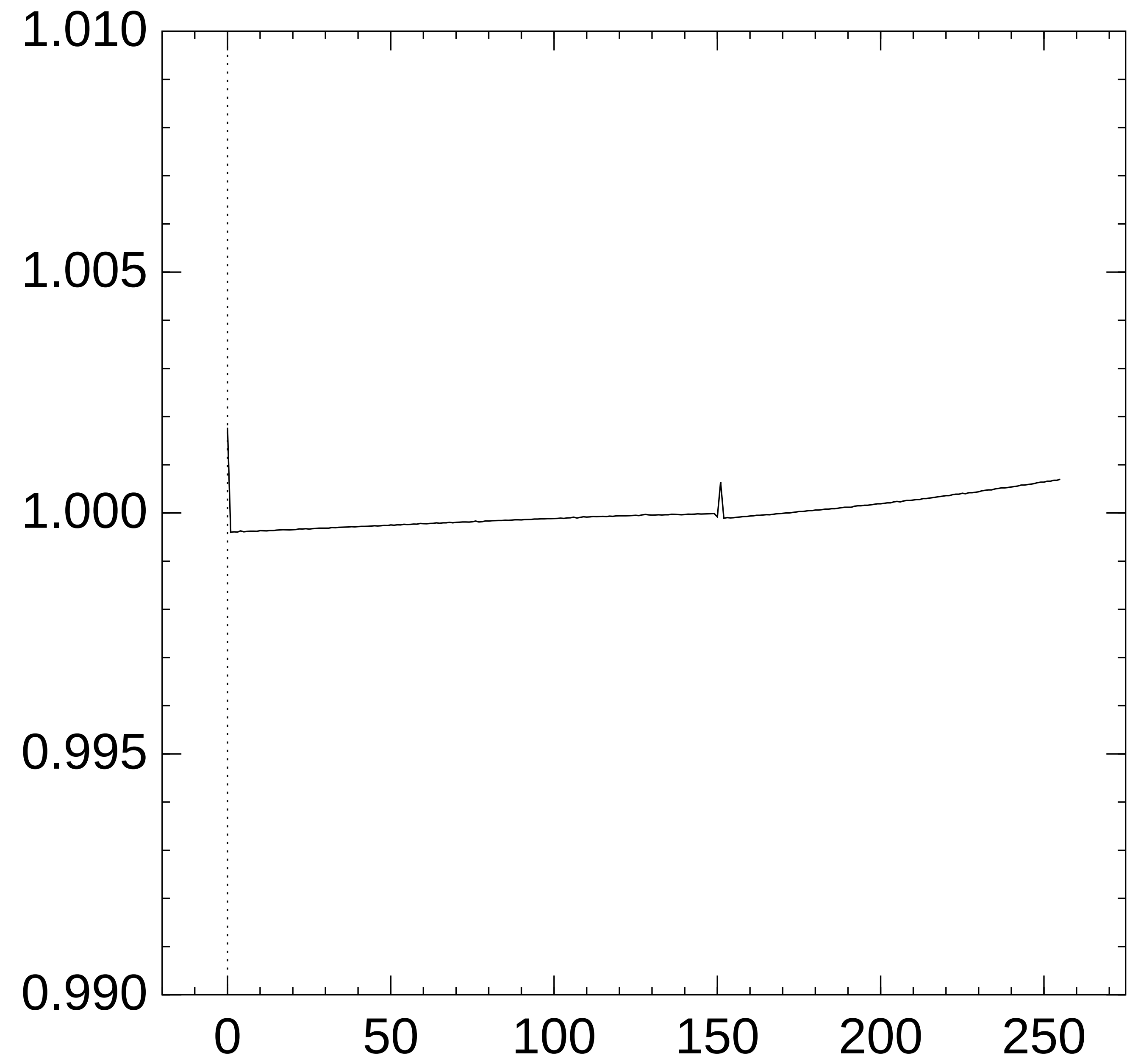
Graph of $256 \Pr[z_{150} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

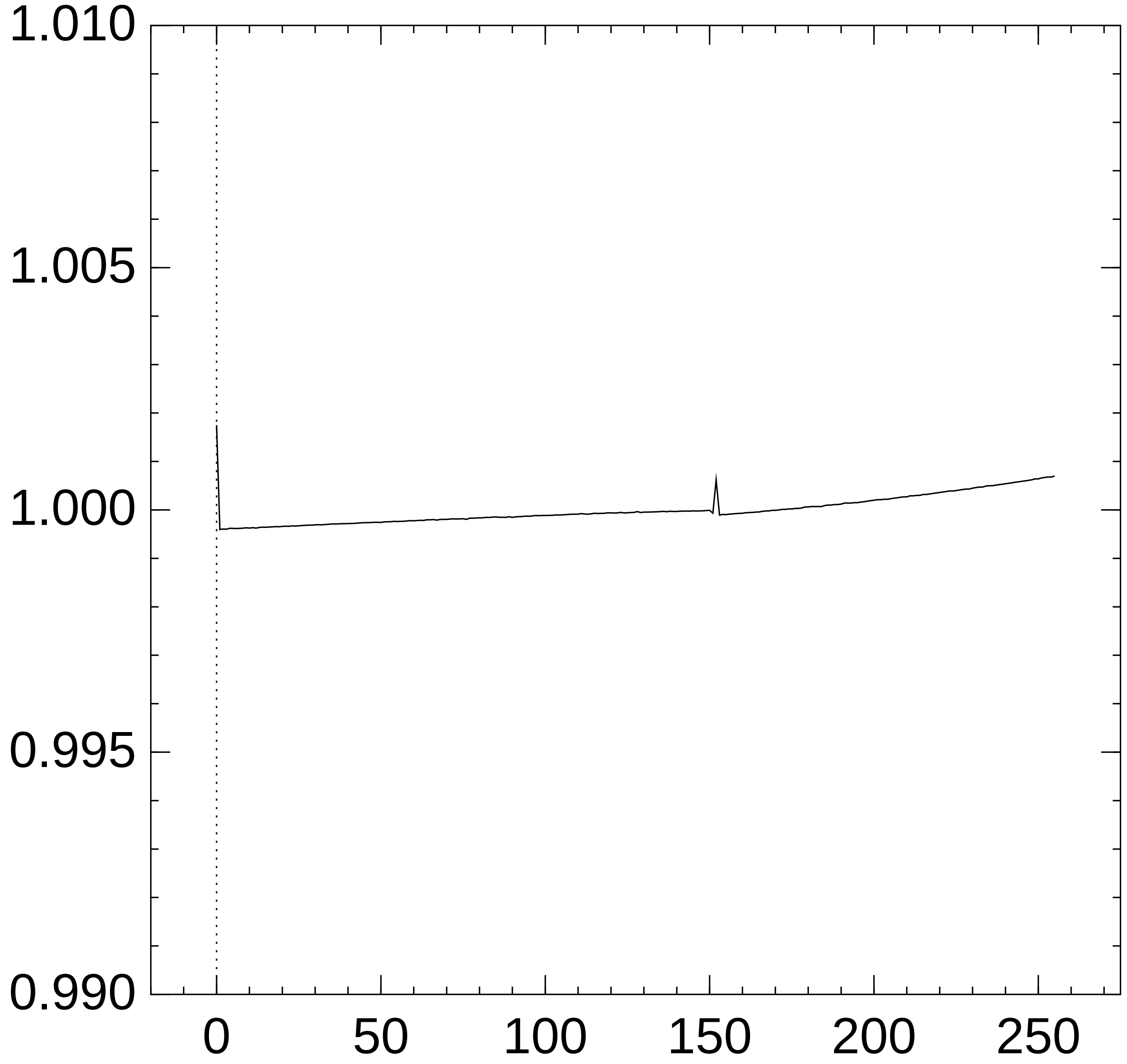
Graph of $256 \Pr[z_{151} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

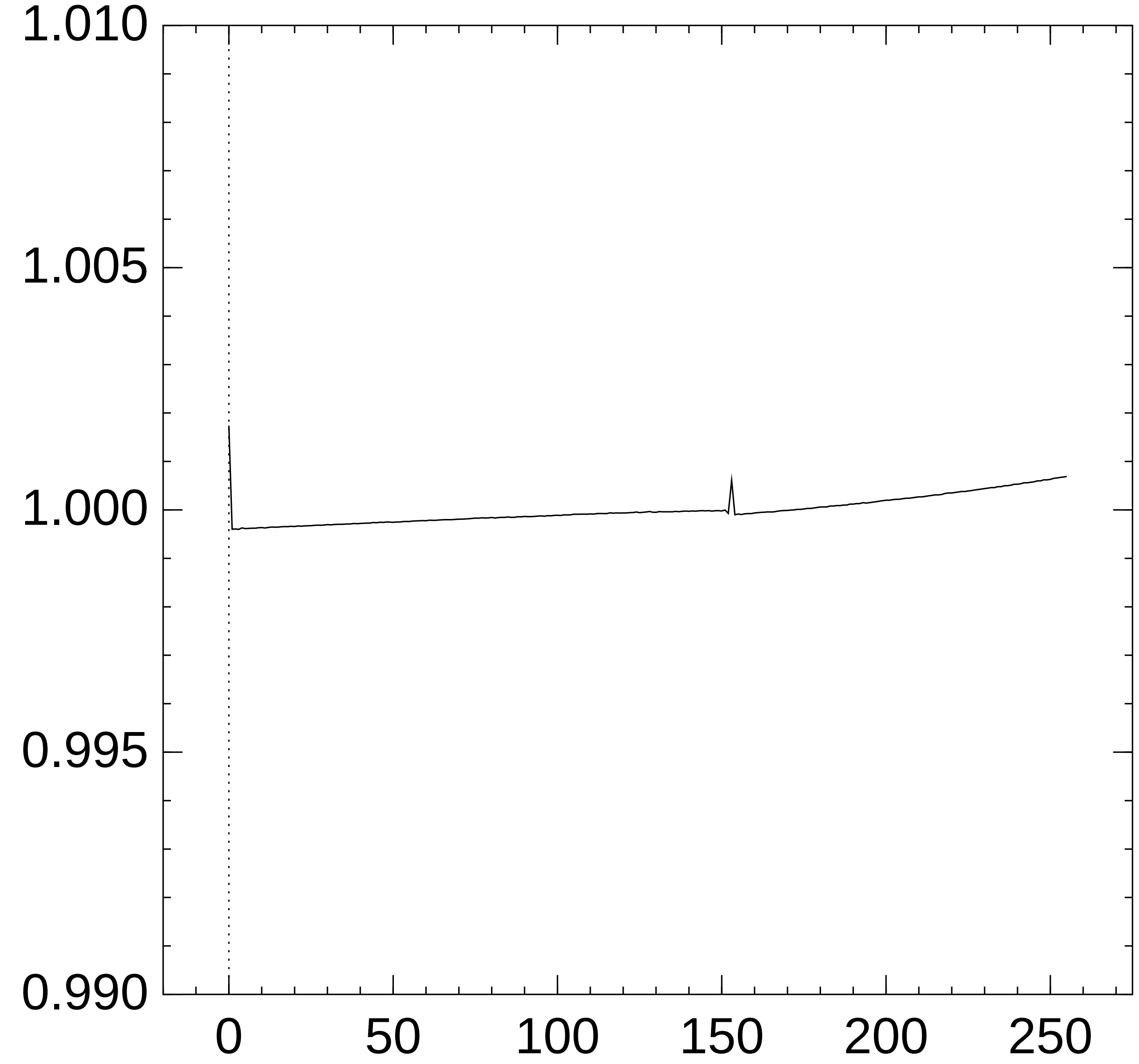
Graph of $256 \Pr[z_{152} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

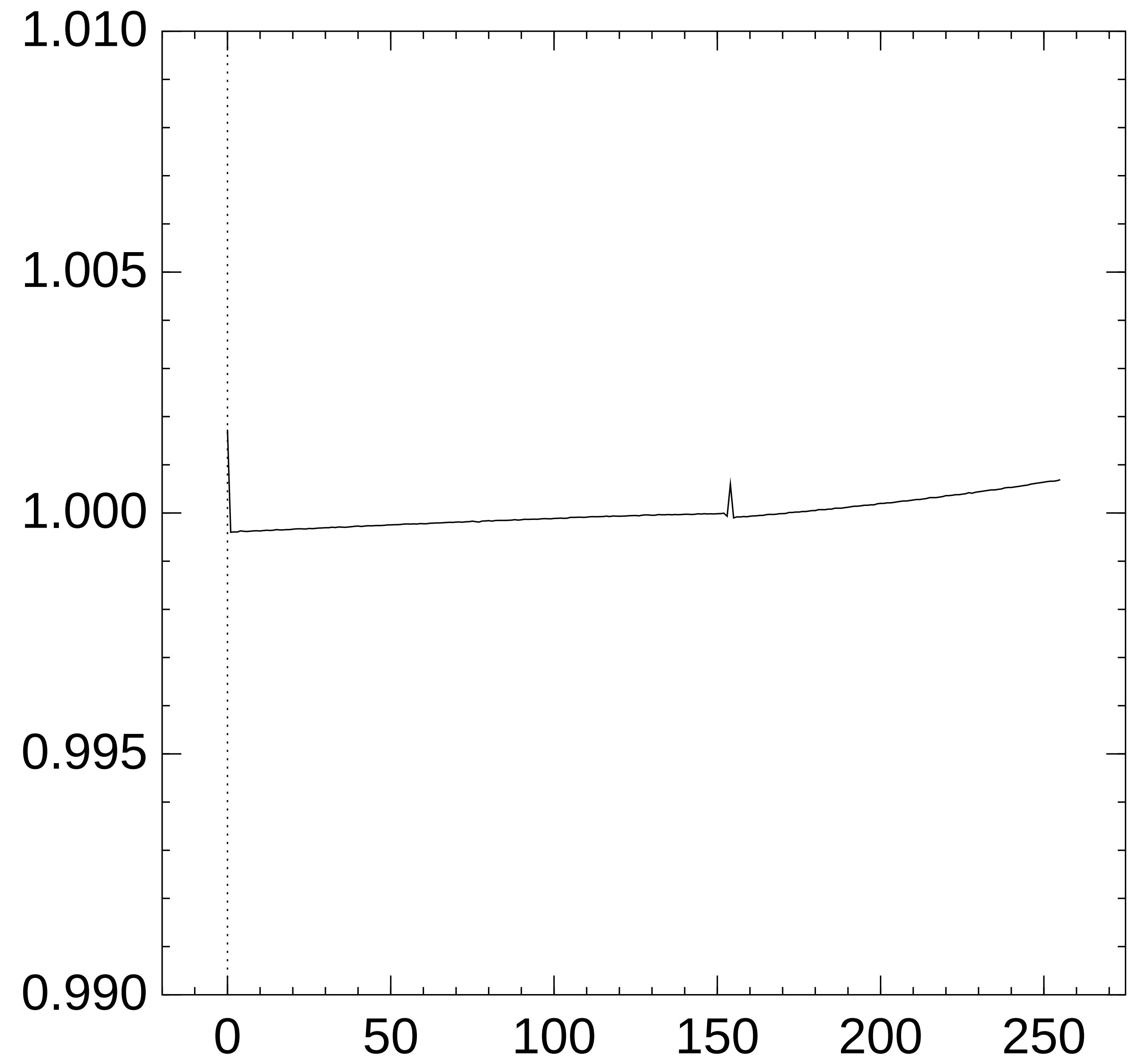
Graph of $256 \Pr[z_{153} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

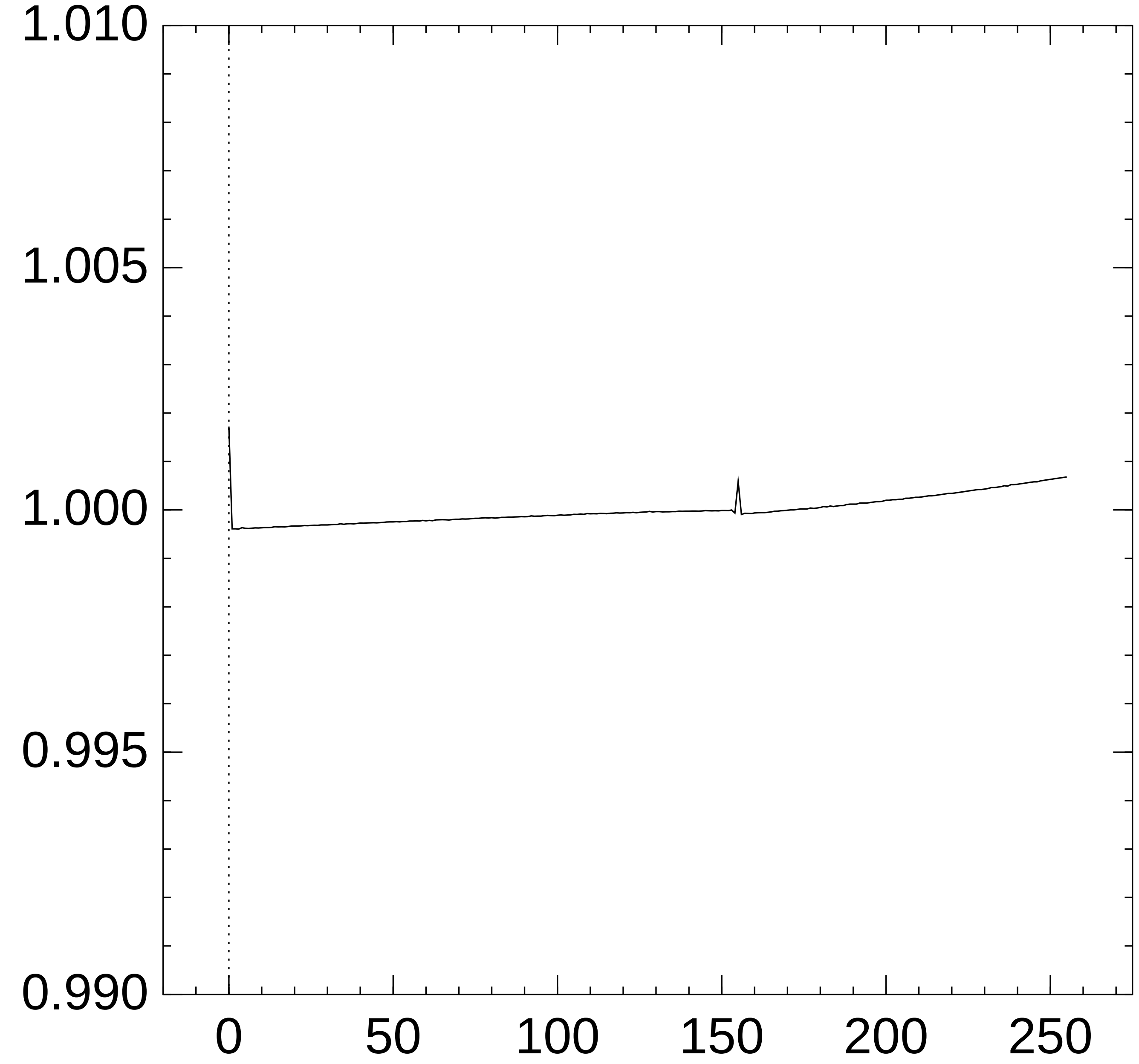
Graph of $256 \Pr[z_{154} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

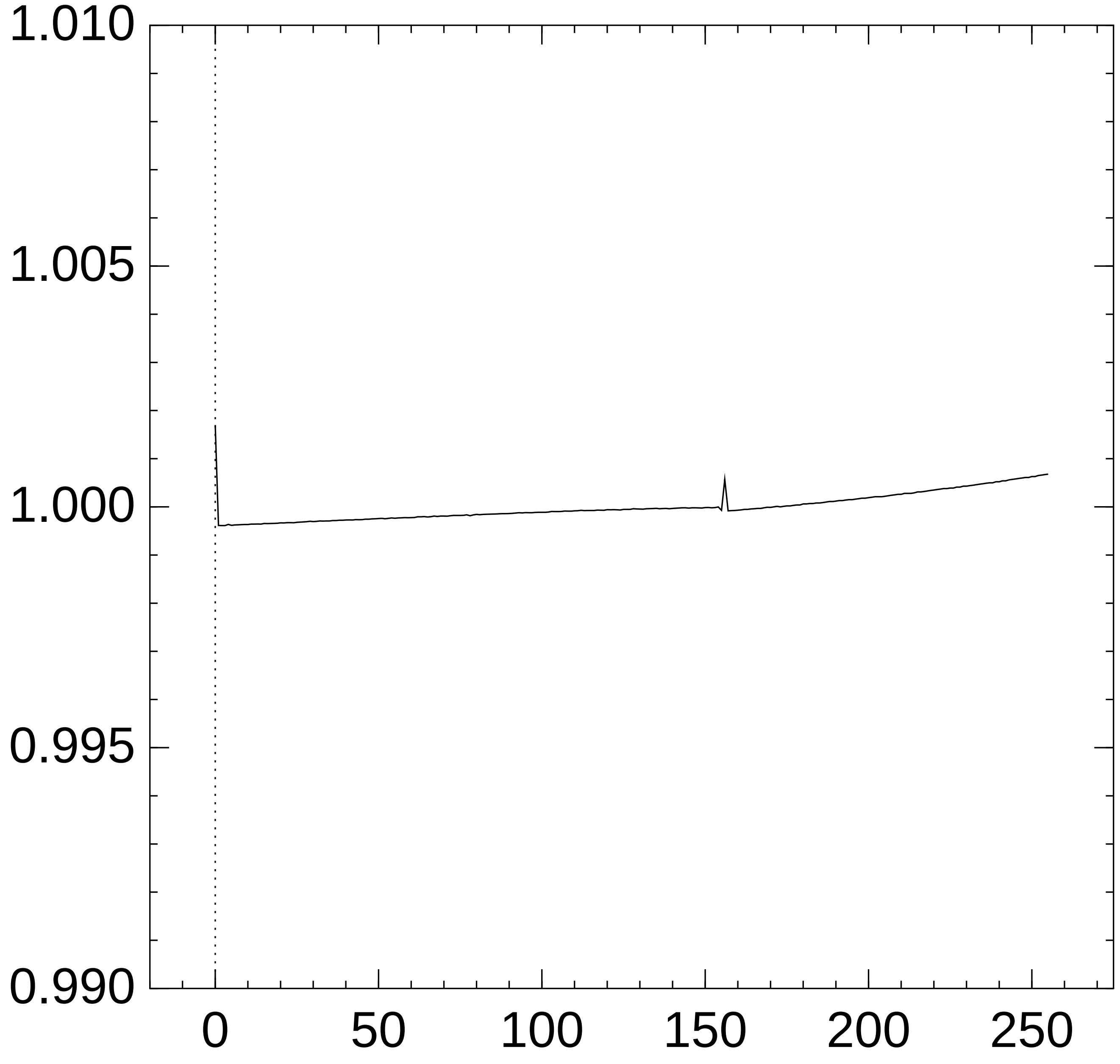
Graph of $256 \Pr[z_{155} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

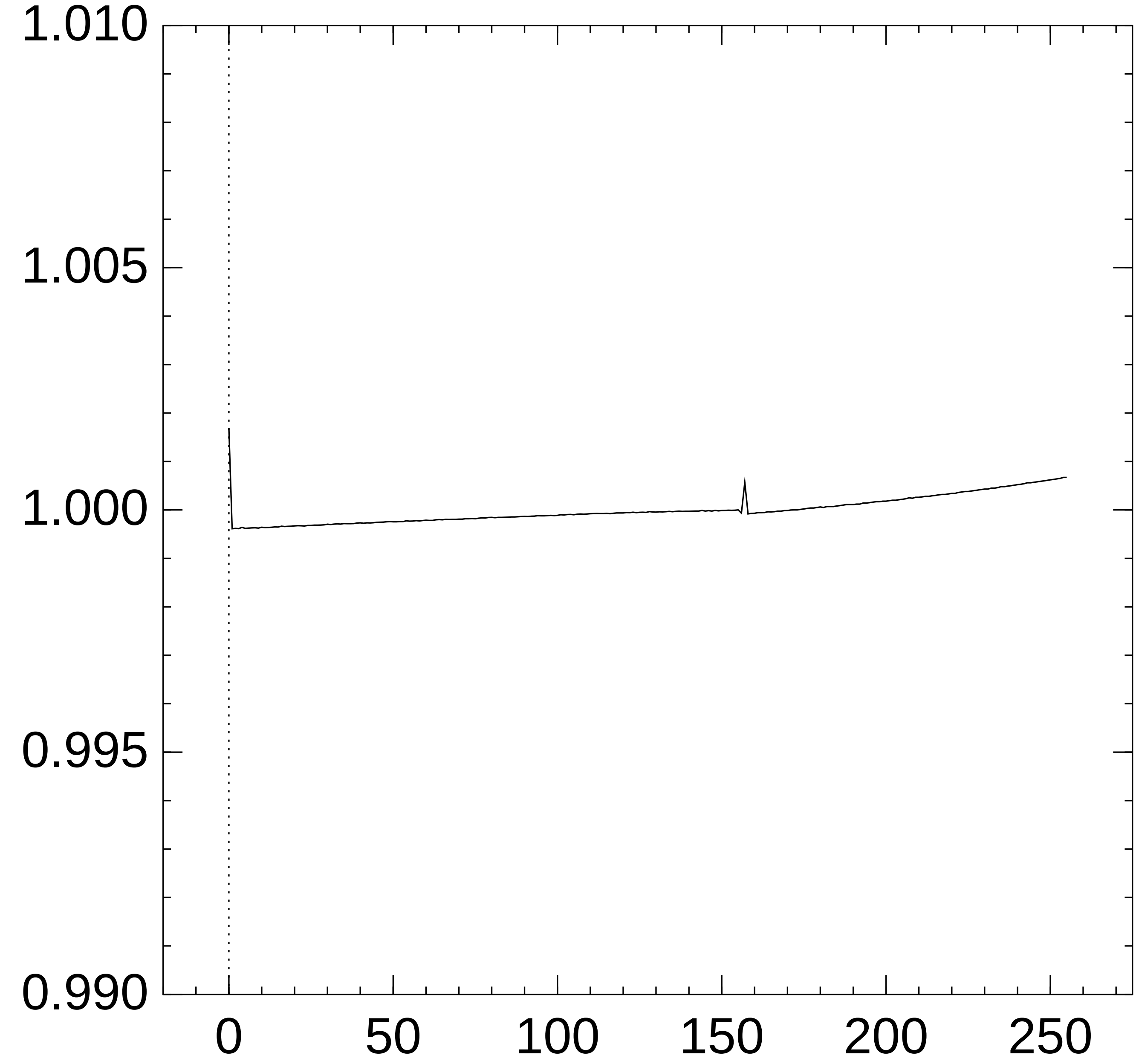
Graph of $256 \Pr[z_{156} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

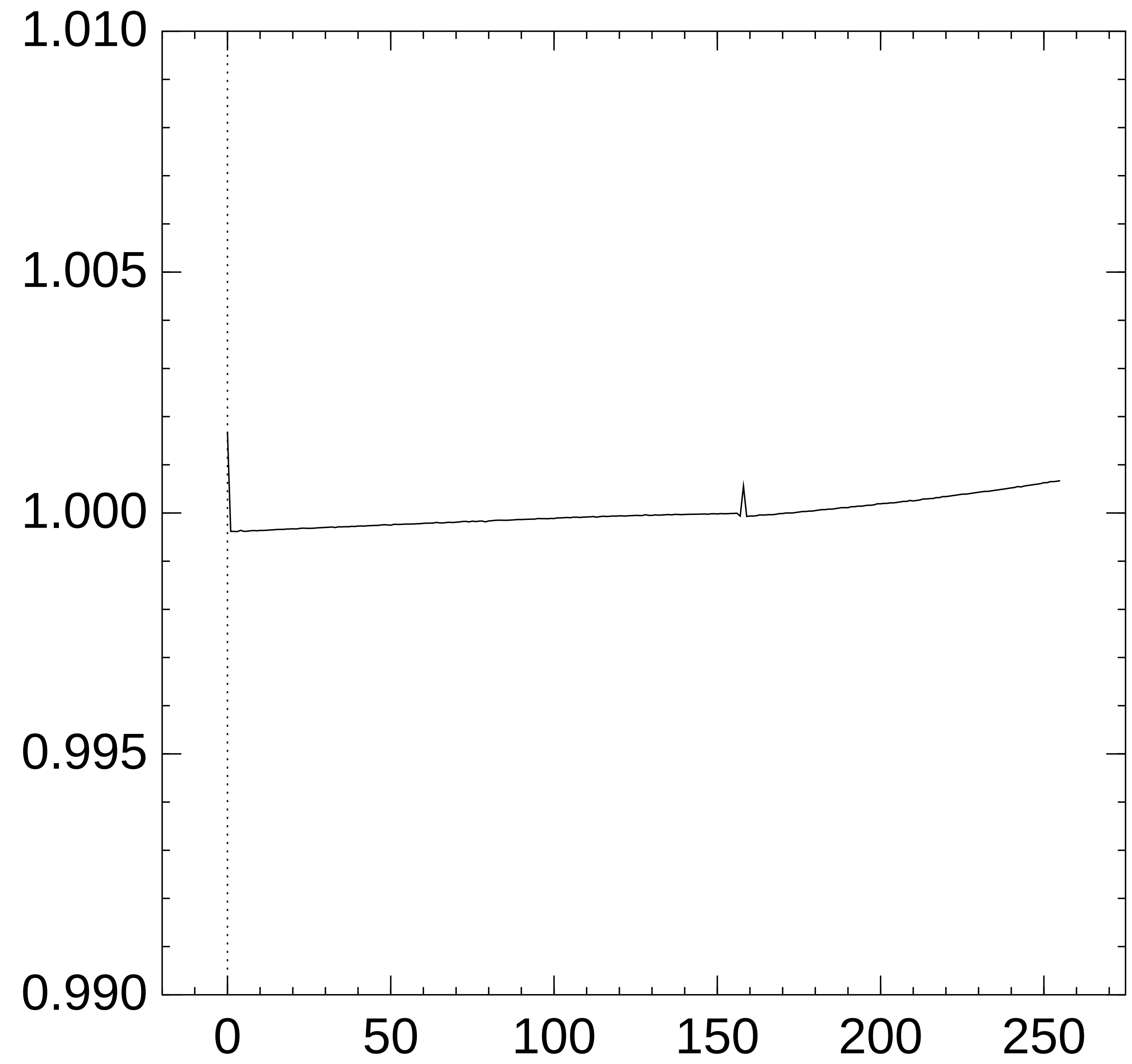
Graph of $256 \Pr[z_{157} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

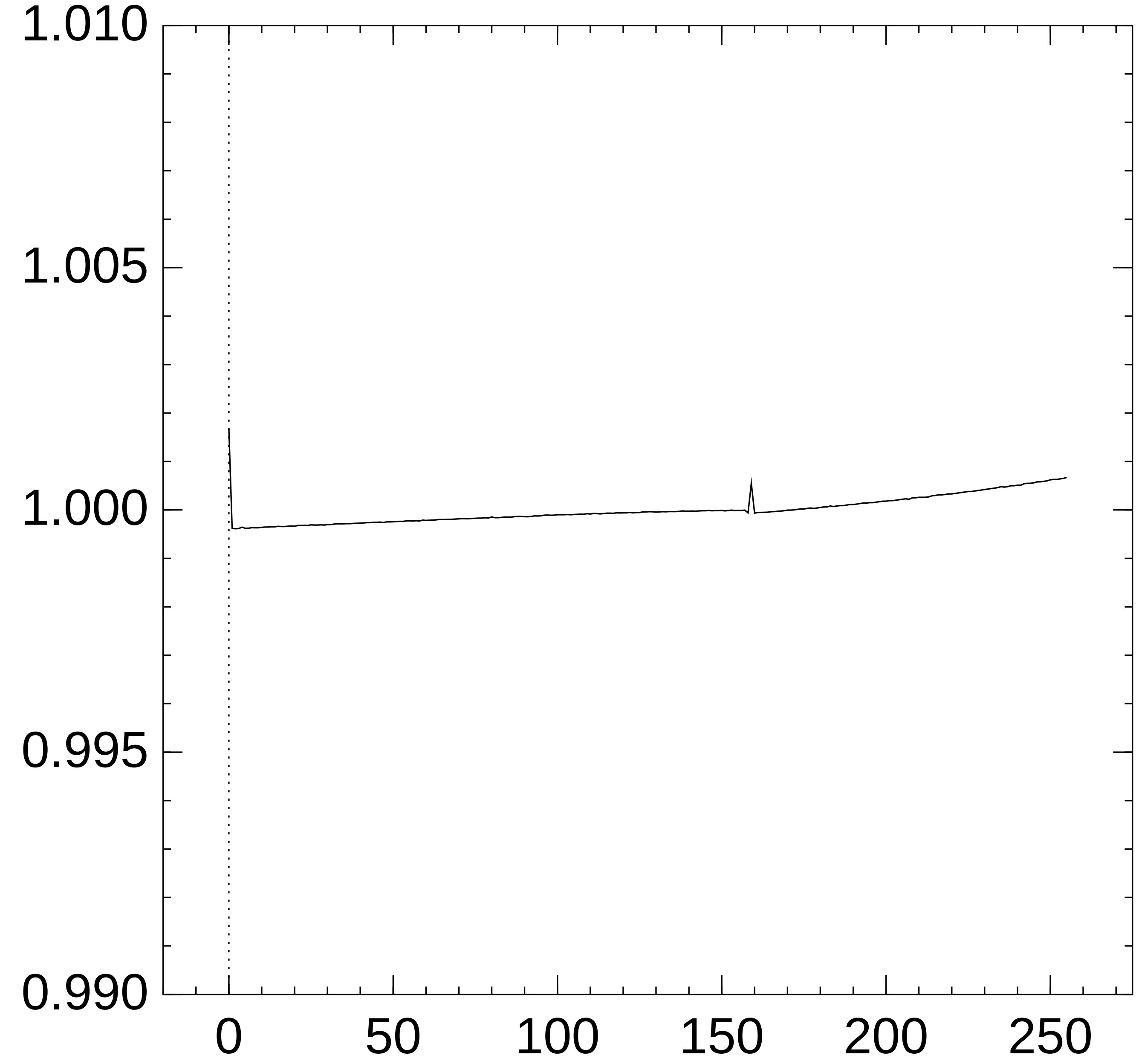
Graph of $256 \Pr[z_{158} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

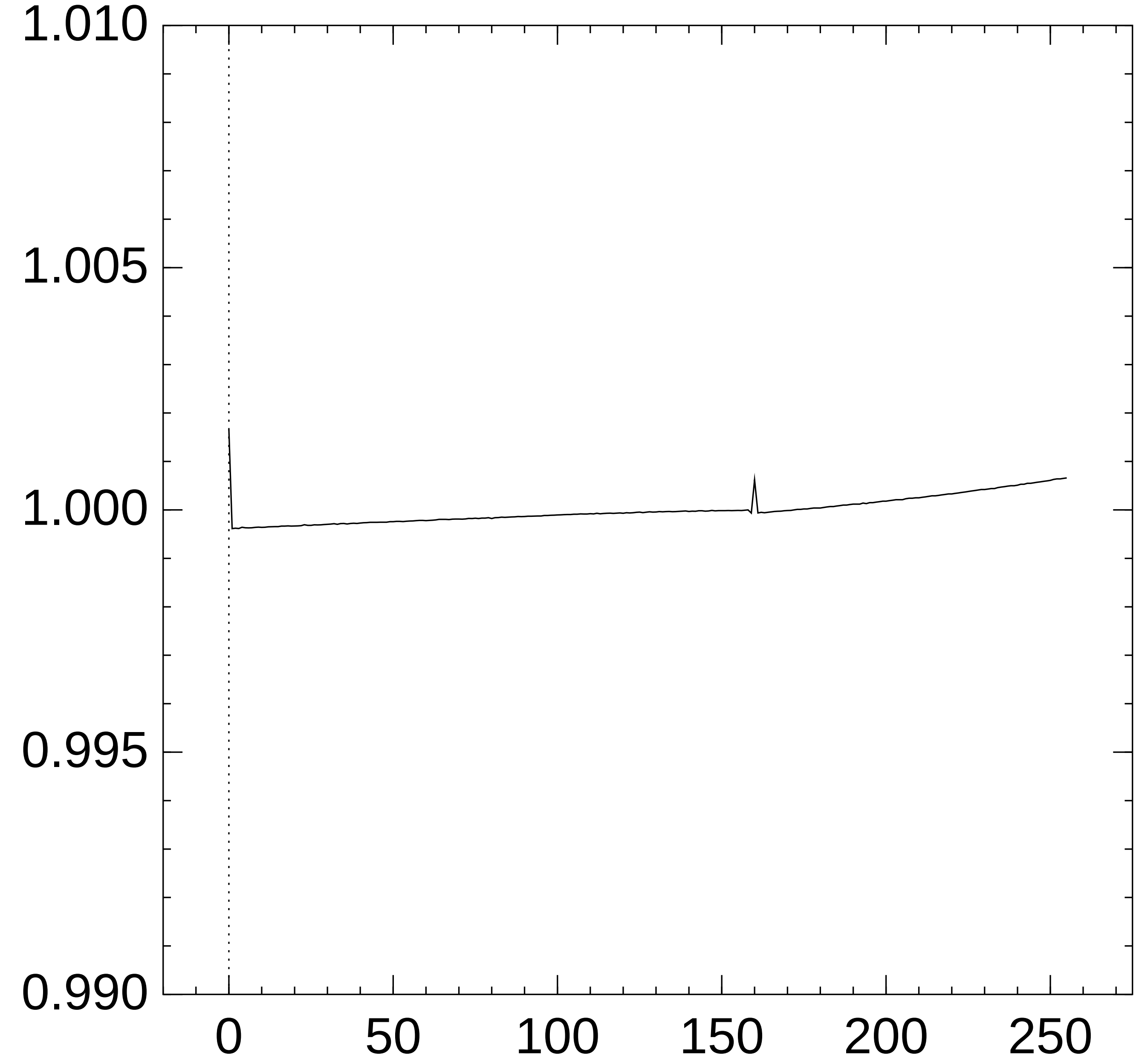
Graph of $256 \Pr[z_{159} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

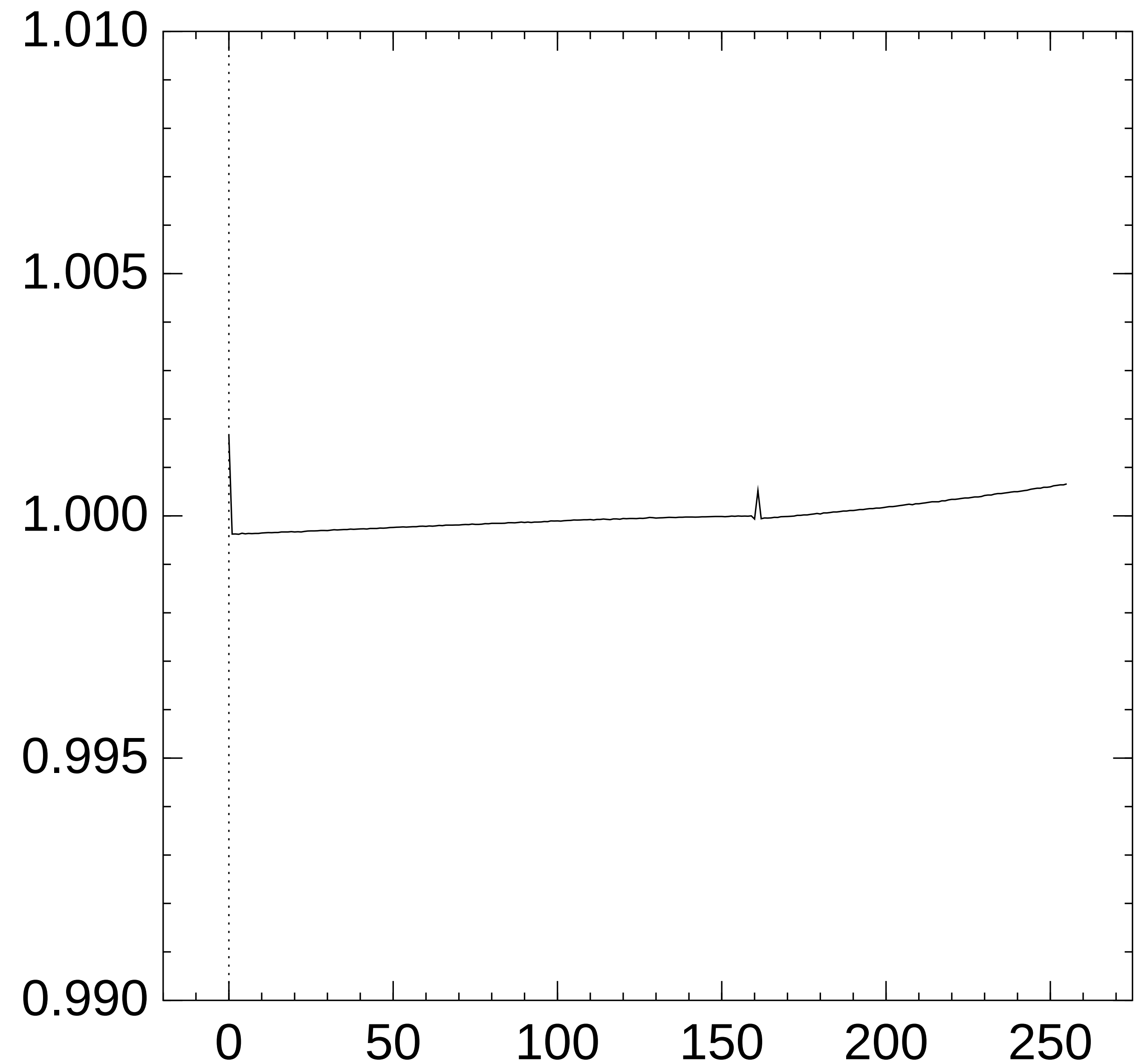
Graph of $256 \Pr[z_{160} = x]$:



2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \ldots, 256\}$, all j;
found $\approx$**65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

$\approx$256 of these biases were found
independently (slightly earlier)
by 2013 Watanabe–Isobe–
Ohigashi–Morii, 2013 Isobe–
Ohigashi–Watanabe–Morii:
$z_{32} \to 224$, $z_{48} \to 208$, etc.;
$z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

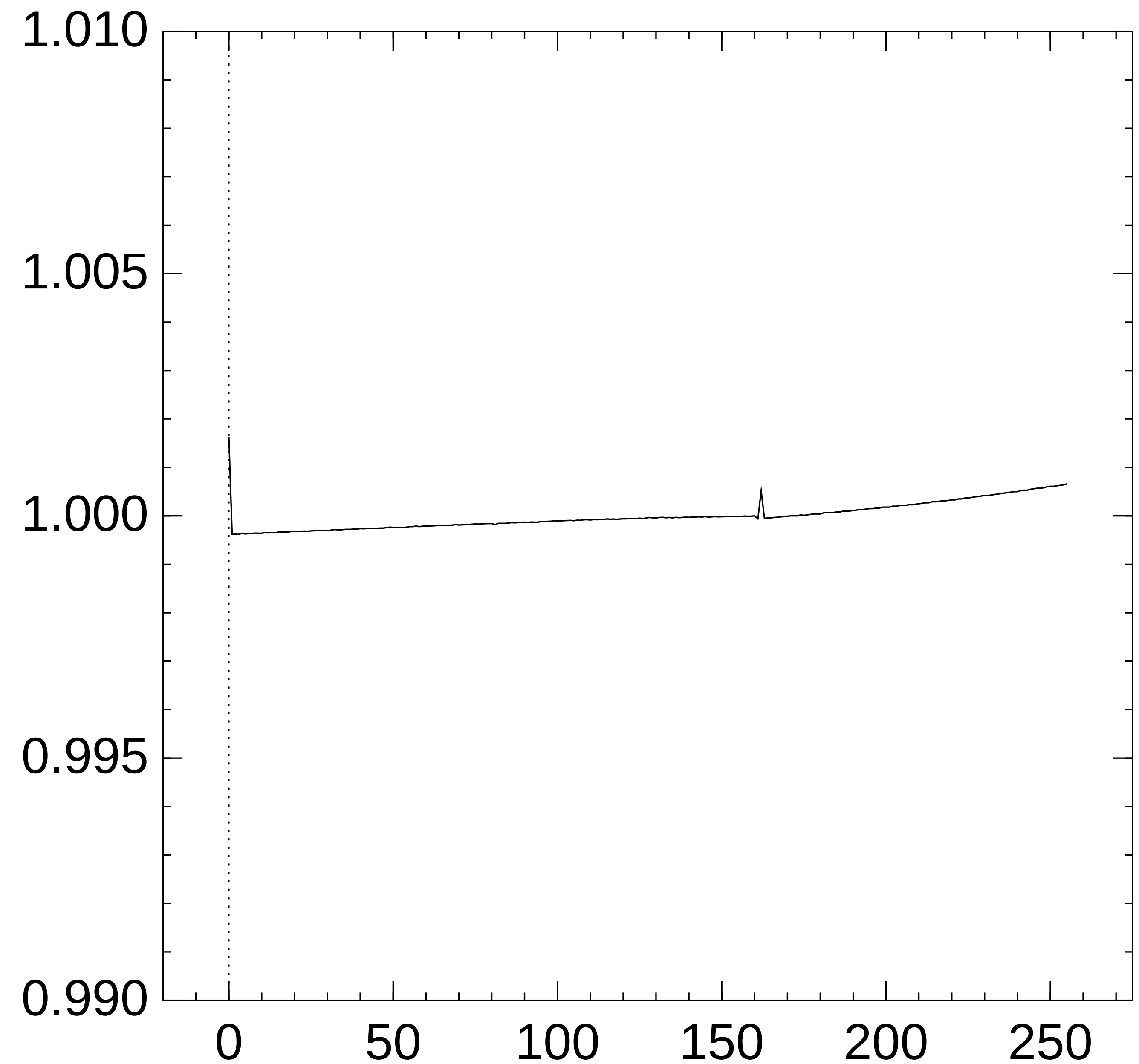
Graph of $256 \Pr[z_{161} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

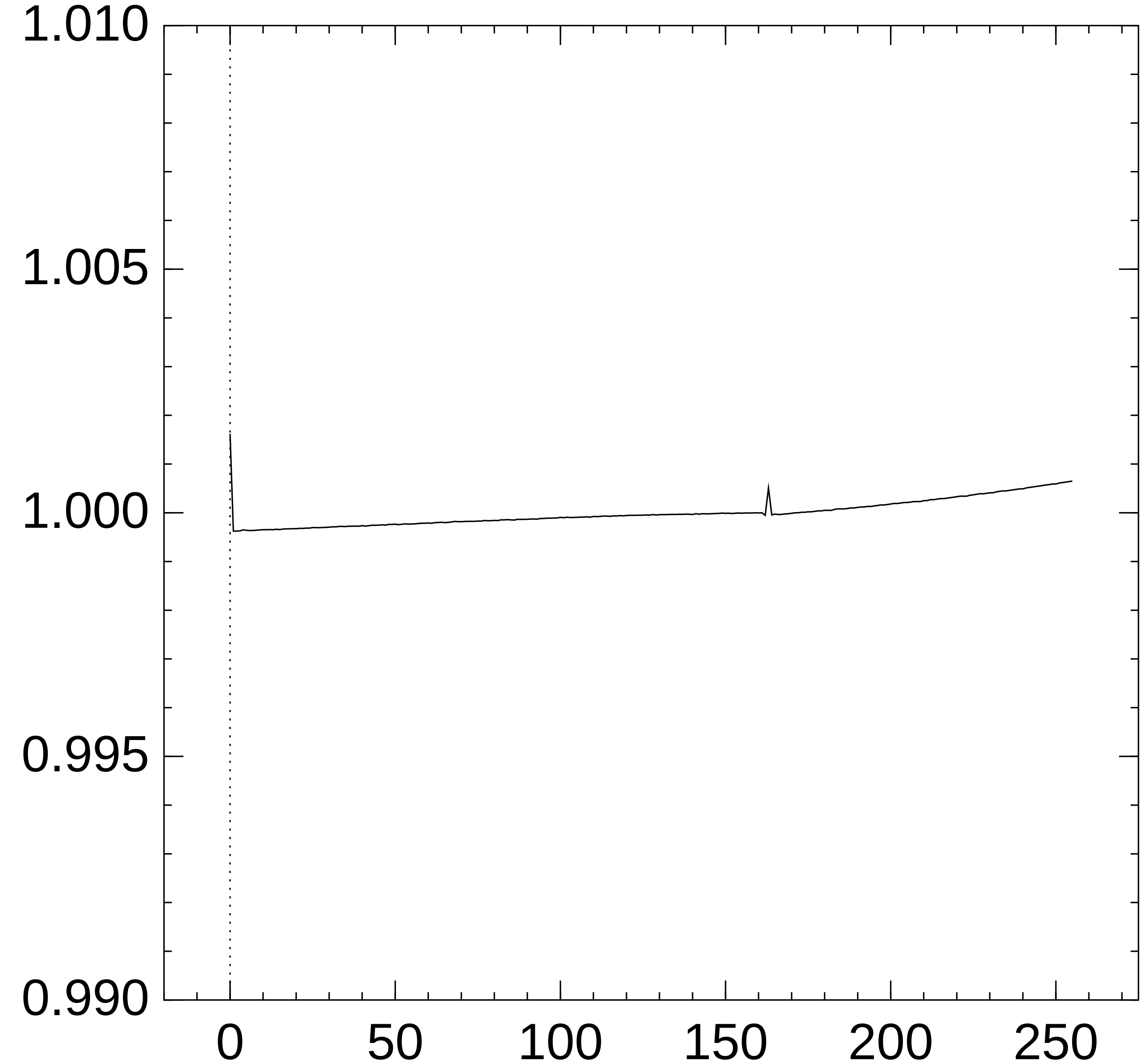
Graph of $256 \Pr[z_{162} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

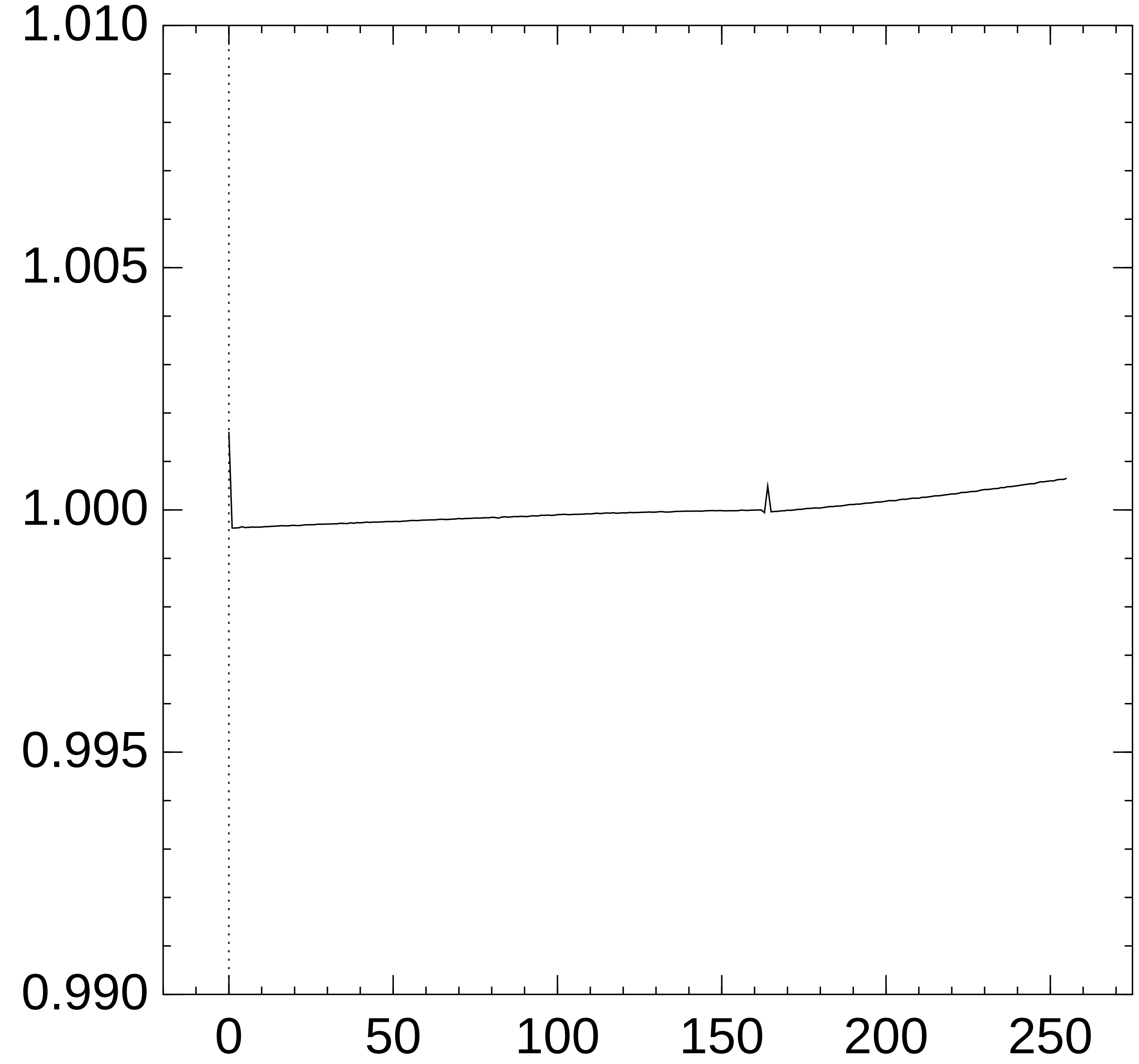
Graph of $256 \Pr[z_{163} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

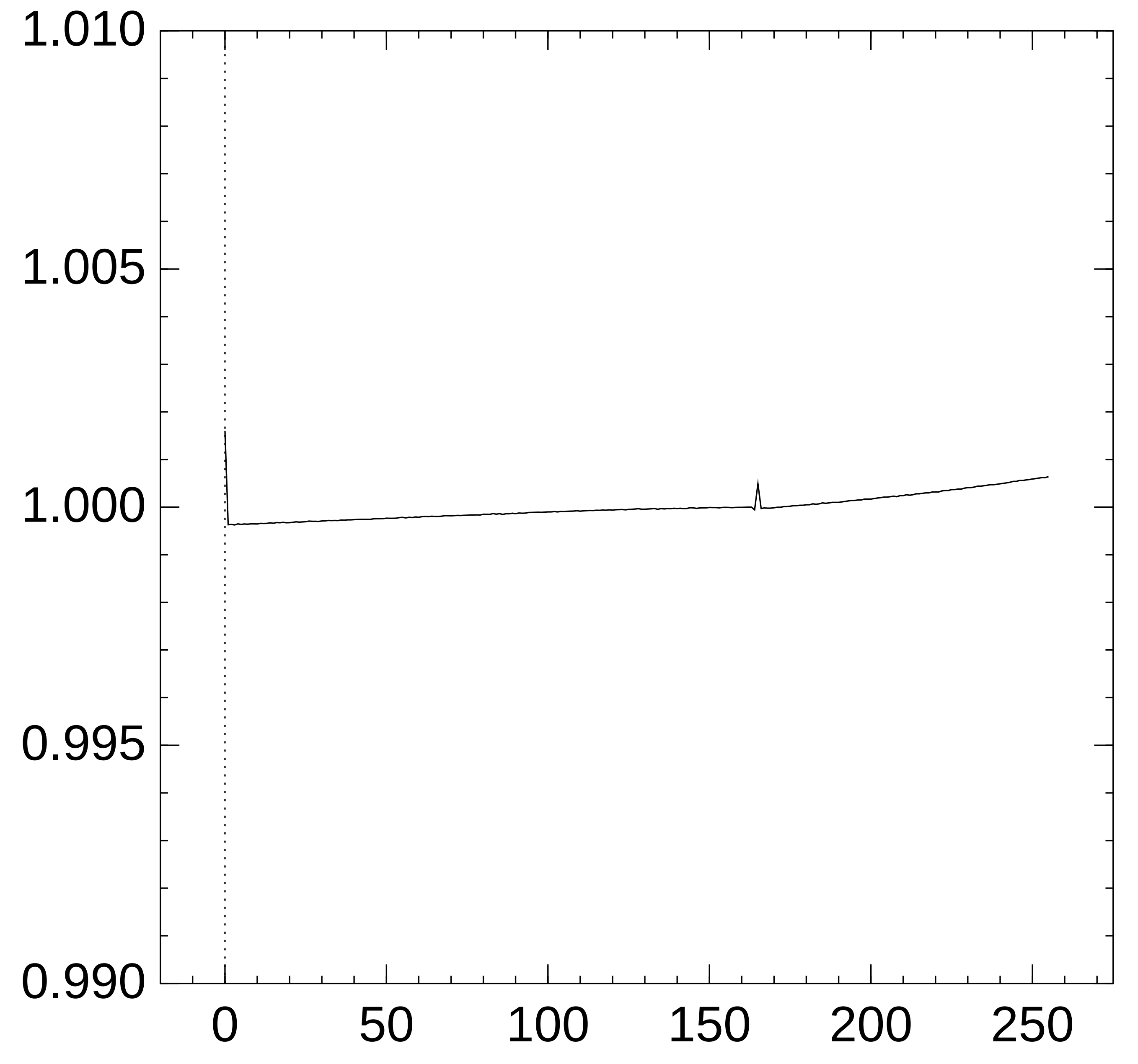
Graph of $256 \Pr[z_{164} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

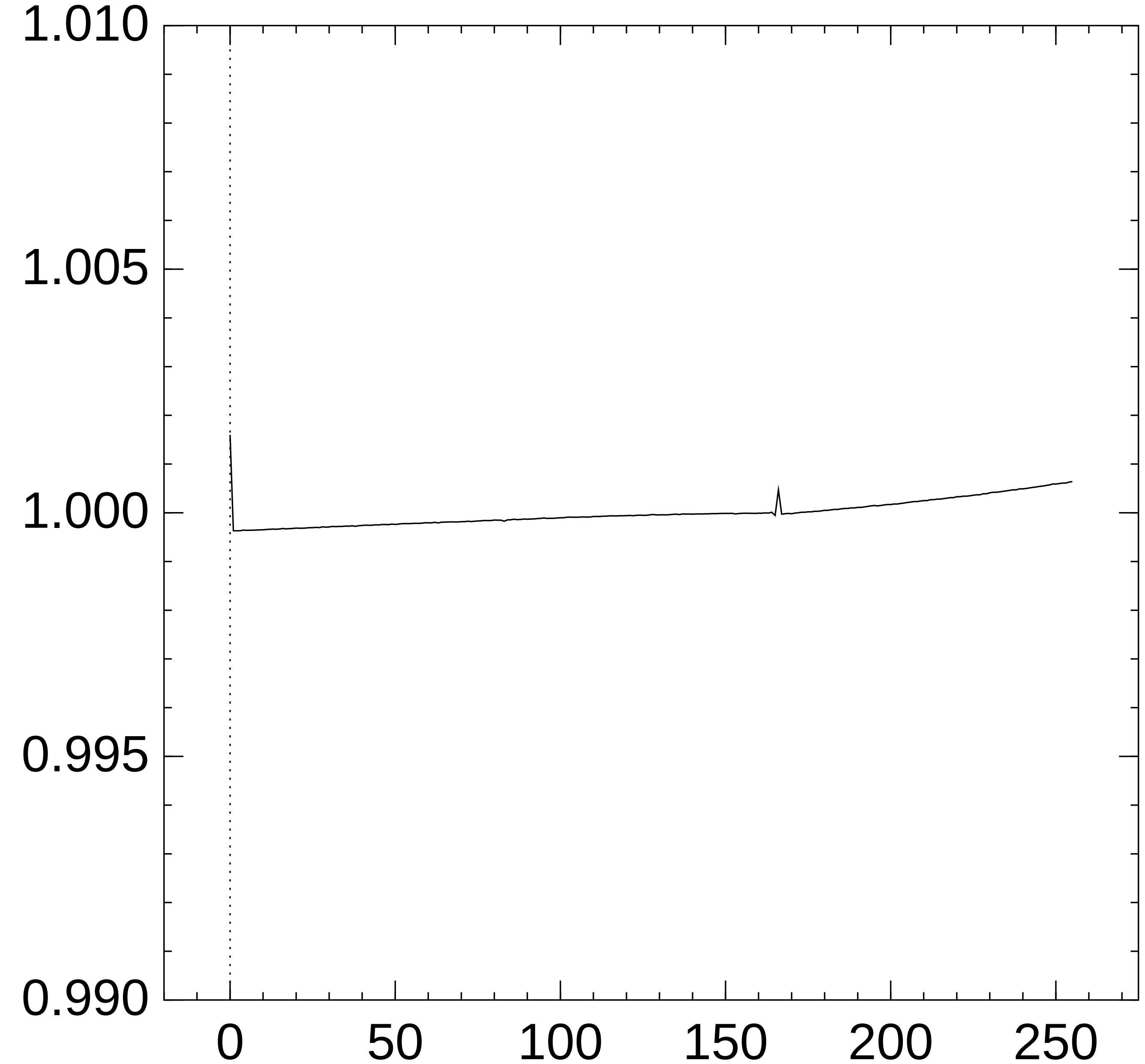
Graph of $256 \Pr[z_{165} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

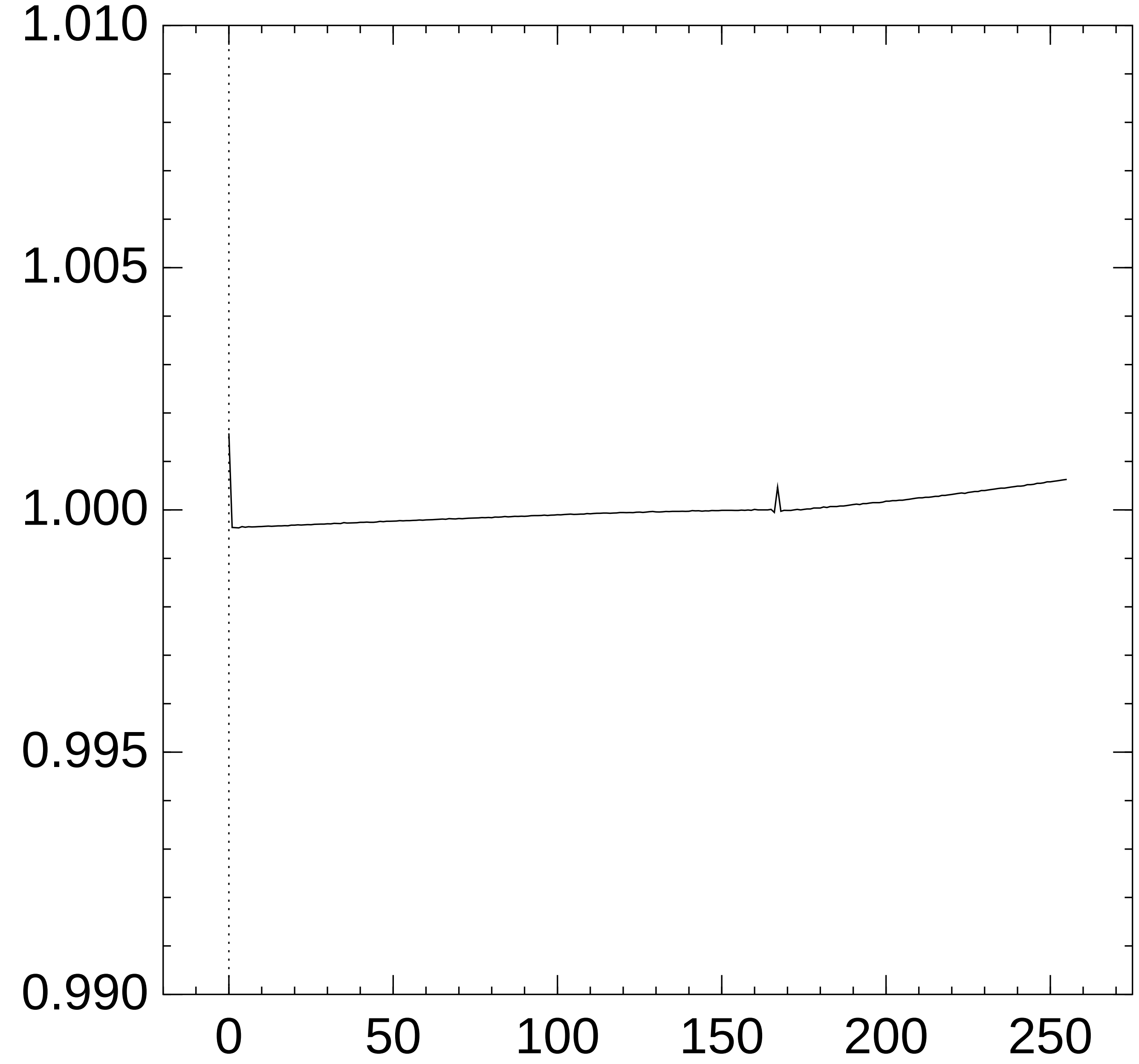
Graph of $256 \Pr[z_{166} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

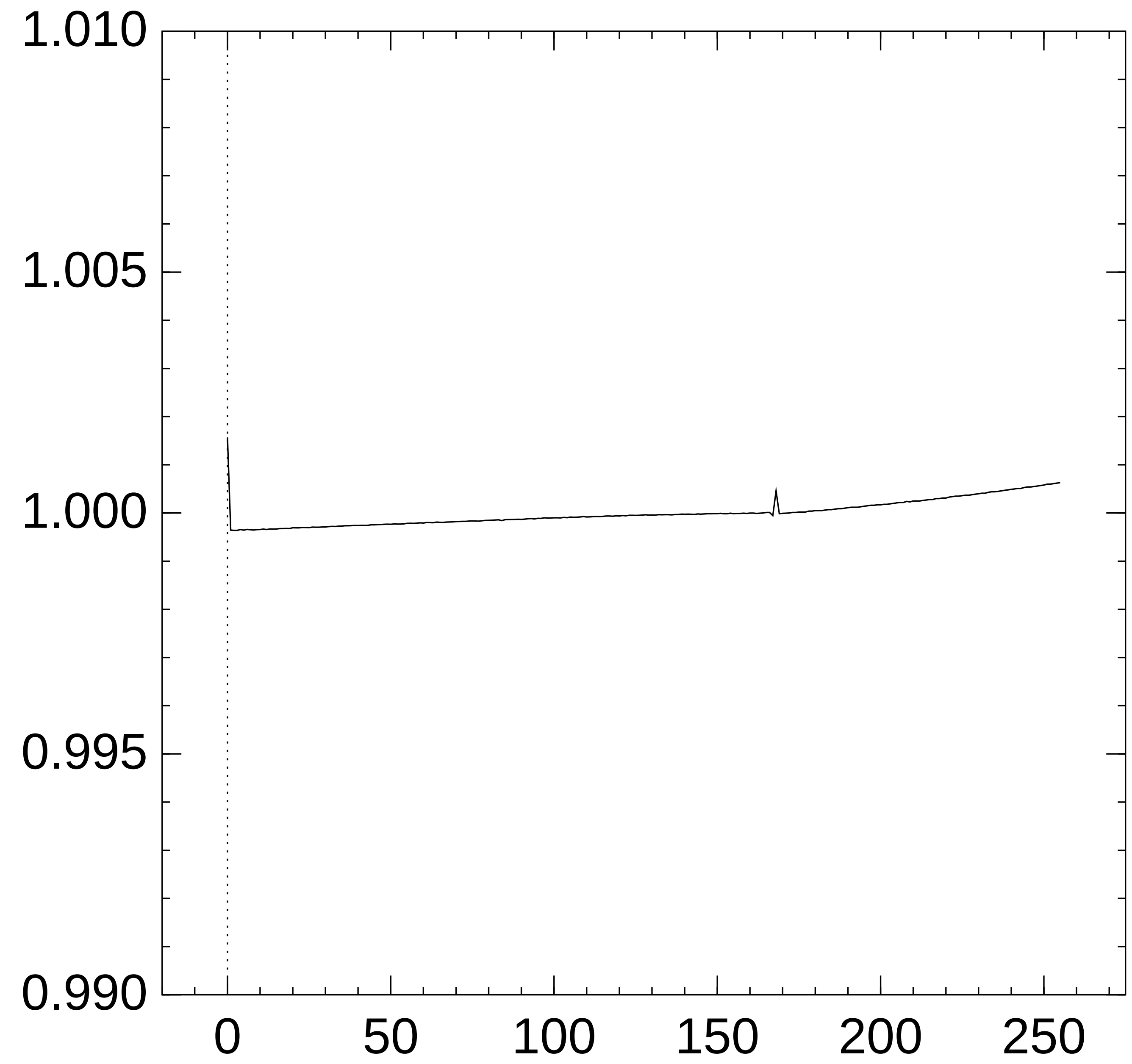
Graph of $256 \Pr[z_{167} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

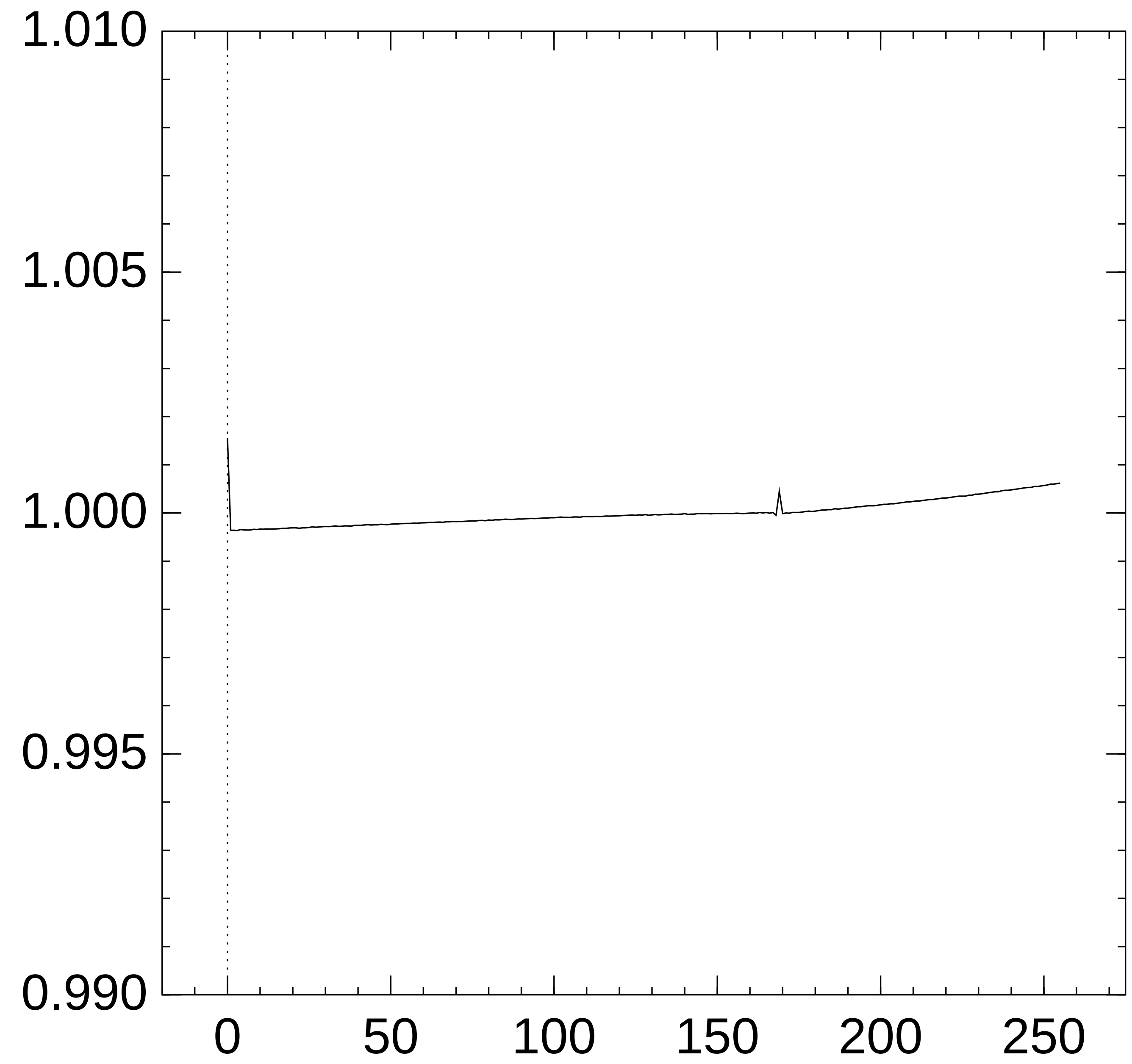
Graph of $256 \Pr[z_{168} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

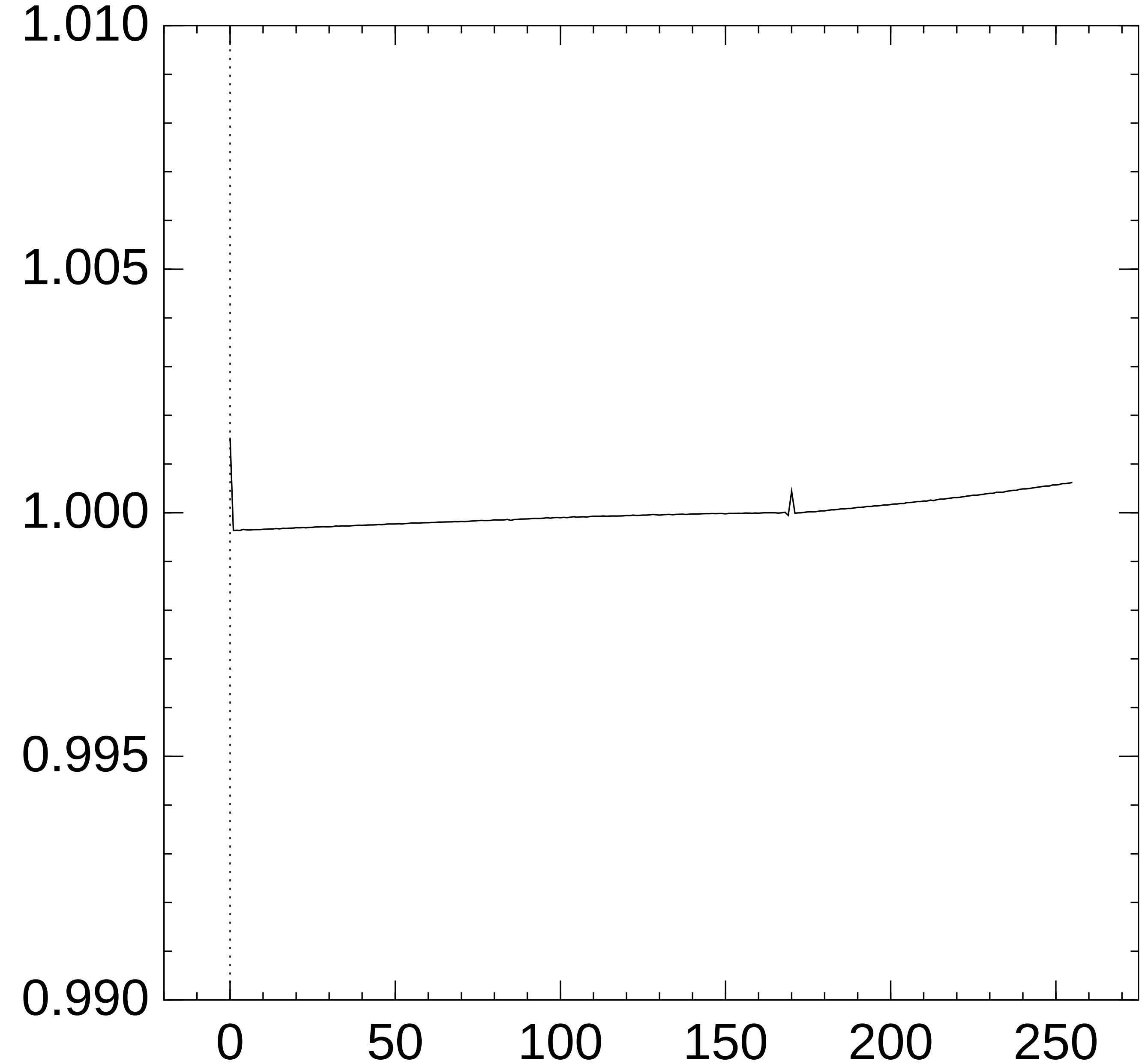
Graph of $256 \Pr[z_{169} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

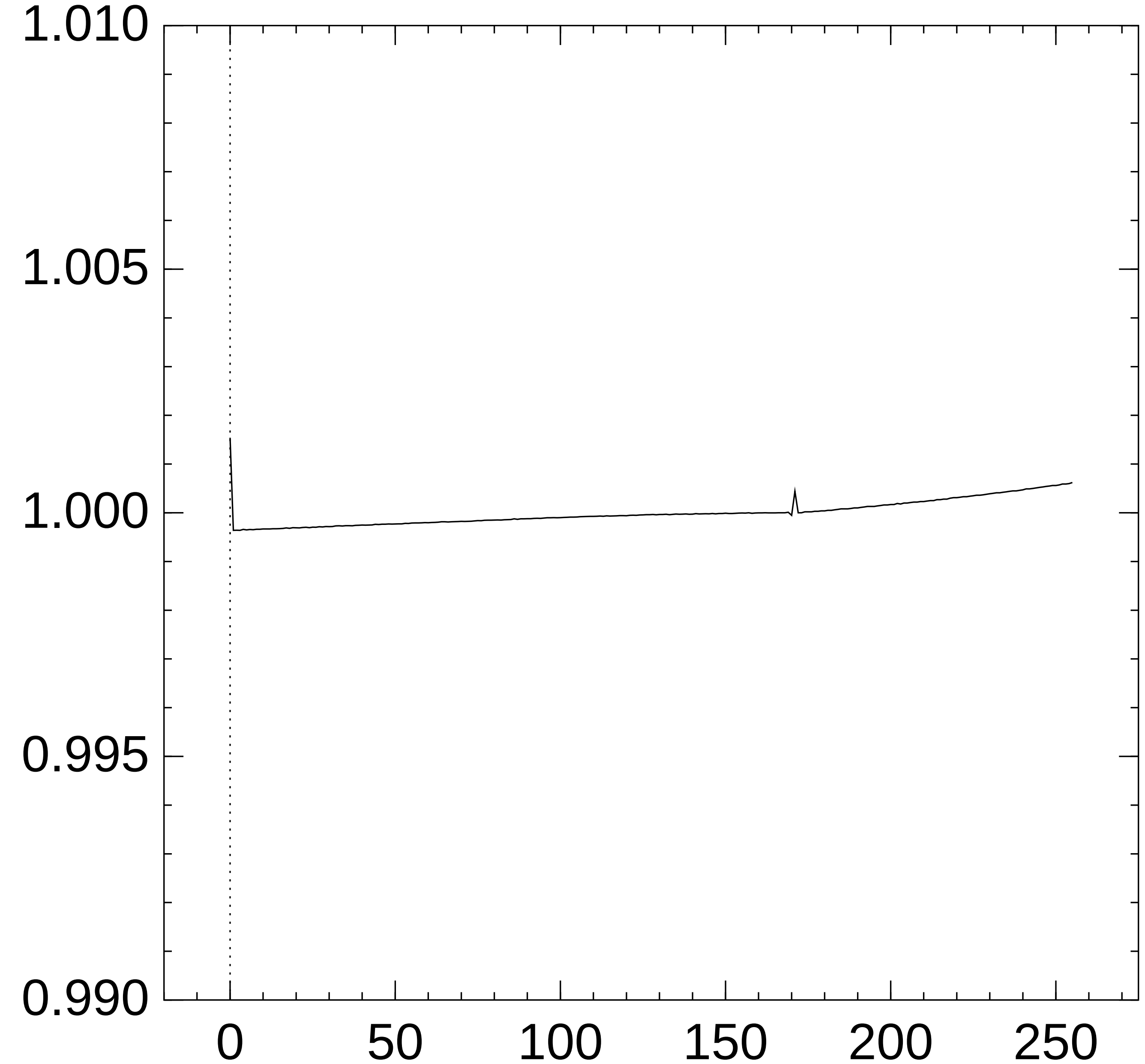
Graph of $256 \Pr[z_{170} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

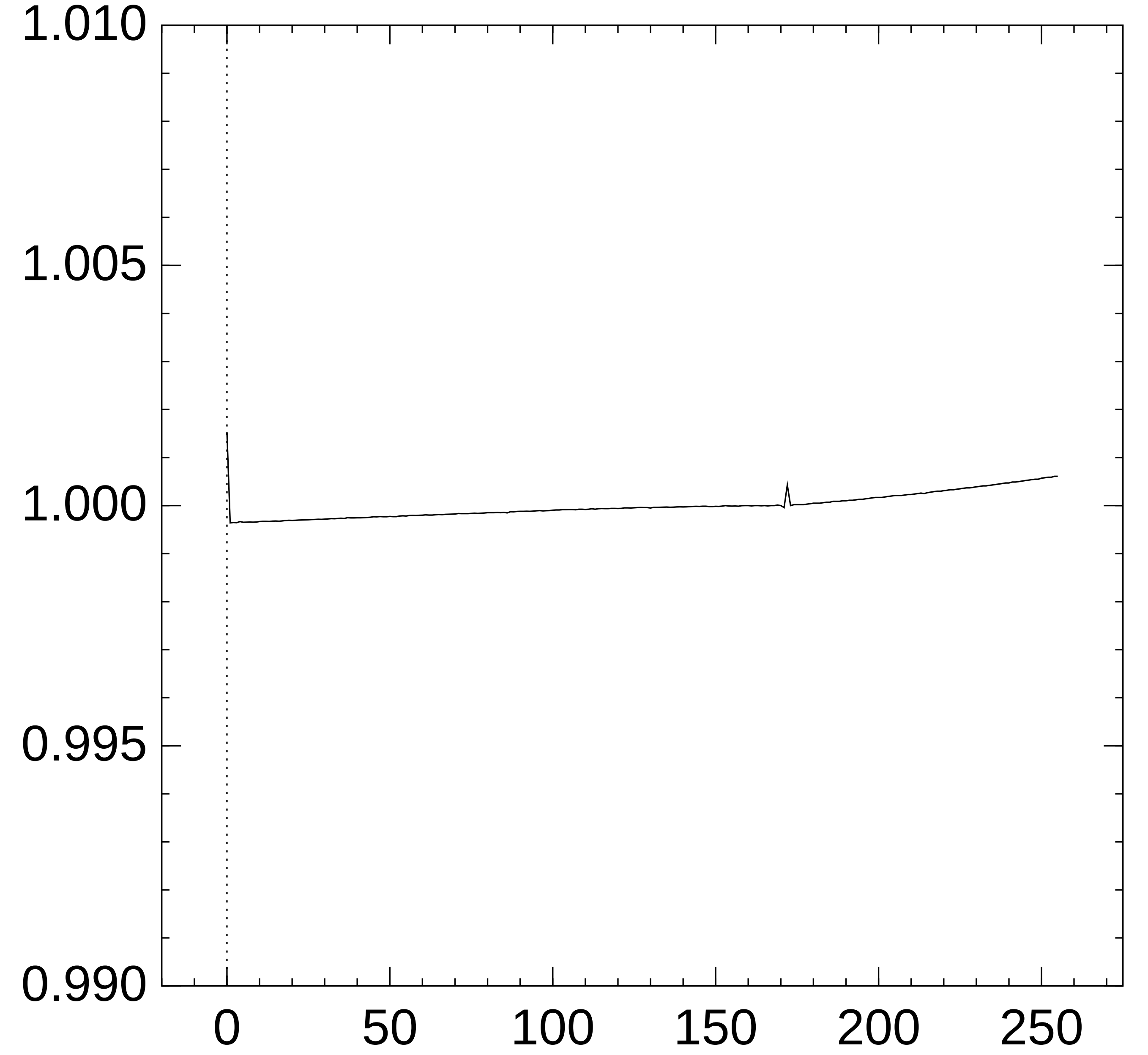
Graph of $256 \Pr[z_{171} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

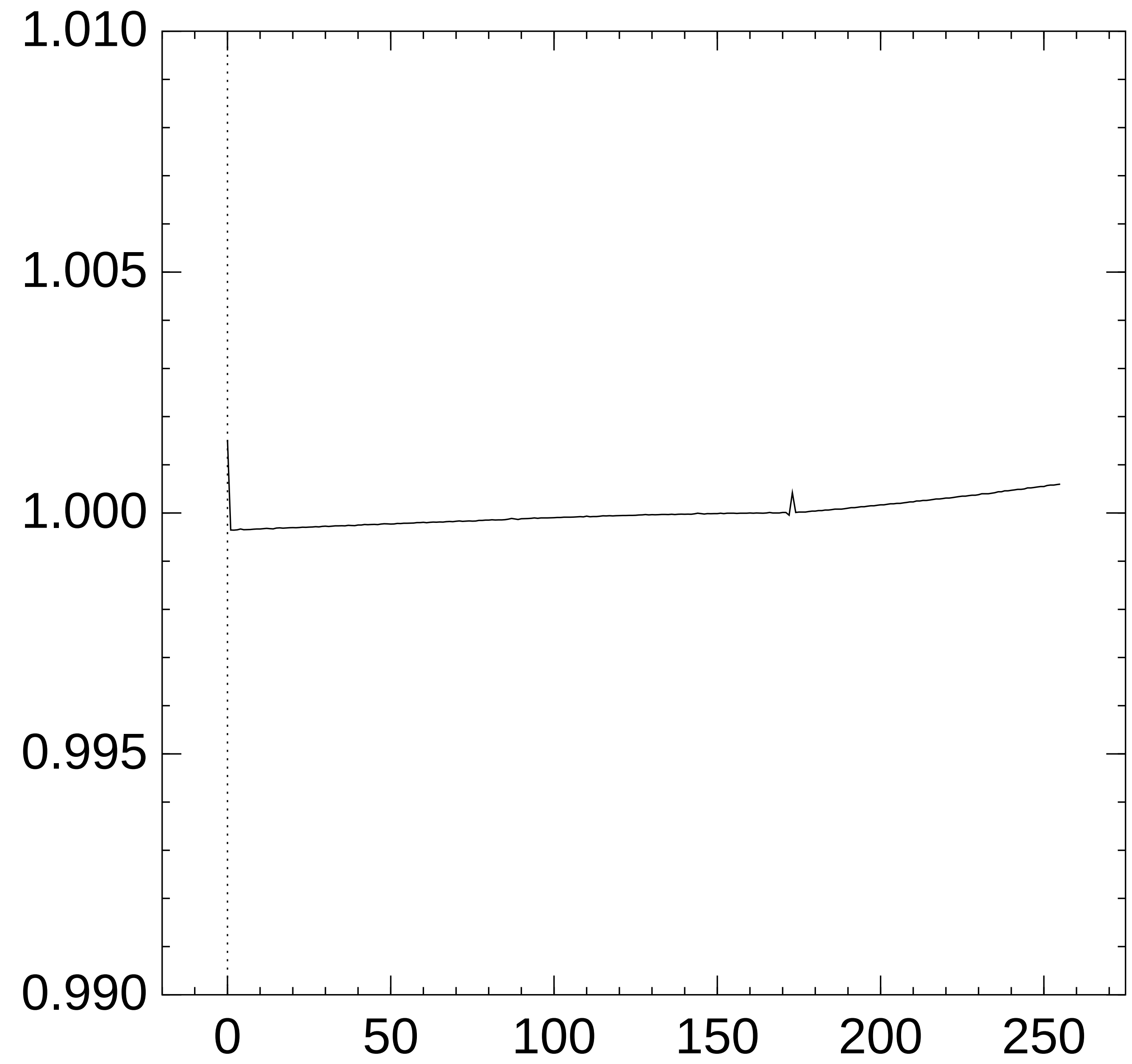
Graph of $256 \Pr[z_{172} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

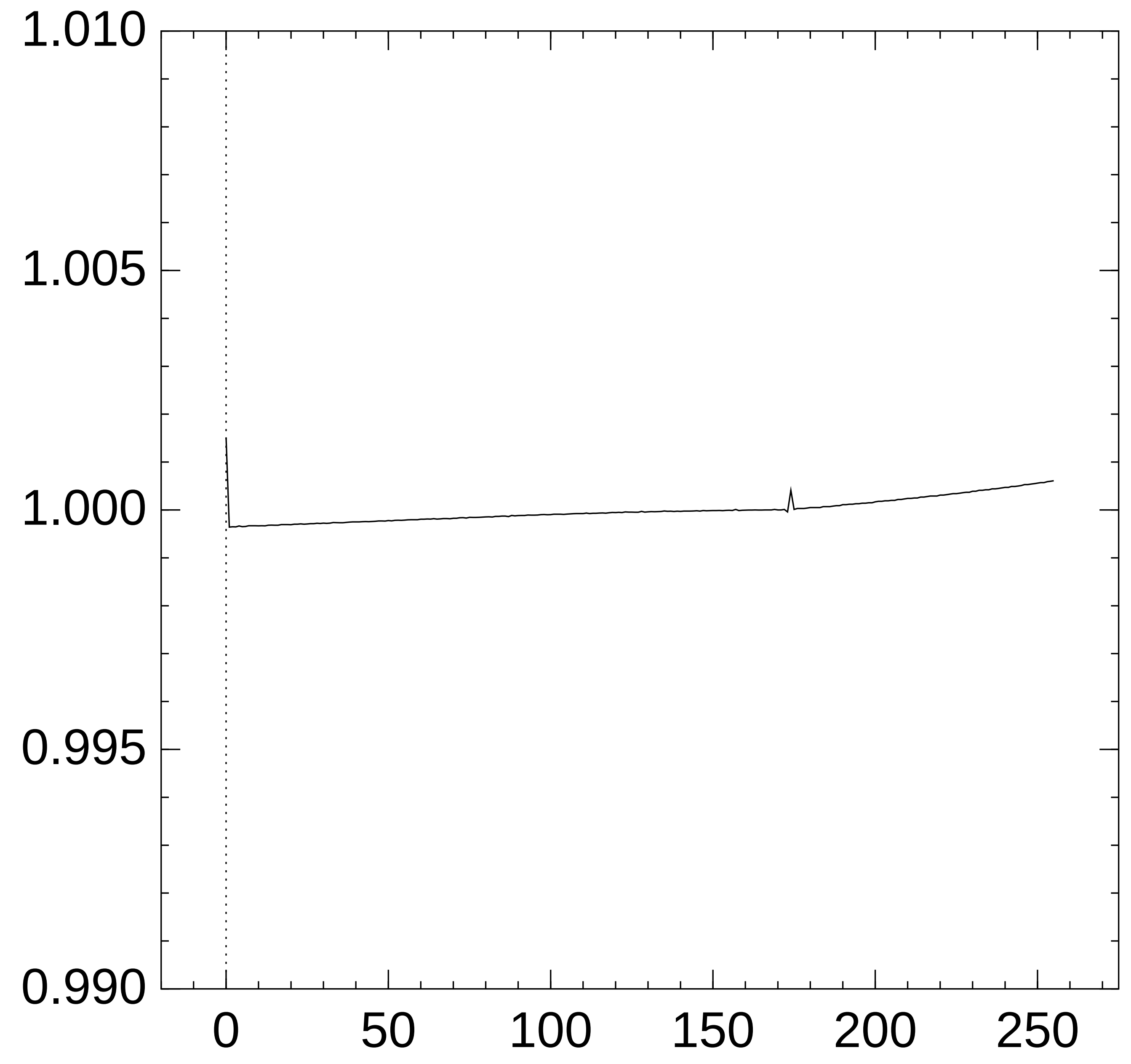
Graph of $256 \Pr[z_{173} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

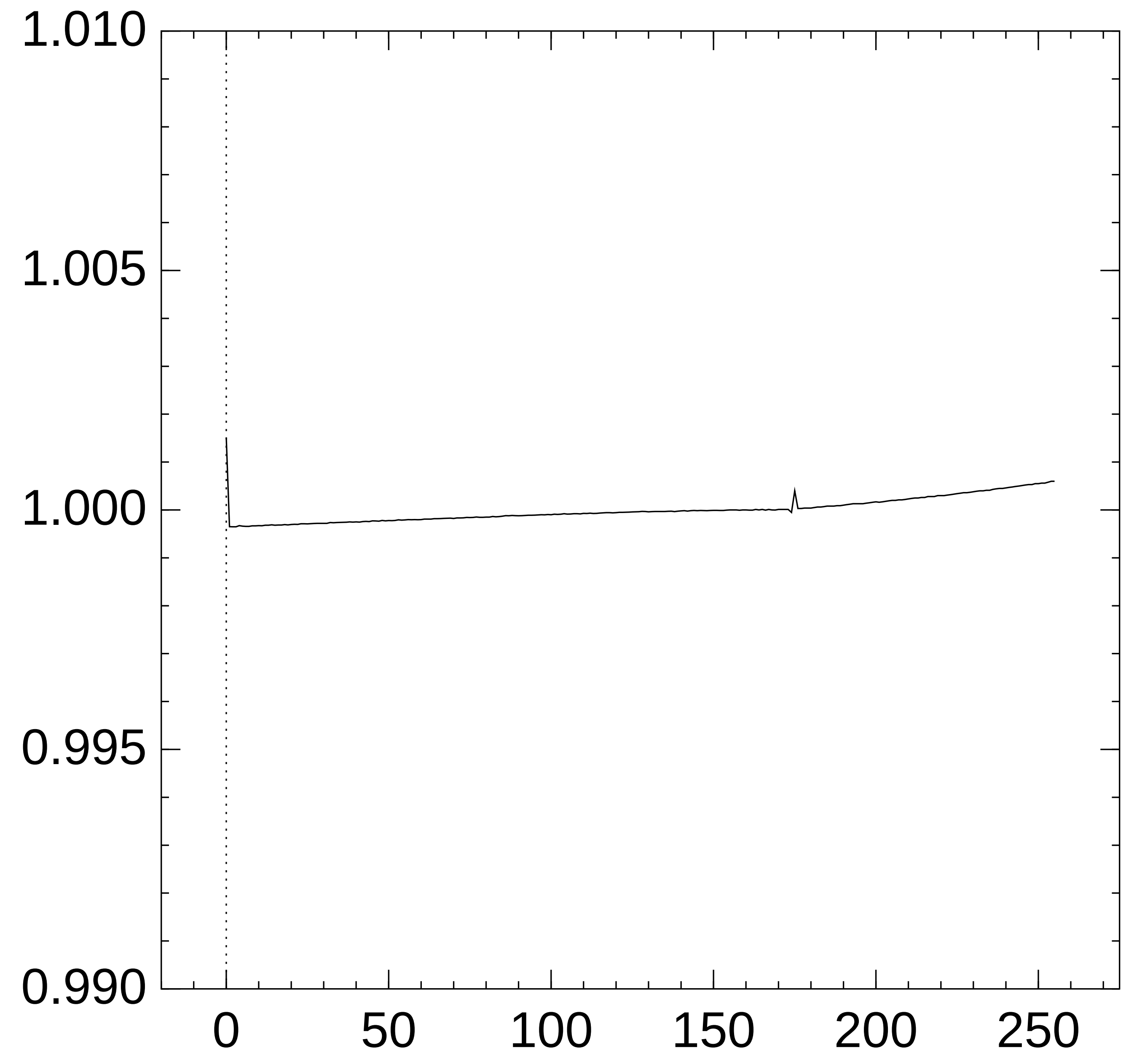
Graph of $256\,\Pr[z_{174} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

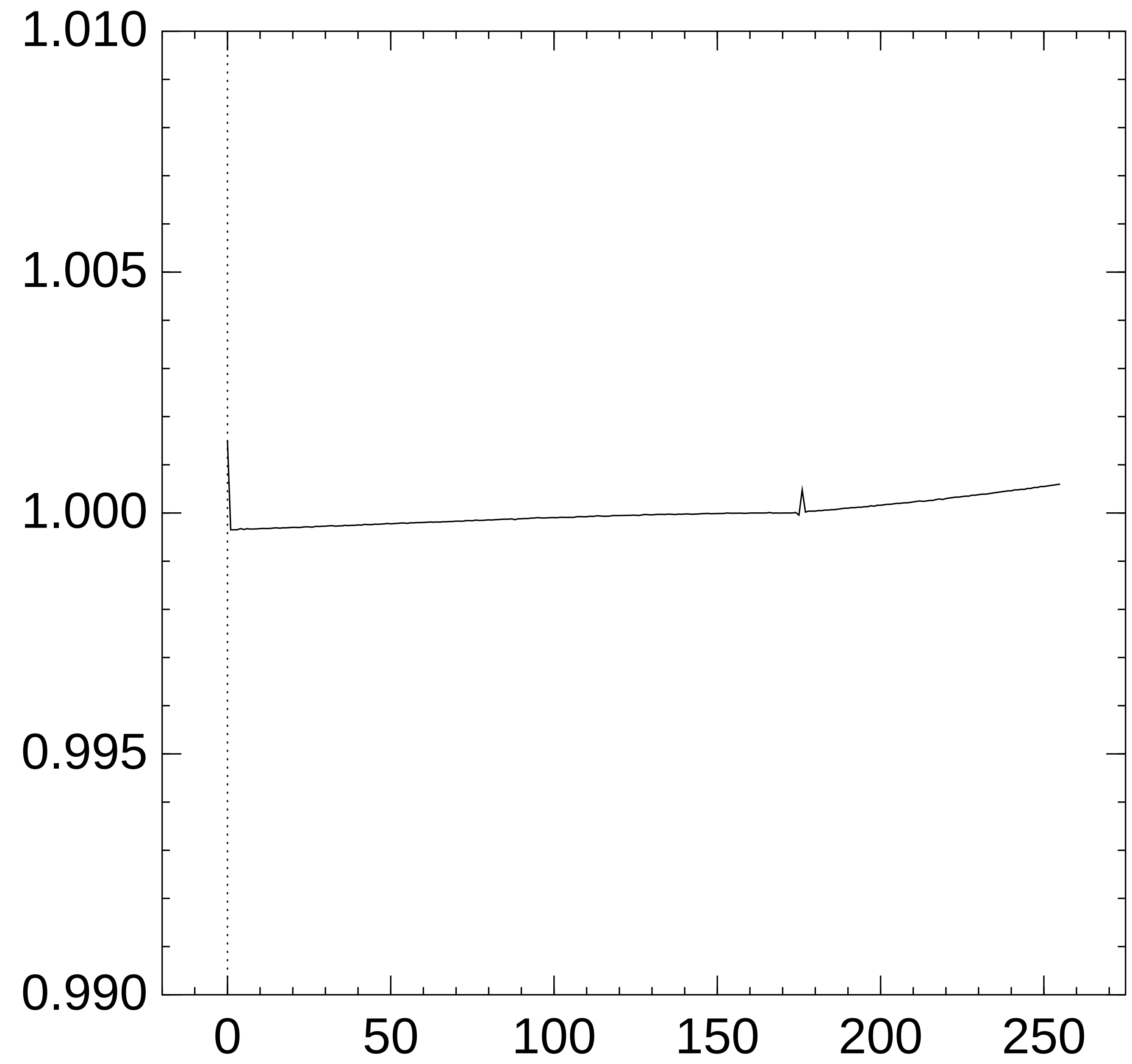
Graph of $256 \Pr[z_{175} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

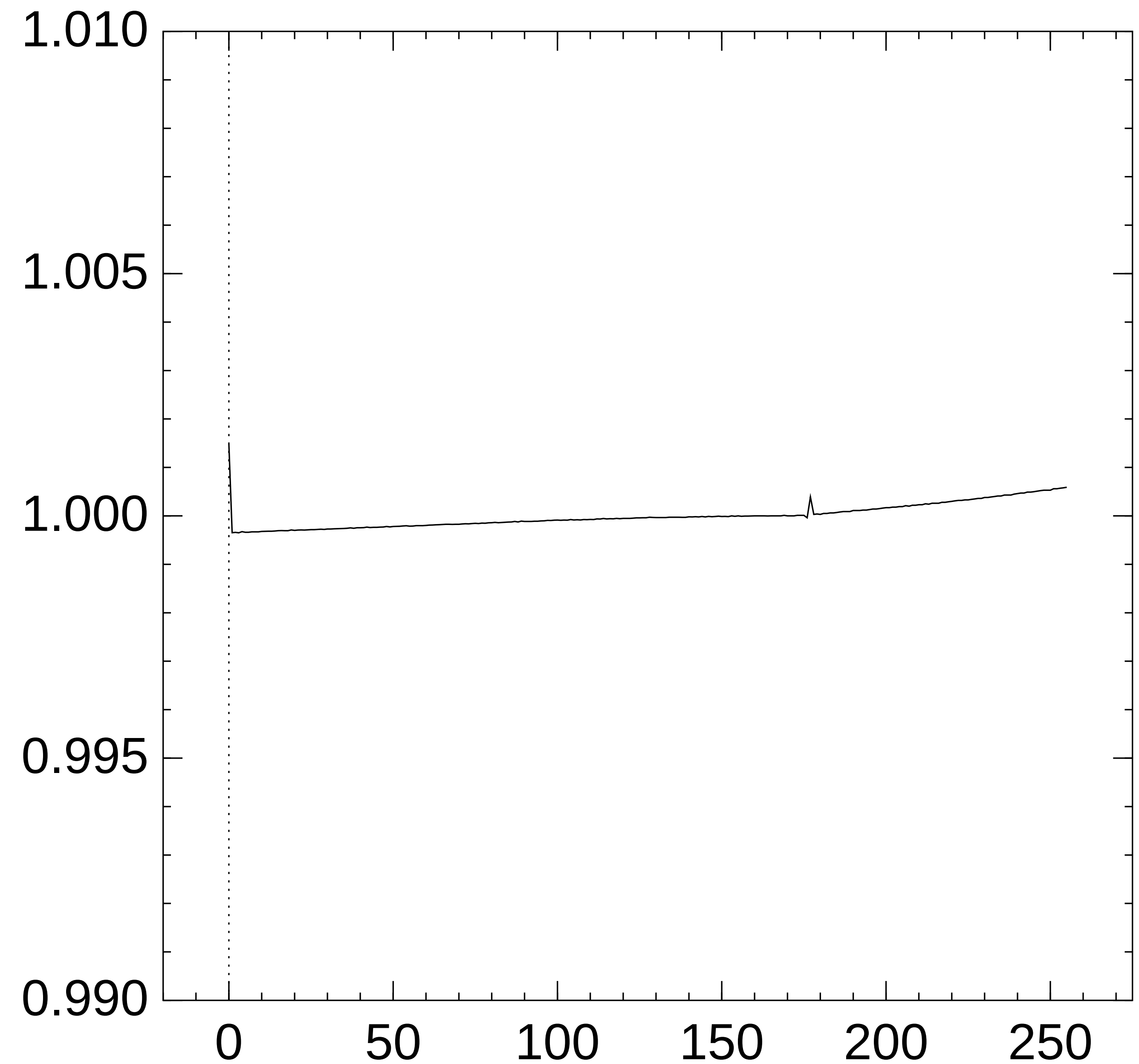
Graph of $256 \Pr[z_{176} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

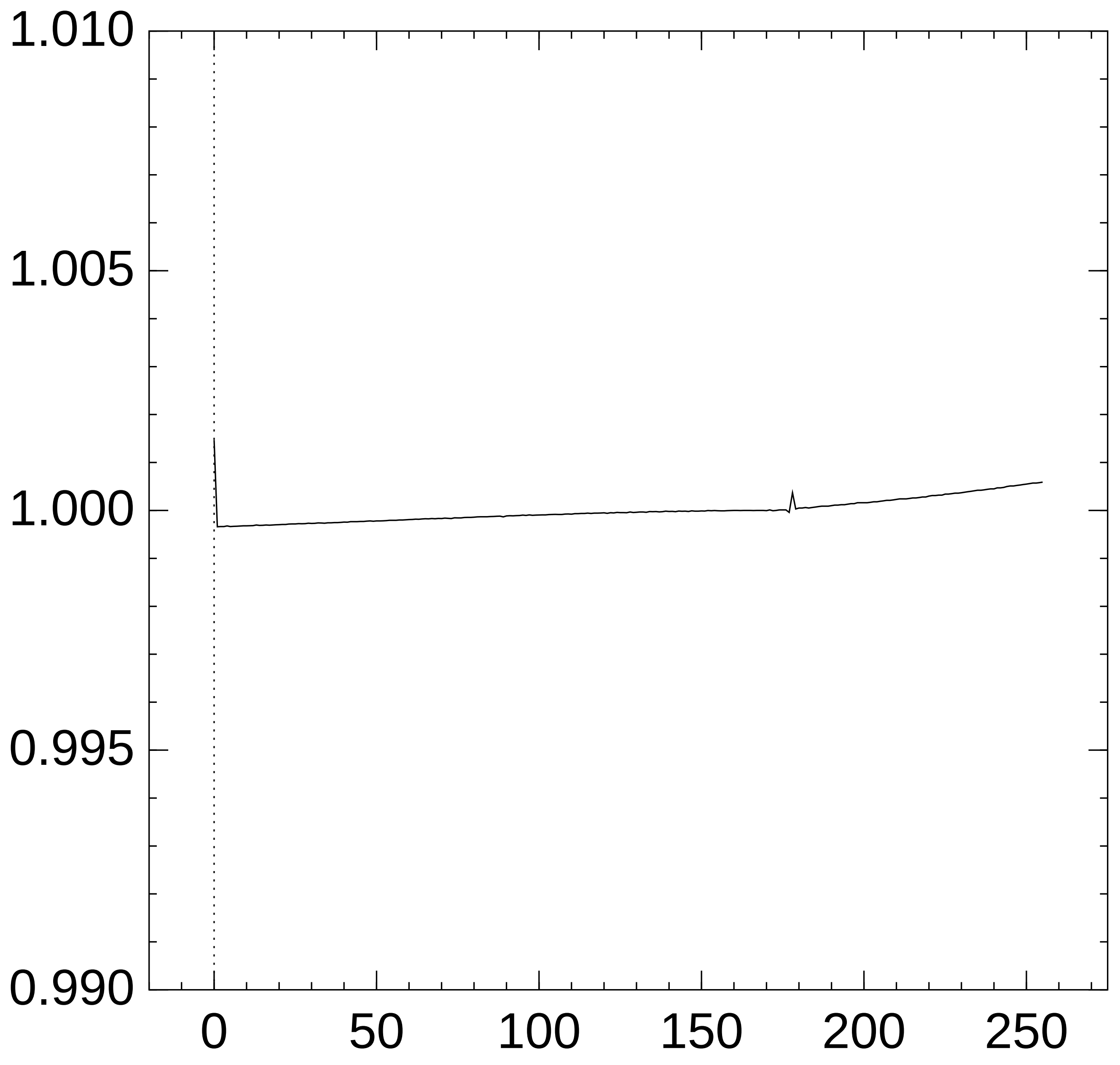
Graph of $256 \Pr[z_{177} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

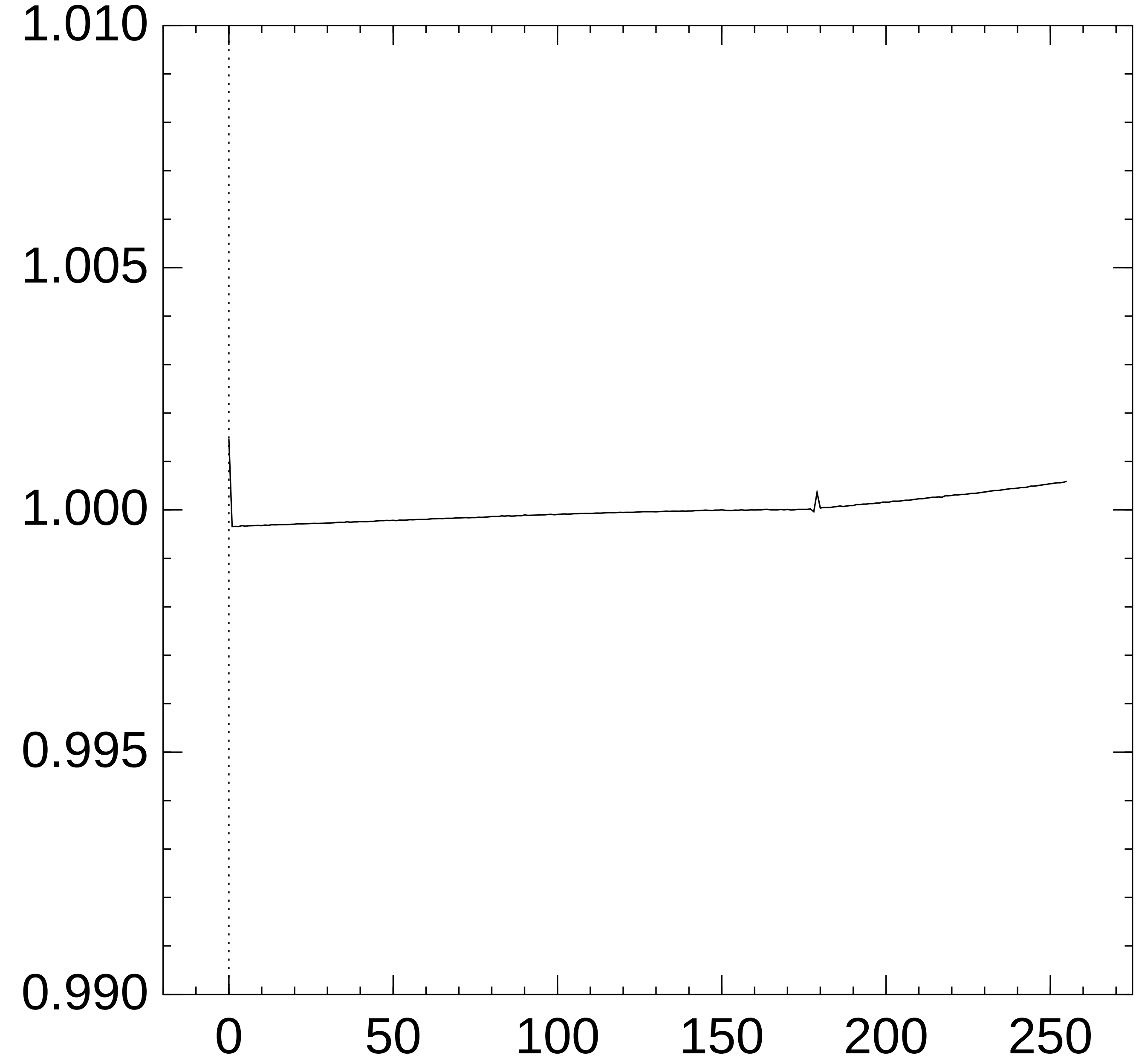
Graph of $256 \Pr[z_{178} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

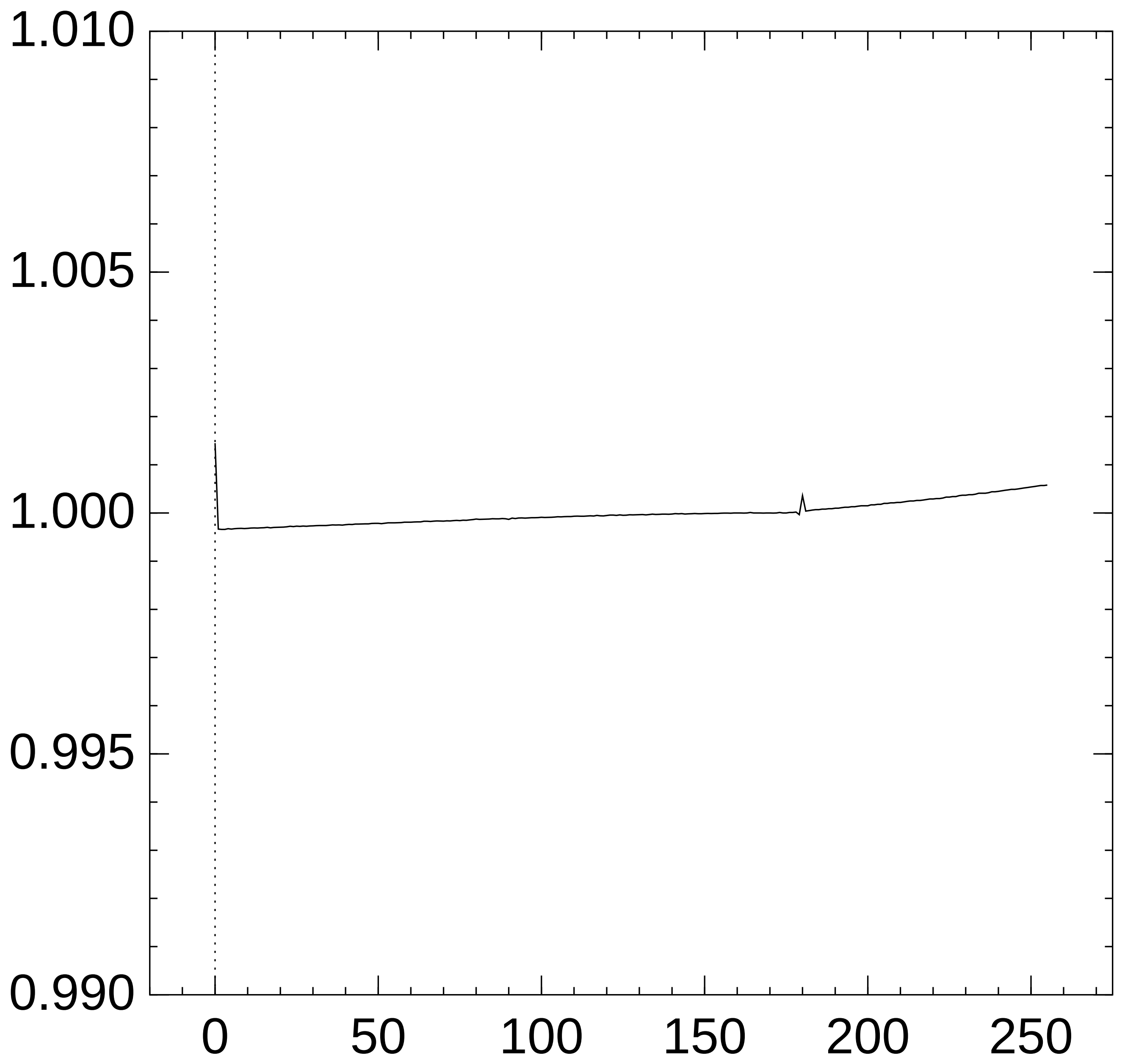
Graph of $256\,\Pr[z_{179} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

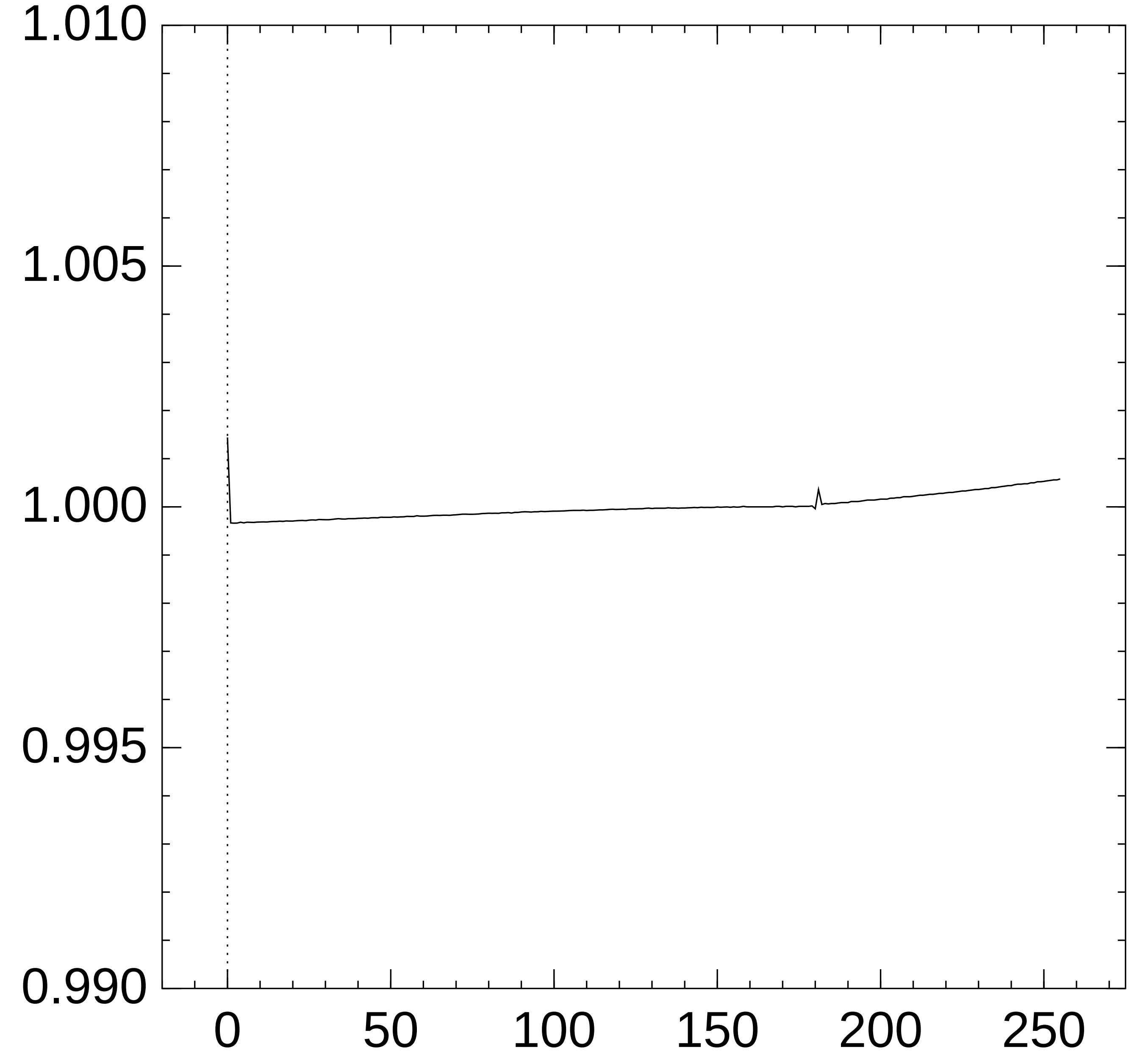
Graph of $256 \Pr[z_{180} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

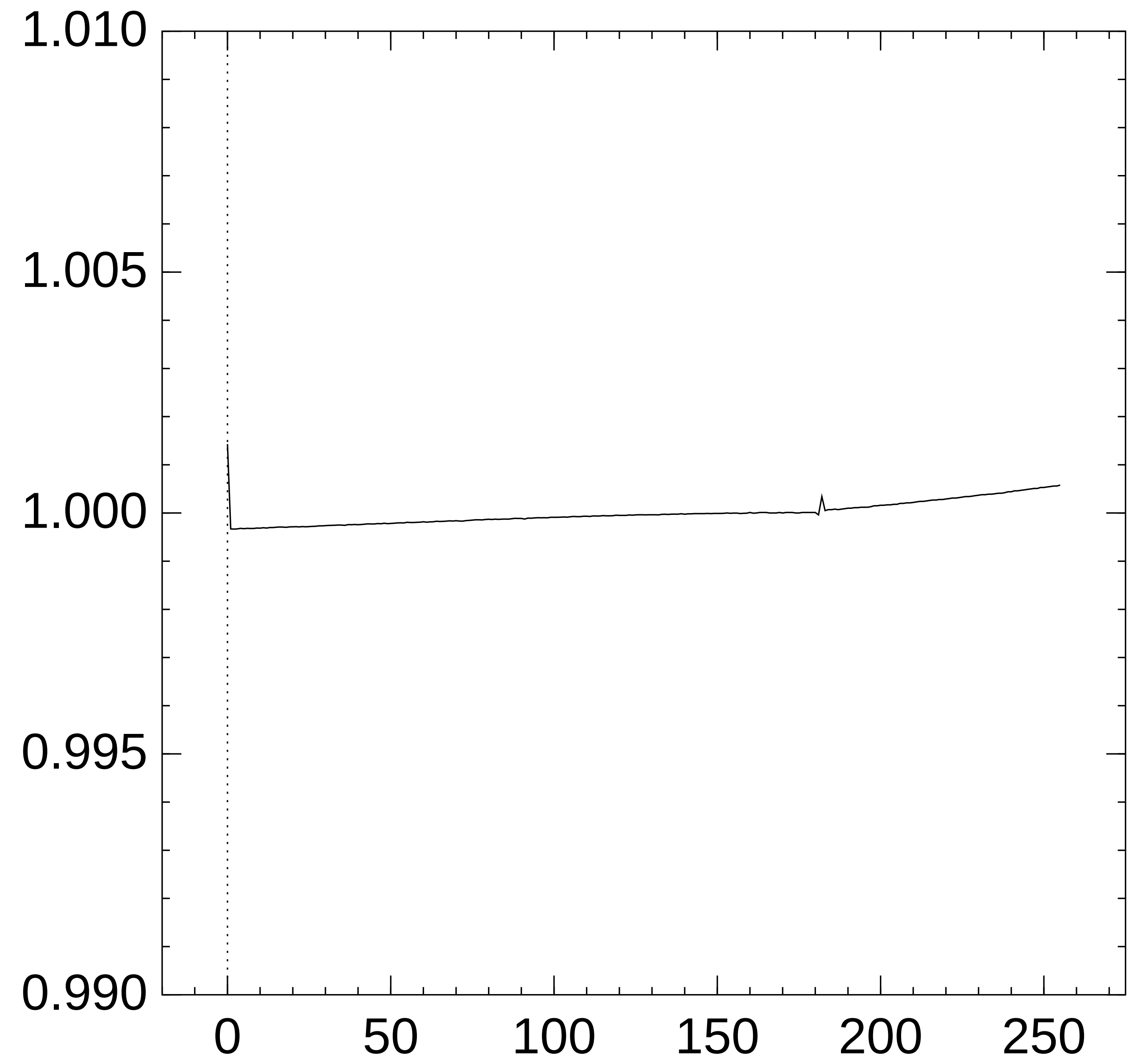
Graph of $256 \Pr[z_{181} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

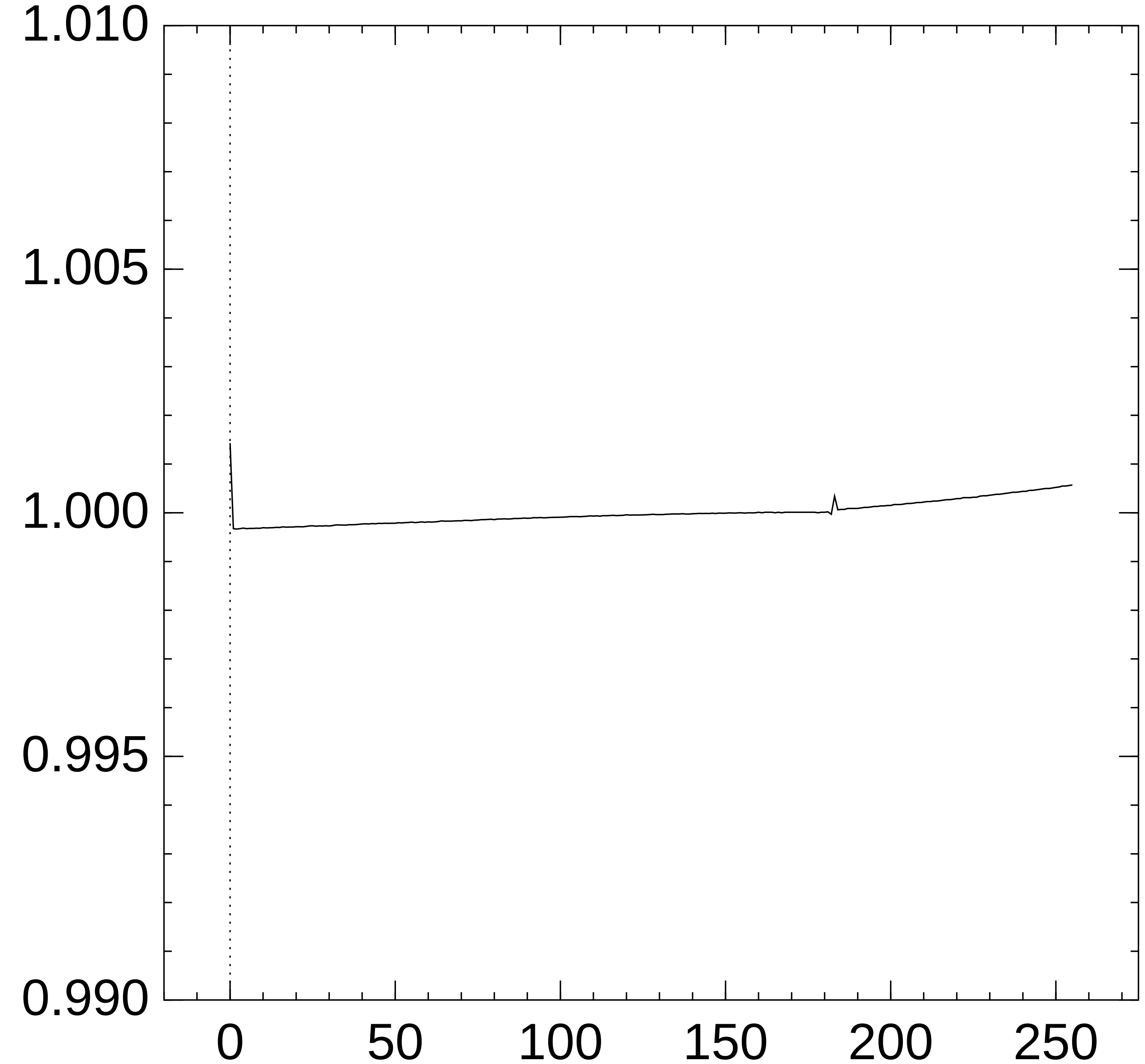
Graph of $256 \Pr[z_{182} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

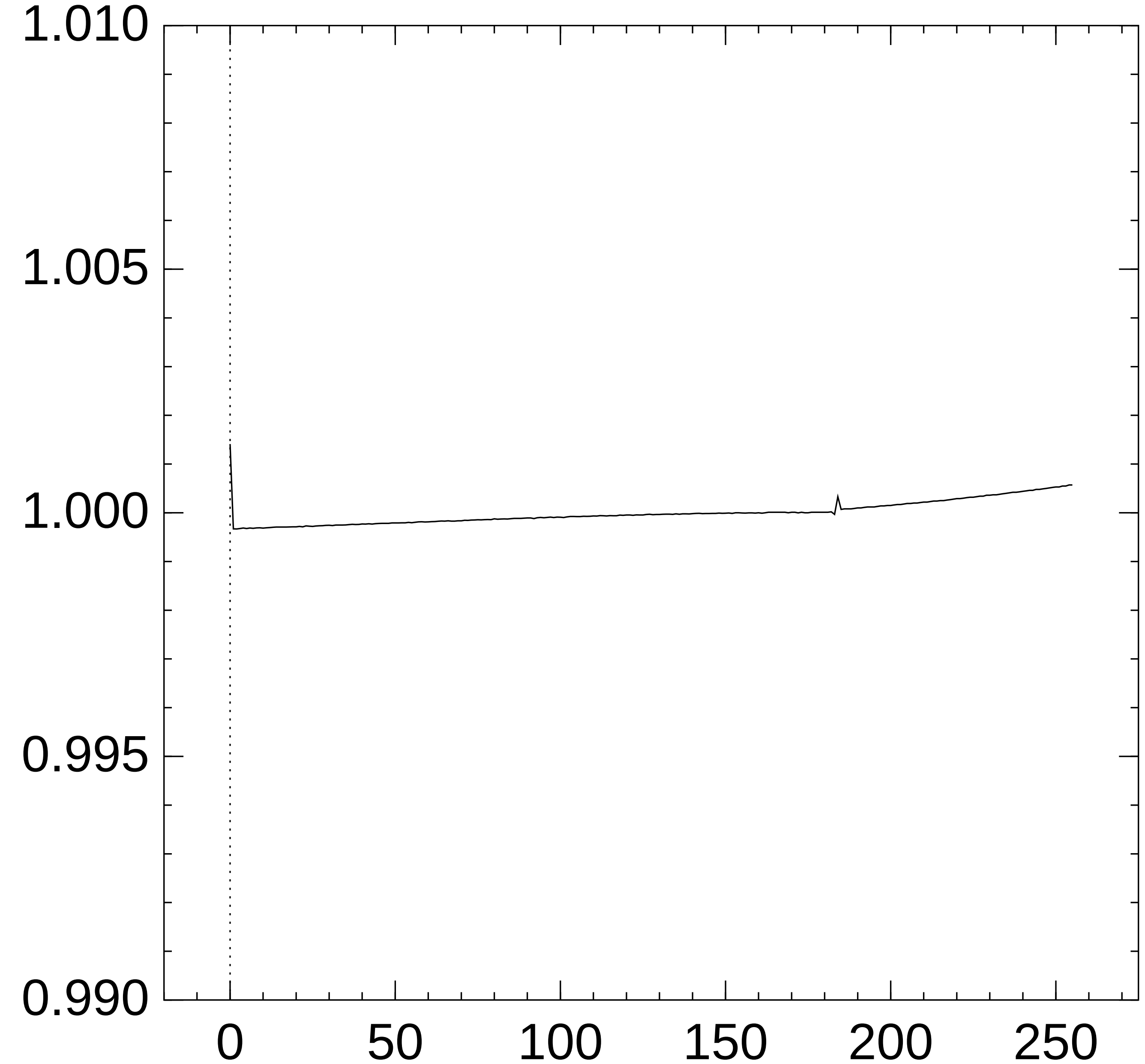
Graph of $256 \Pr[z_{183} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

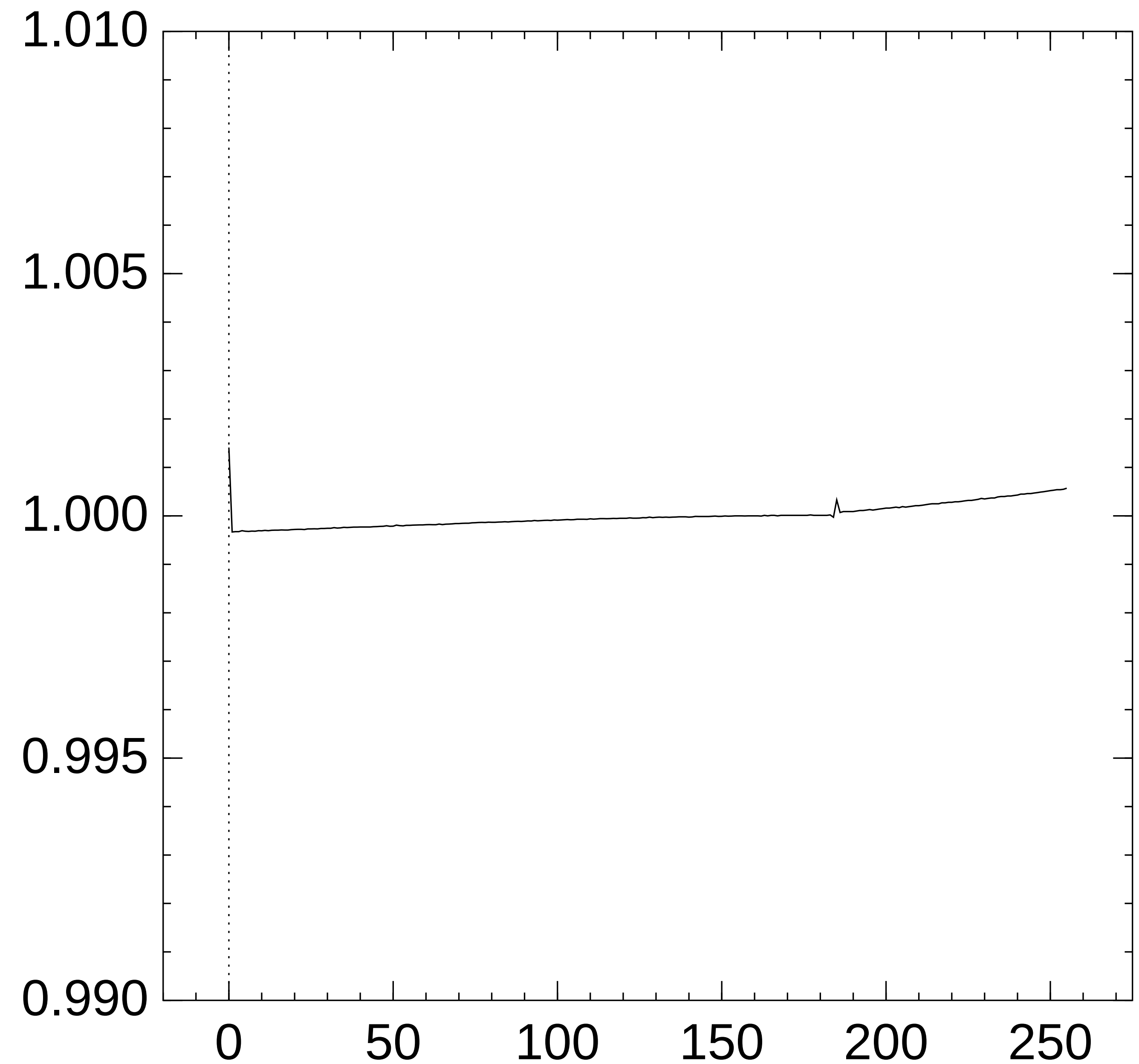
Graph of $256 \Pr[z_{184} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

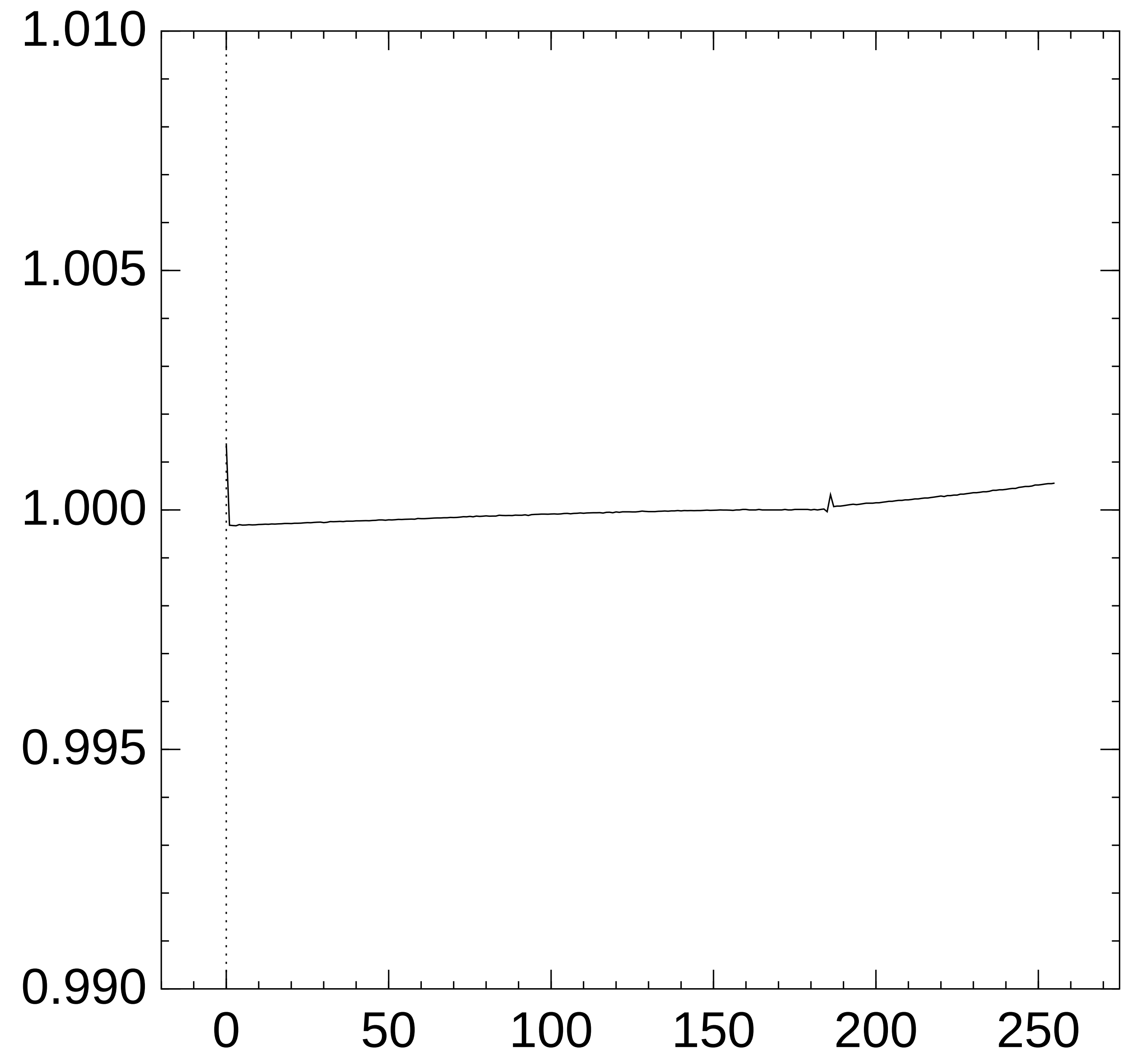
Graph of $256 \Pr[z_{185} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

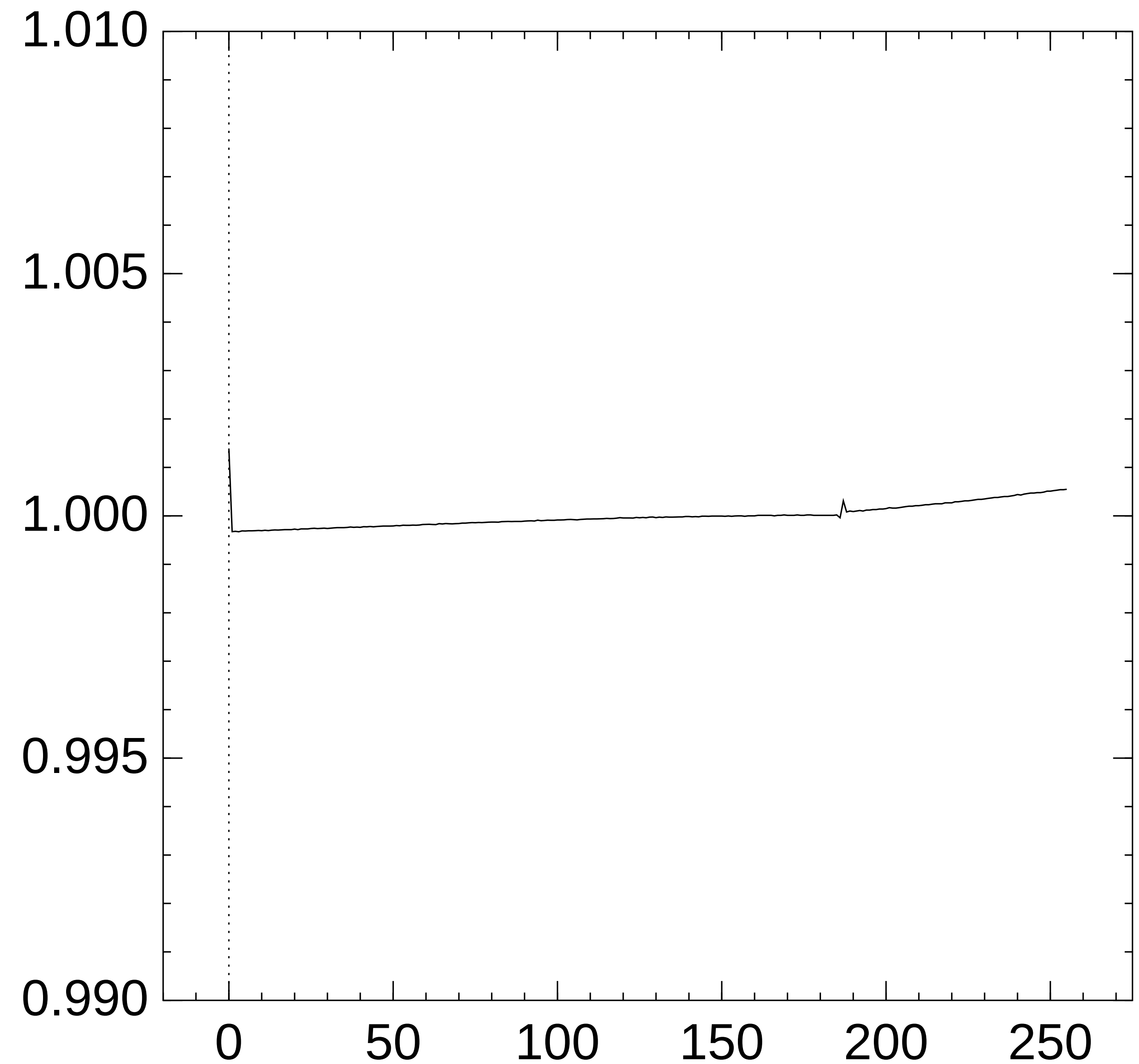
Graph of $256 \Pr[z_{186} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

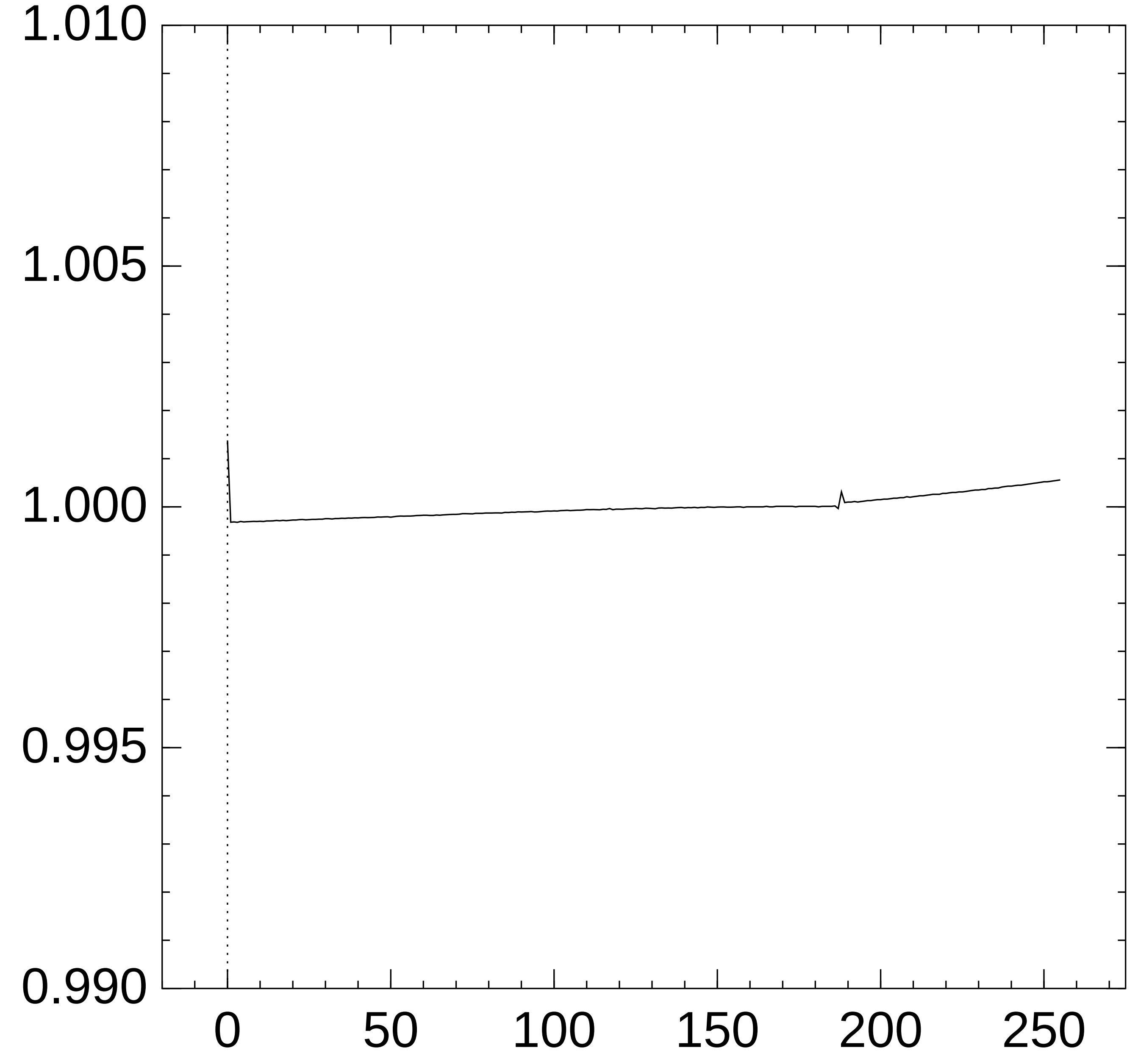
Graph of $256 \Pr[z_{187} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

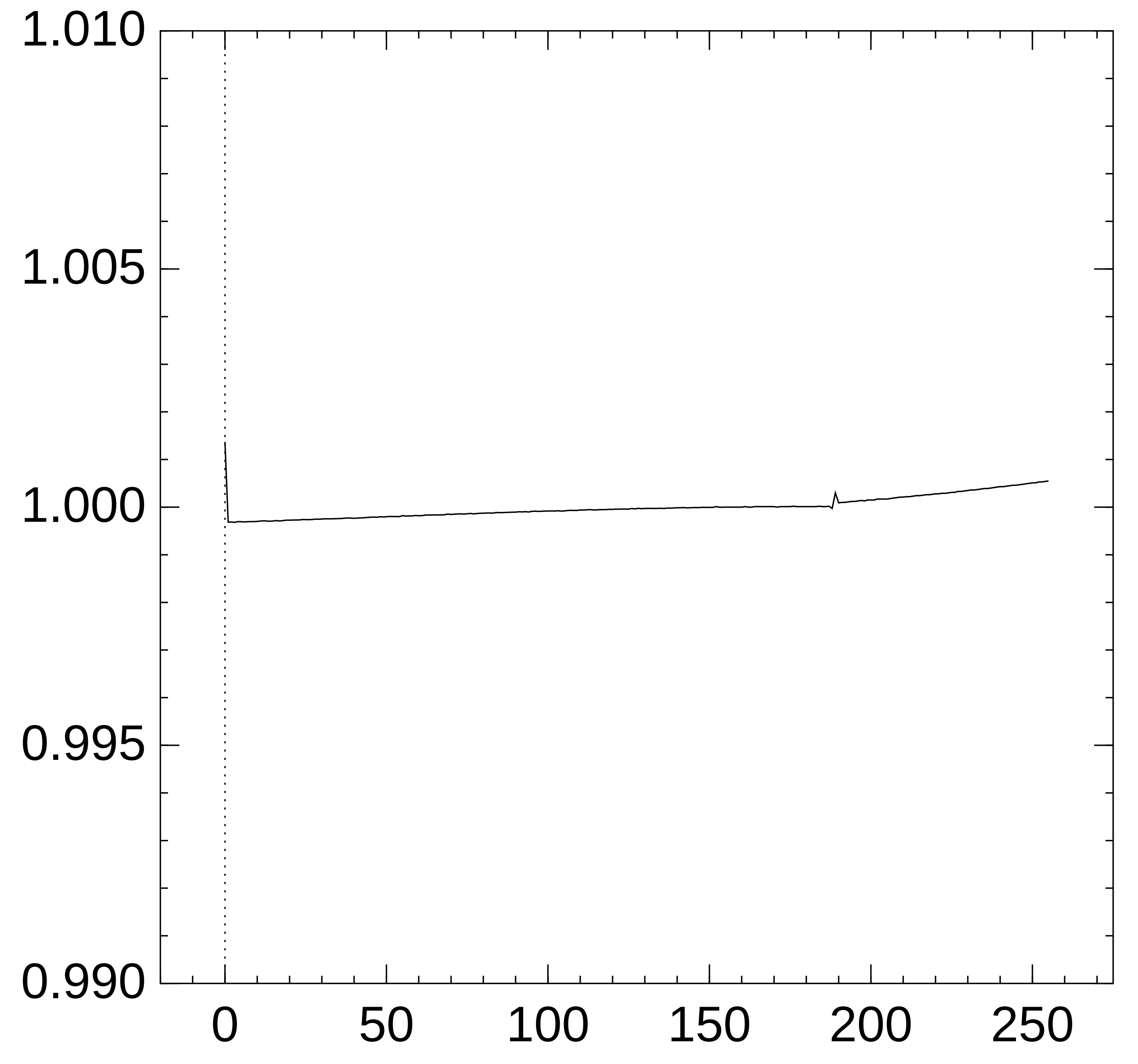
Graph of $256 \Pr[z_{188} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

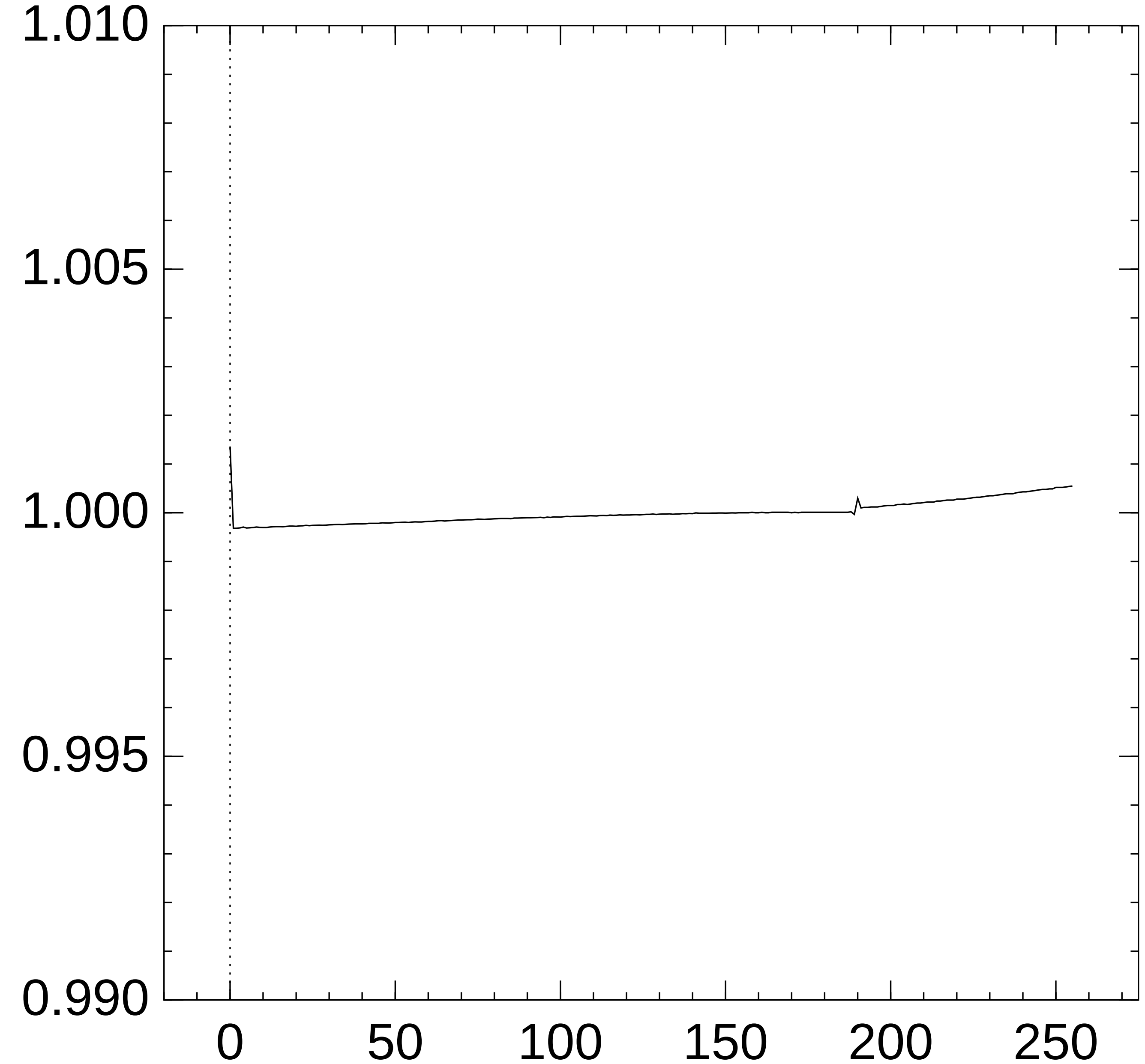
Graph of $256 \Pr[z_{189} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

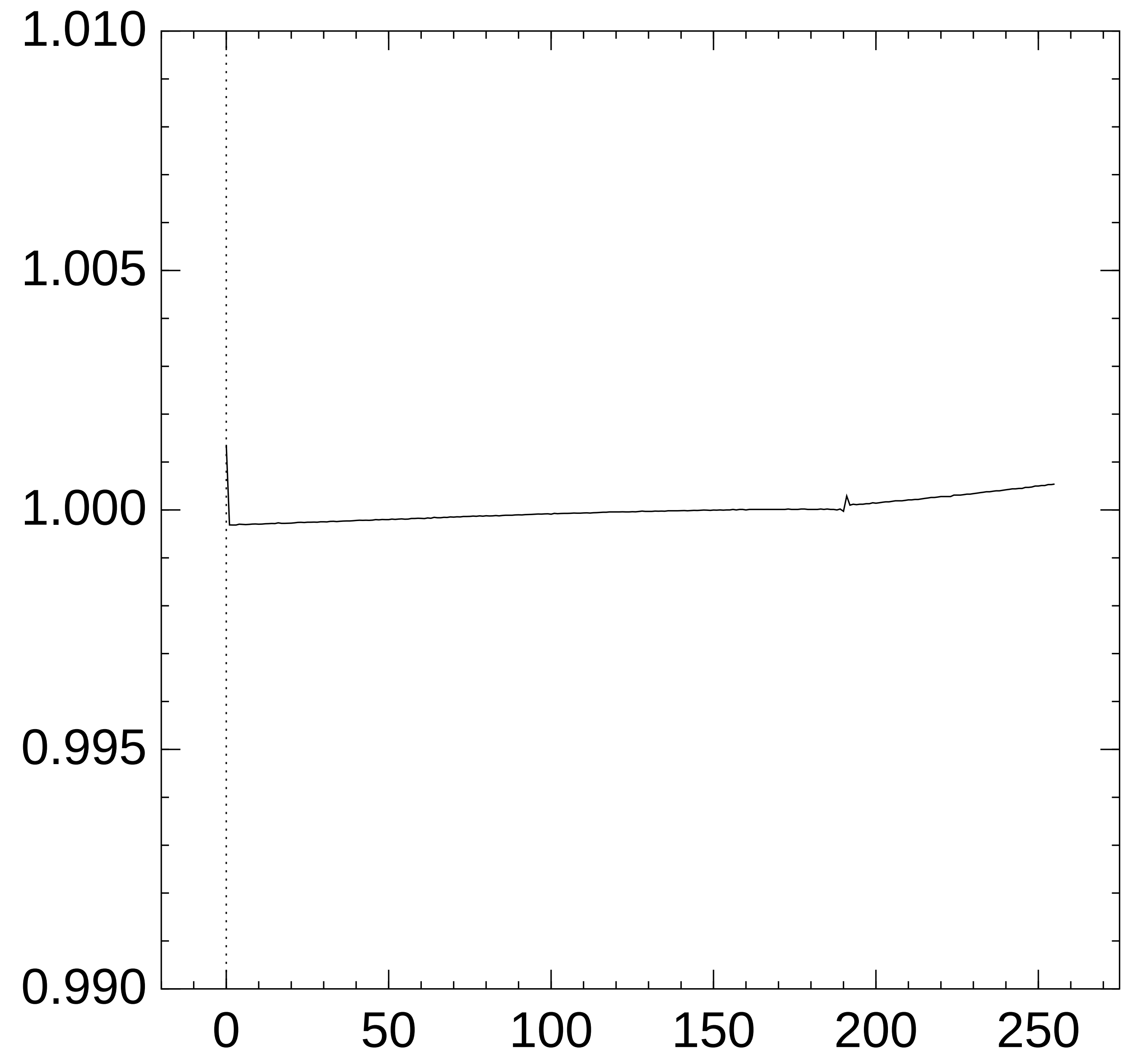
Graph of $256 \Pr[z_{190} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

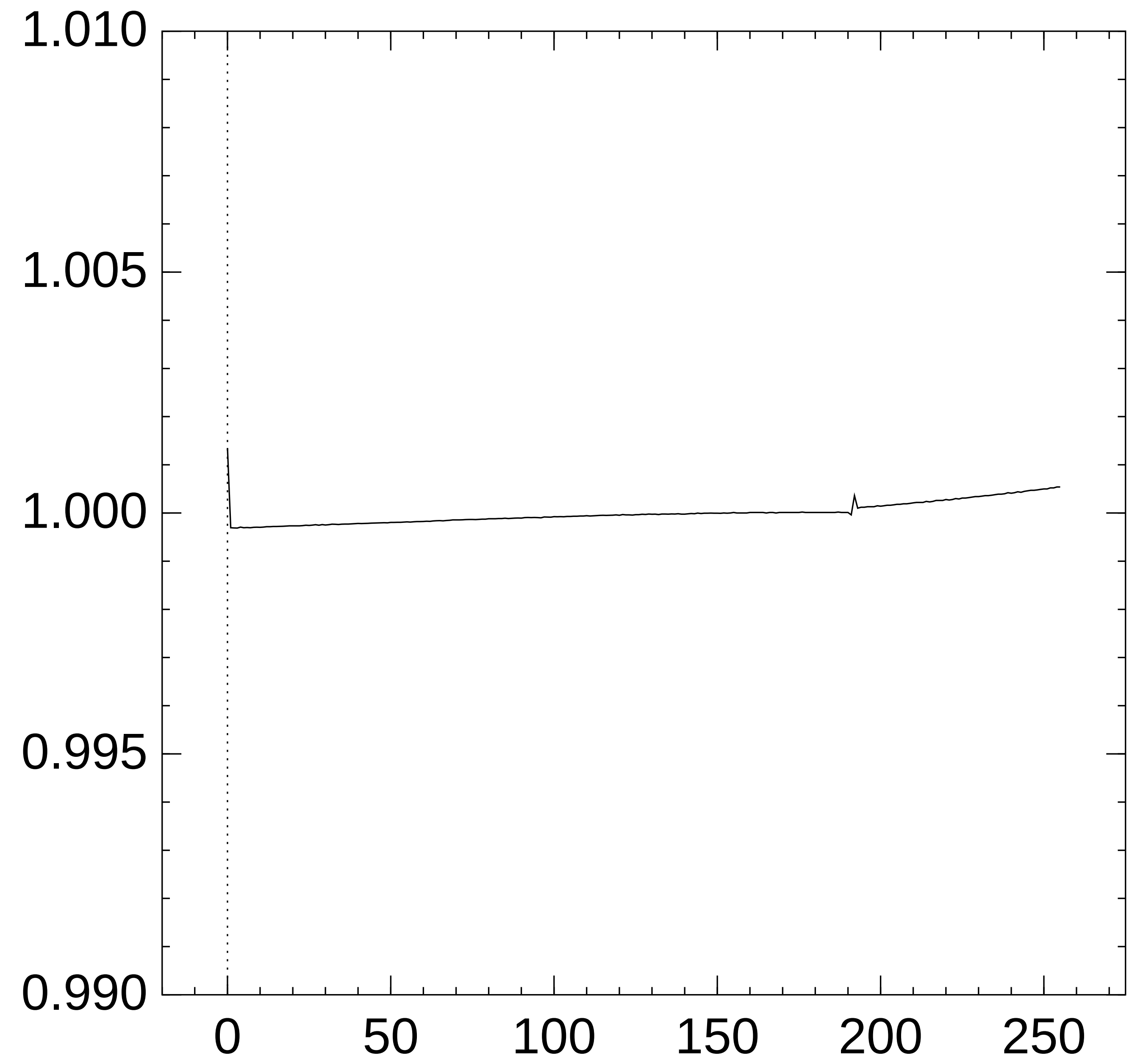
Graph of $256 \Pr[z_{191} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

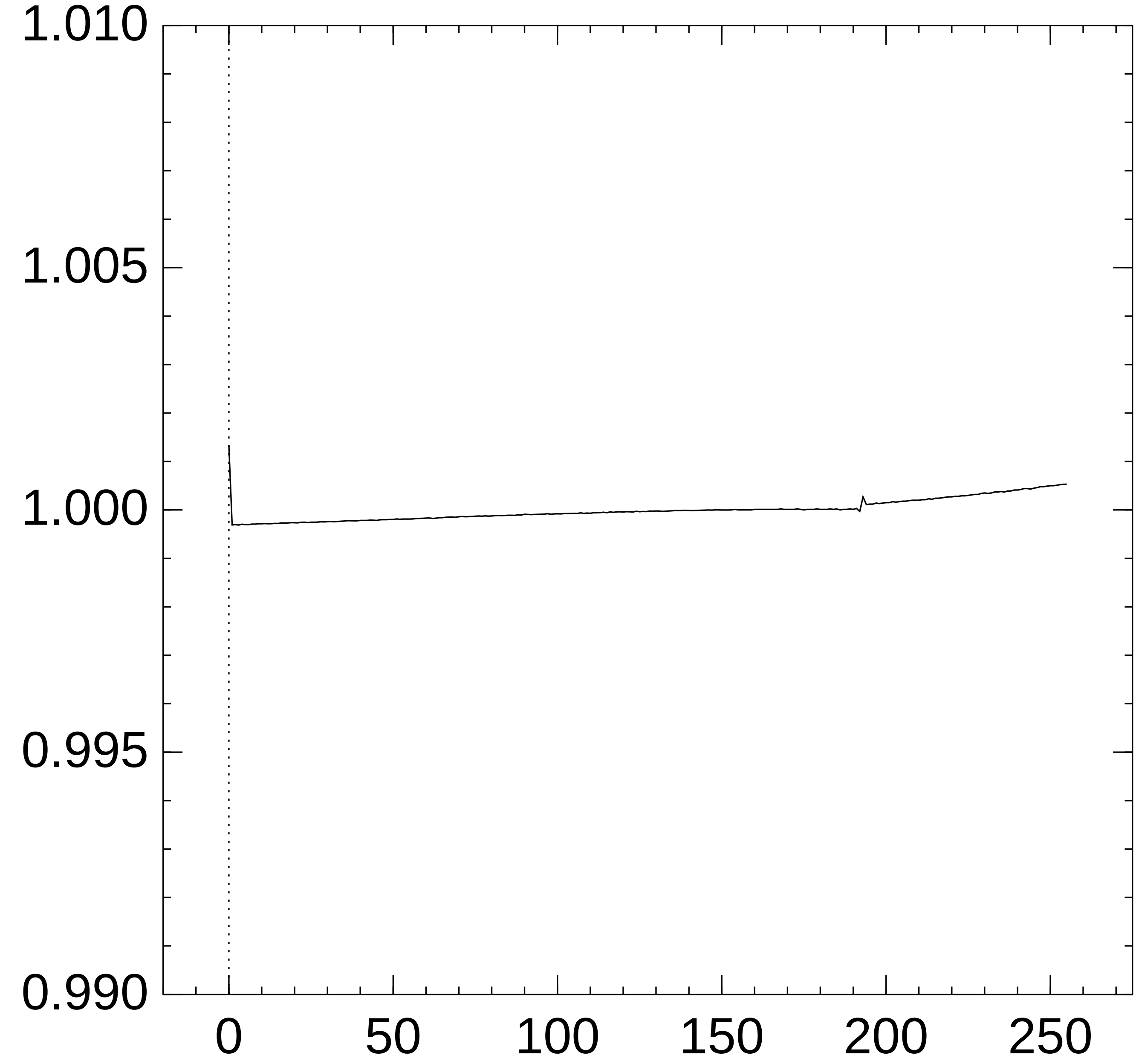
Graph of $256 \Pr[z_{192} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

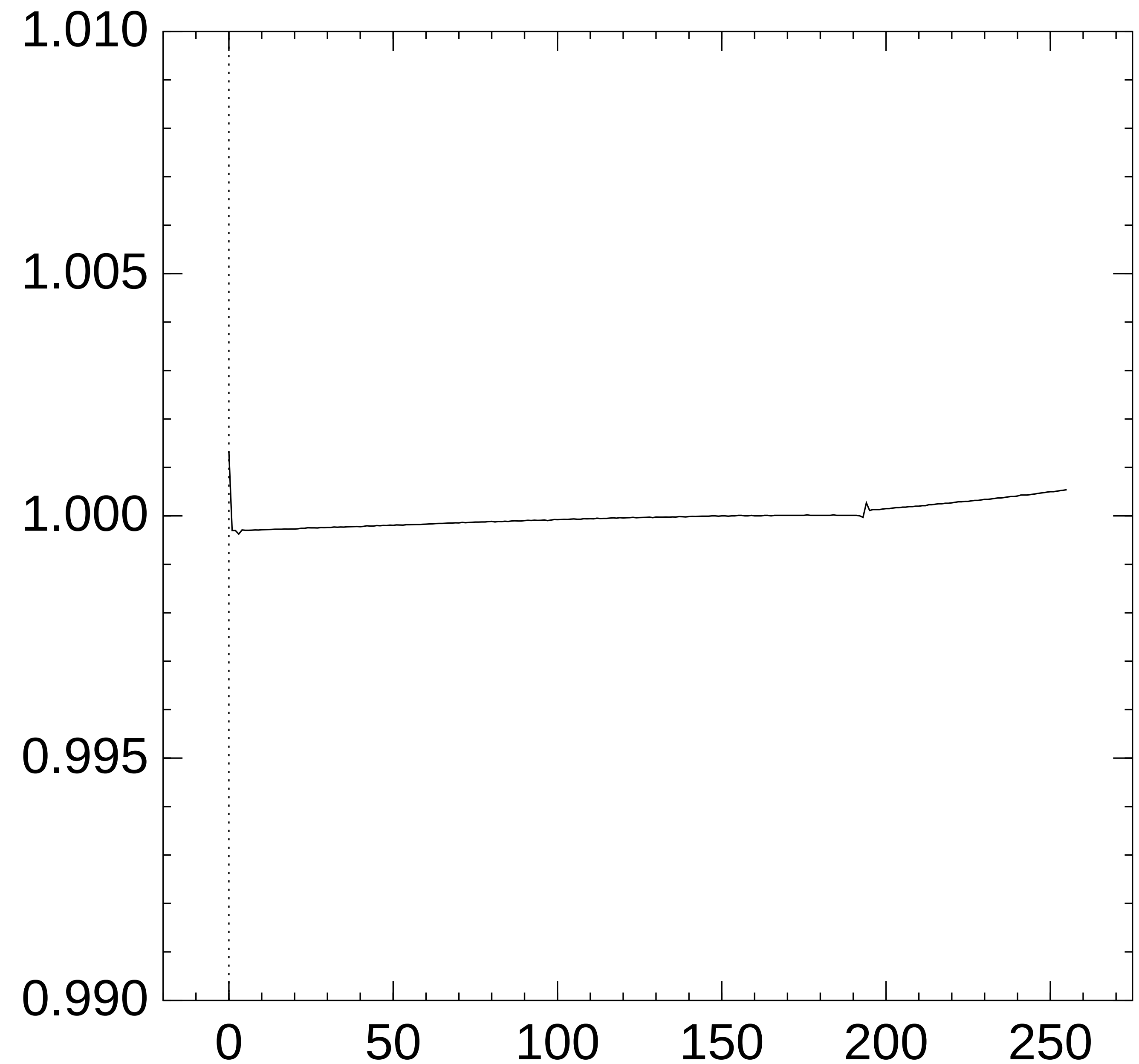
Graph of $256 \Pr[z_{193} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

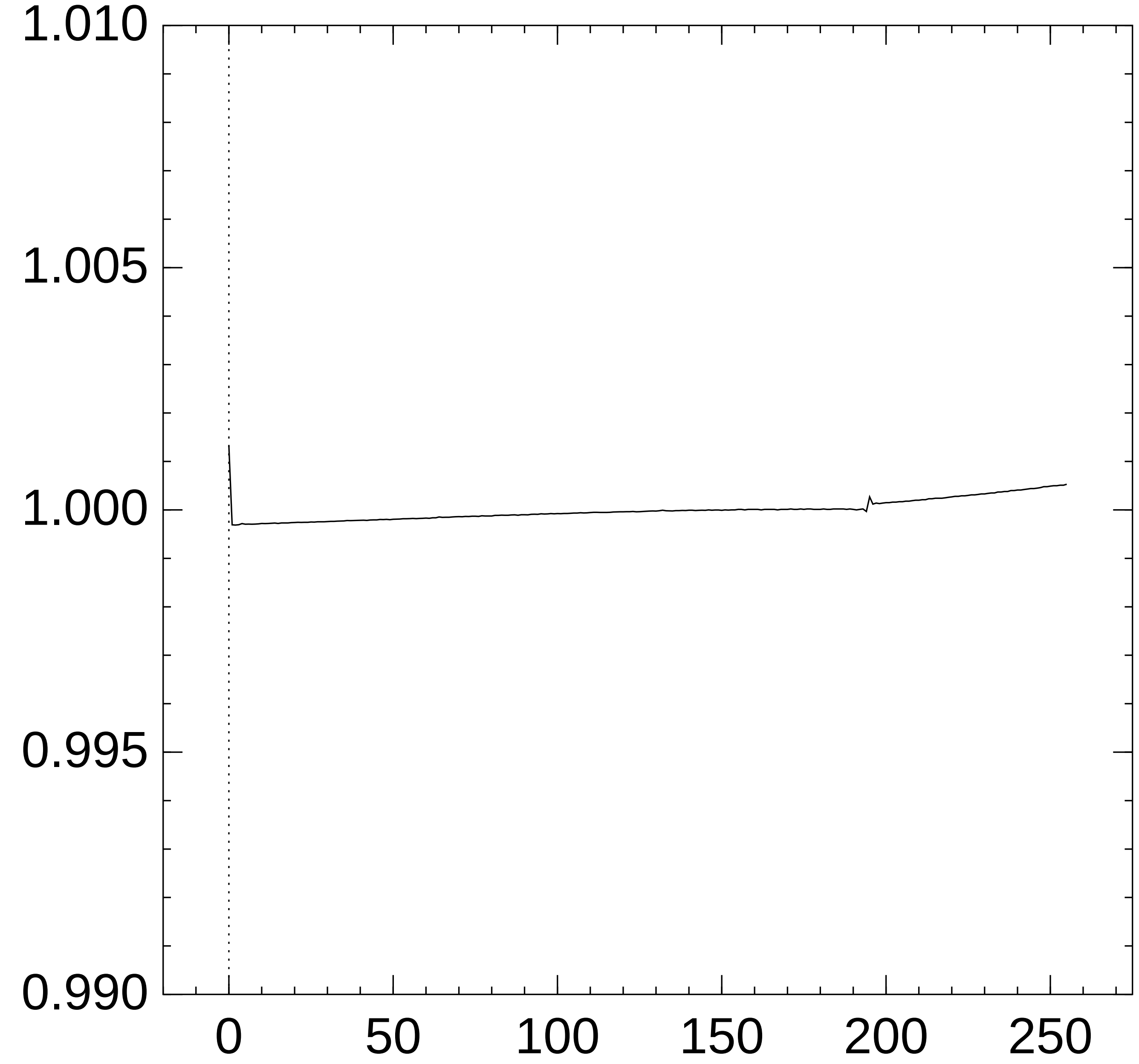
Graph of $256 \Pr[z_{194} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

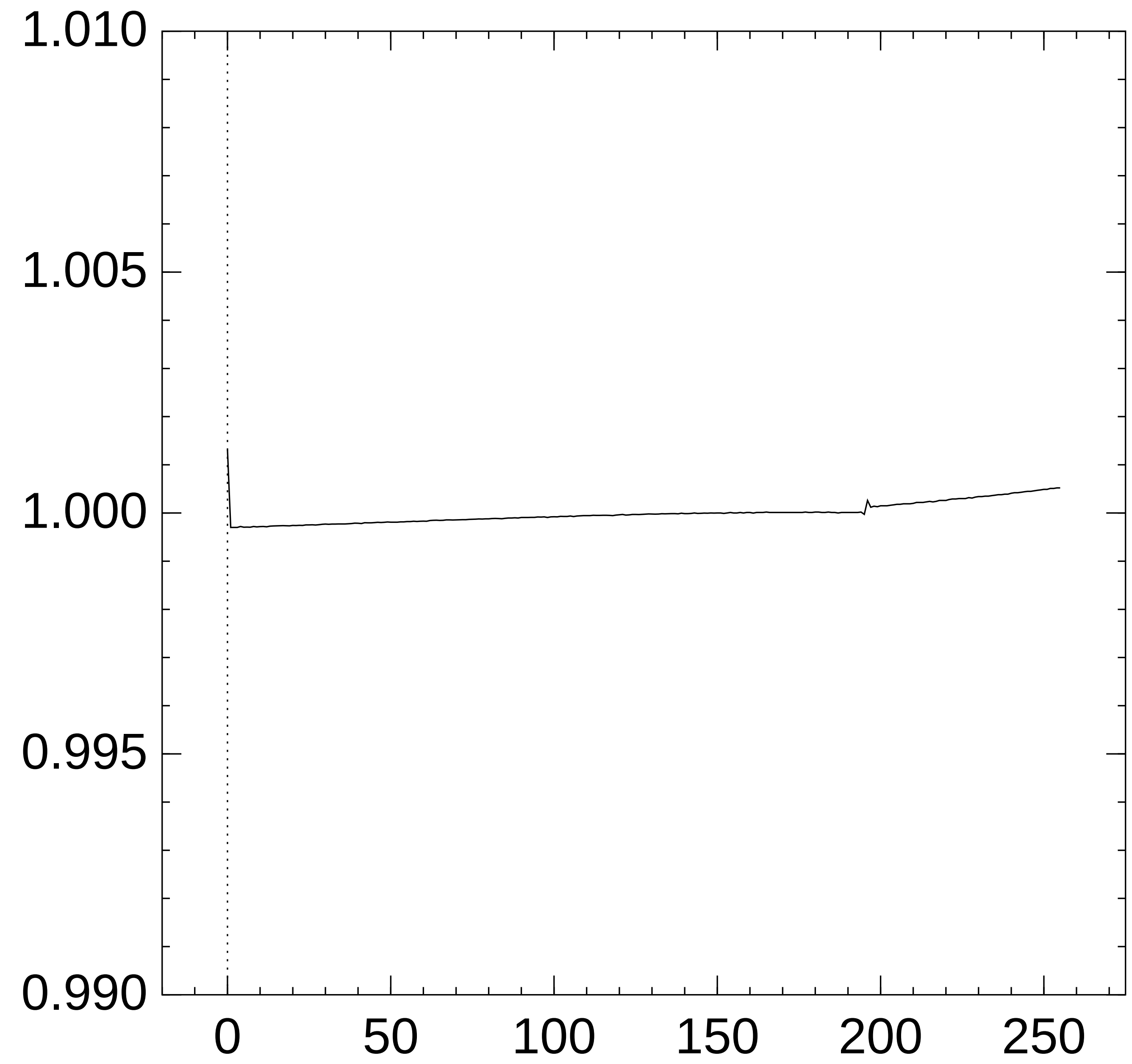
Graph of $256 \Pr[z_{195} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

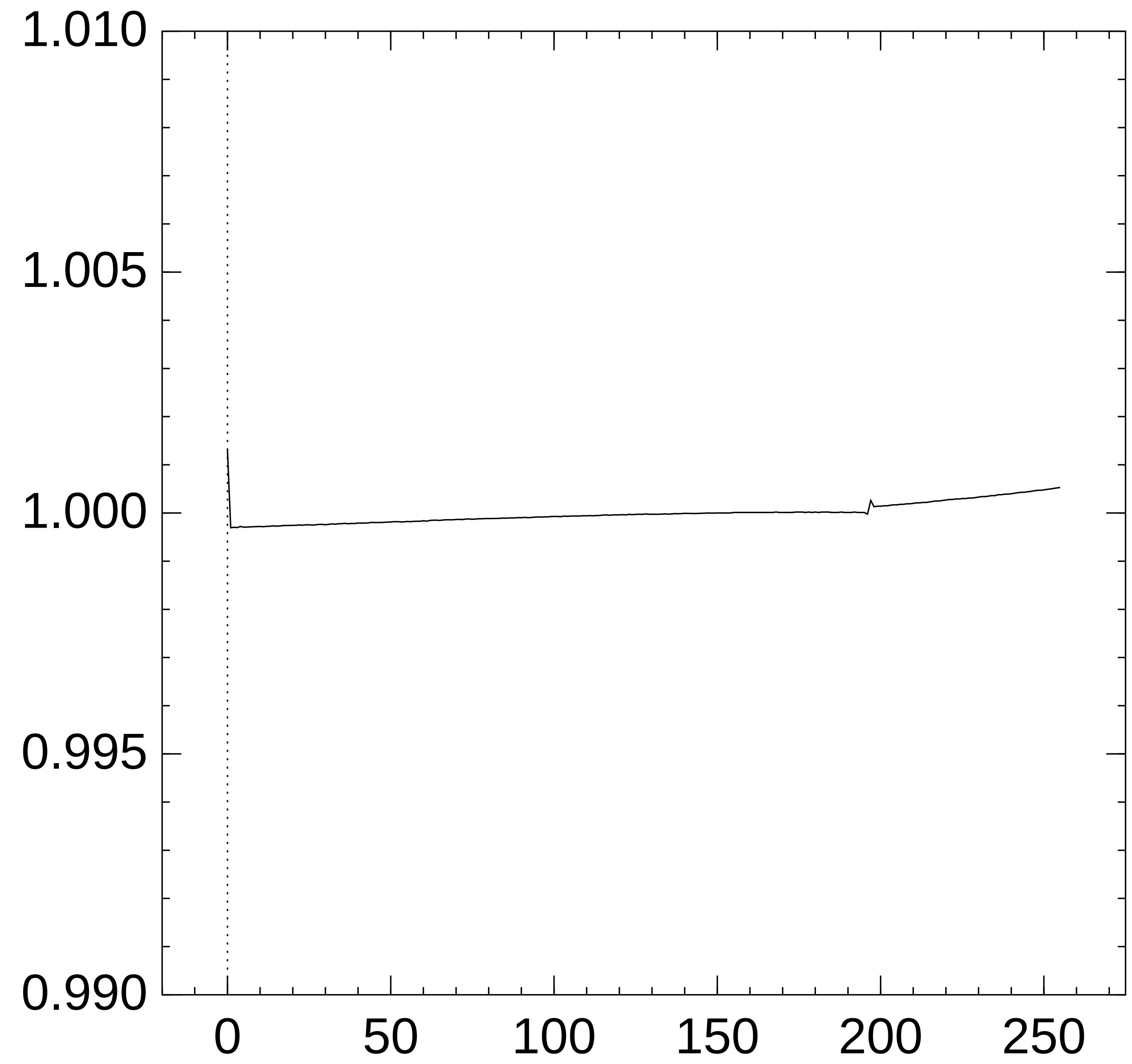
Graph of $256 \Pr[z_{196} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

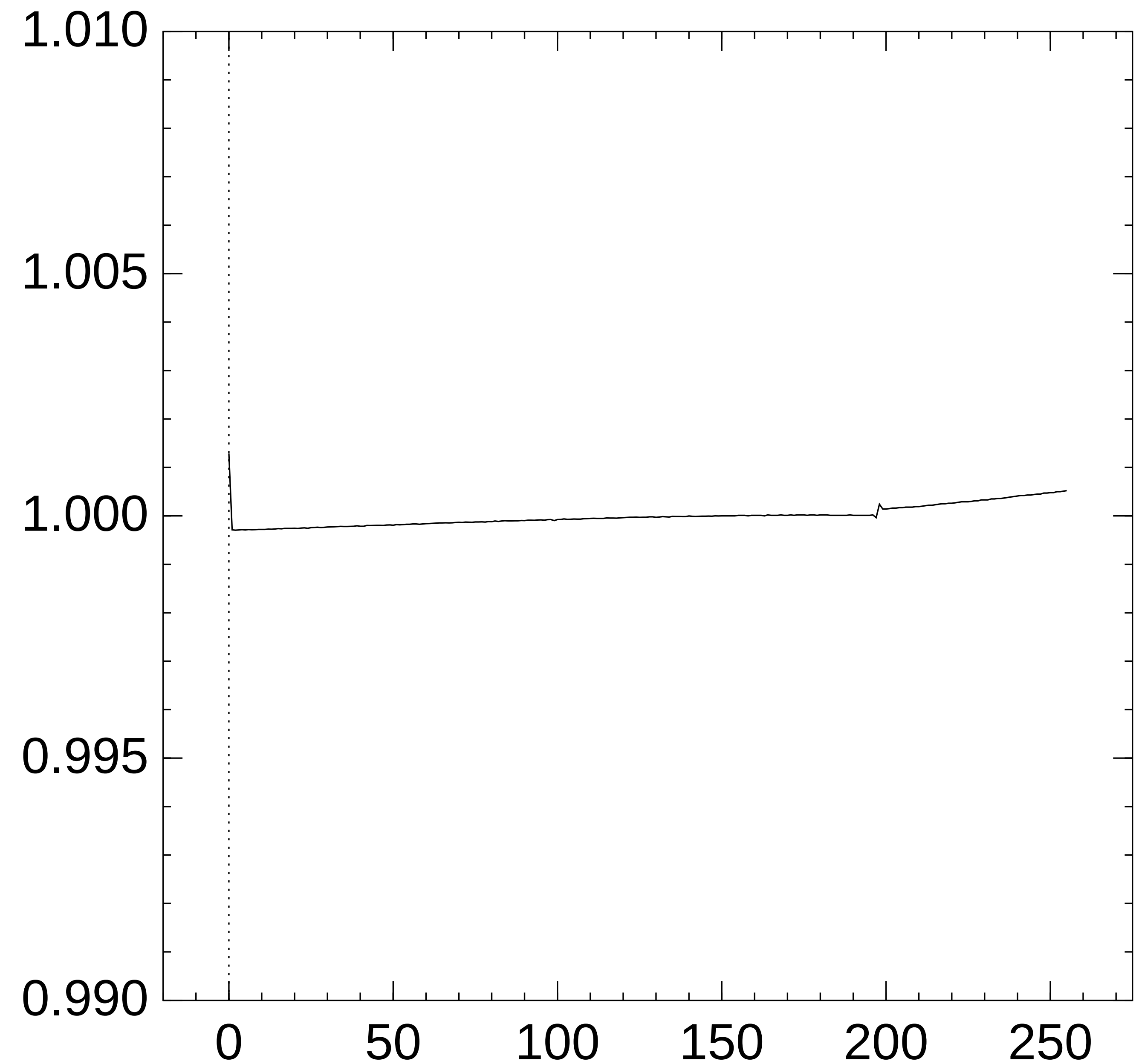
Graph of $256 \Pr[z_{197} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

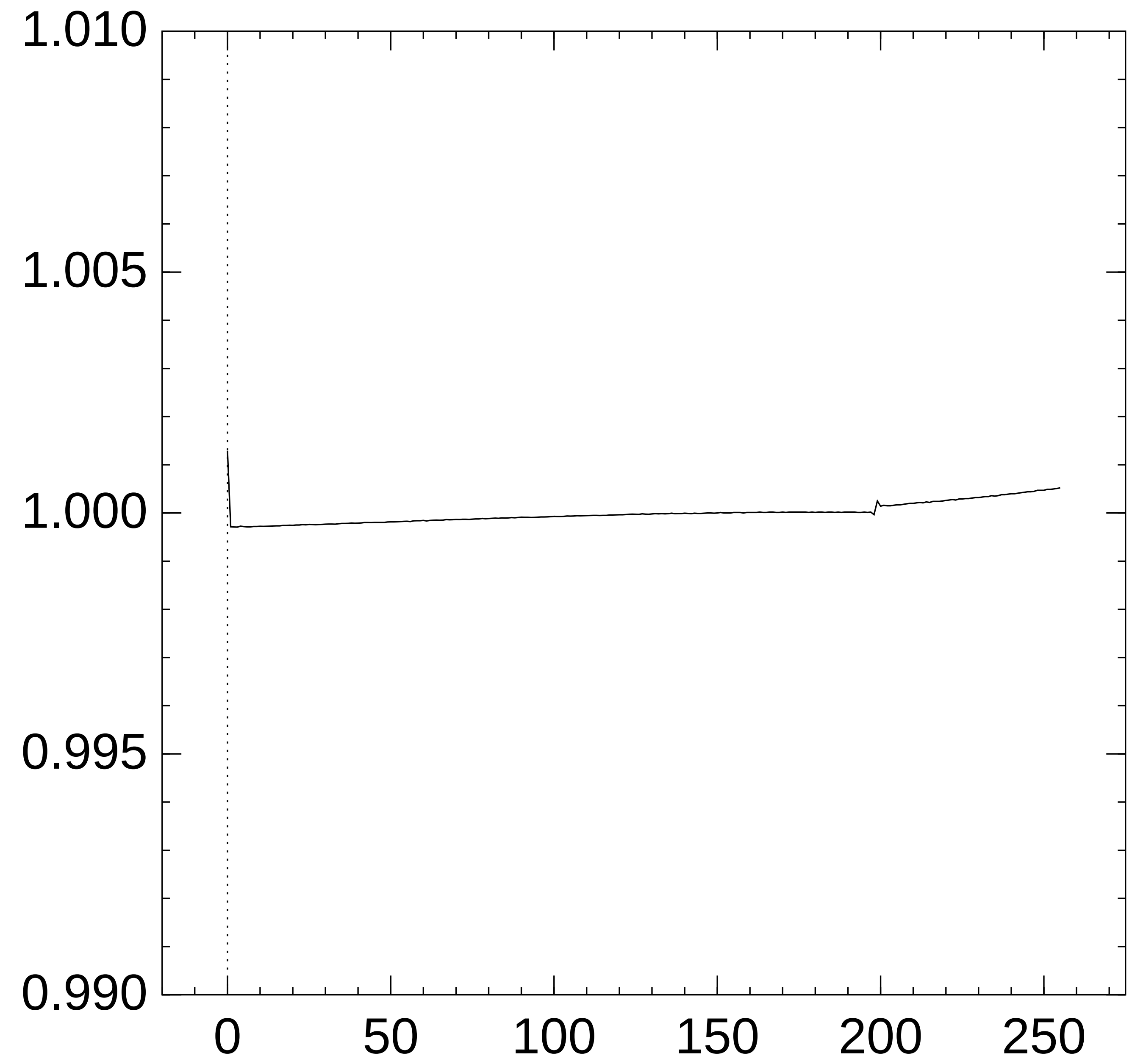
Graph of $256 \Pr[z_{198} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

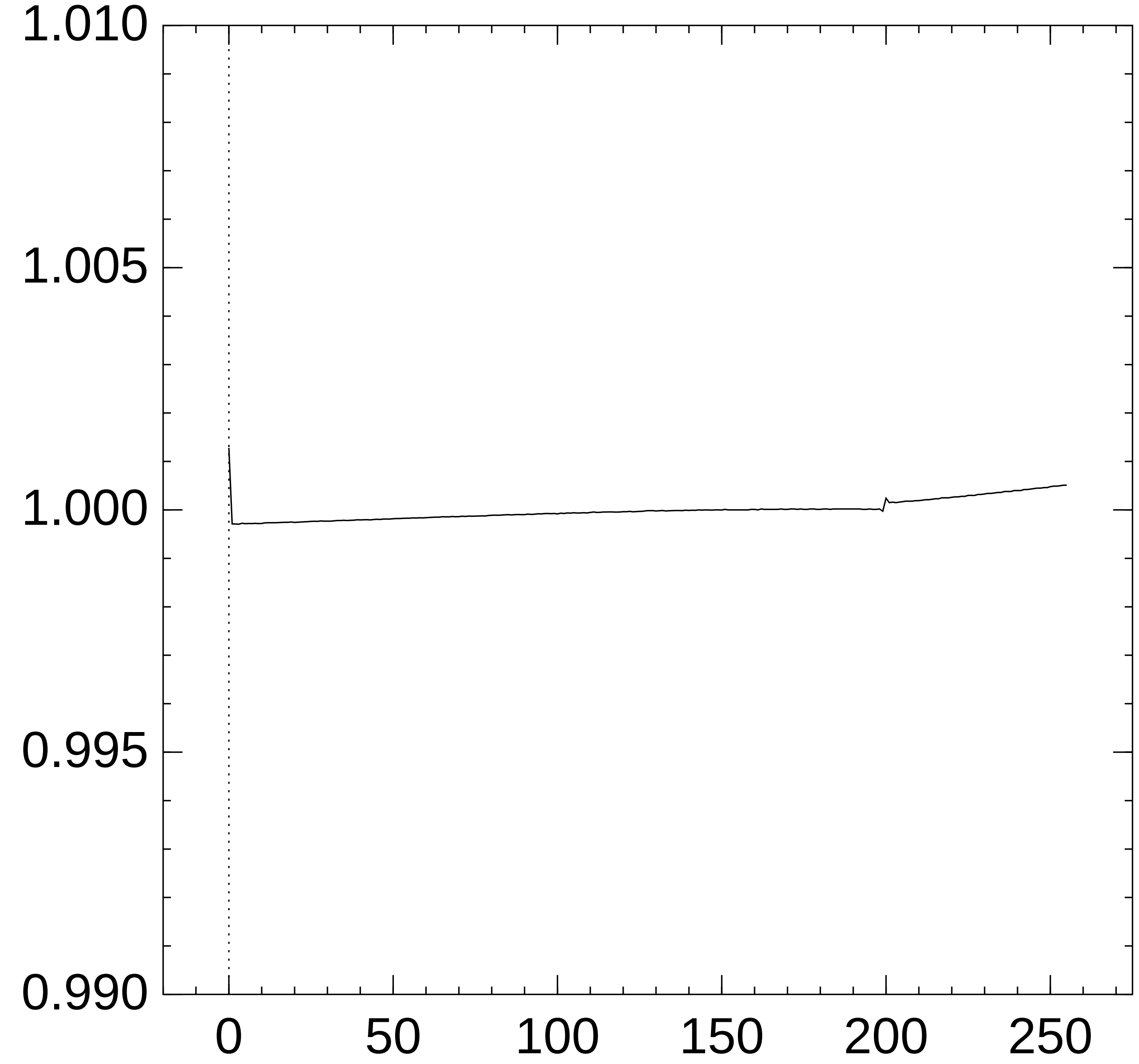
Graph of $256 \Pr[z_{199} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

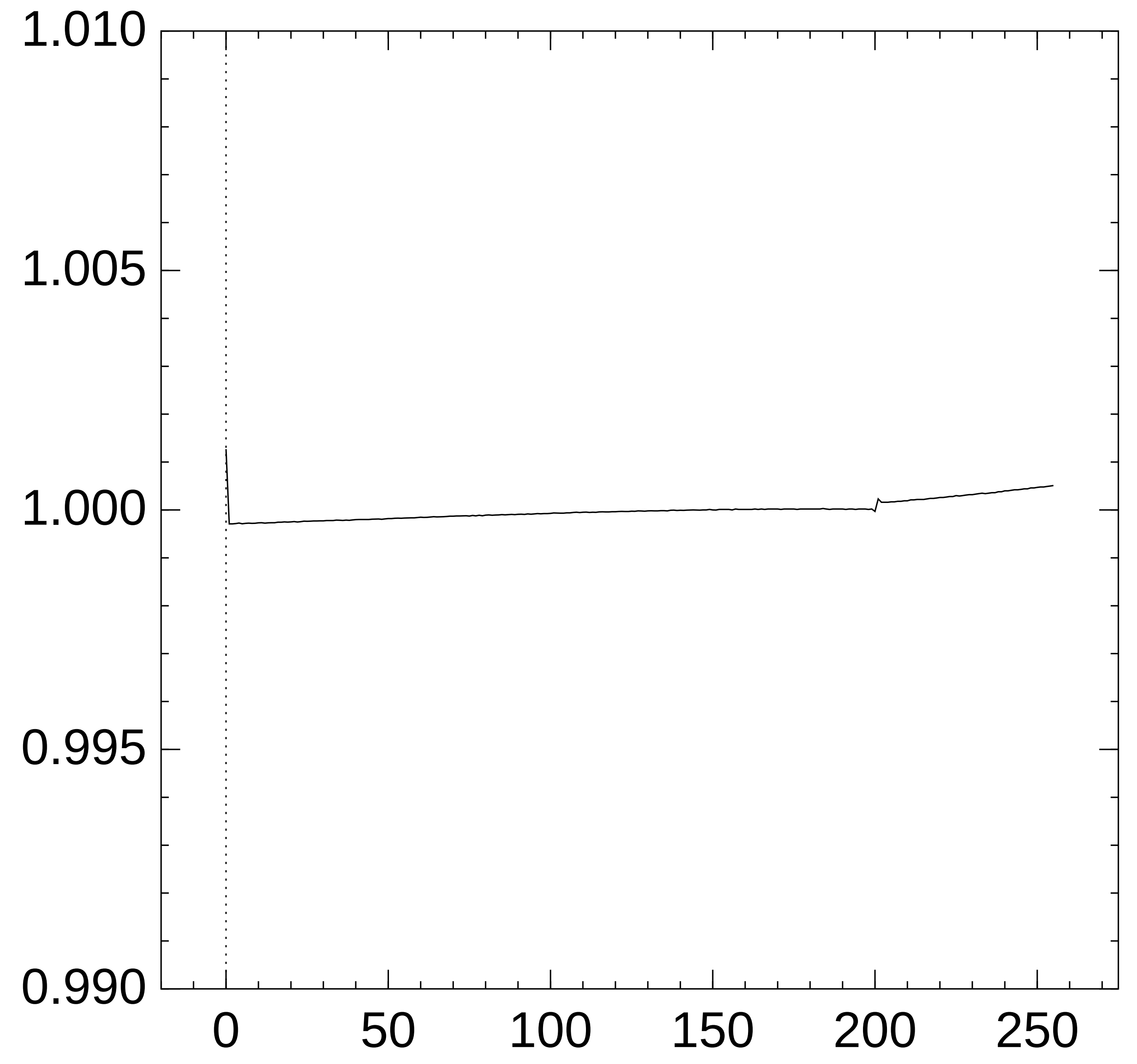
Graph of $256 \Pr[z_{200} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

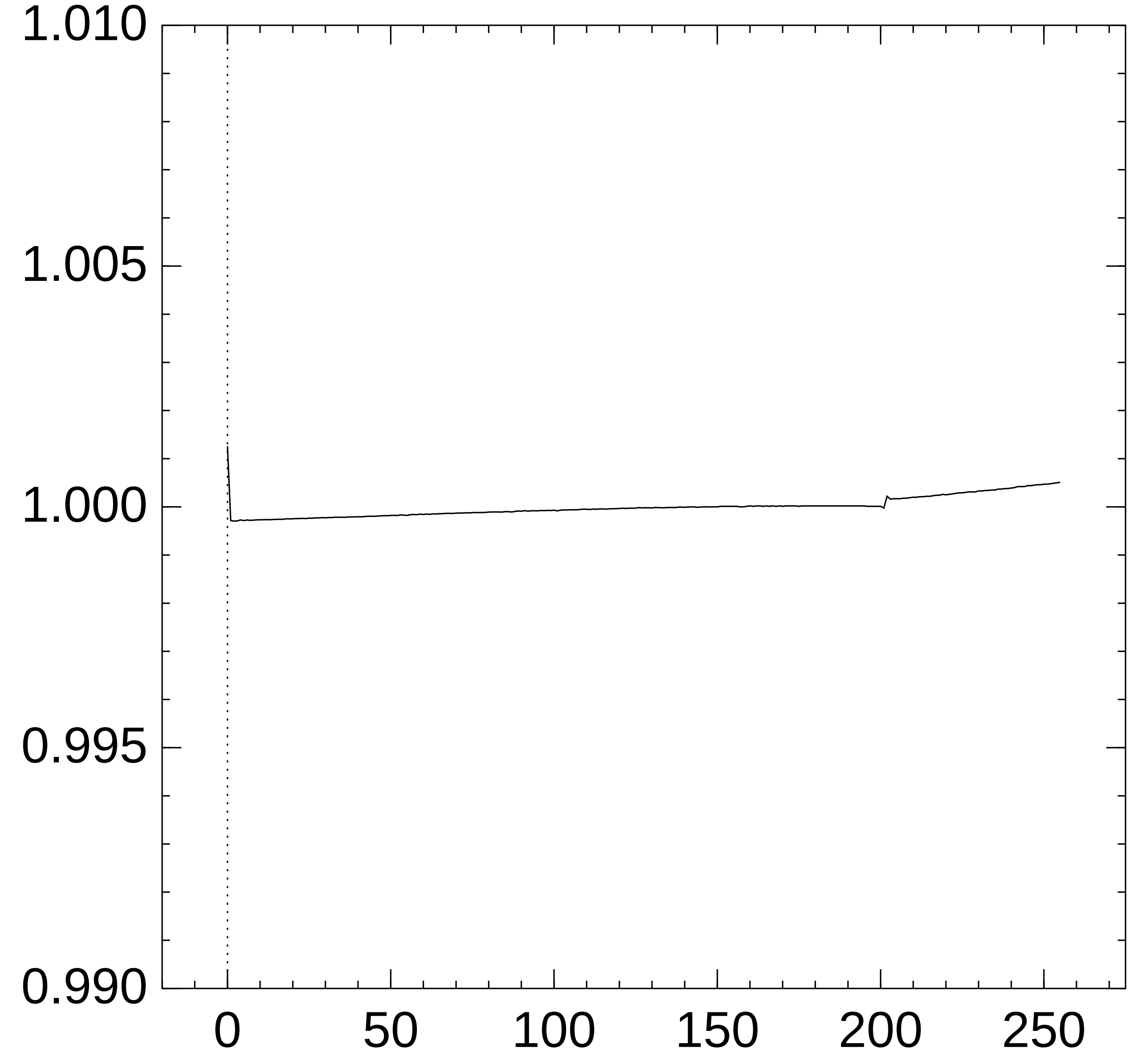
Graph of $256 \Pr[z_{201} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

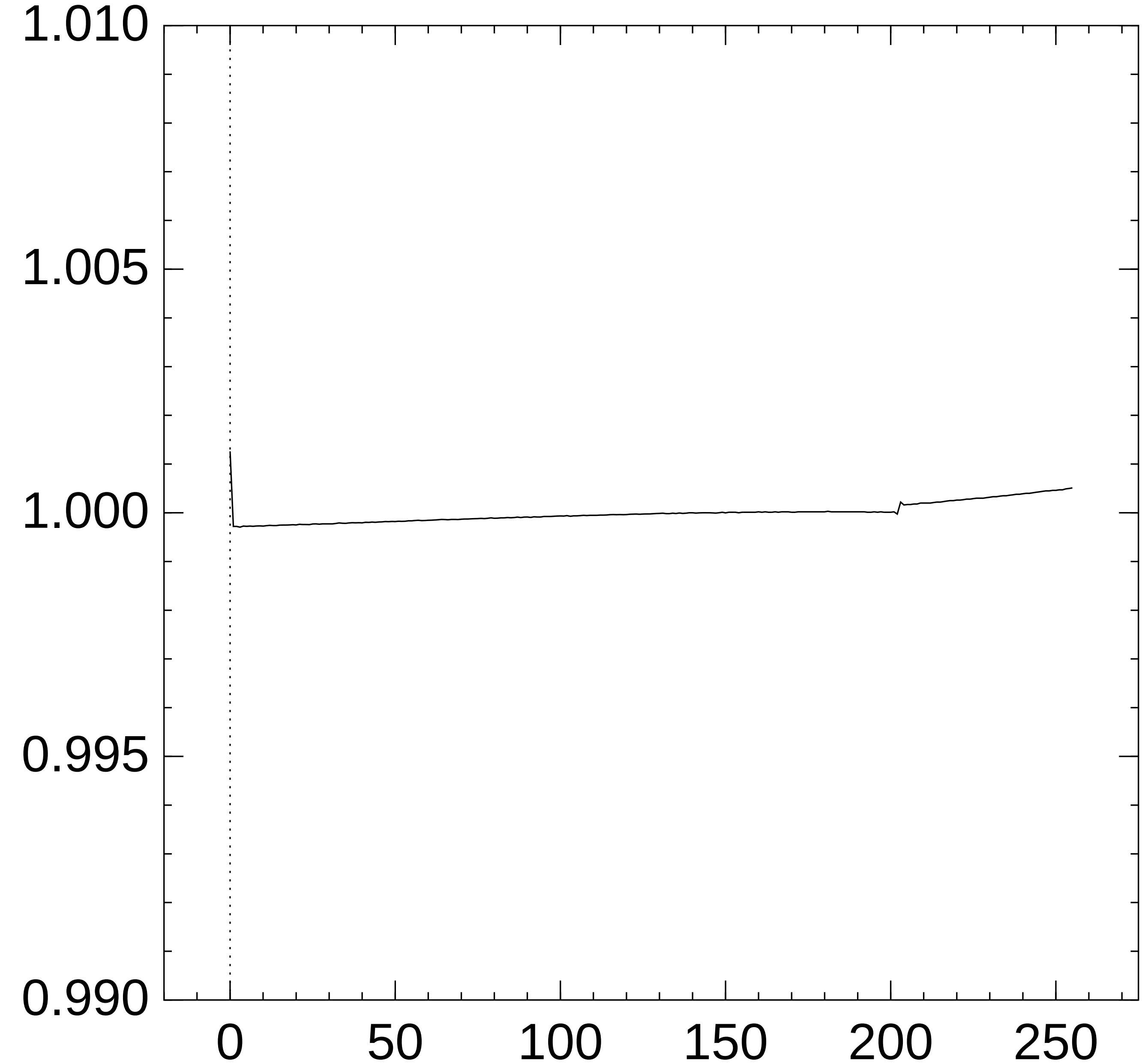
Graph of $256 \Pr[z_{202} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

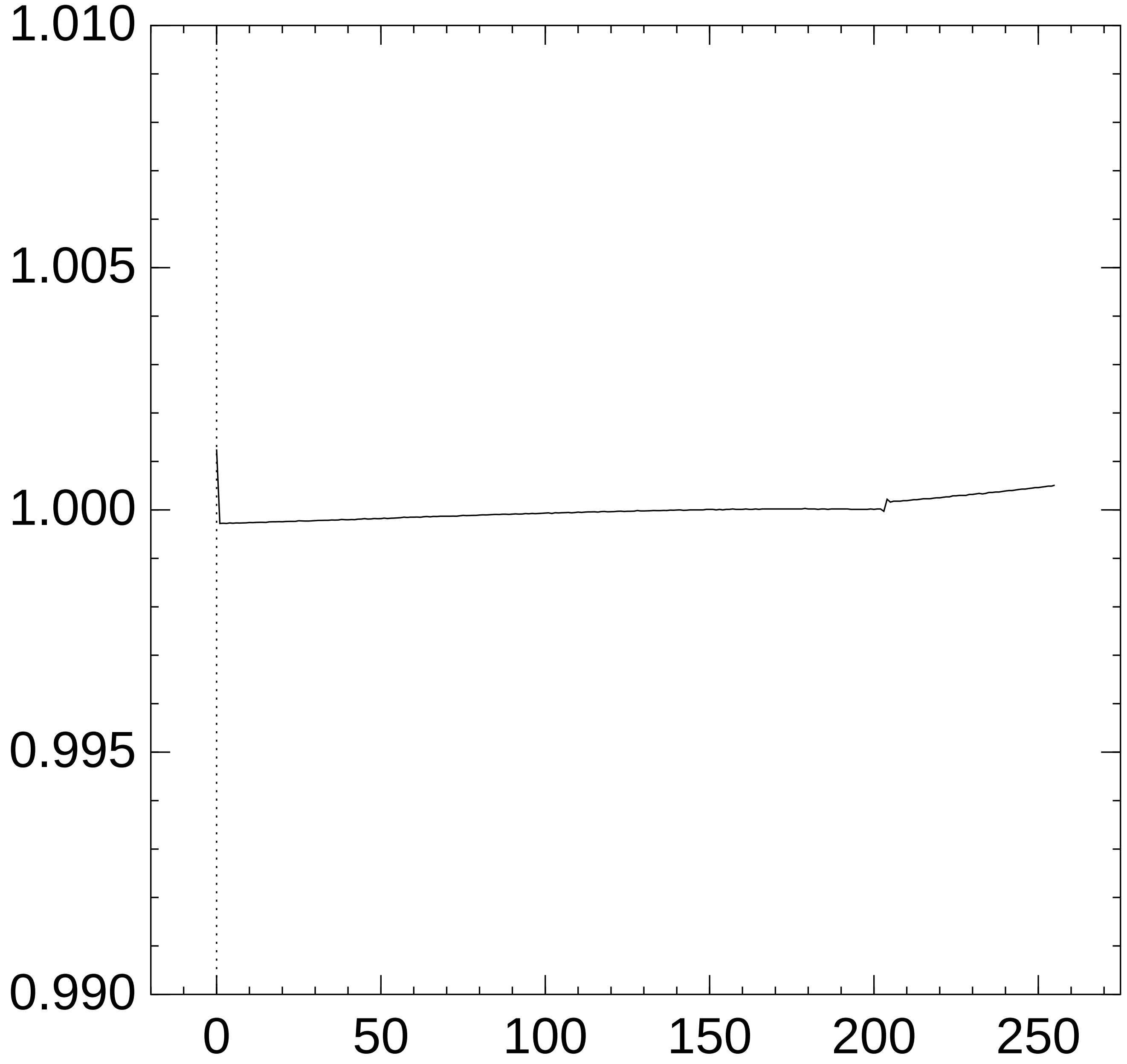
Graph of $256 \Pr[z_{203} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

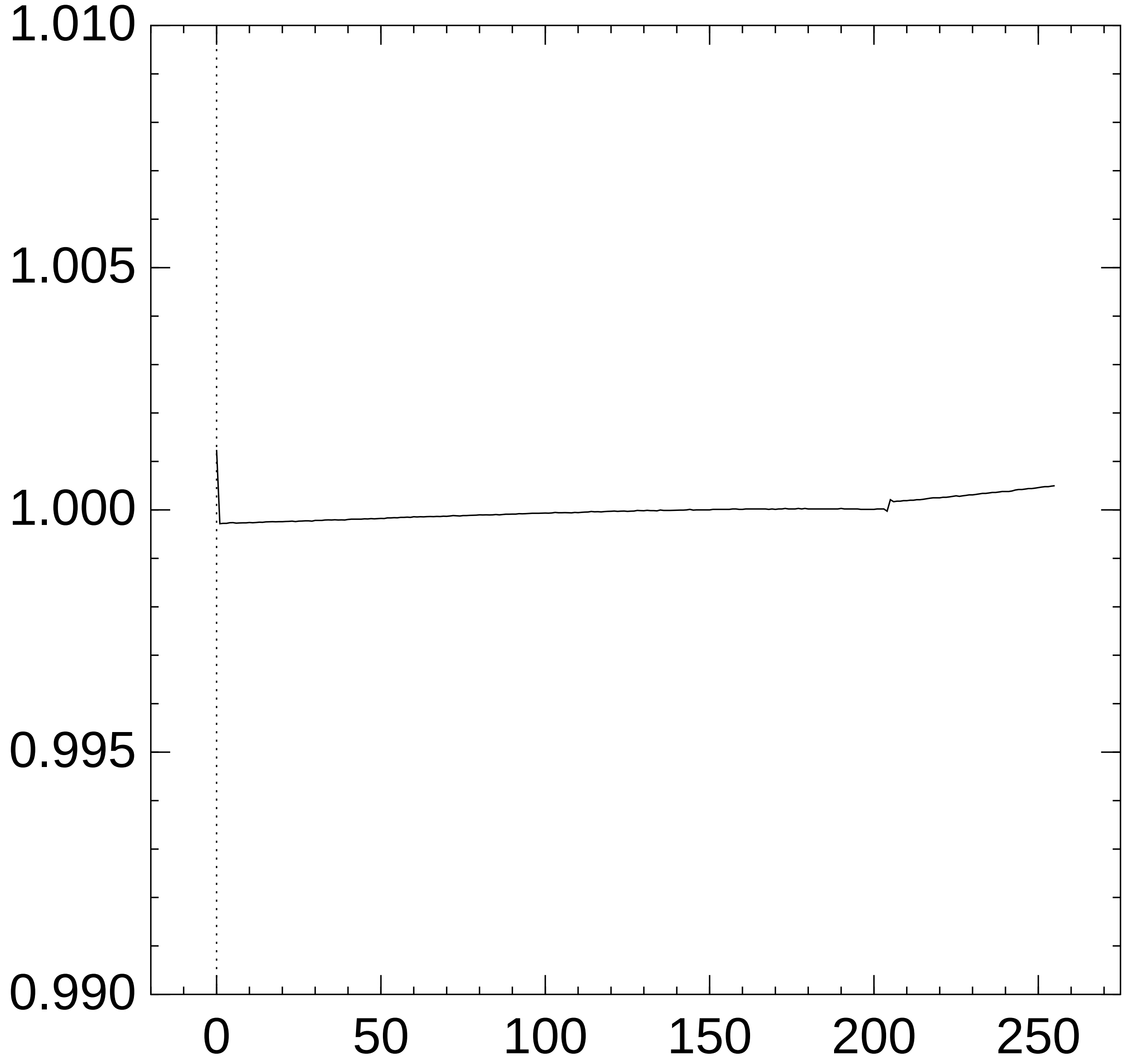
Graph of $256 \Pr[z_{204} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

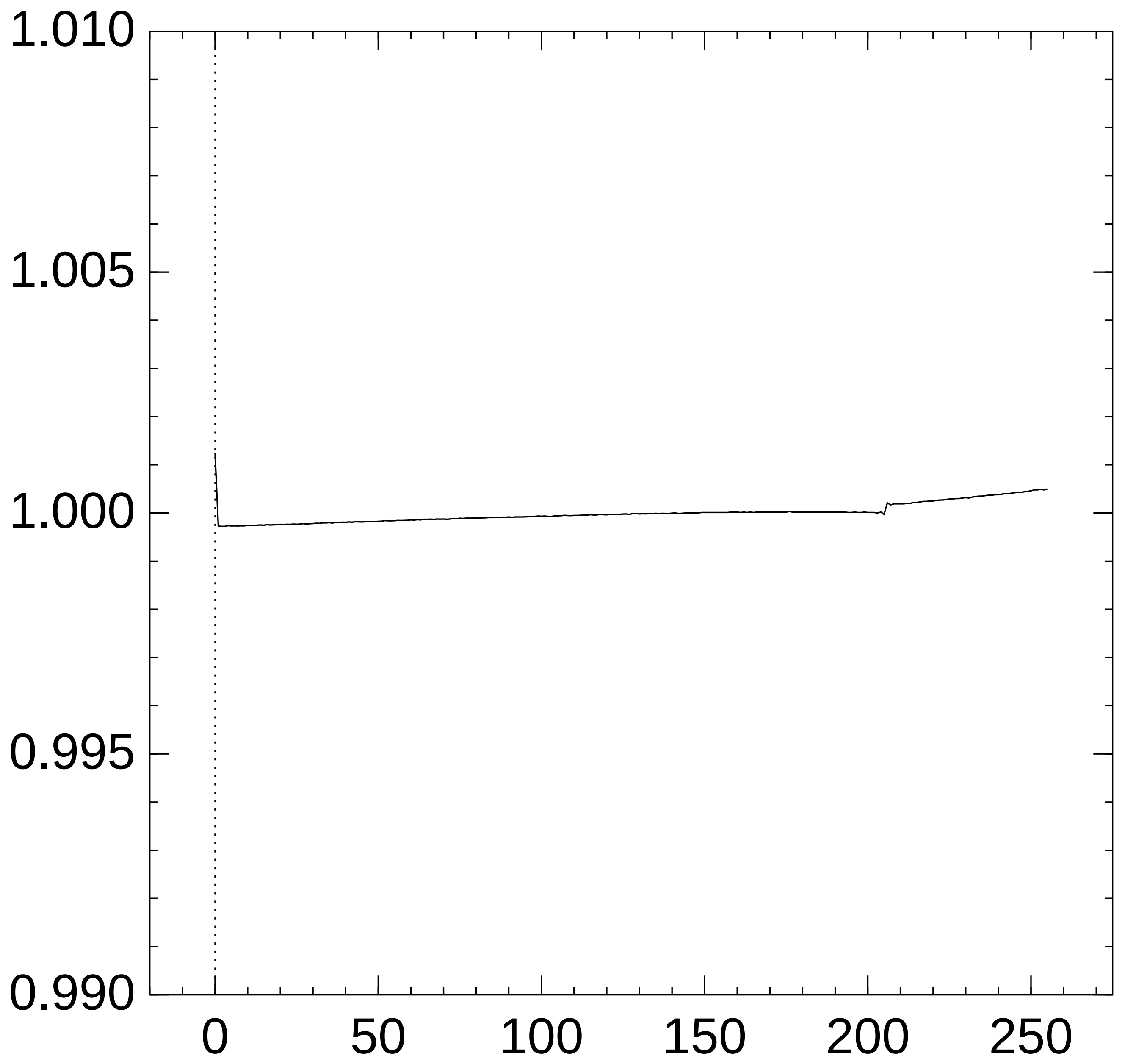
Graph of $256 \Pr[z_{205} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

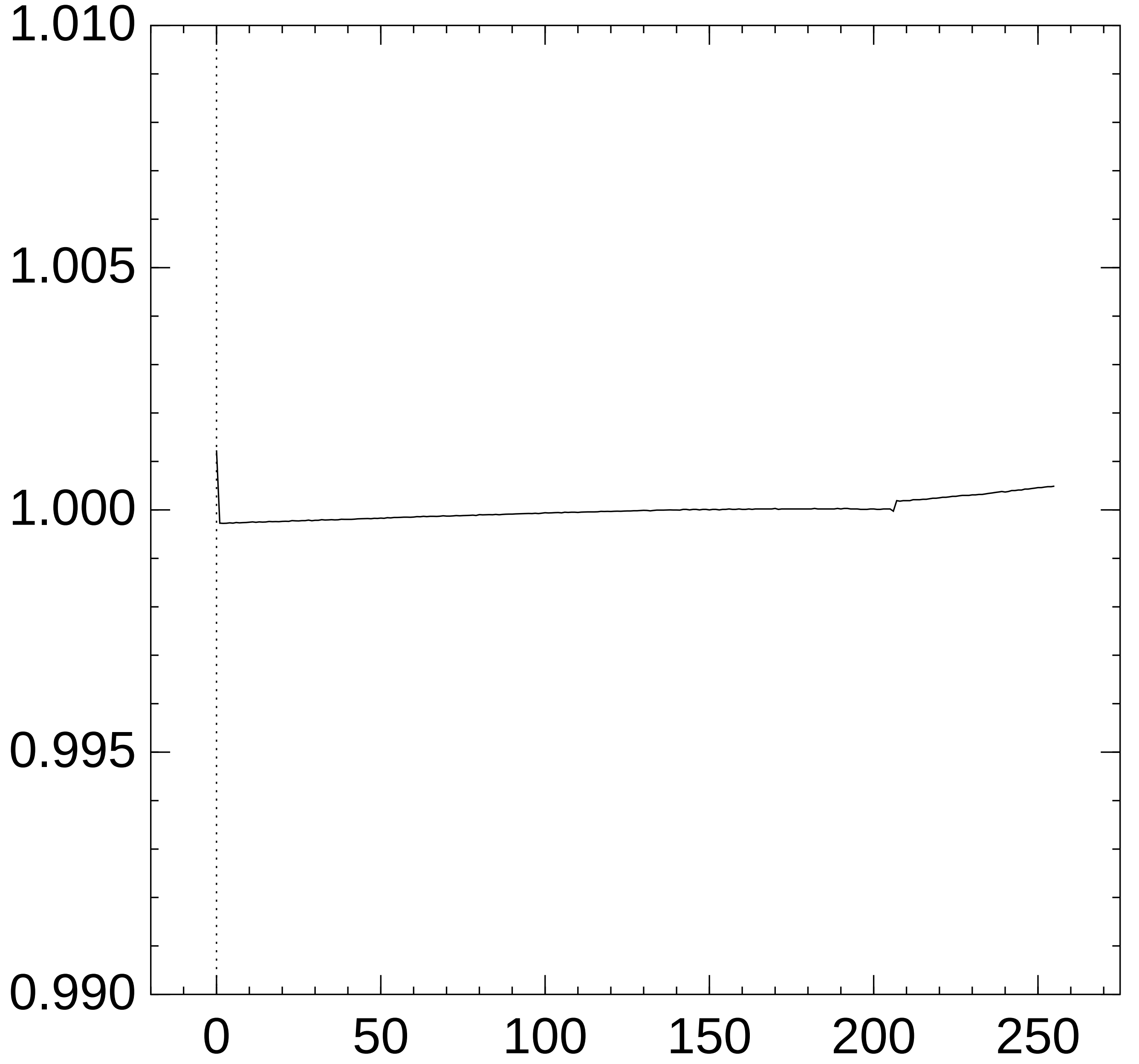
Graph of $256 \Pr[z_{206} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

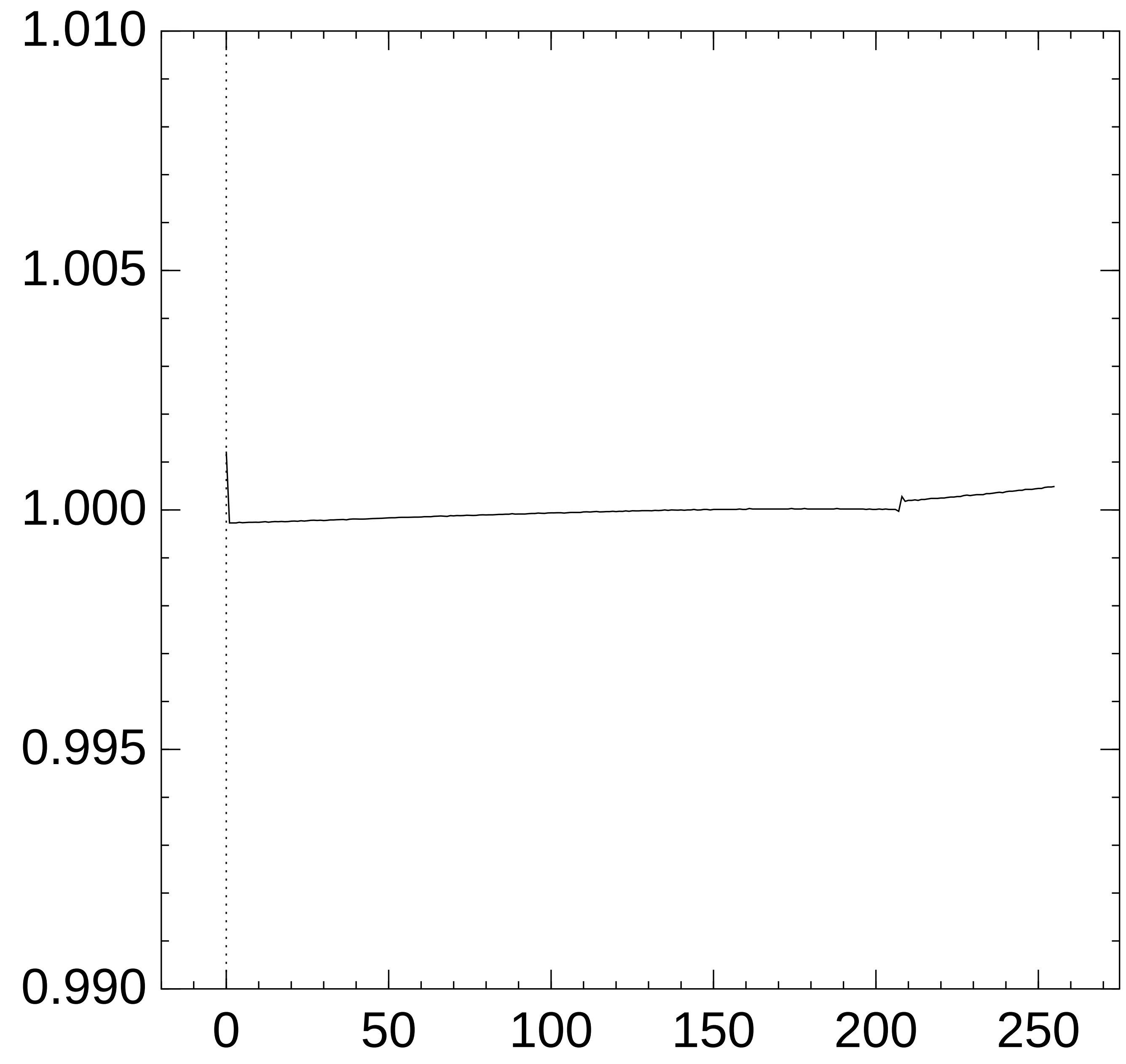
Graph of $256 \Pr[z_{207} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

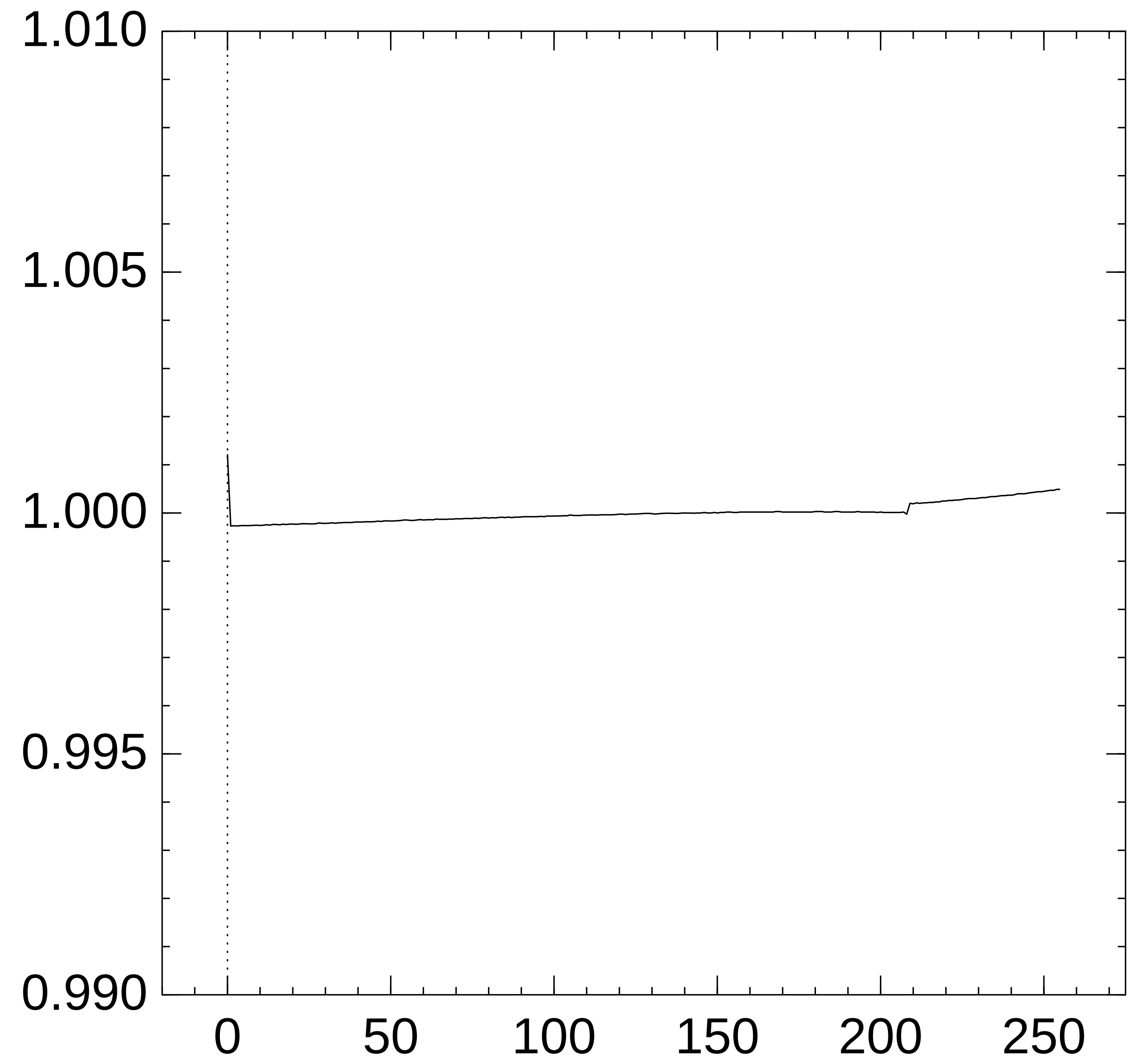
Graph of $256\,\Pr[z_{208} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

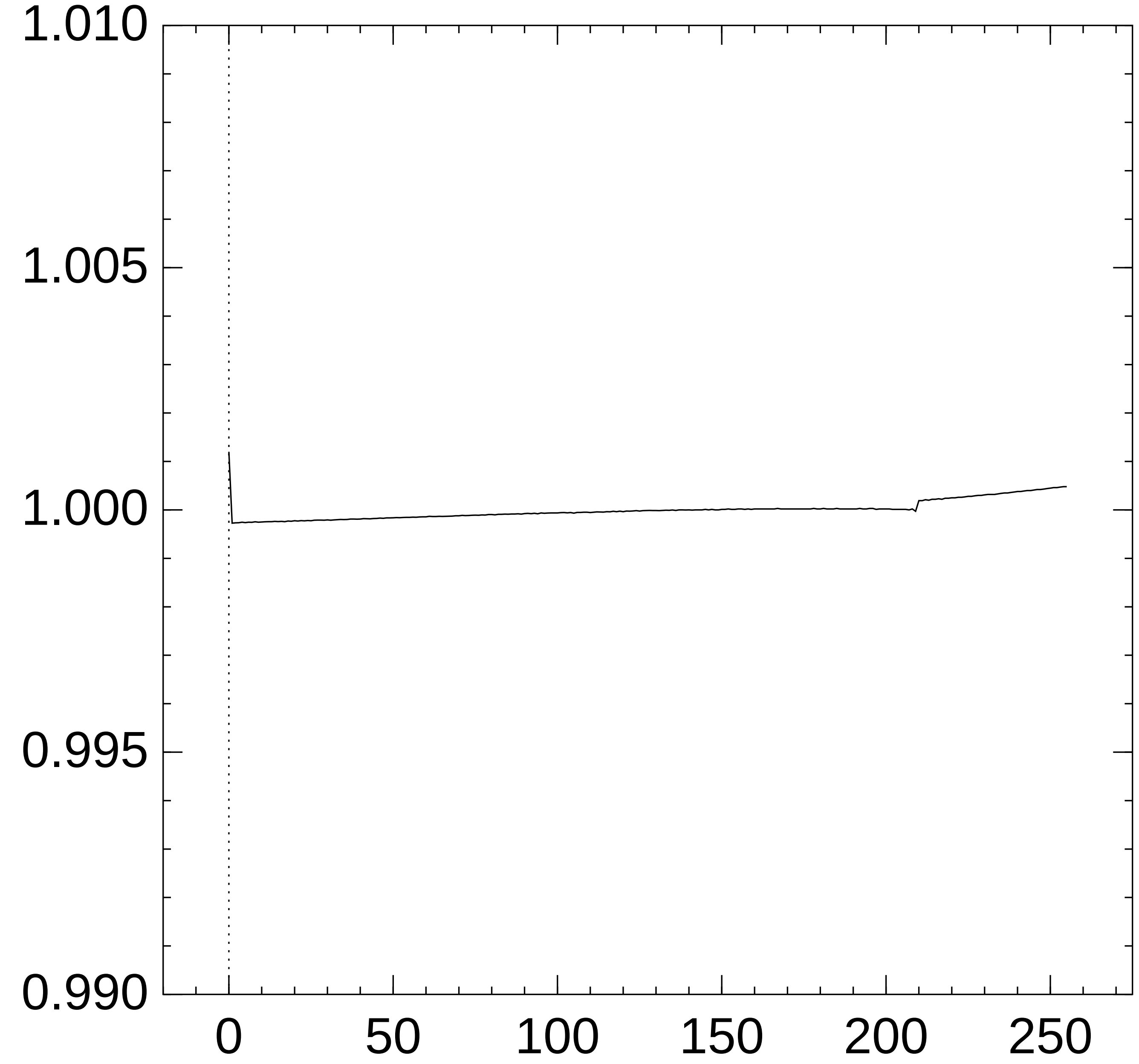
Graph of $256 \Pr[z_{209} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

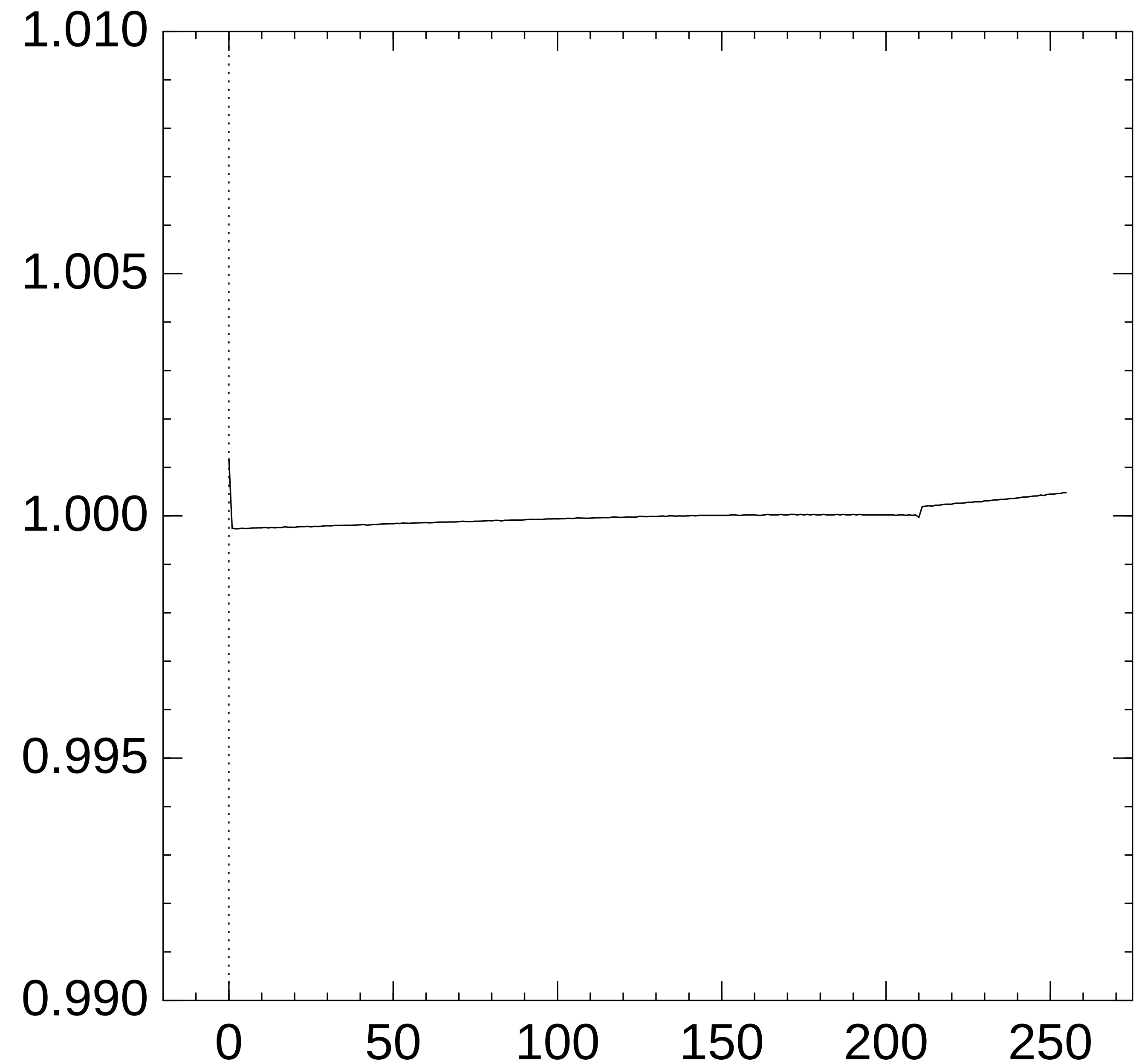
Graph of $256 \Pr[z_{210} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

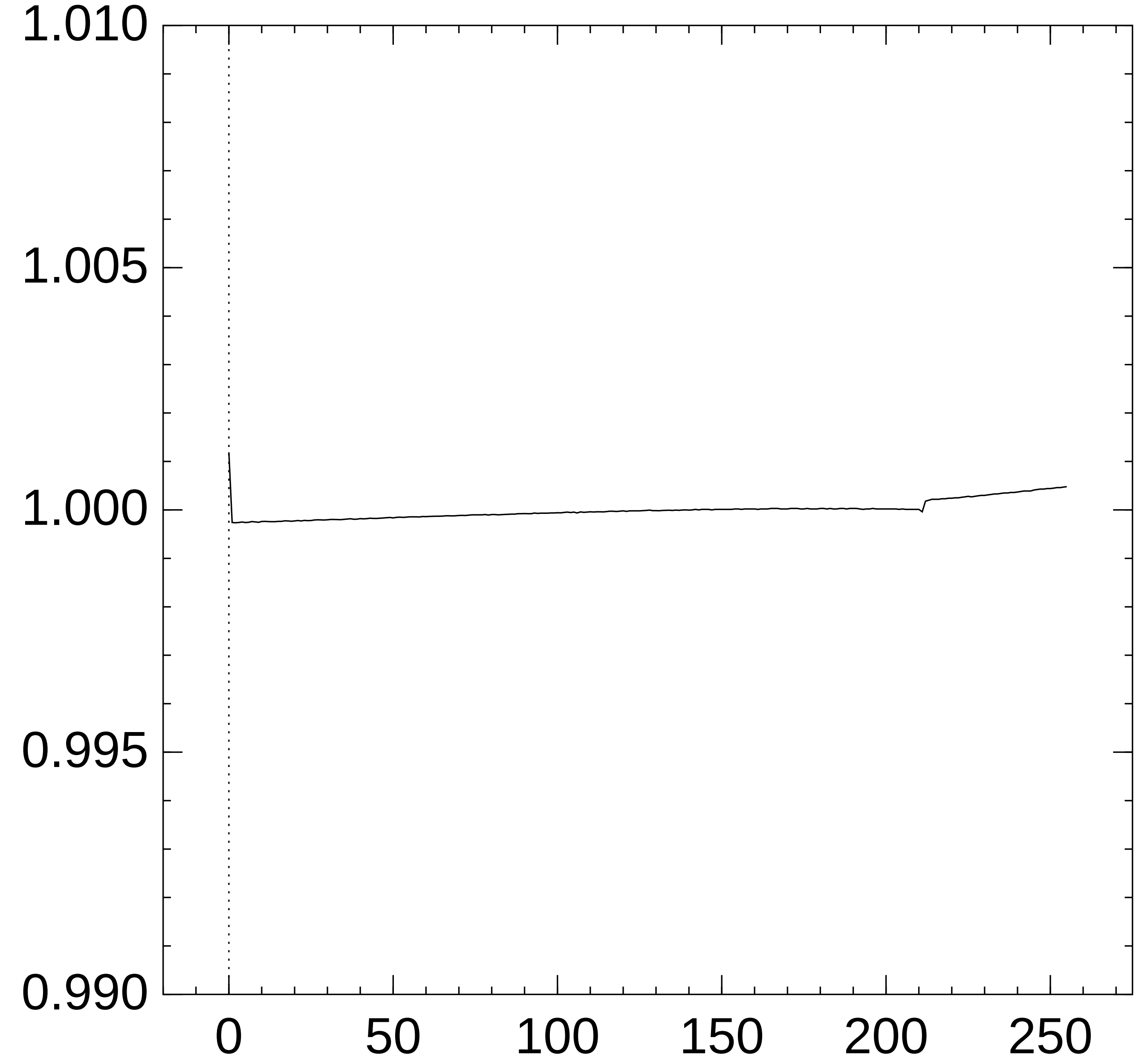
Graph of $256\,\Pr[z_{211} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

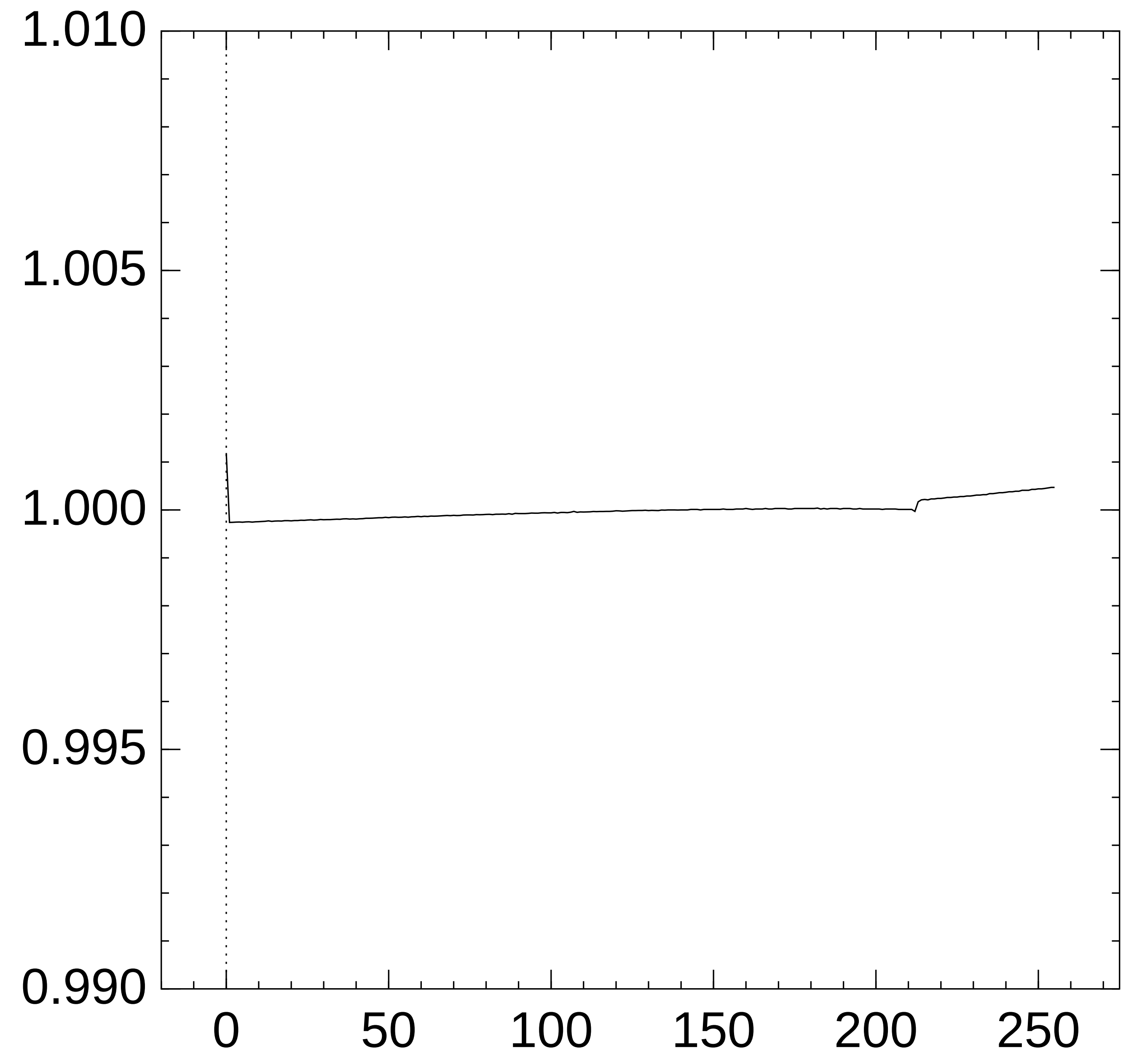
Graph of $256 \Pr[z_{212} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

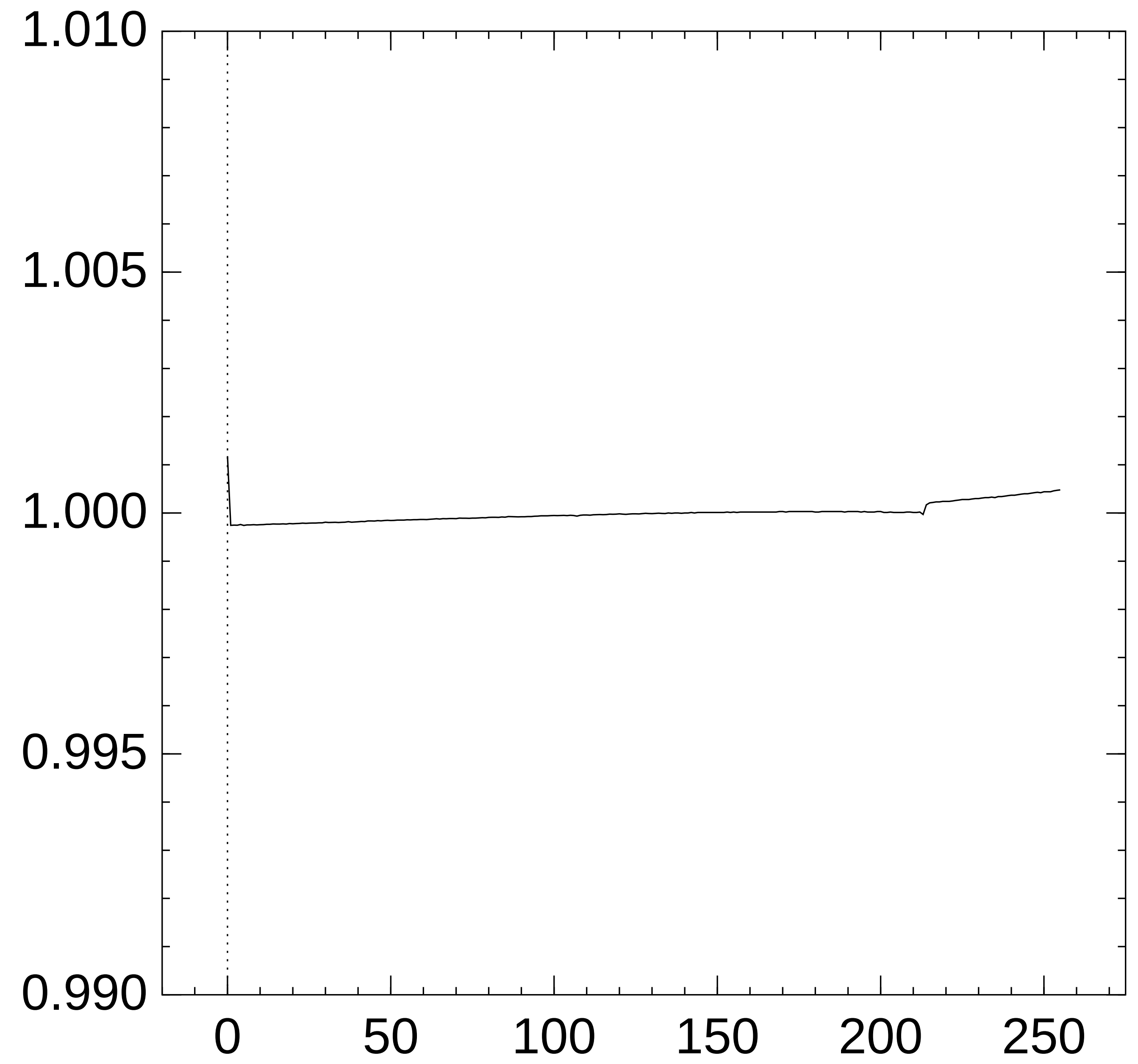
Graph of $256 \Pr[z_{213} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

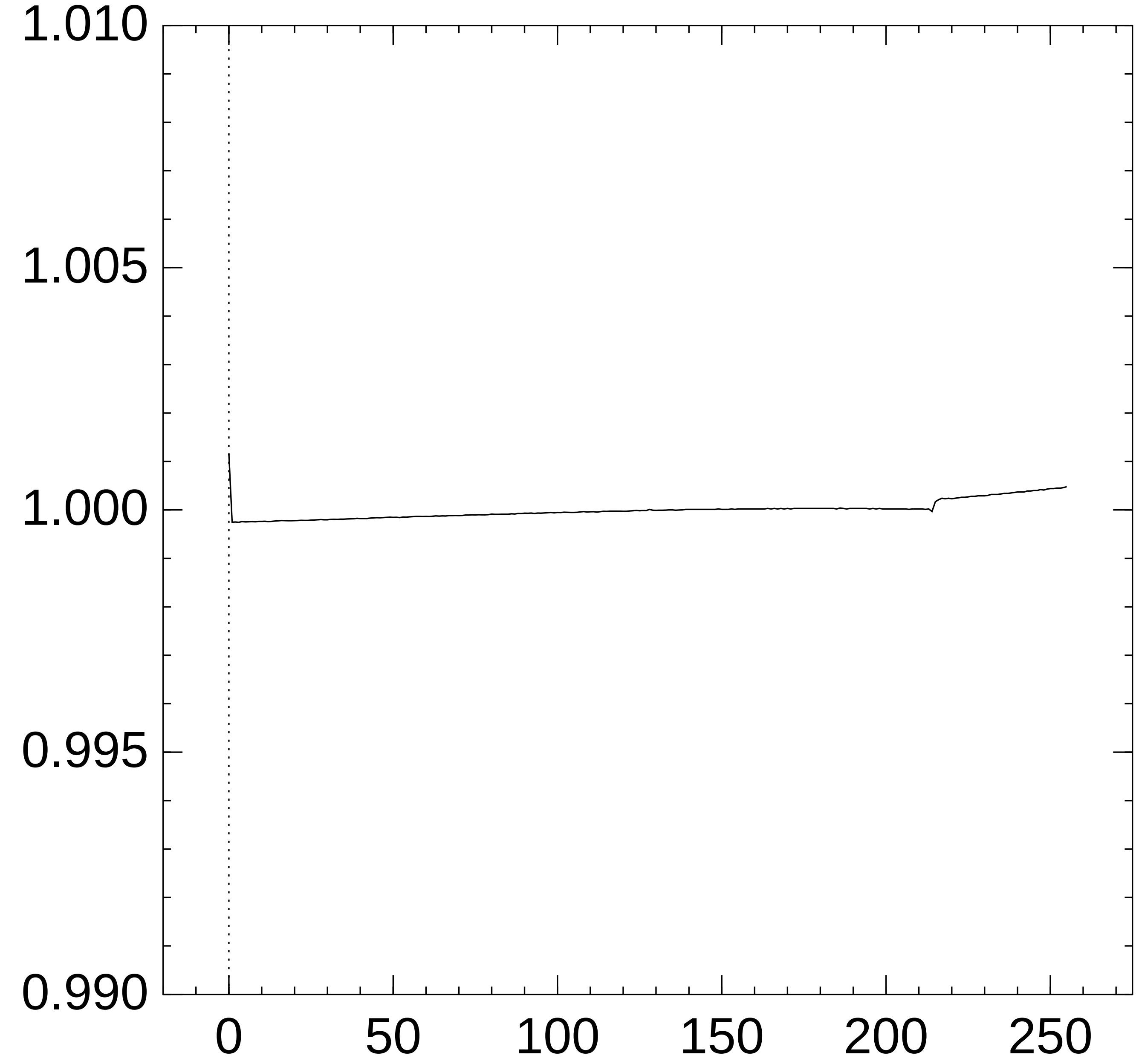
Graph of $256 \Pr[z_{214} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

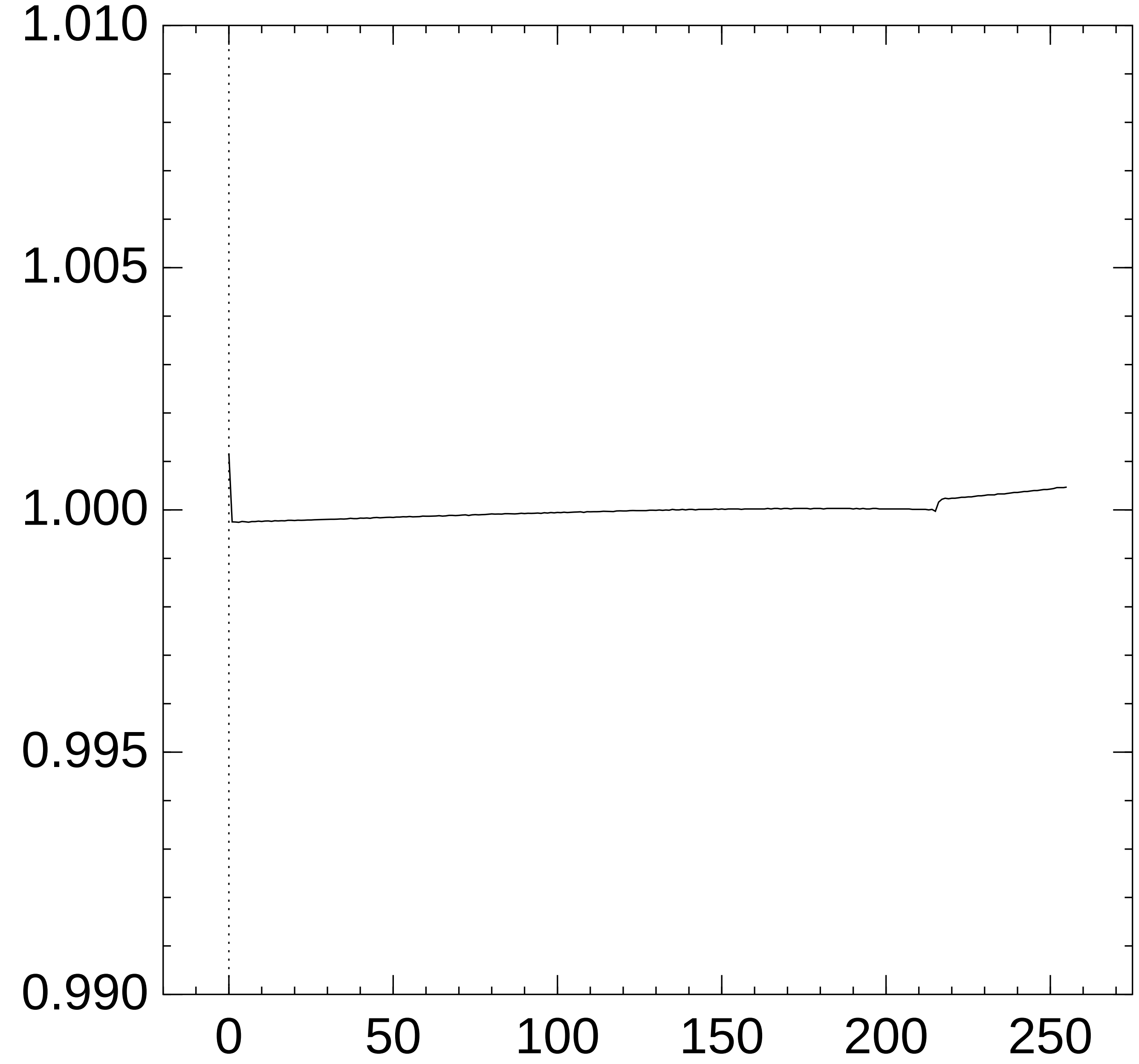
Graph of $256 \Pr[z_{215} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

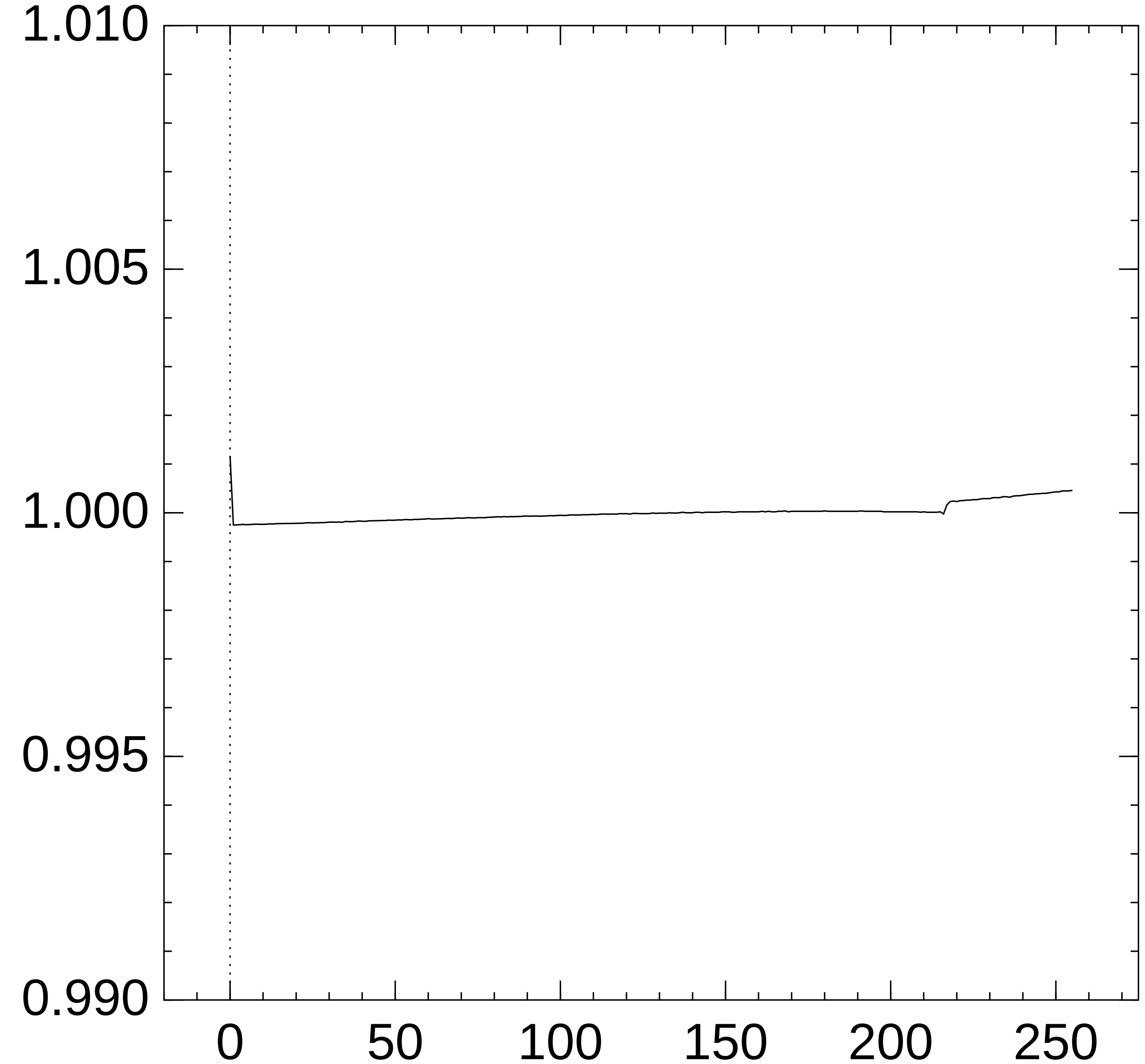
Graph of $256 \Pr[z_{216} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

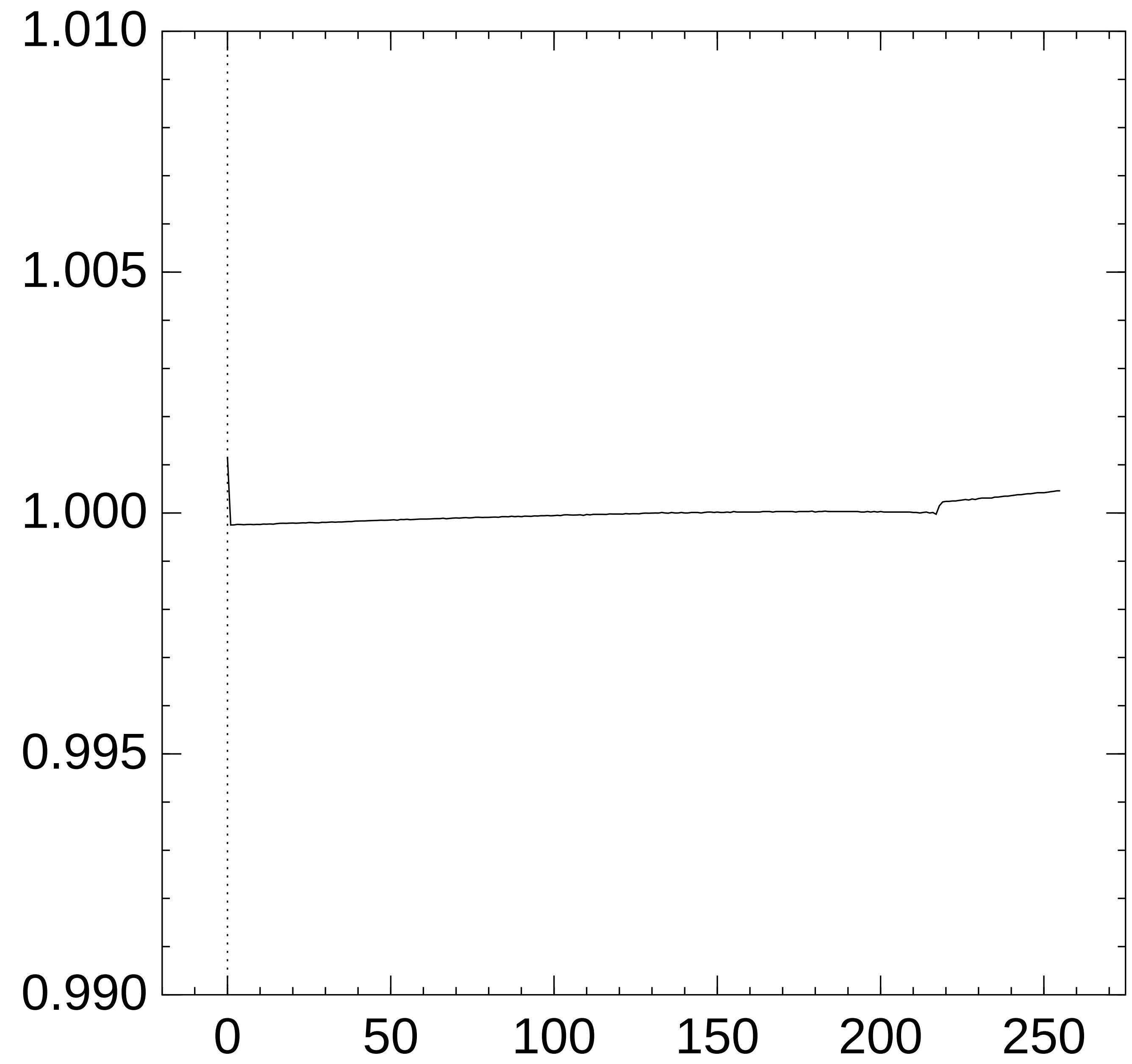
Graph of $256 \Pr[z_{217} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

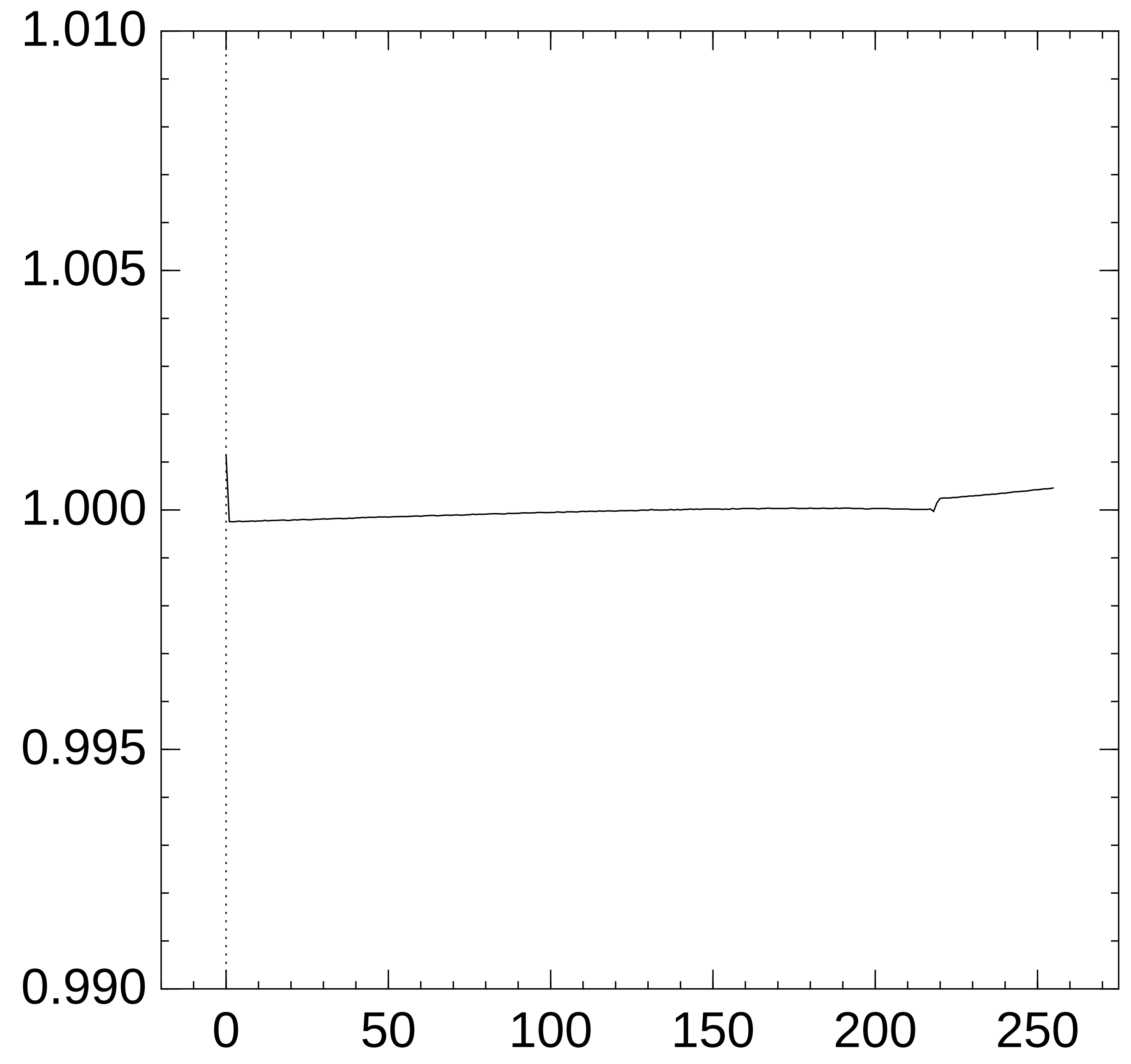
Graph of $256 \Pr[z_{218} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

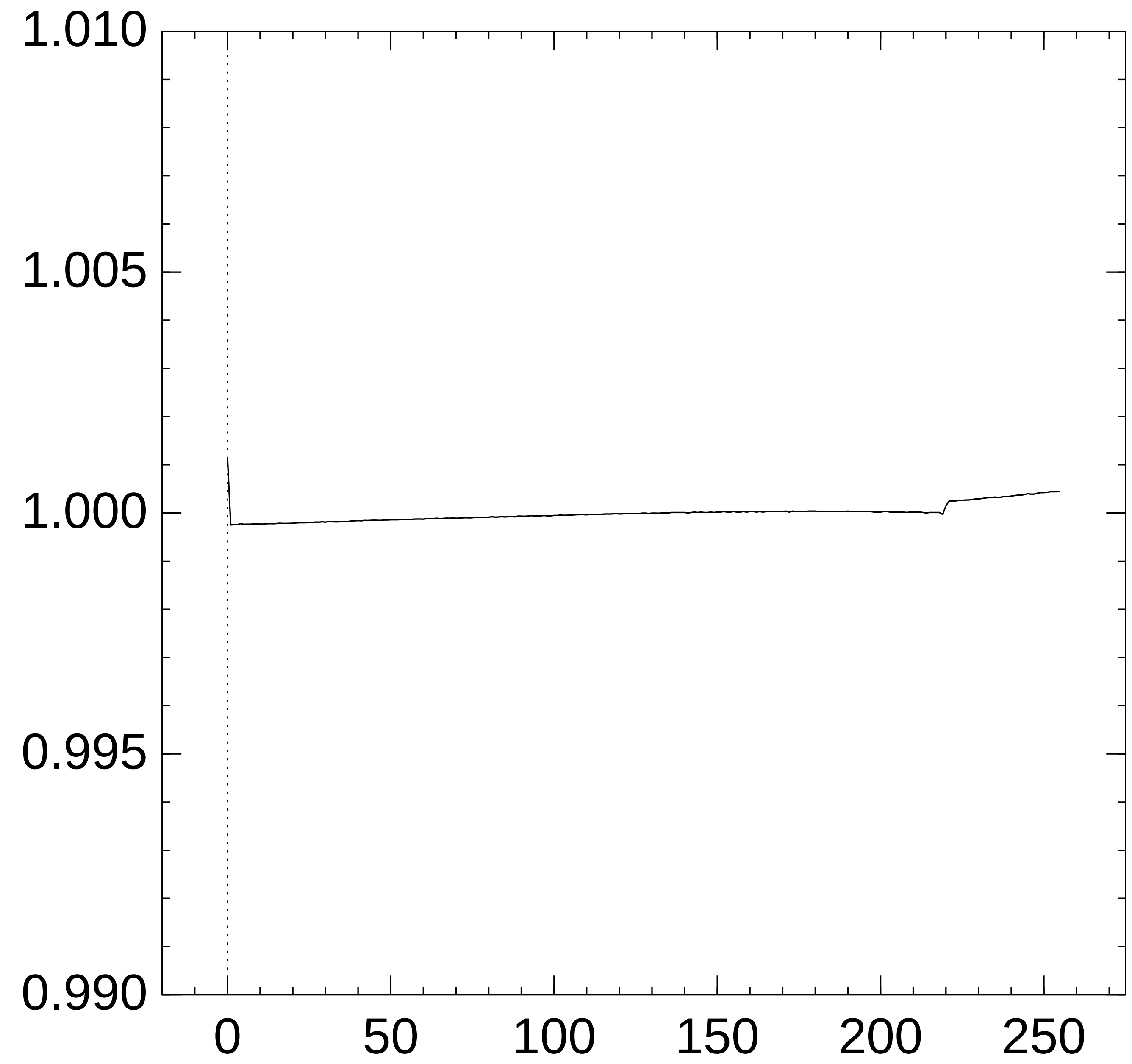
Graph of $256 \Pr[z_{219} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

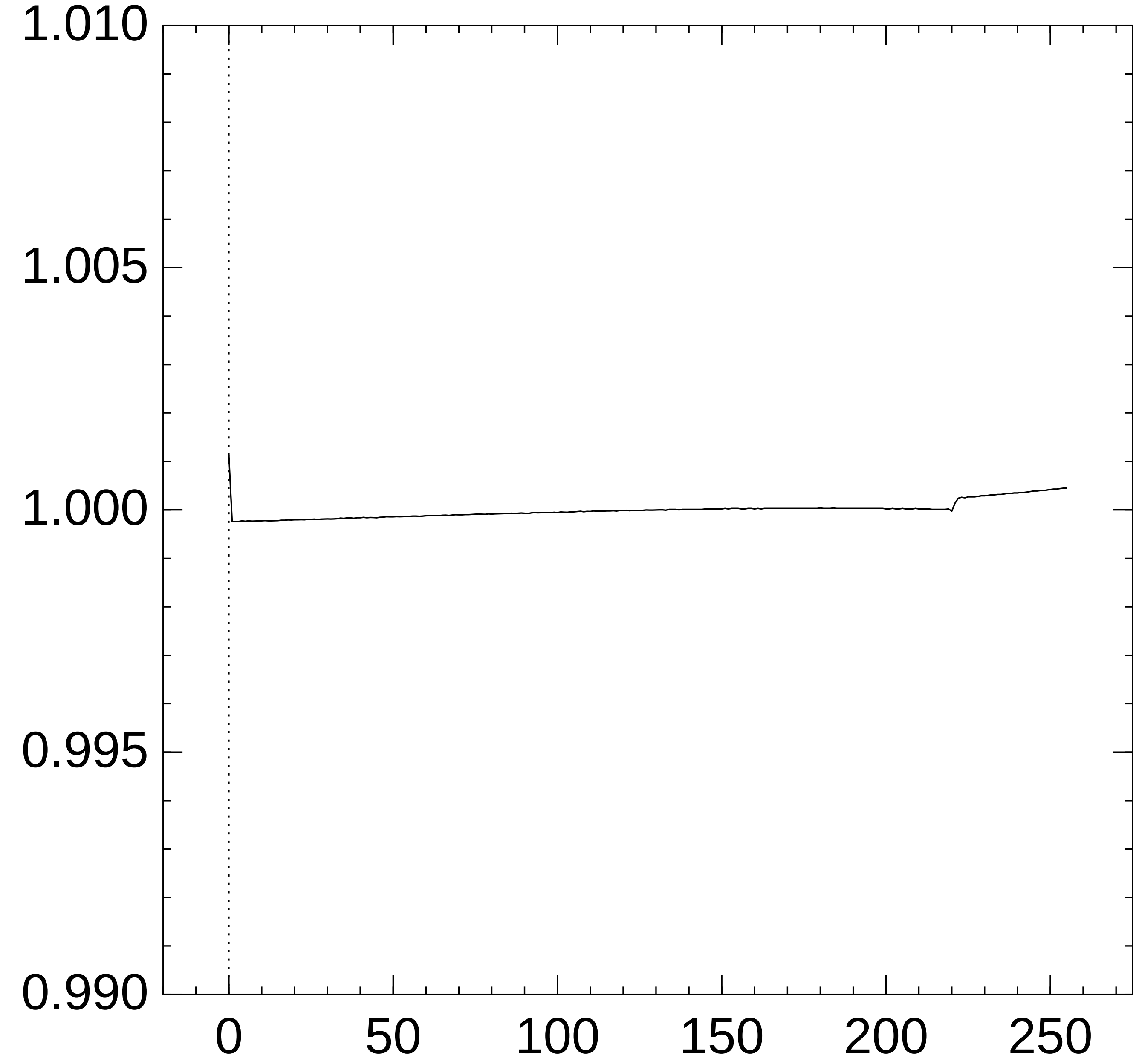
Graph of $256 \Pr[z_{220} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

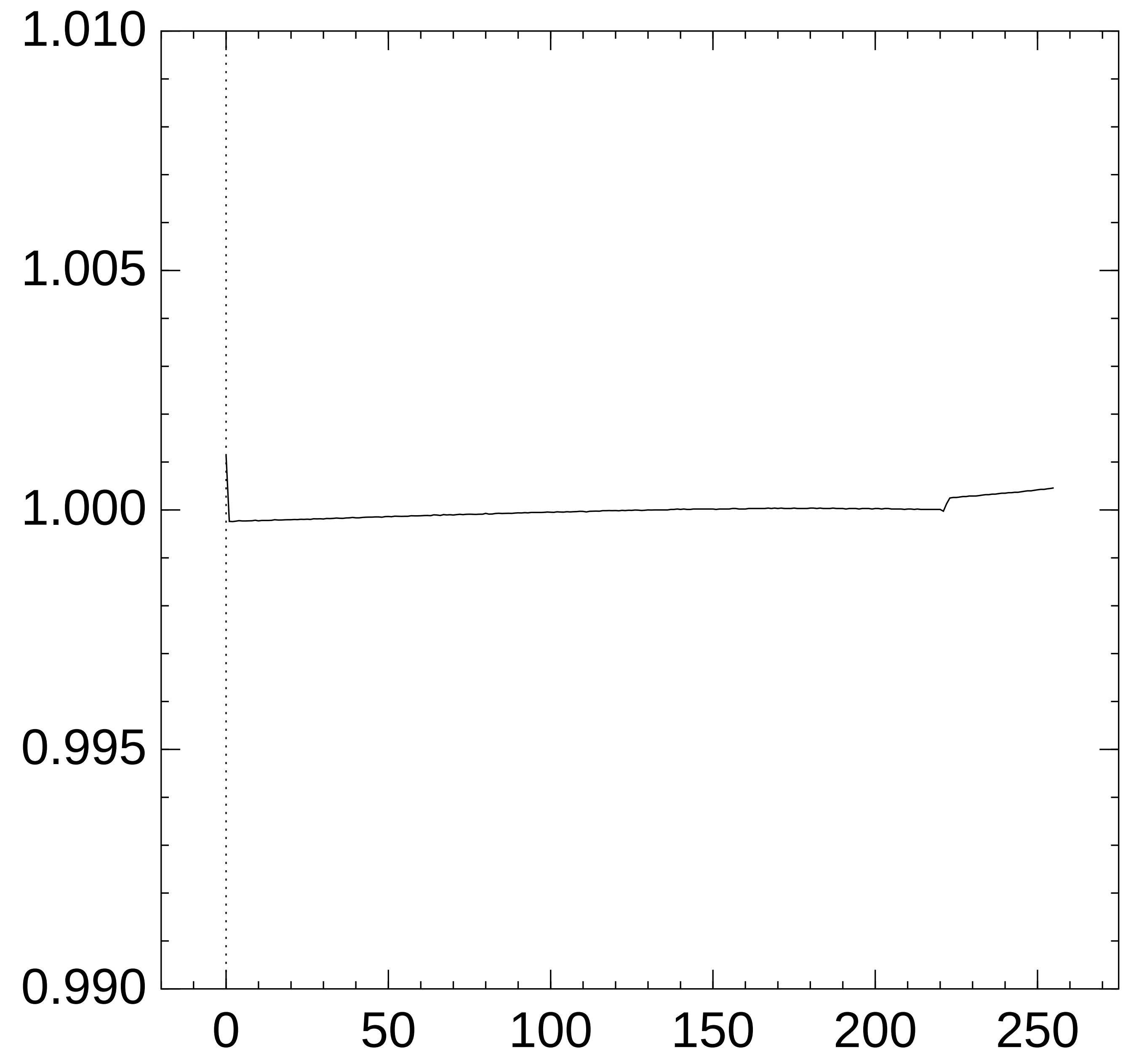
Graph of $256\Pr[z_{221} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

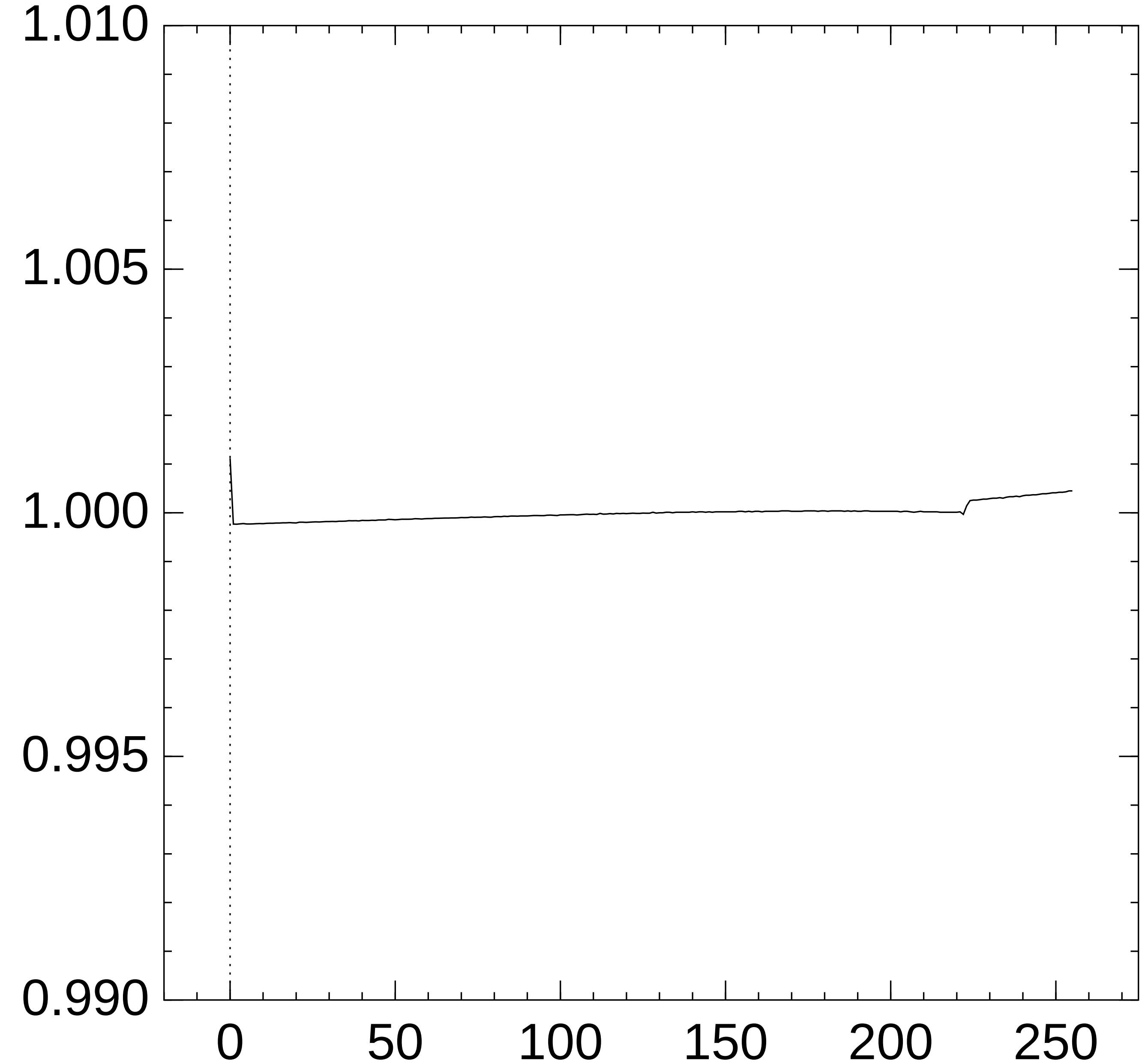
Graph of $256 \Pr[z_{222} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

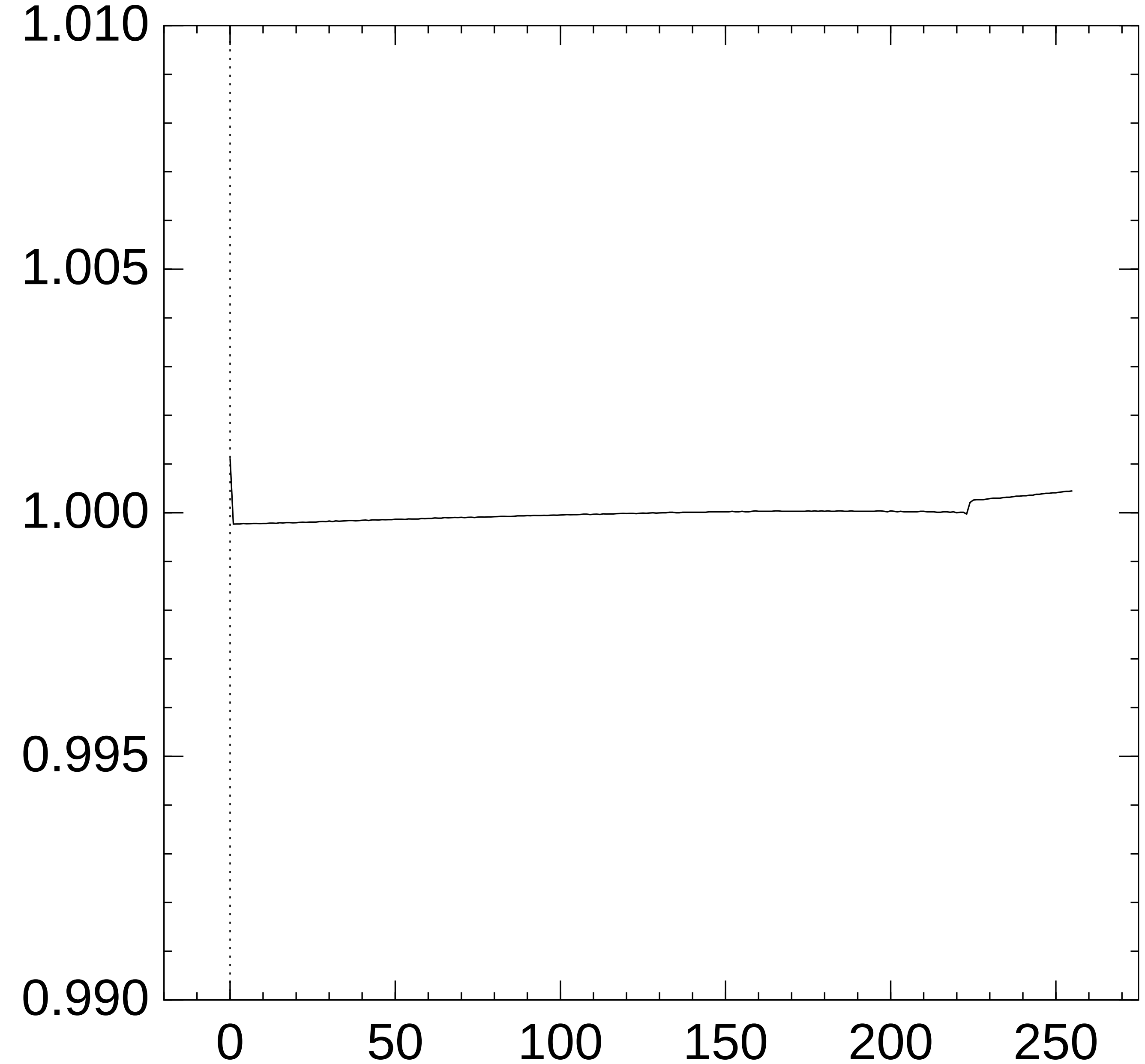
Graph of $256 \Pr[z_{223} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

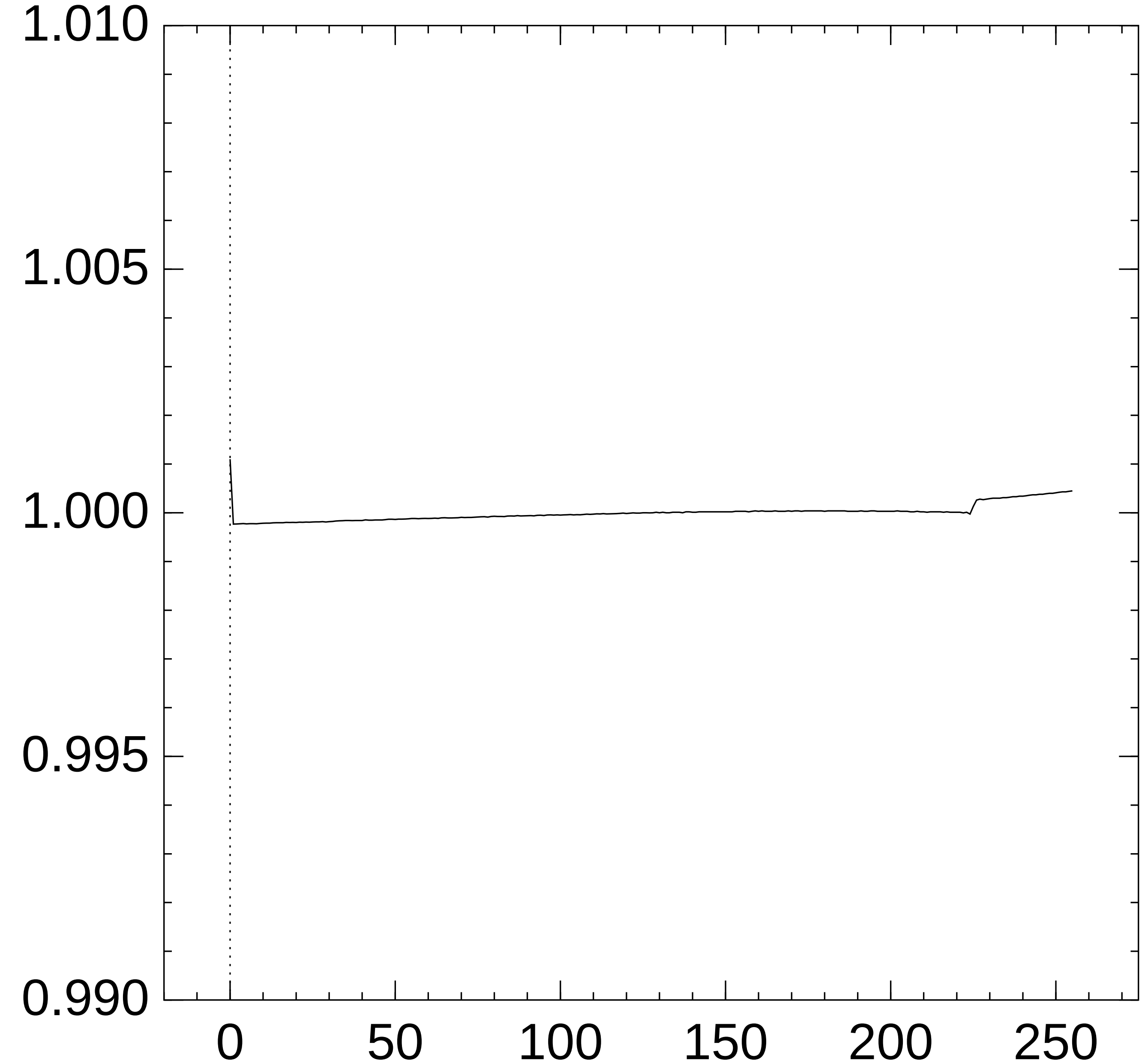
Graph of $256 \Pr[z_{224} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

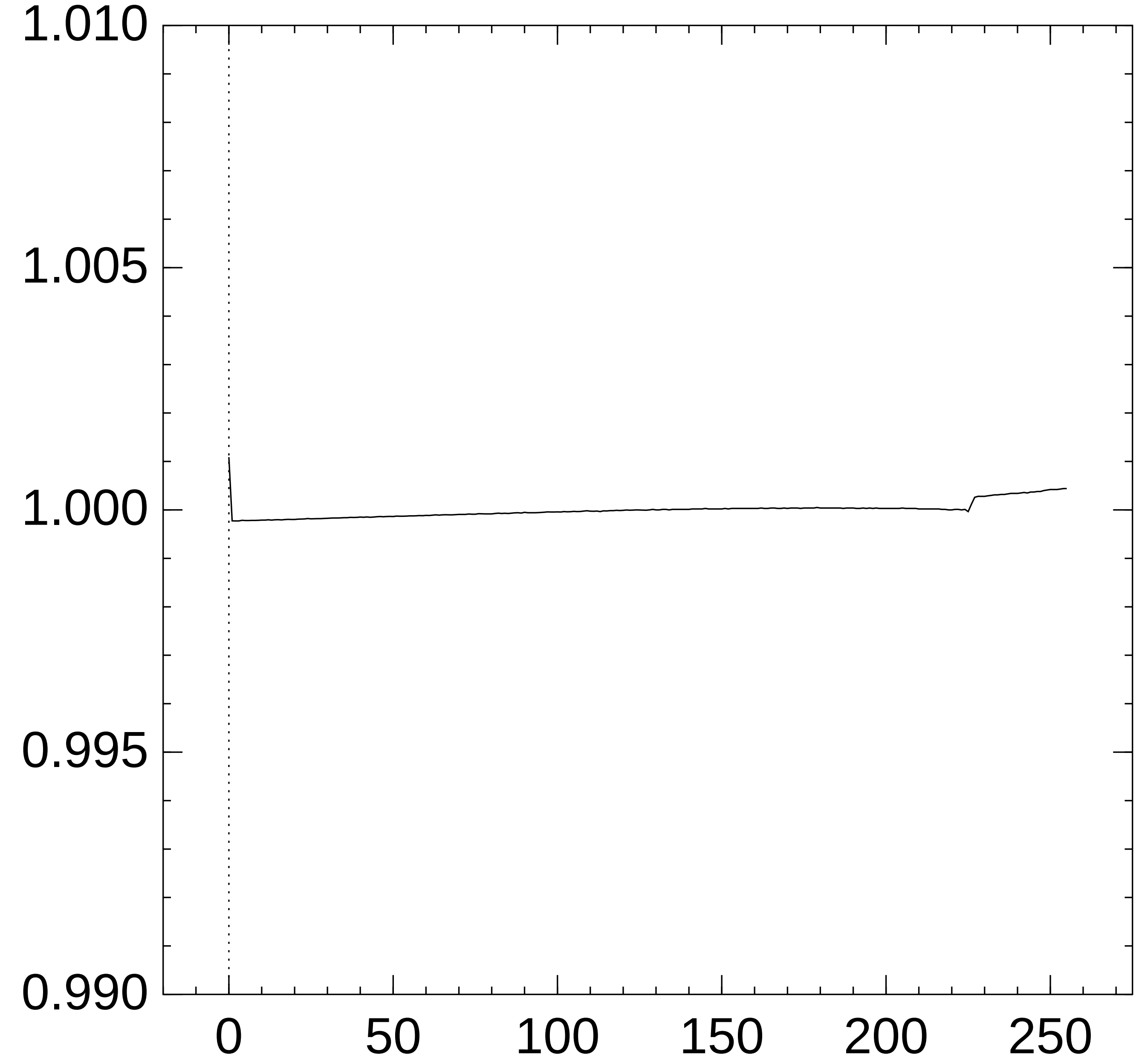
Graph of $256 \Pr[z_{225} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

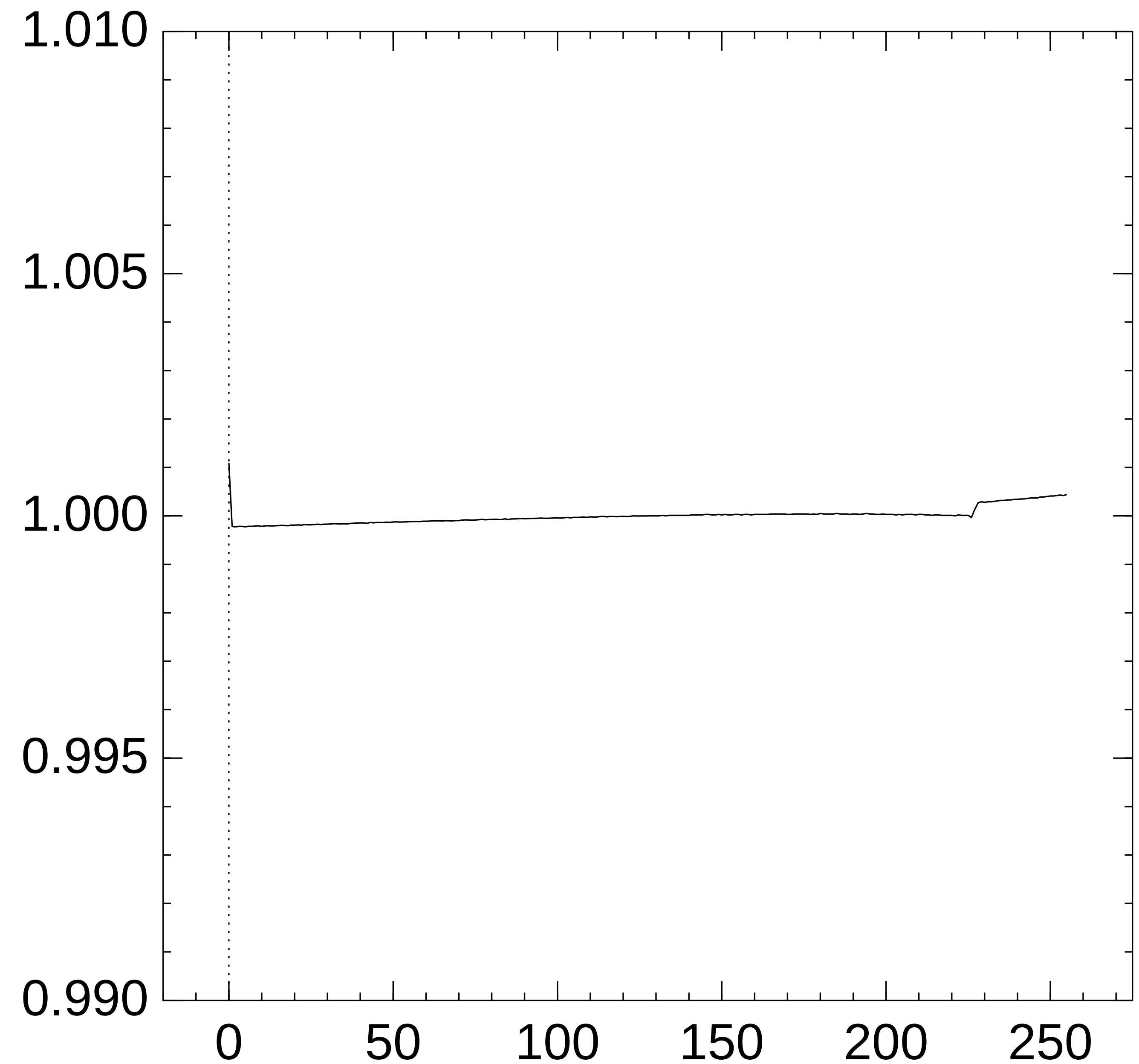
Graph of $256 \Pr[z_{226} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

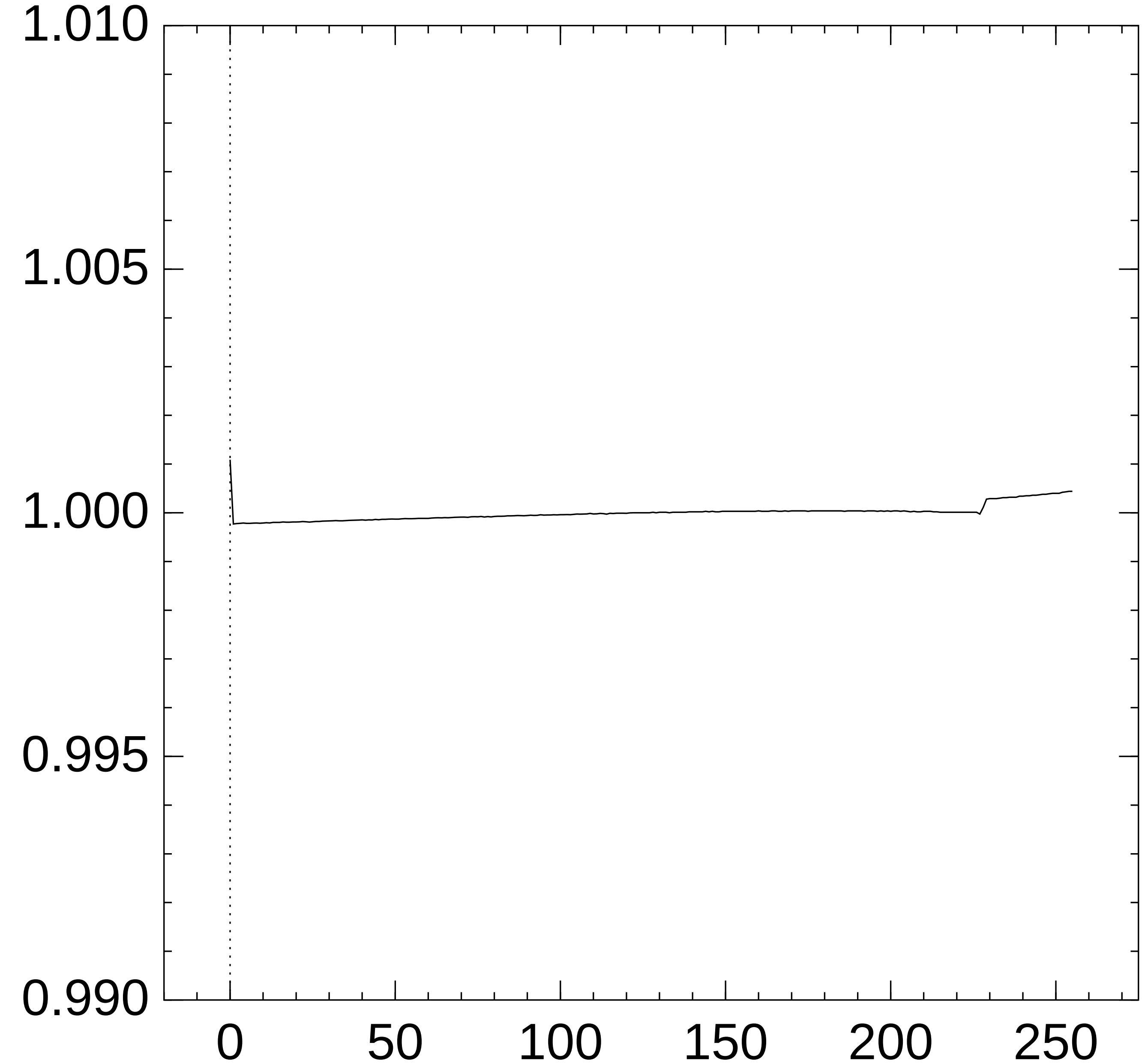
Graph of $256 \Pr[z_{227} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

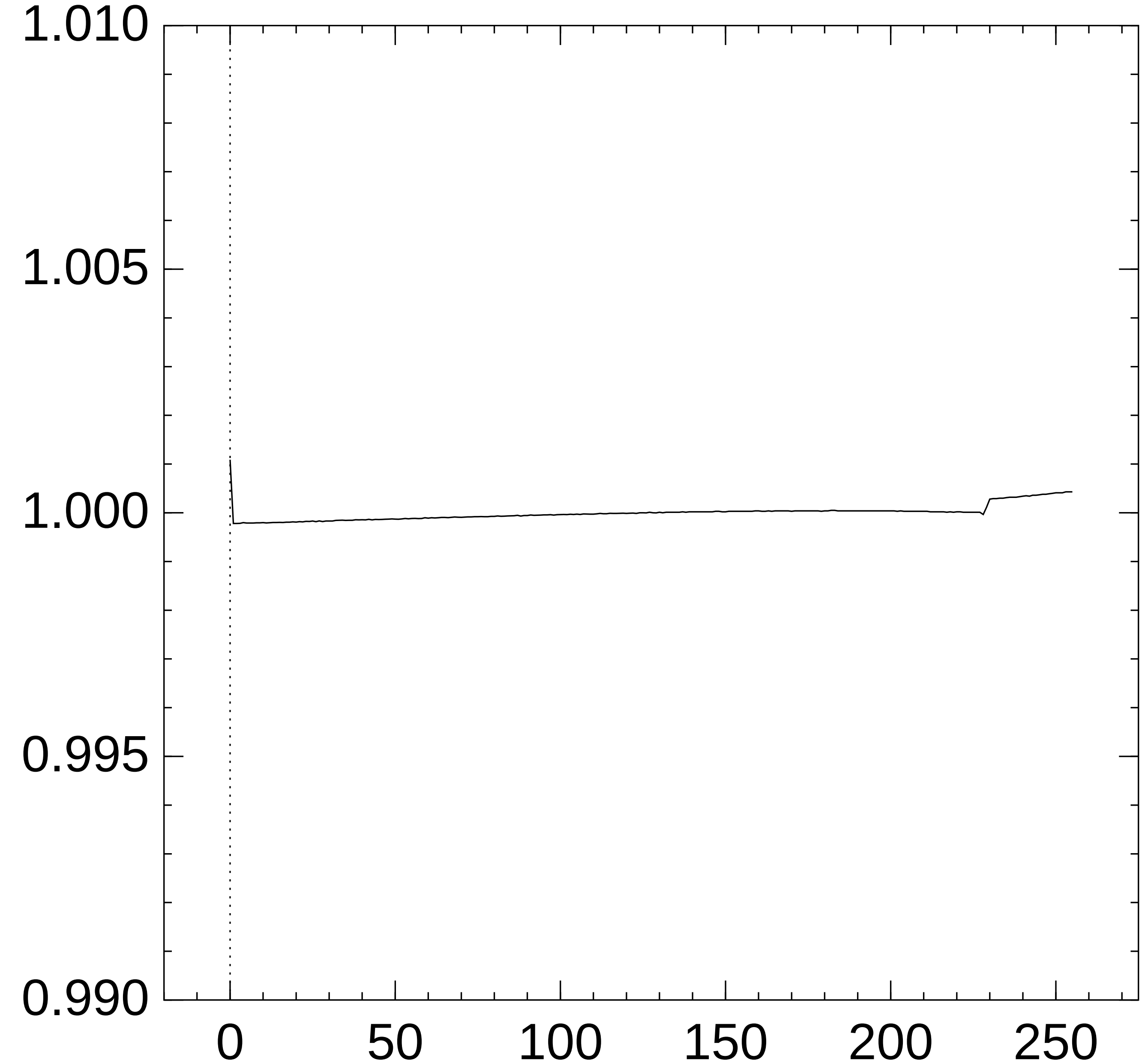
Graph of $256 \Pr[z_{228} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

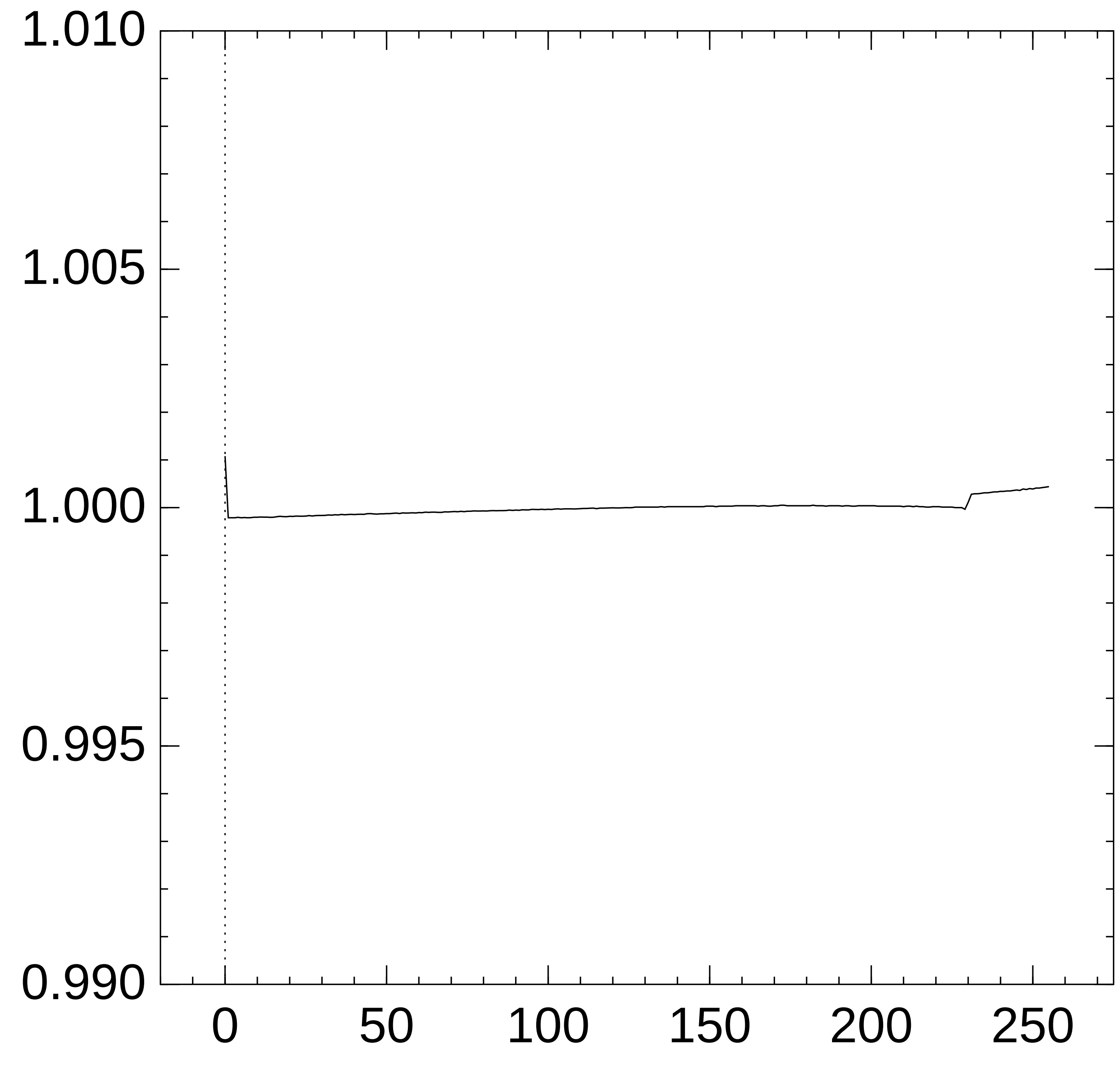
Graph of $256 \Pr[z_{229} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

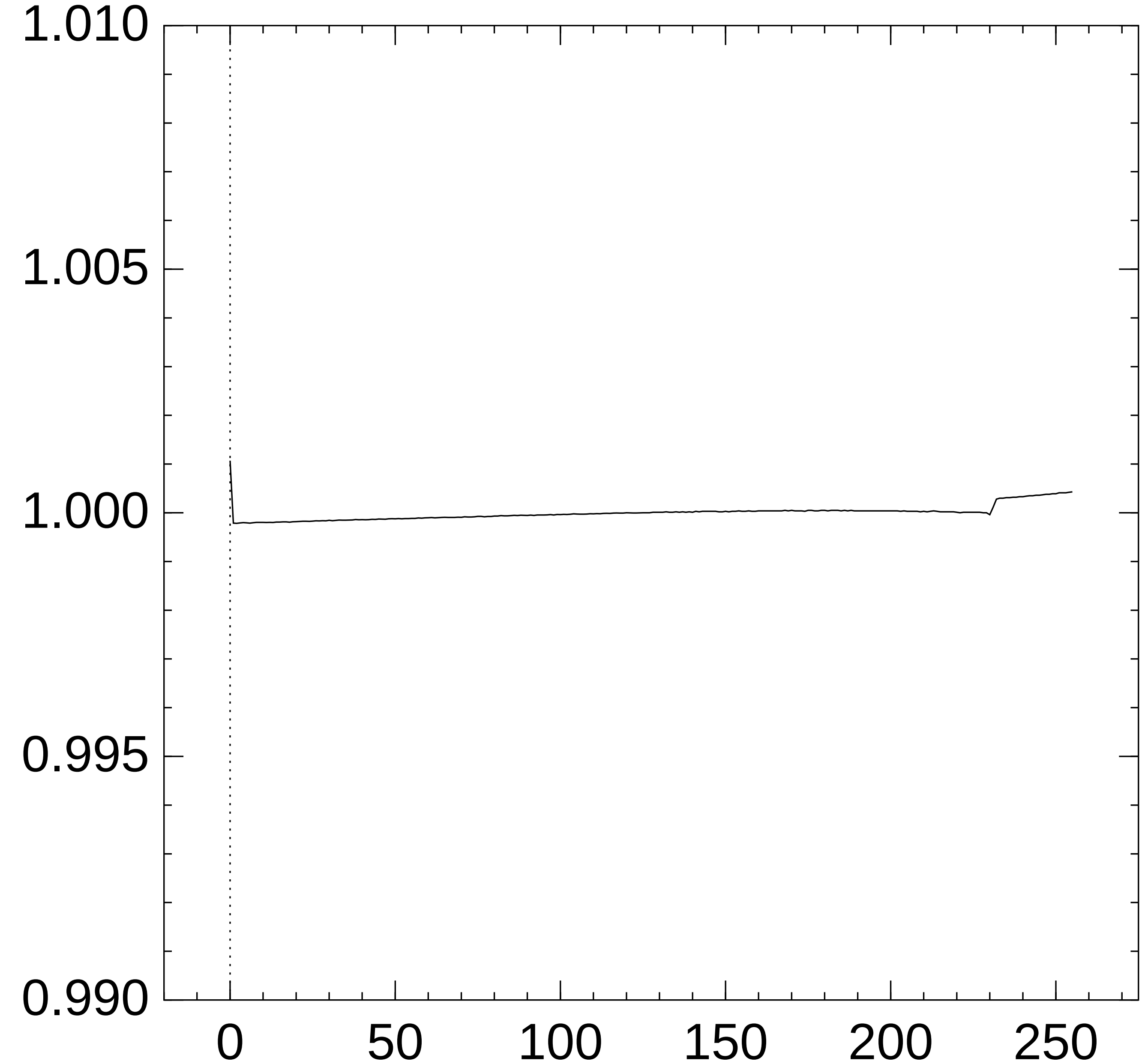
Graph of $256 \Pr[z_{230} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

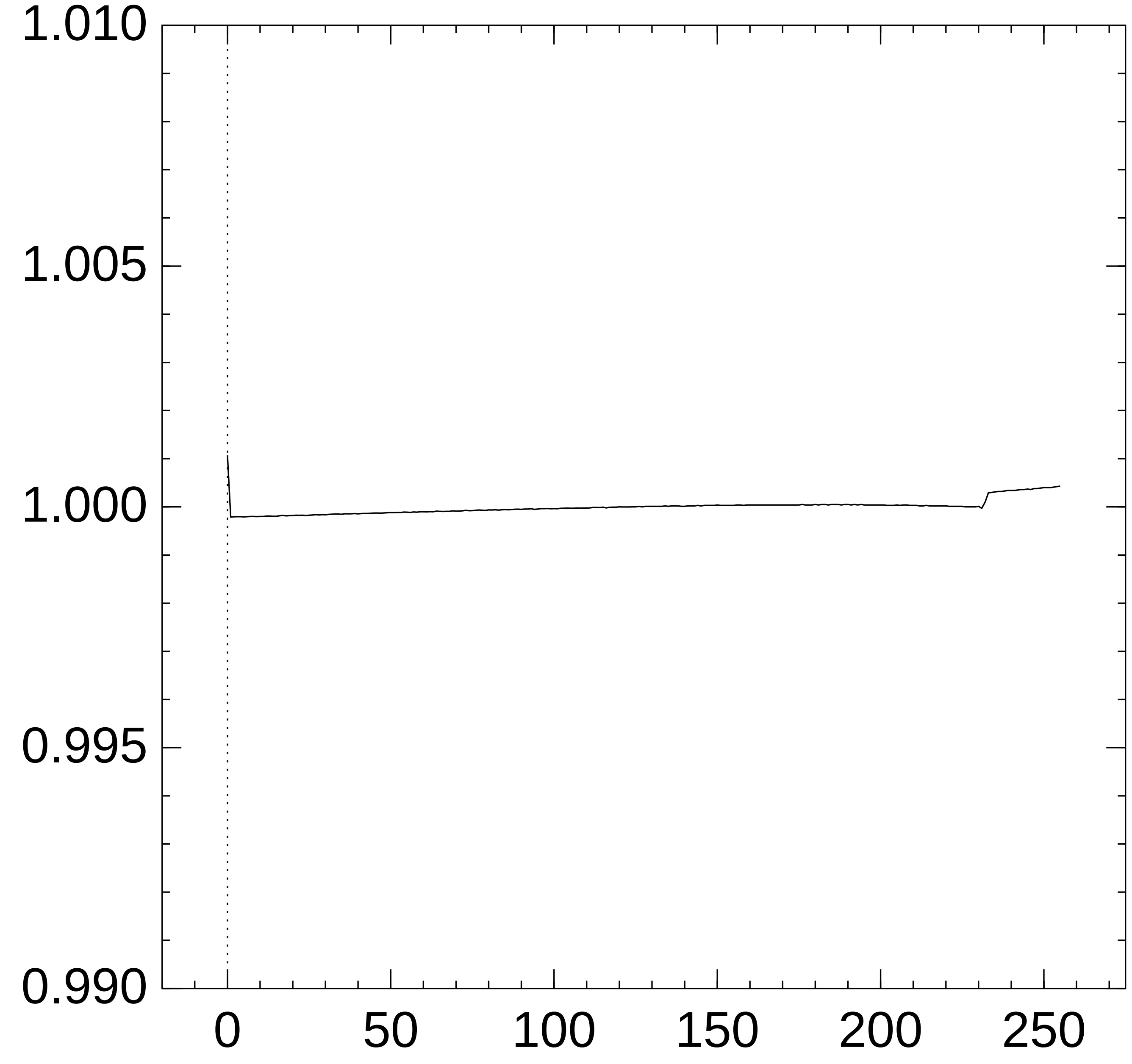
Graph of $256 \Pr[z_{231} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

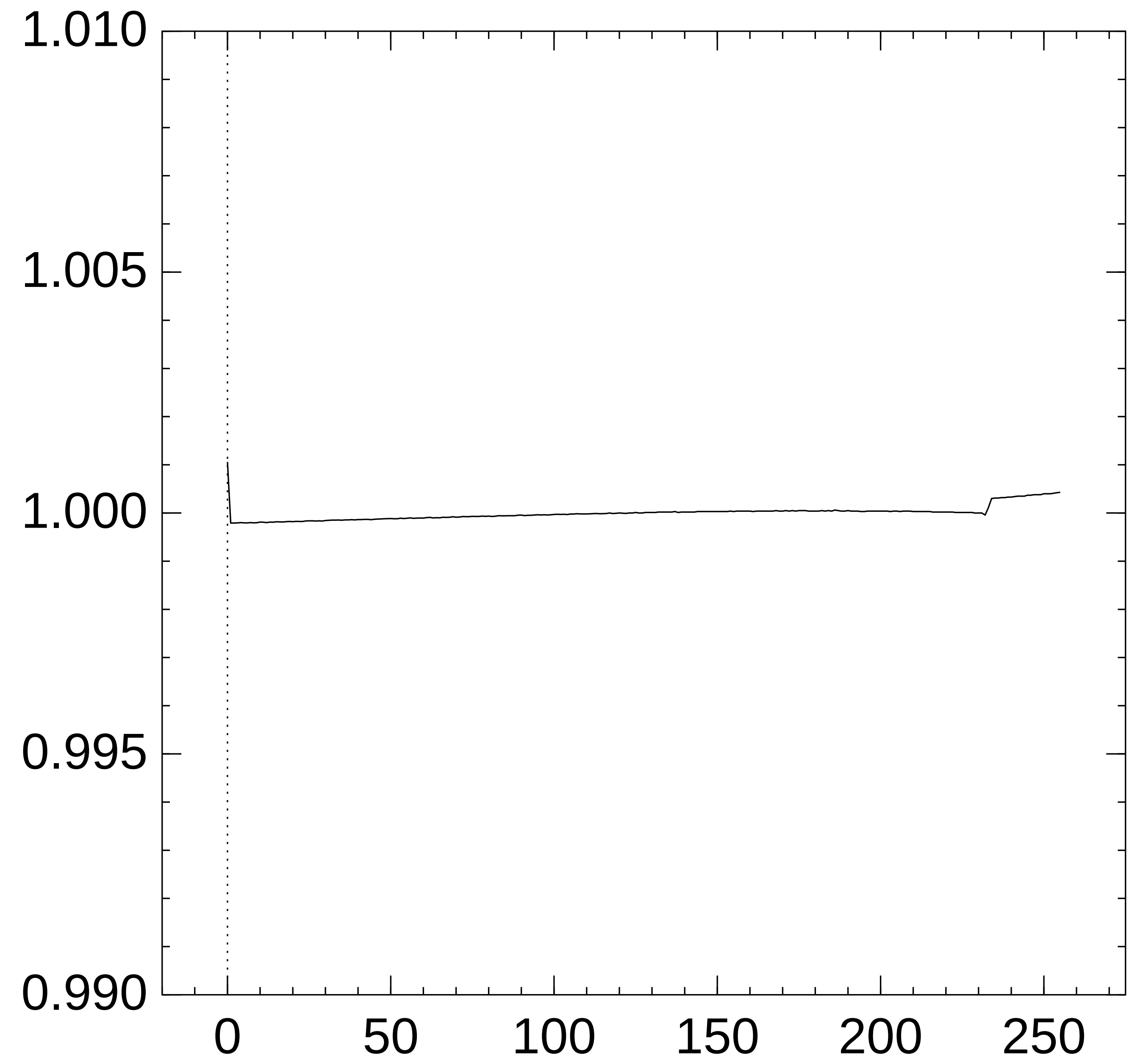
Graph of $256 \Pr[z_{232} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

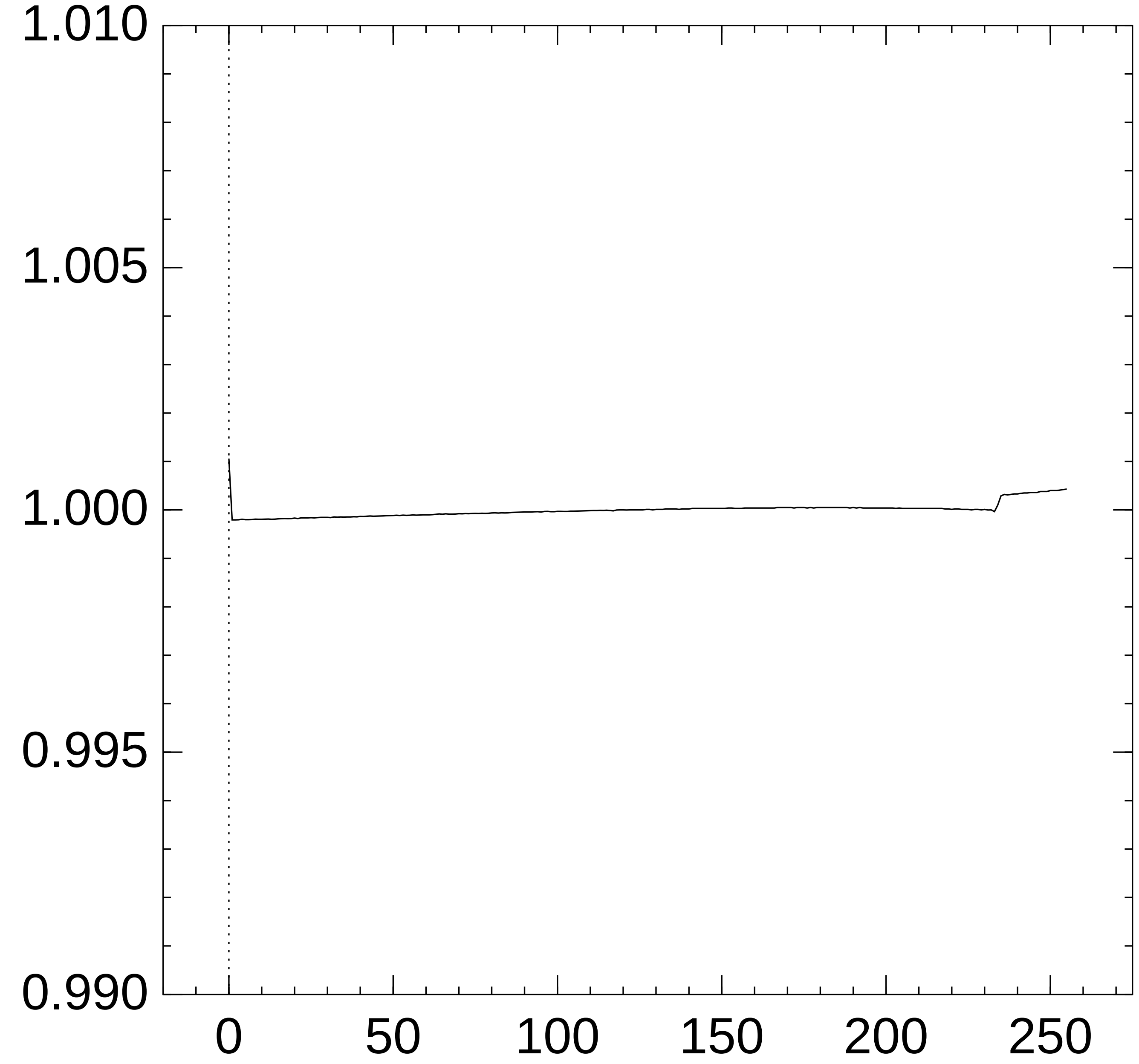
Graph of $256 \Pr[z_{233} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

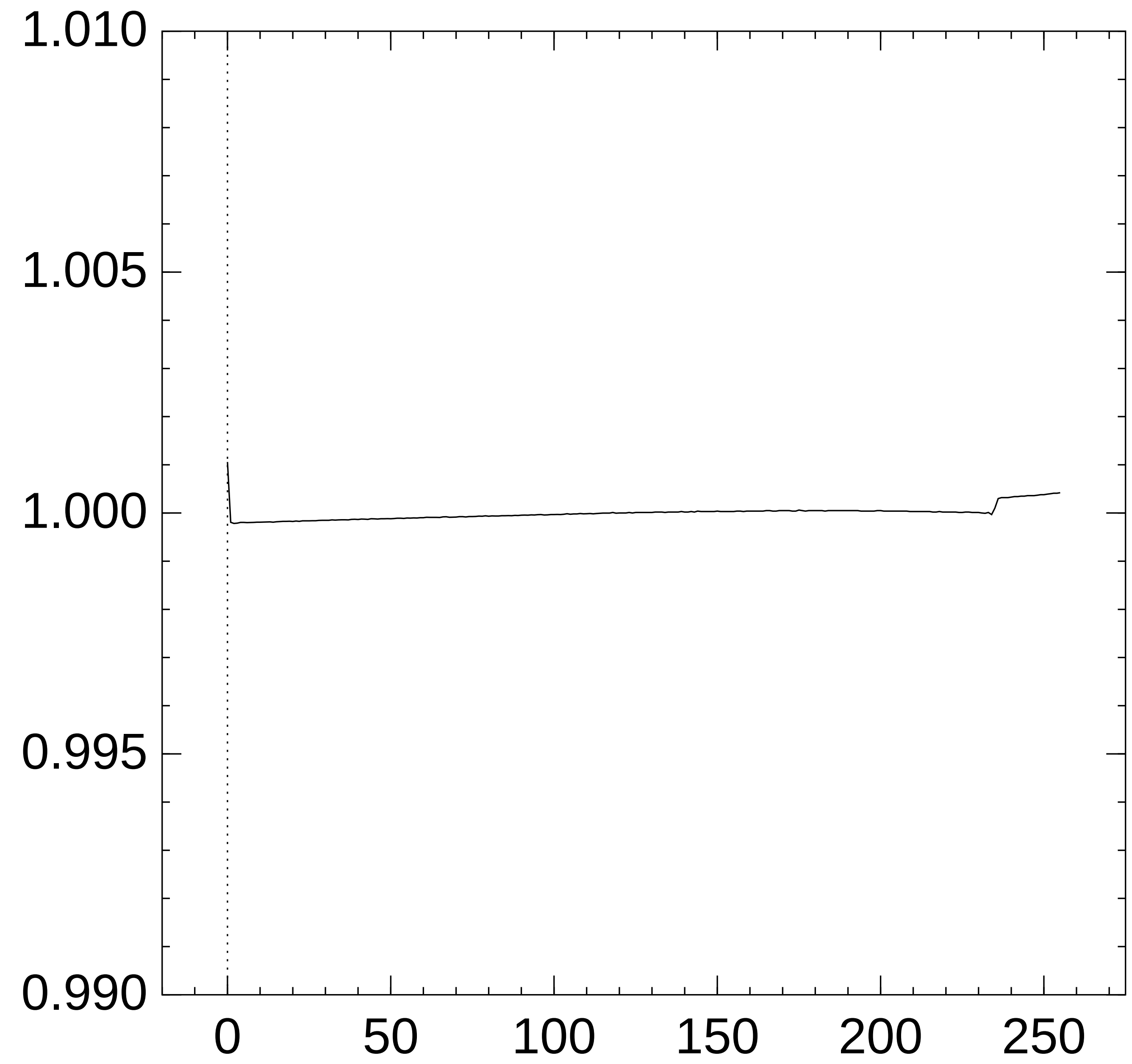
Graph of $256 \Pr[z_{234} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

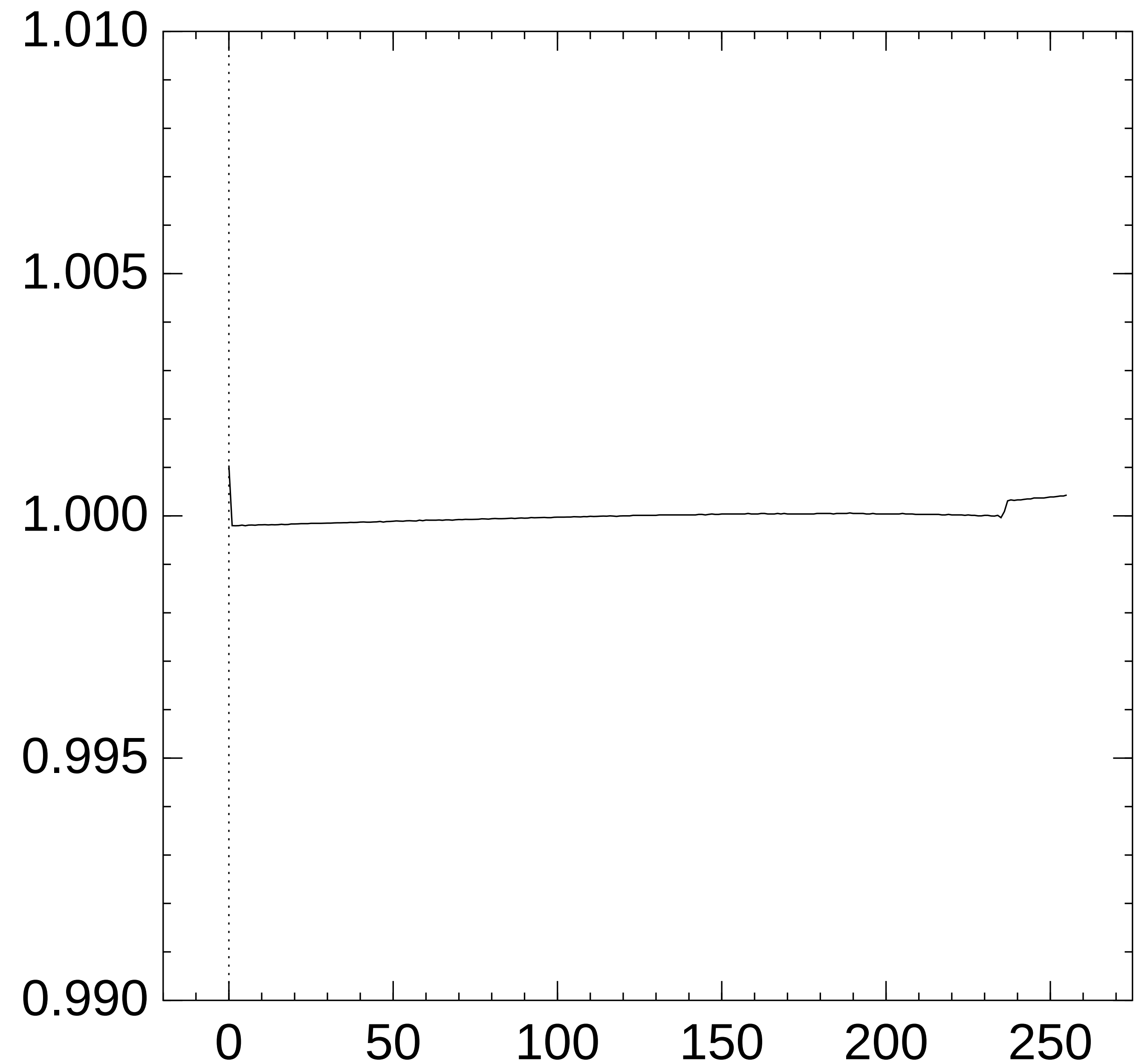
Graph of $256 \Pr[z_{235} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

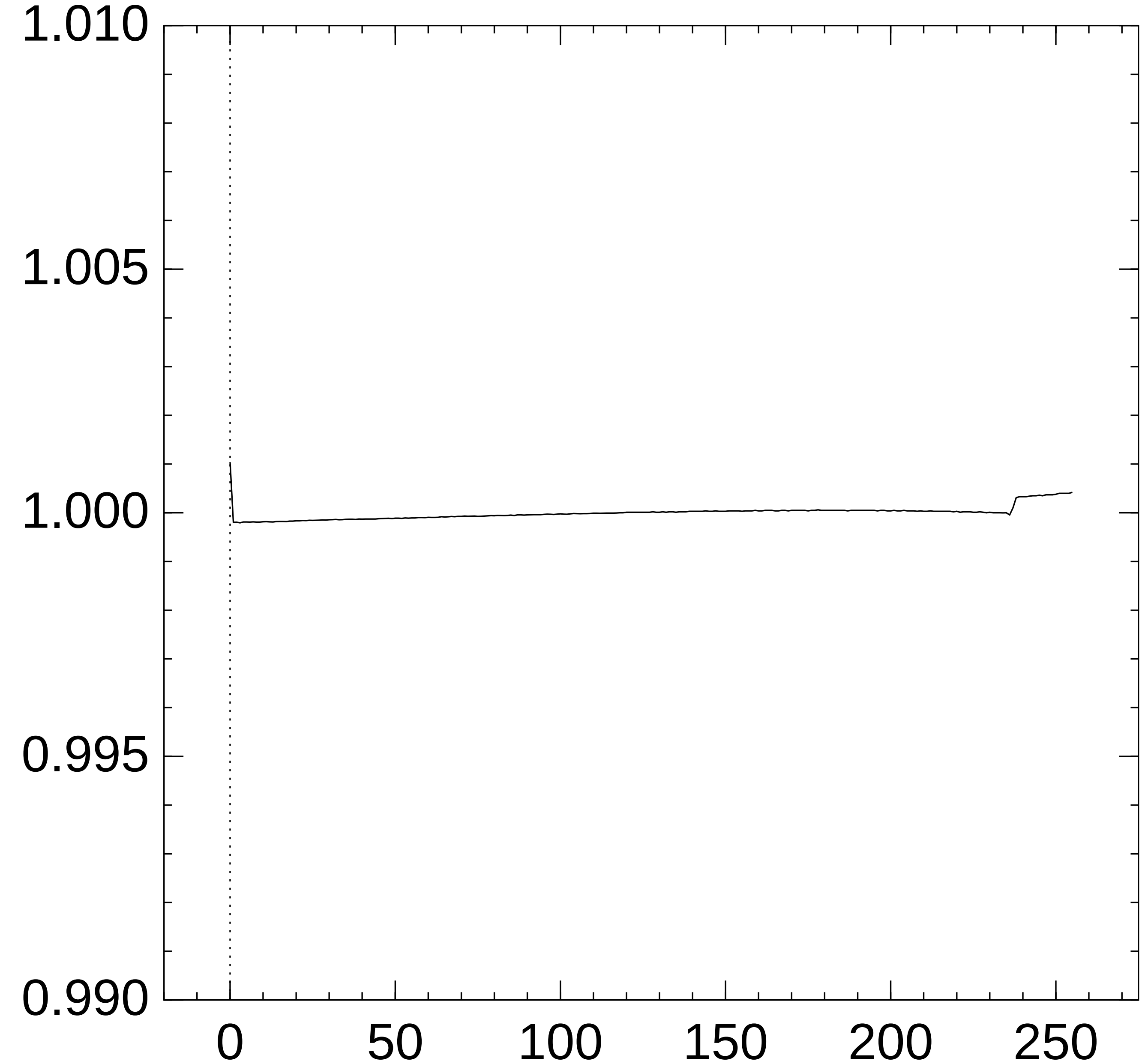
Graph of $256 \Pr[z_{236} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

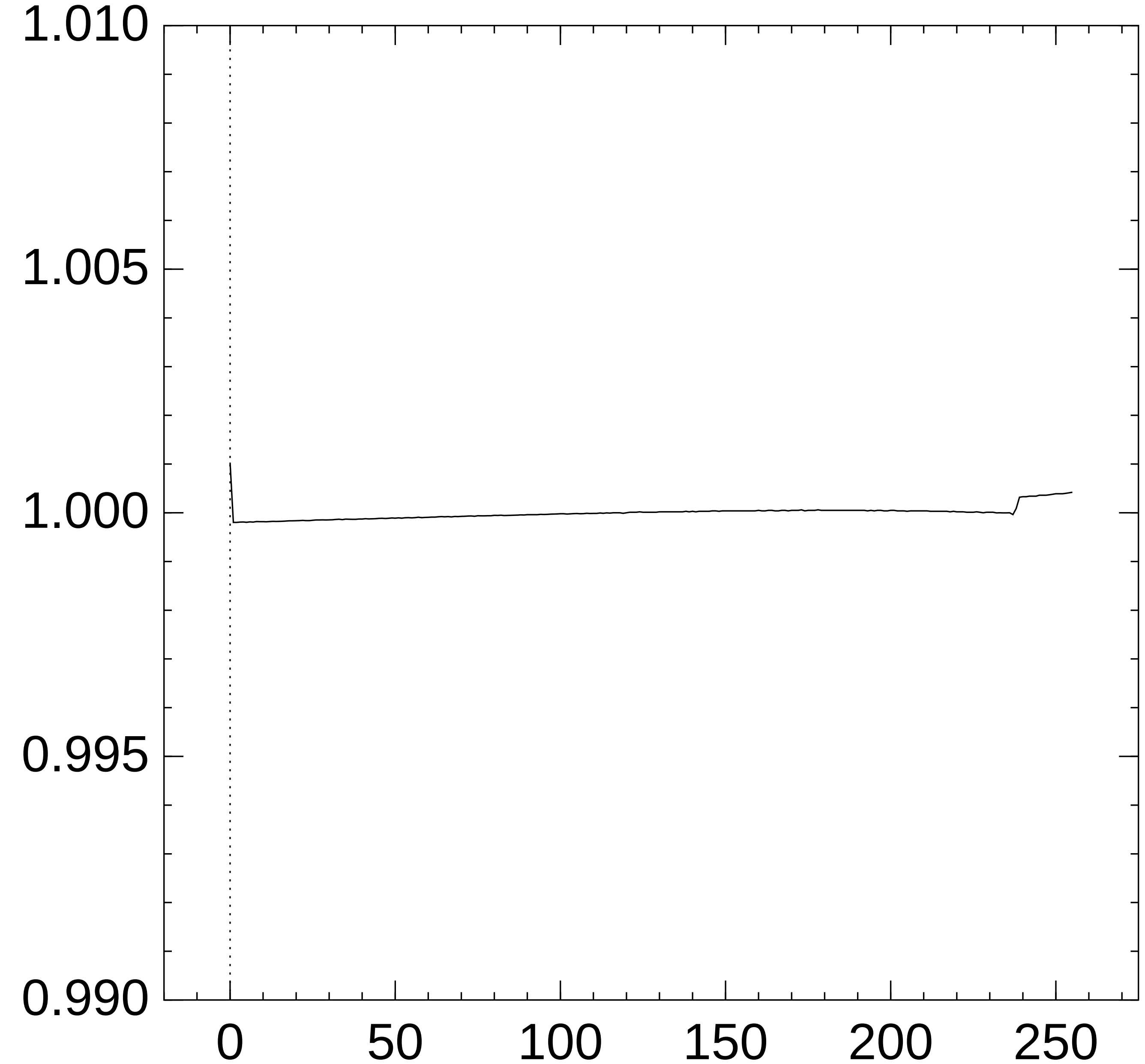
Graph of $256 \Pr[z_{237} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

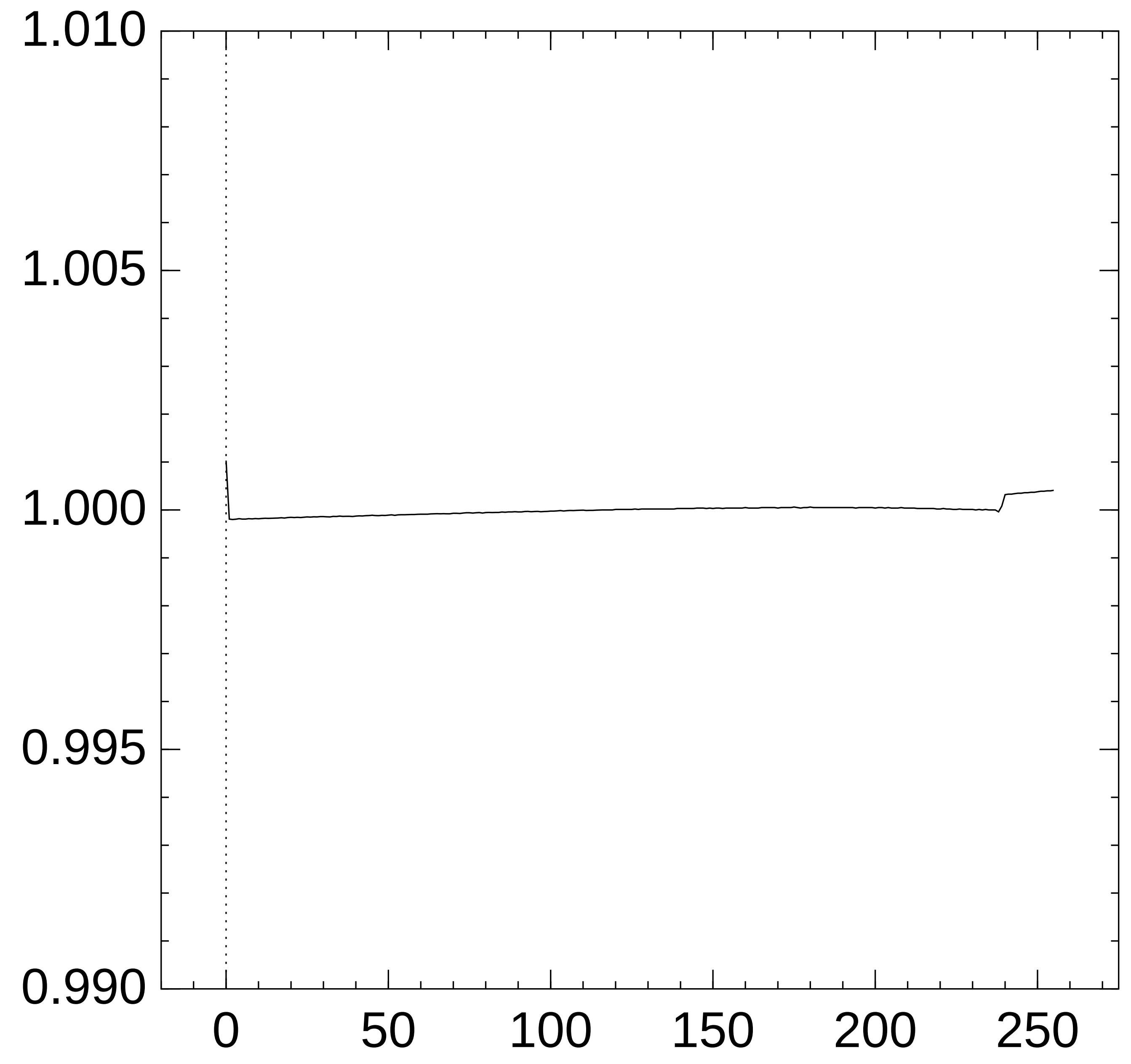
Graph of $256 \Pr[z_{238} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

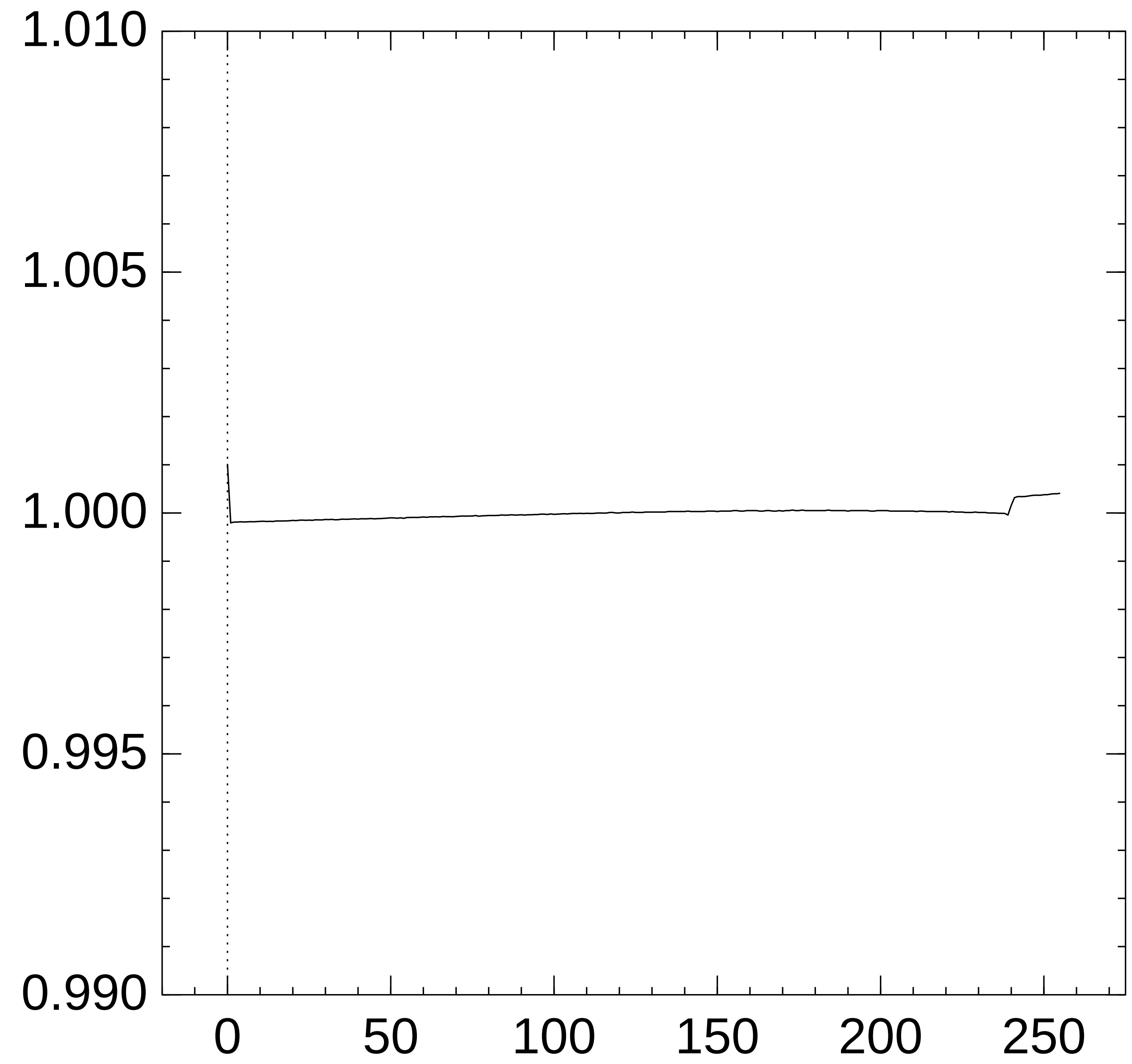
Graph of $256 \Pr[z_{239} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

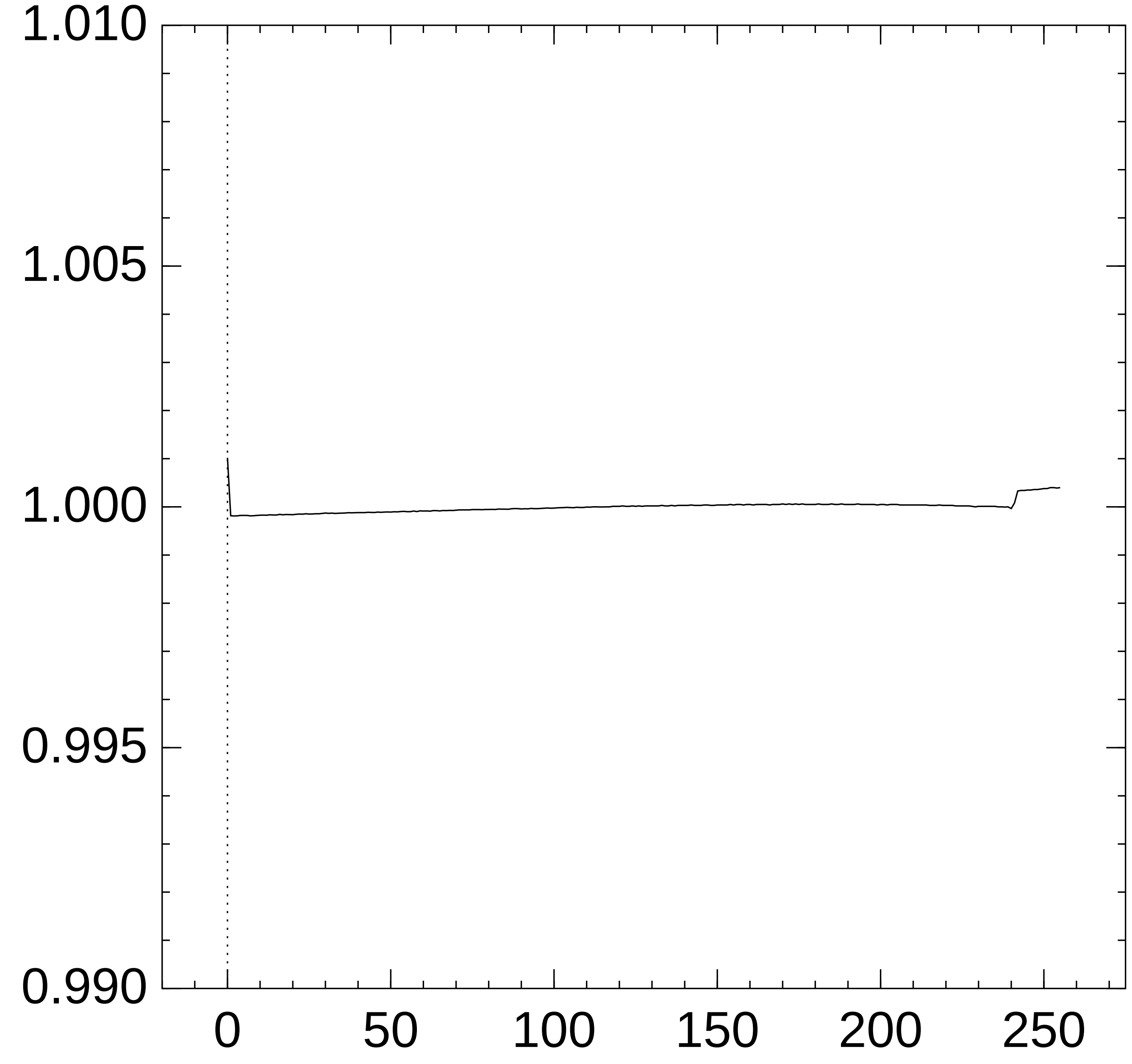
Graph of $256 \Pr[z_{240} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

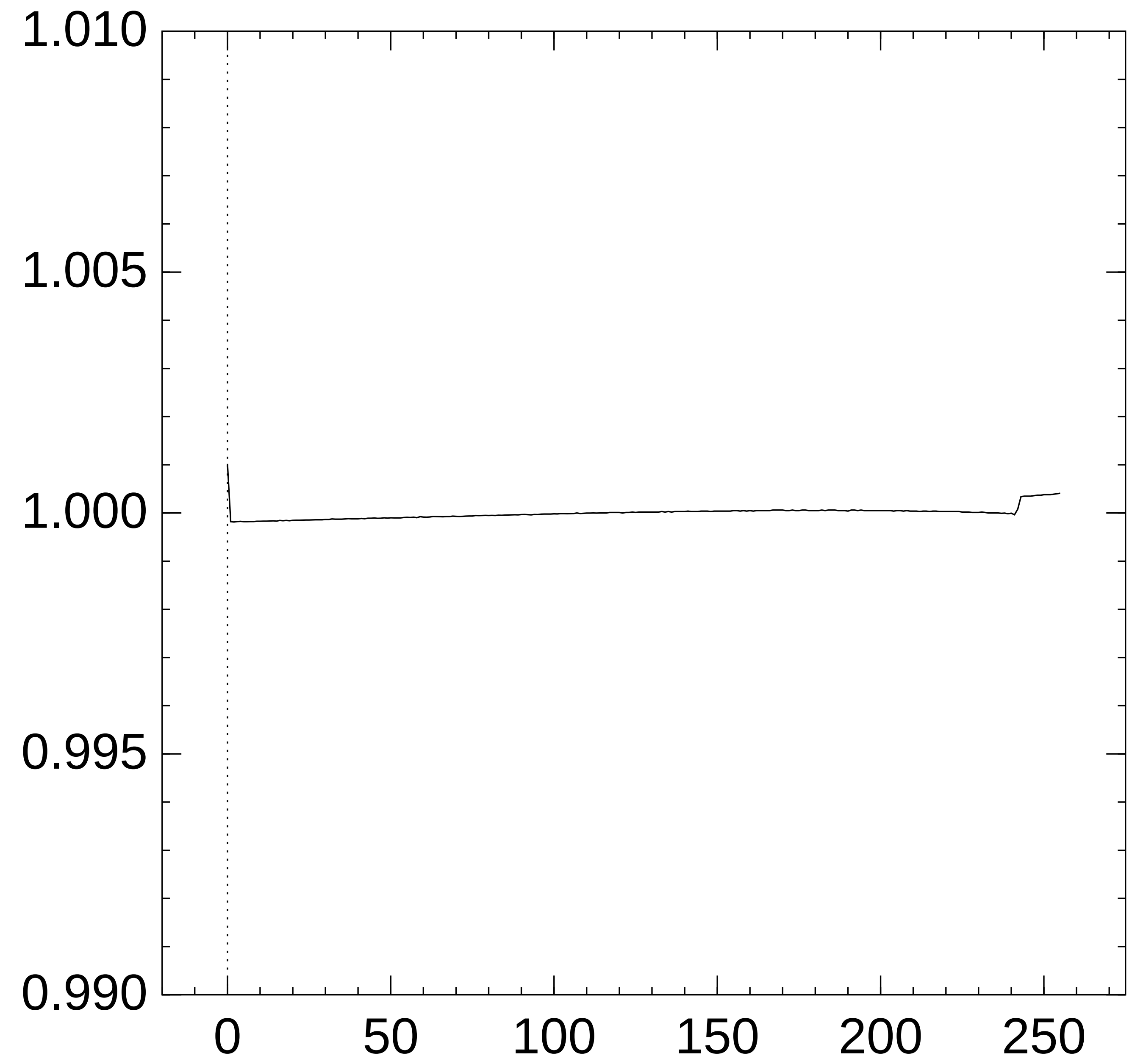
Graph of $256 \Pr[z_{241} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

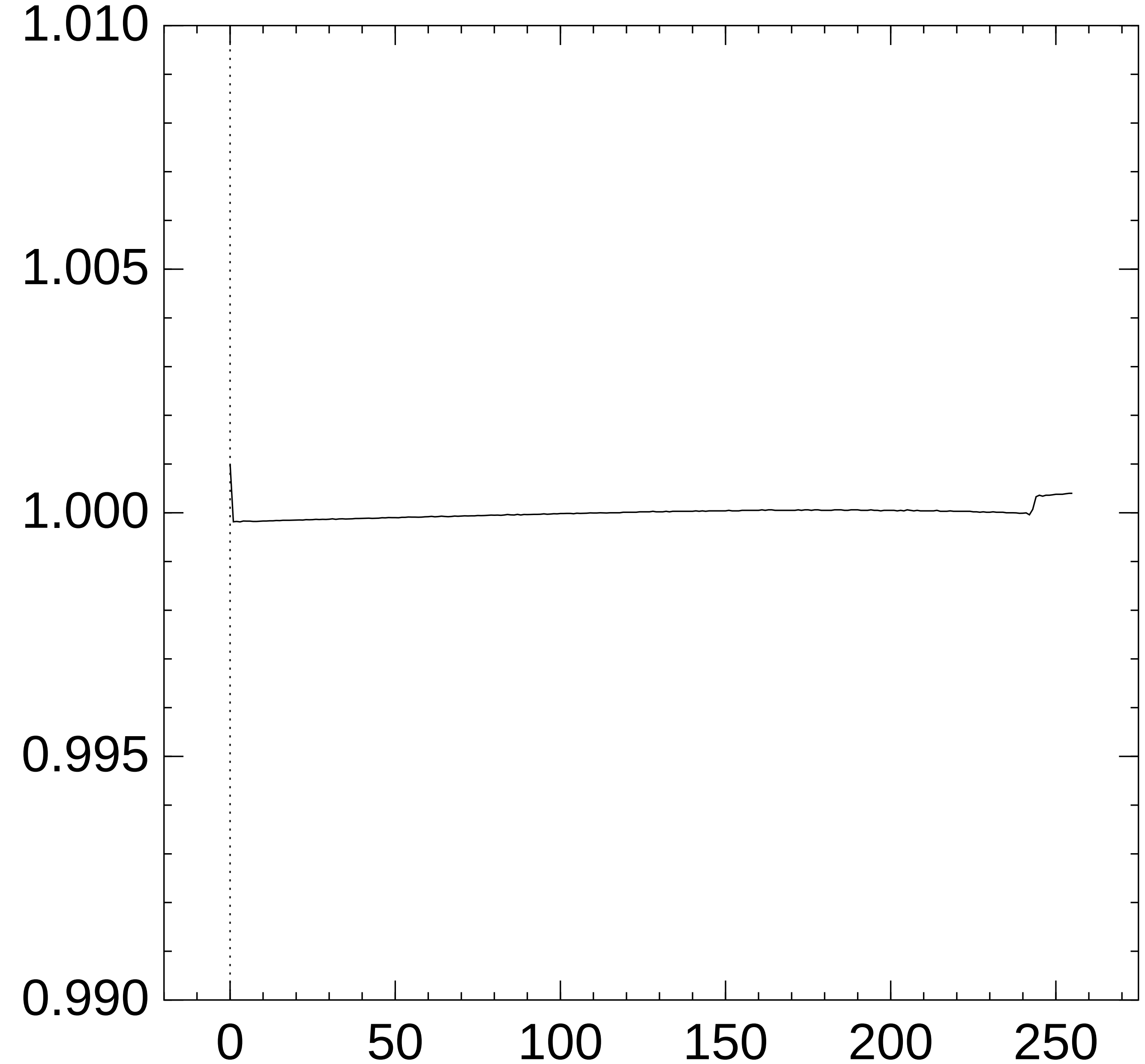
Graph of $256 \Pr[z_{242} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

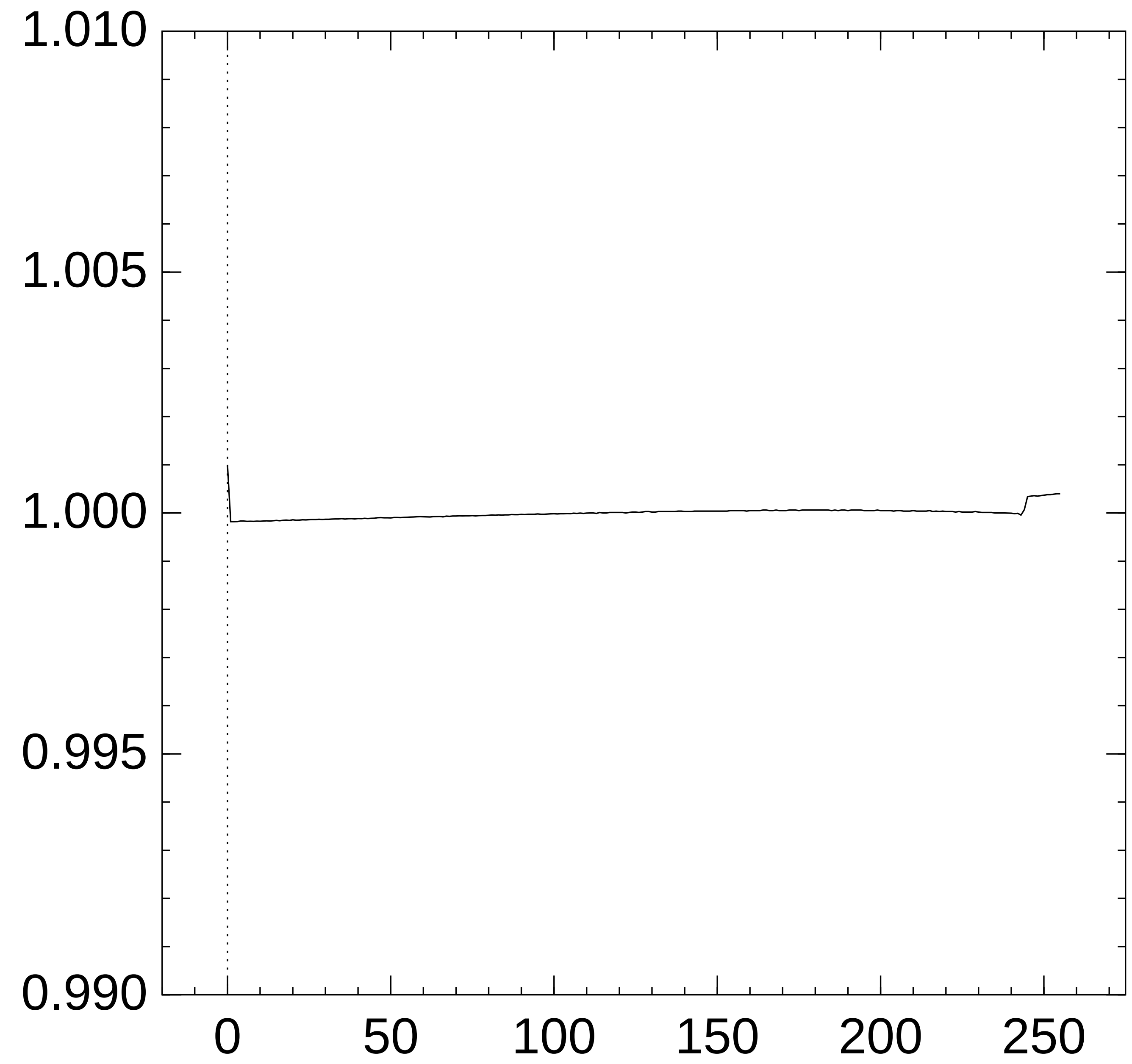
Graph of $256 \Pr[z_{243} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

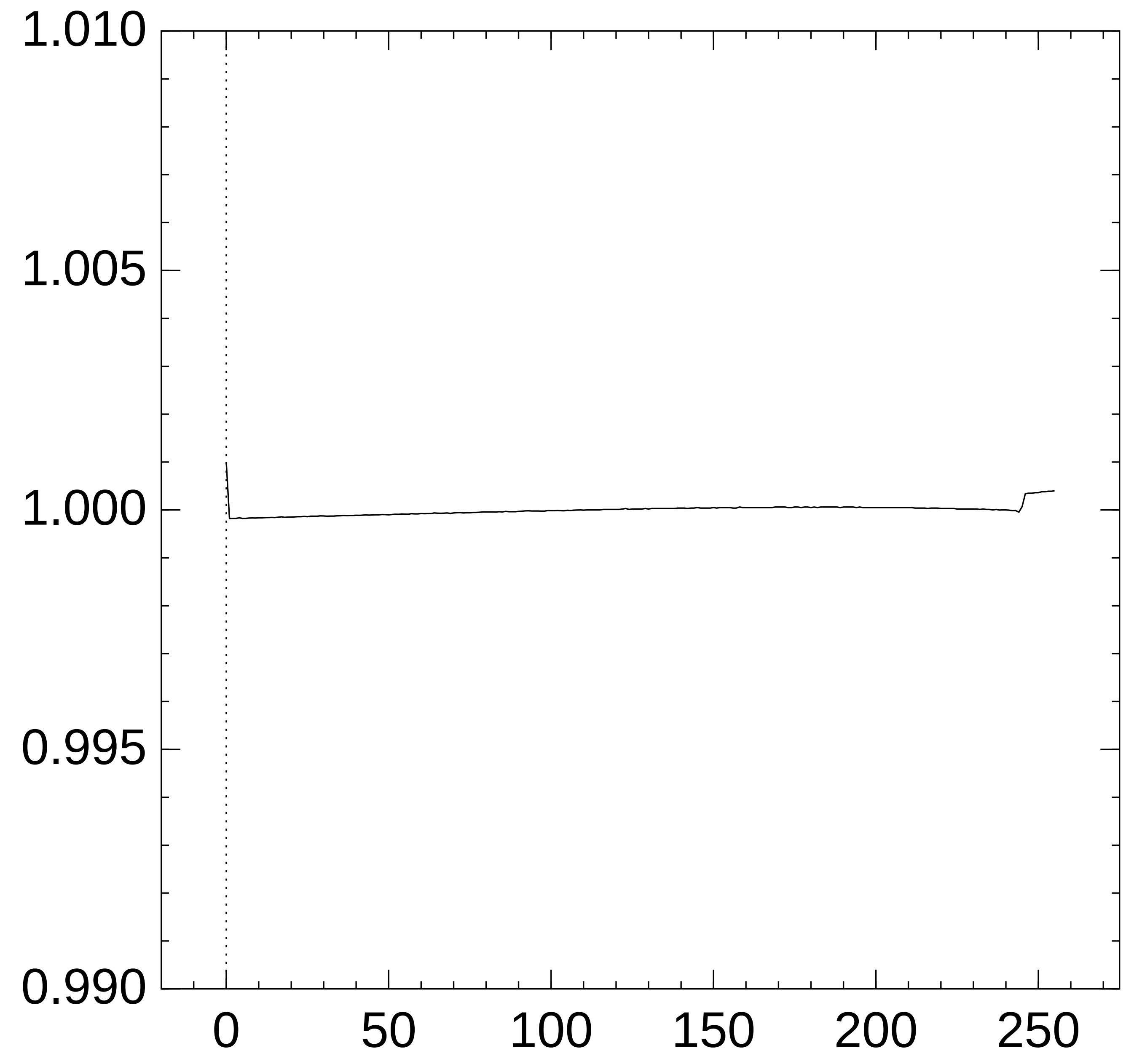
Graph of $256 \Pr[z_{244} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

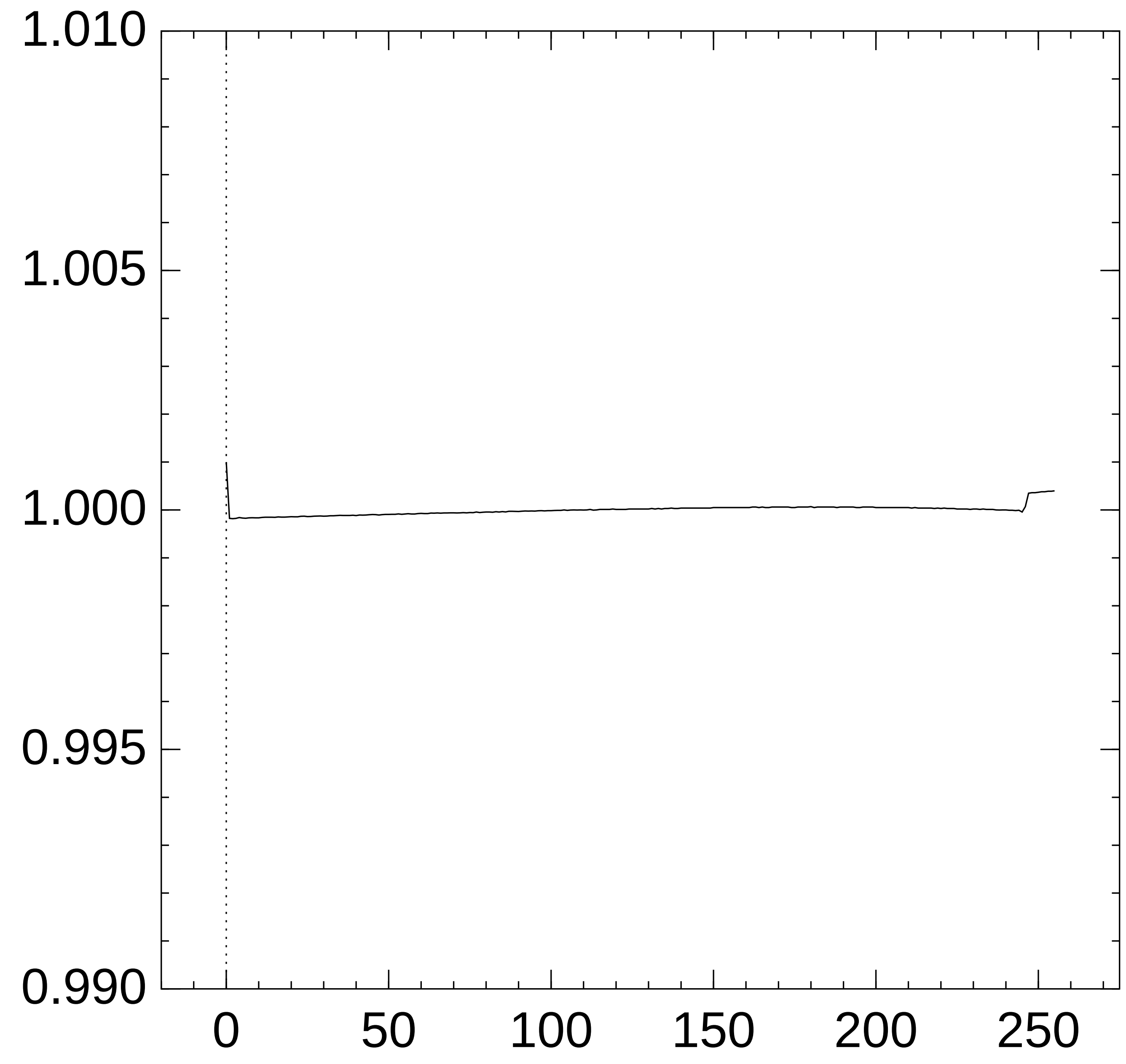
Graph of $256 \Pr[z_{245} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

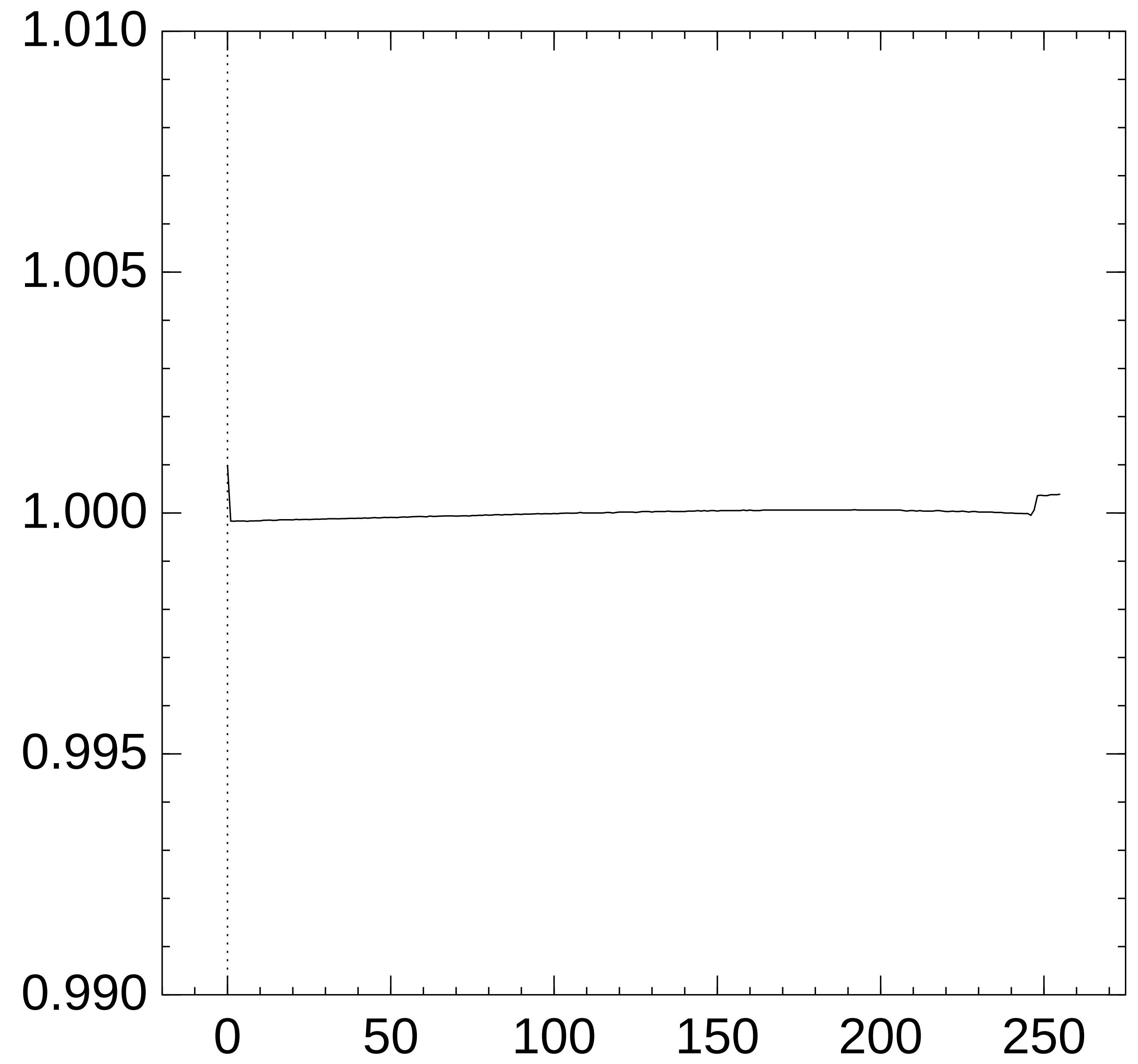
Graph of $256 \Pr[z_{246} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

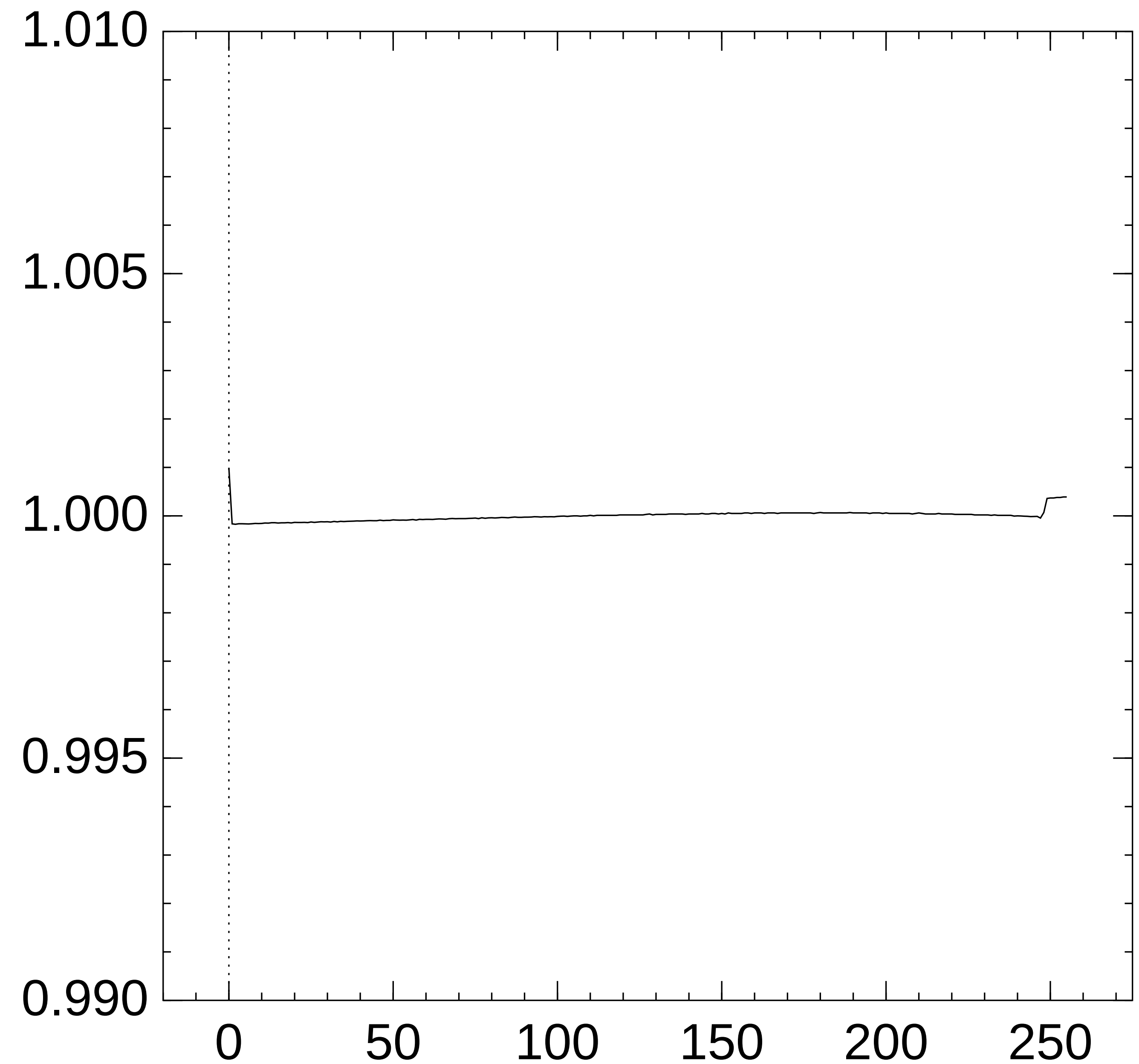
Graph of $256 \Pr[z_{247} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

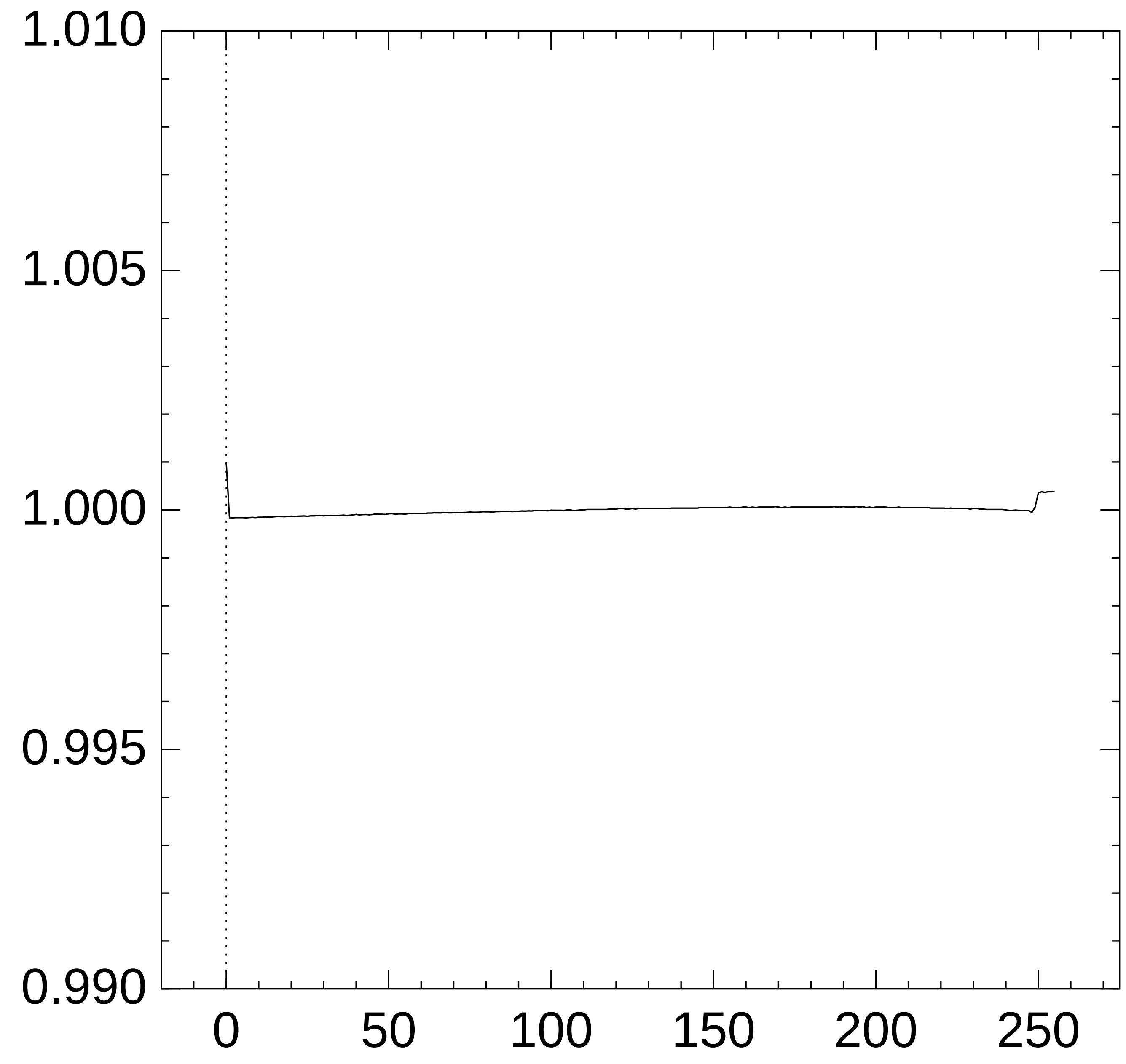
Graph of $256 \Pr[z_{248} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

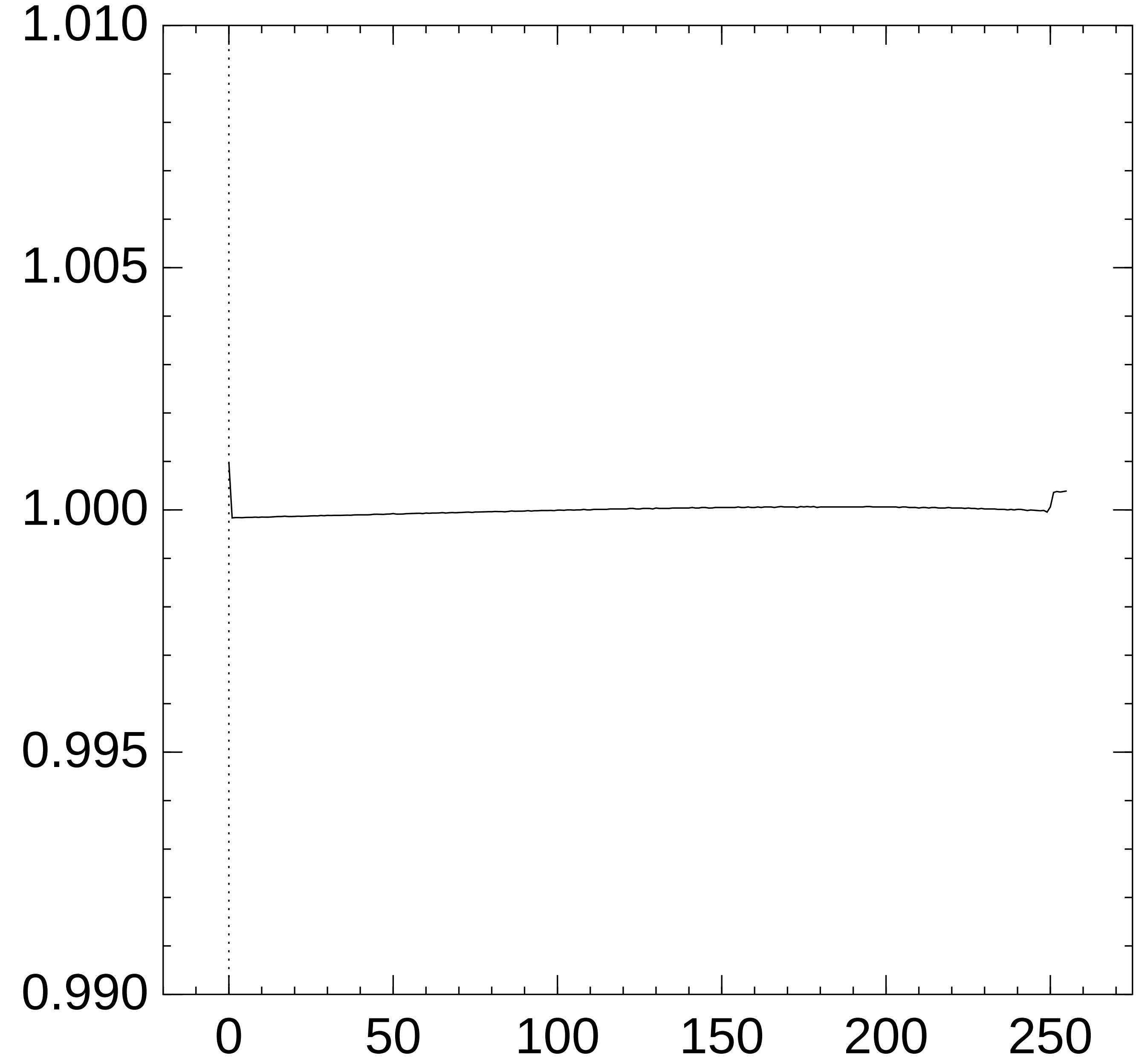
Graph of $256 \Pr[z_{249} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

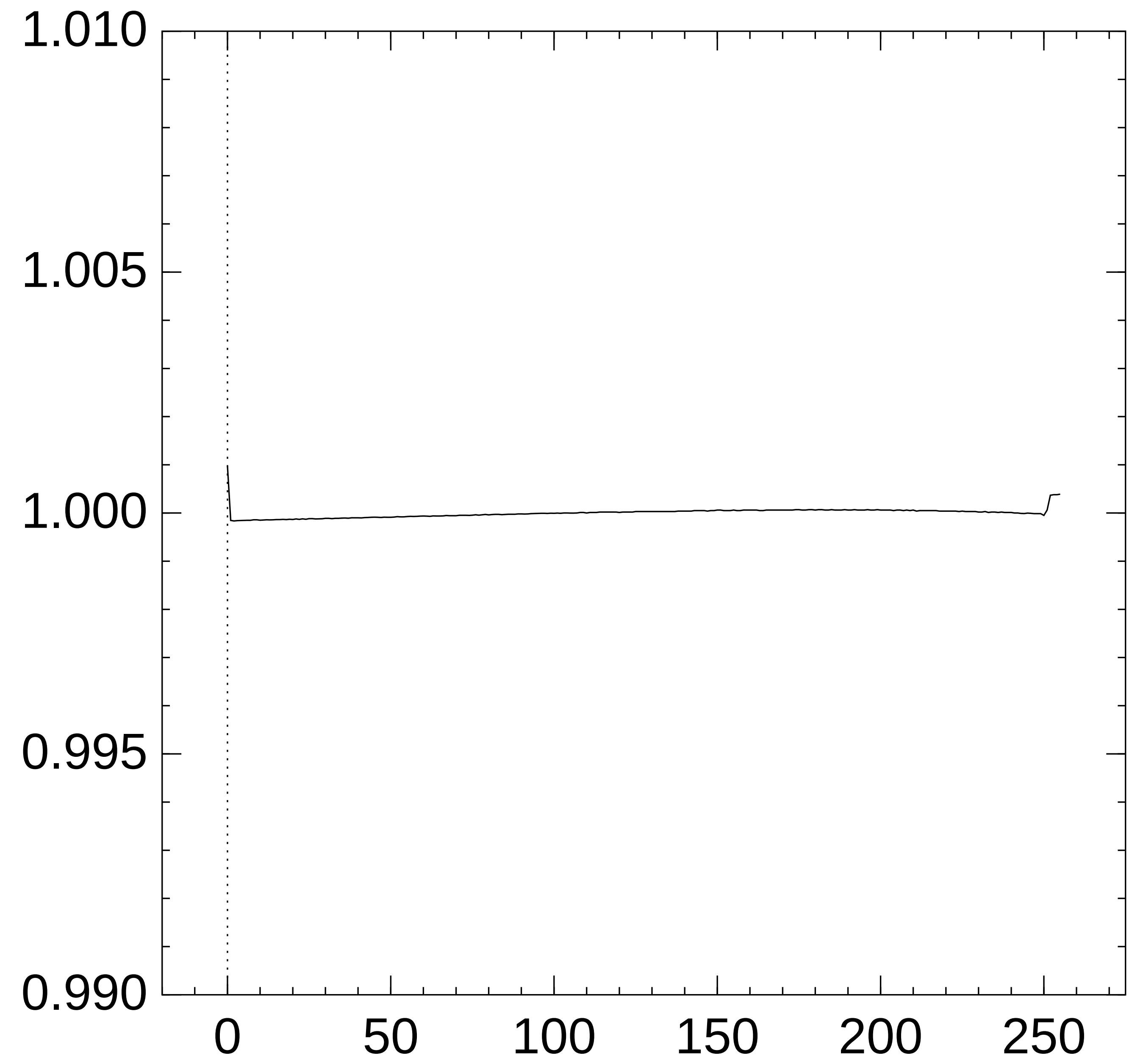
Graph of $256 \Pr[z_{250} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

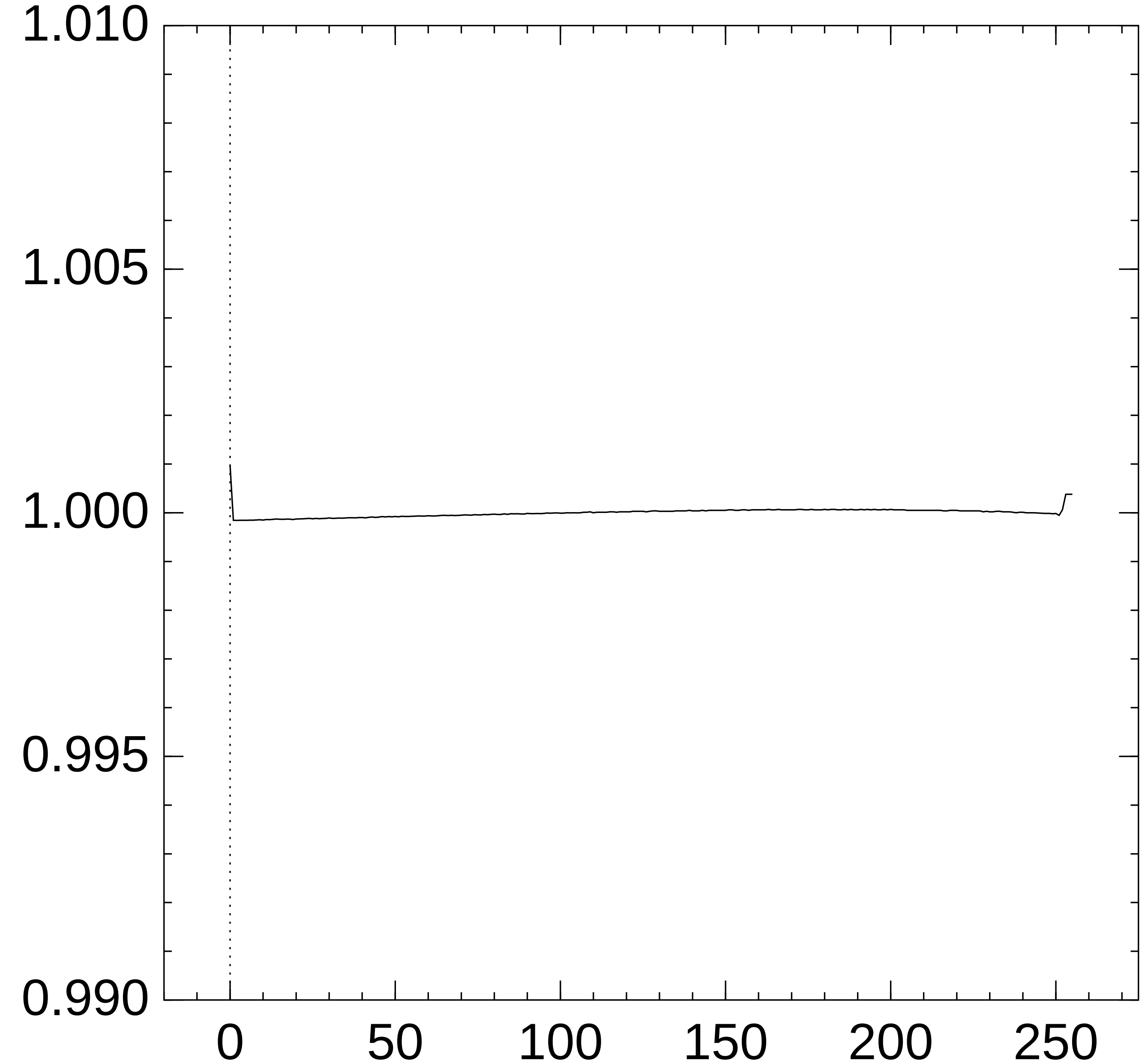
Graph of $256 \Pr[z_{251} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

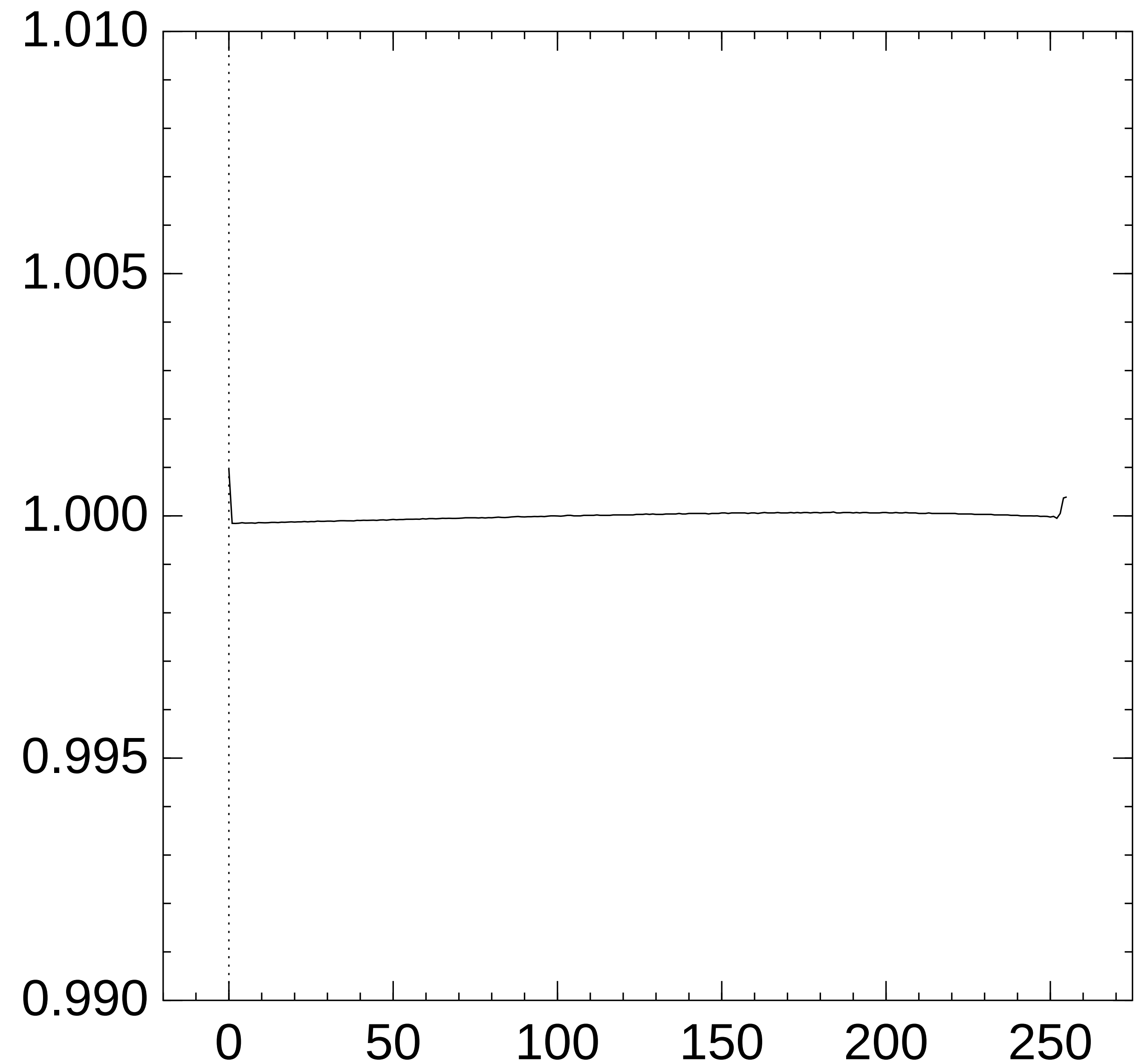
Graph of $256\Pr[z_{252} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

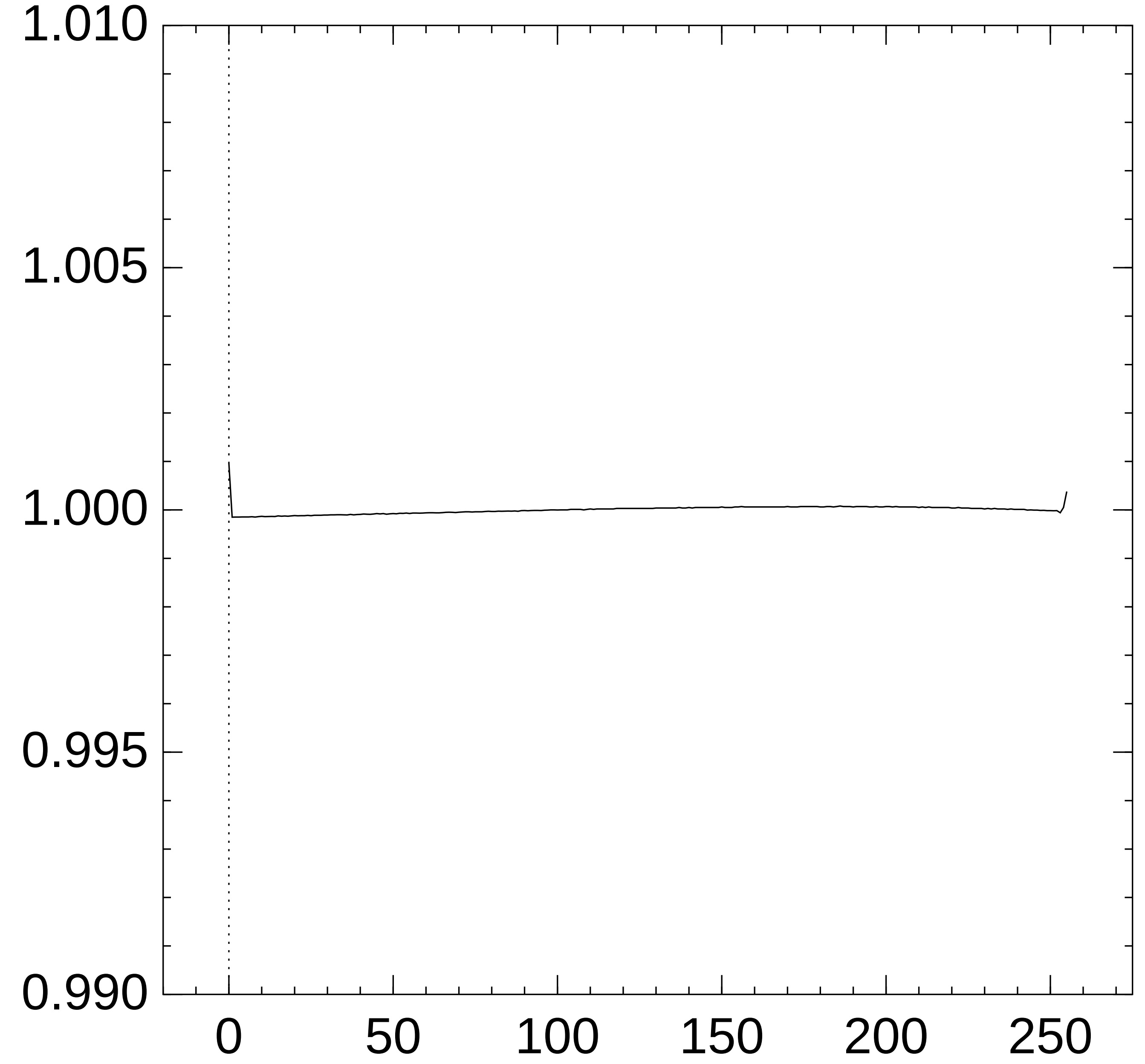
Graph of $256 \Pr[z_{253} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found ≈**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

≈256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

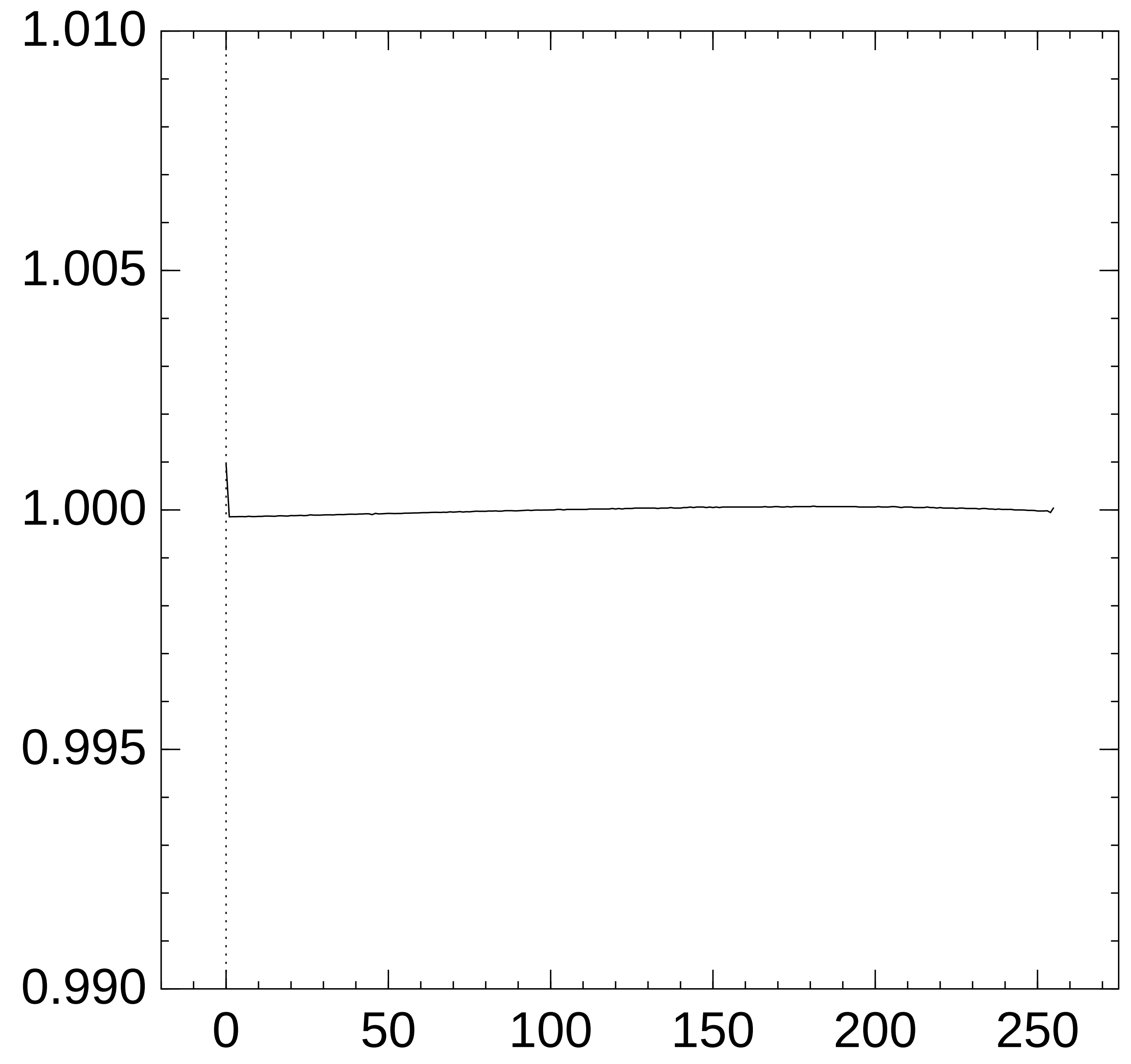
Graph of $256 \Pr[z_{254} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

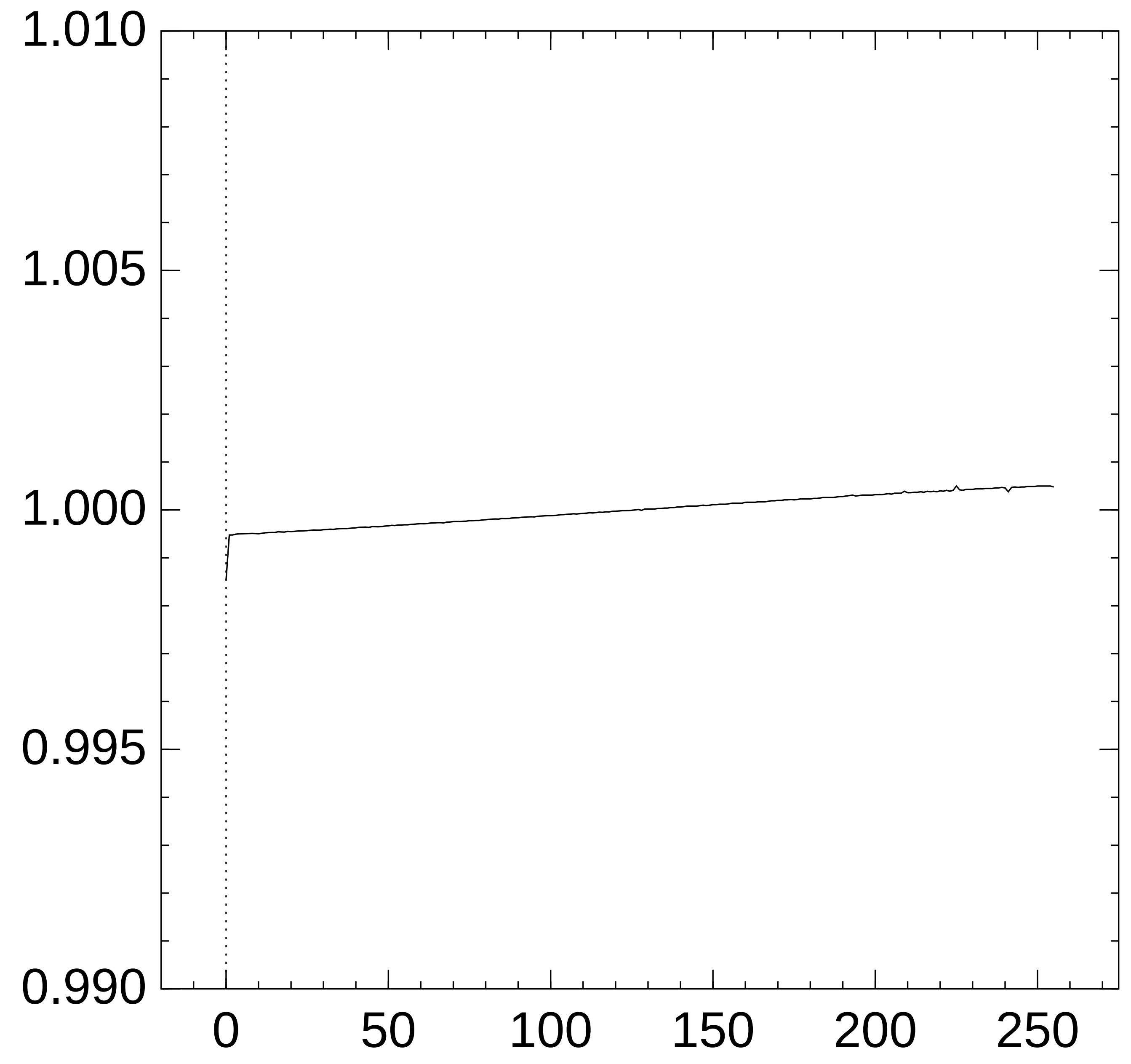
Graph of $256 \Pr[z_{255} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt: *accurately* computed $\Pr[z_i = j]$ for all $i \in \{1, \ldots, 256\}$, all j; found $\approx$**65536** single-byte biases; used *all* of them in SSL attack via proper Bayesian analysis.

$\approx$256 of these biases were found independently (slightly earlier) by 2013 Watanabe–Isobe–Ohigashi–Morii, 2013 Isobe–Ohigashi–Watanabe–Morii: $z_{32} \to 224$, $z_{48} \to 208$, etc.; $z_3 \to 131$; $z_i \to i$; $z_{256} \not\to 0$.

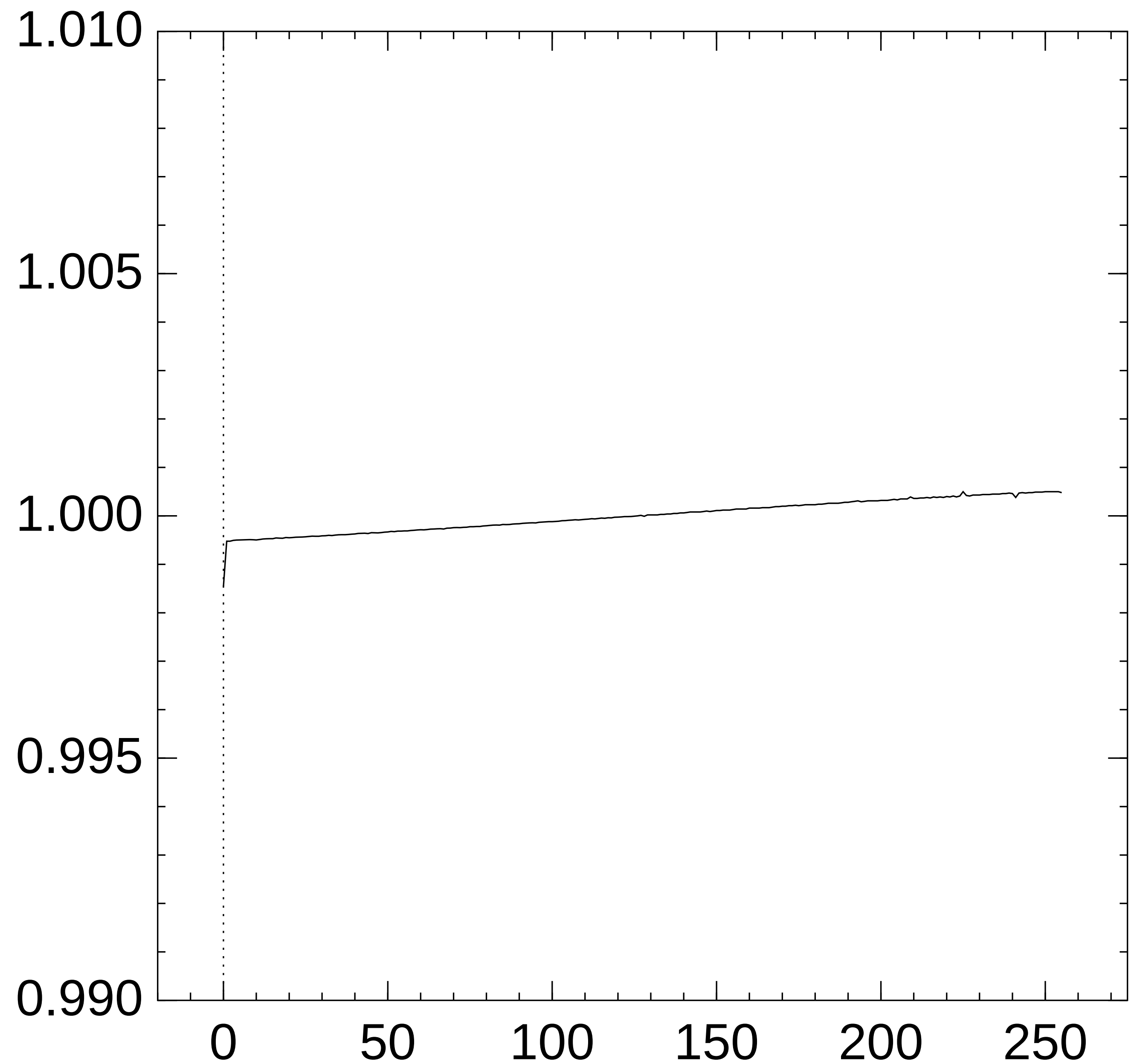
Graph of $256 \Pr[z_{256} = x]$:



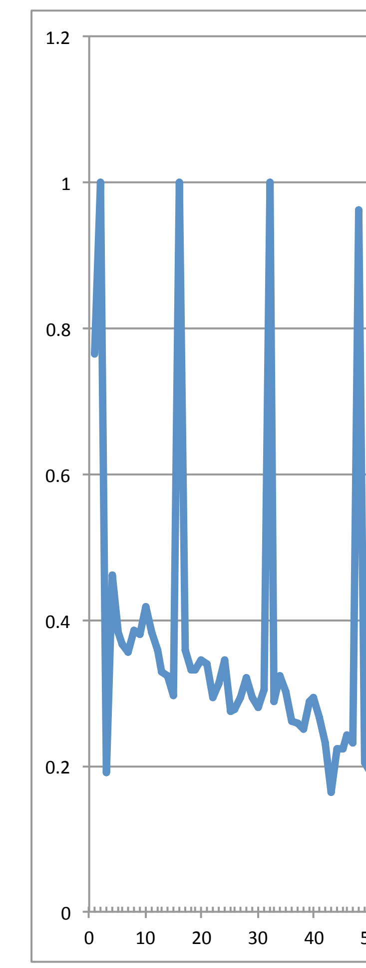
Fardan–Bernstein–
n–Poettering–Schuldt:
ely computed $\Pr[z_i = j]$
$\in \{1, \ldots, 256\}$, all j;
65536 single-byte biases;
of them in SSL attack
er Bayesian analysis.

these biases were found
dently (slightly earlier)
Watanabe–Isobe–
i–Morii, 2013 Isobe–
i–Watanabe–Morii:
24, $z_{48} \to 208$, etc.;
1; $z_i \to i$; $z_{256} \not\to 0$.

Graph of $256 \Pr[z_{256} = x]$:



2013 AlF
Patersor
success
for recov
from 2^{24}
no prior

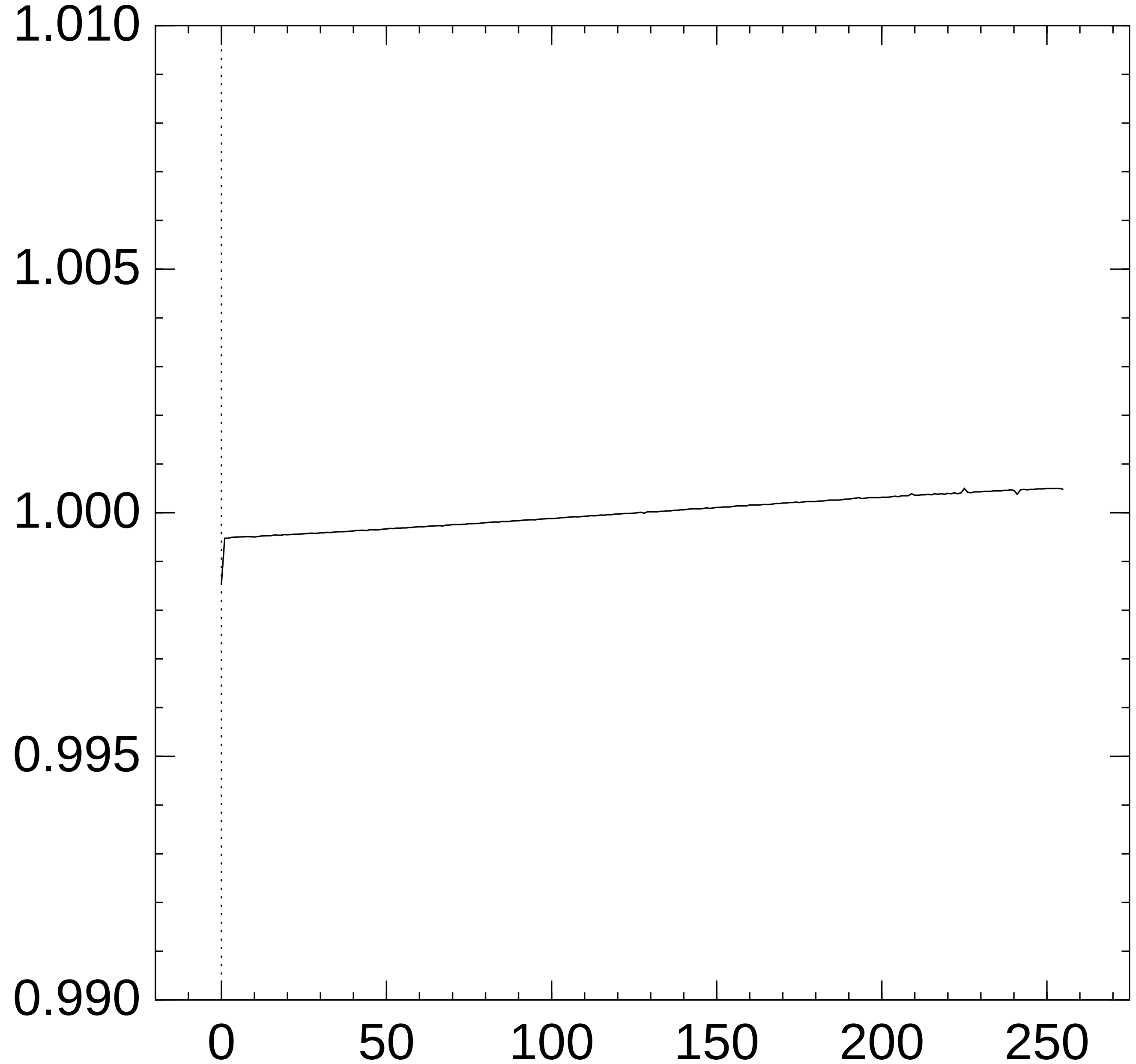


Later by

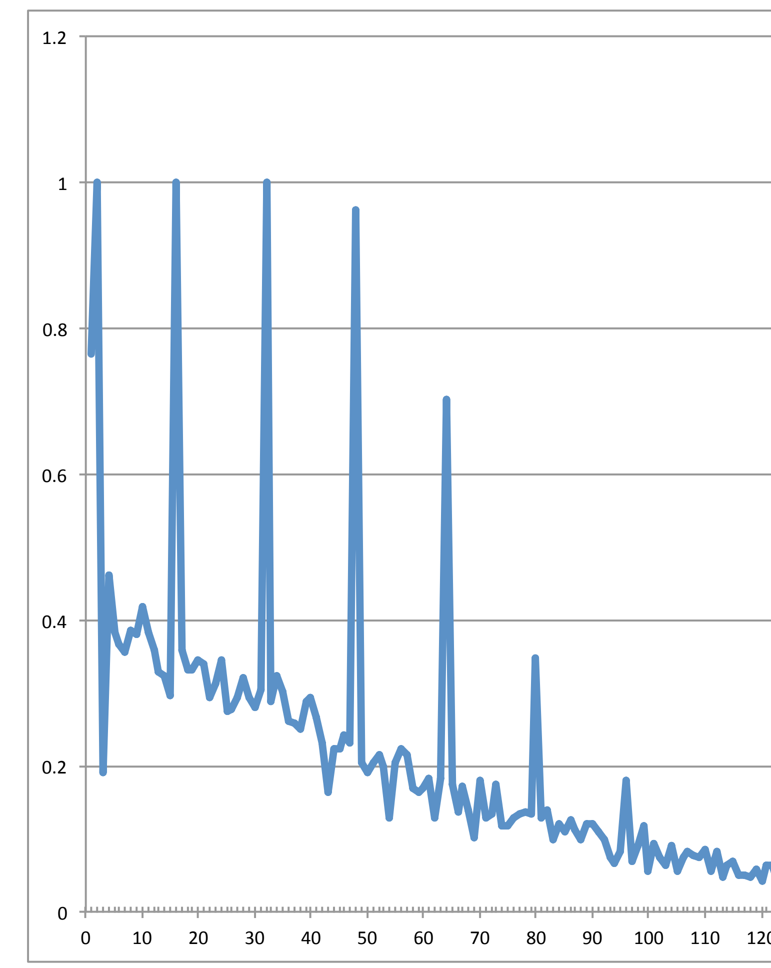
ernstein–
ng–Schuldt:
ted Pr$[z_i = j]$
256}, all j;
ngle-byte biases;
n SSL attack
n analysis.

ses were found
ghtly earlier)
e–Isobe–
013 Isobe–
be–Morii:
208, etc.;
$z_{256} \not\to 0$.

Graph of $256\,\Pr[z_{256} = x]$:



2013 AlFardan–Be
Paterson–Poetteri
success probability
for recovering byte
from 2^{24} ciphertex
no prior plaintext



Later bytes: see p

t:

$= j]$

j;

iases;

ack

.

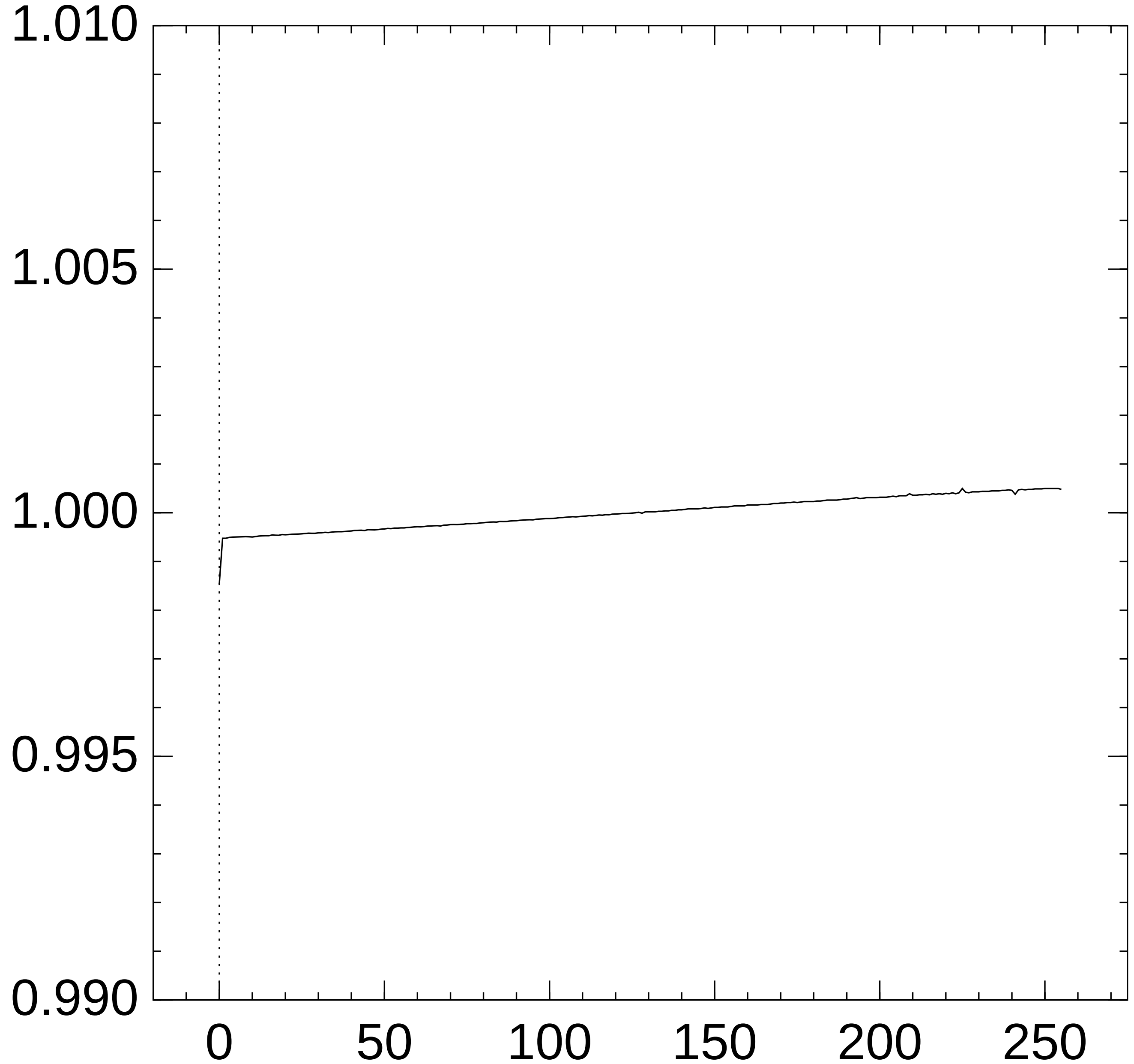
ound

er)

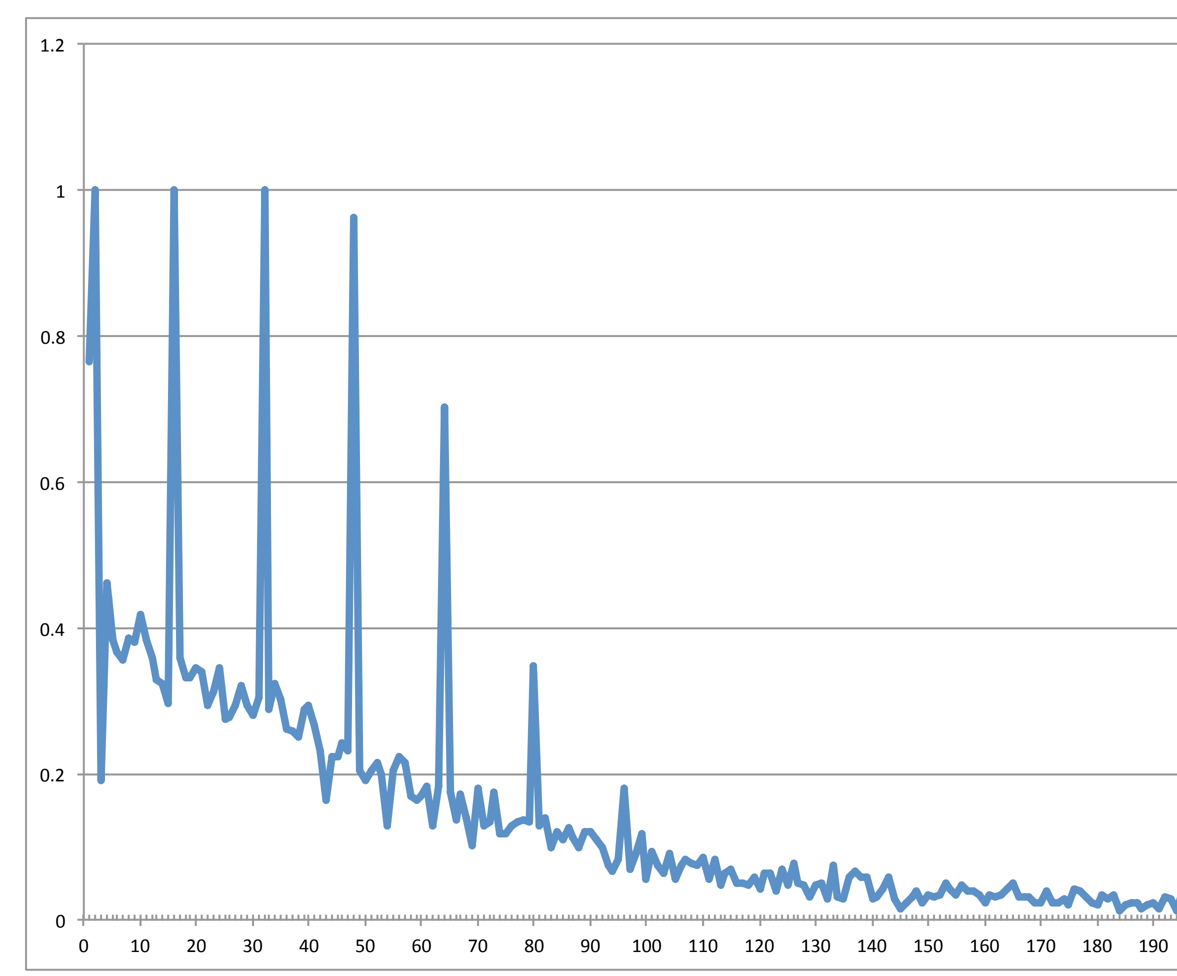
.

.

Graph of $256\,\Pr[z_{256} = x]$:

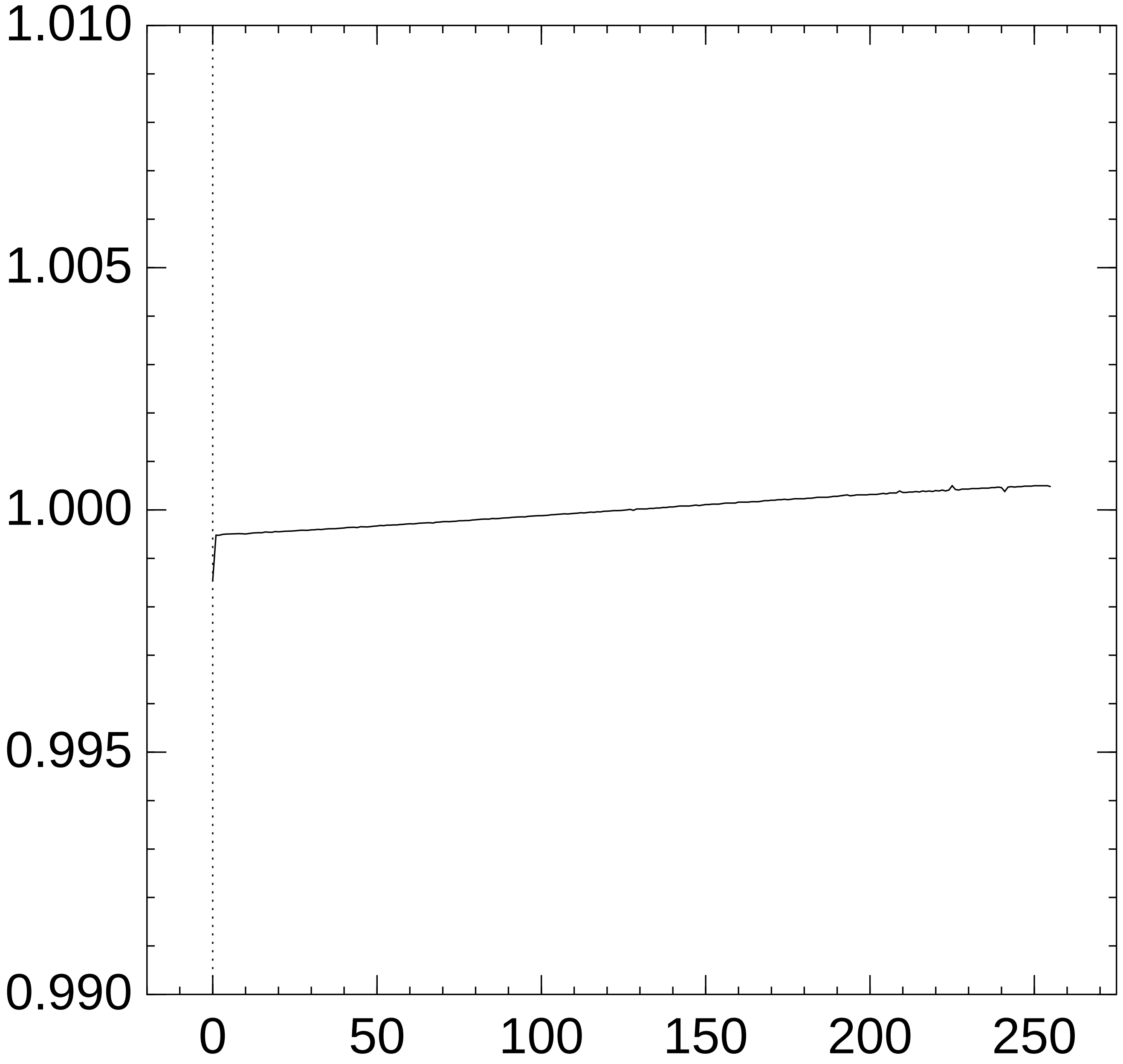


2013 AlFardan–Bernstein–Paterson–Poettering–Schuld
success probability (256 tria
for recovering byte x of plai
from 2^{24} ciphertexts (with
no prior plaintext knowledge

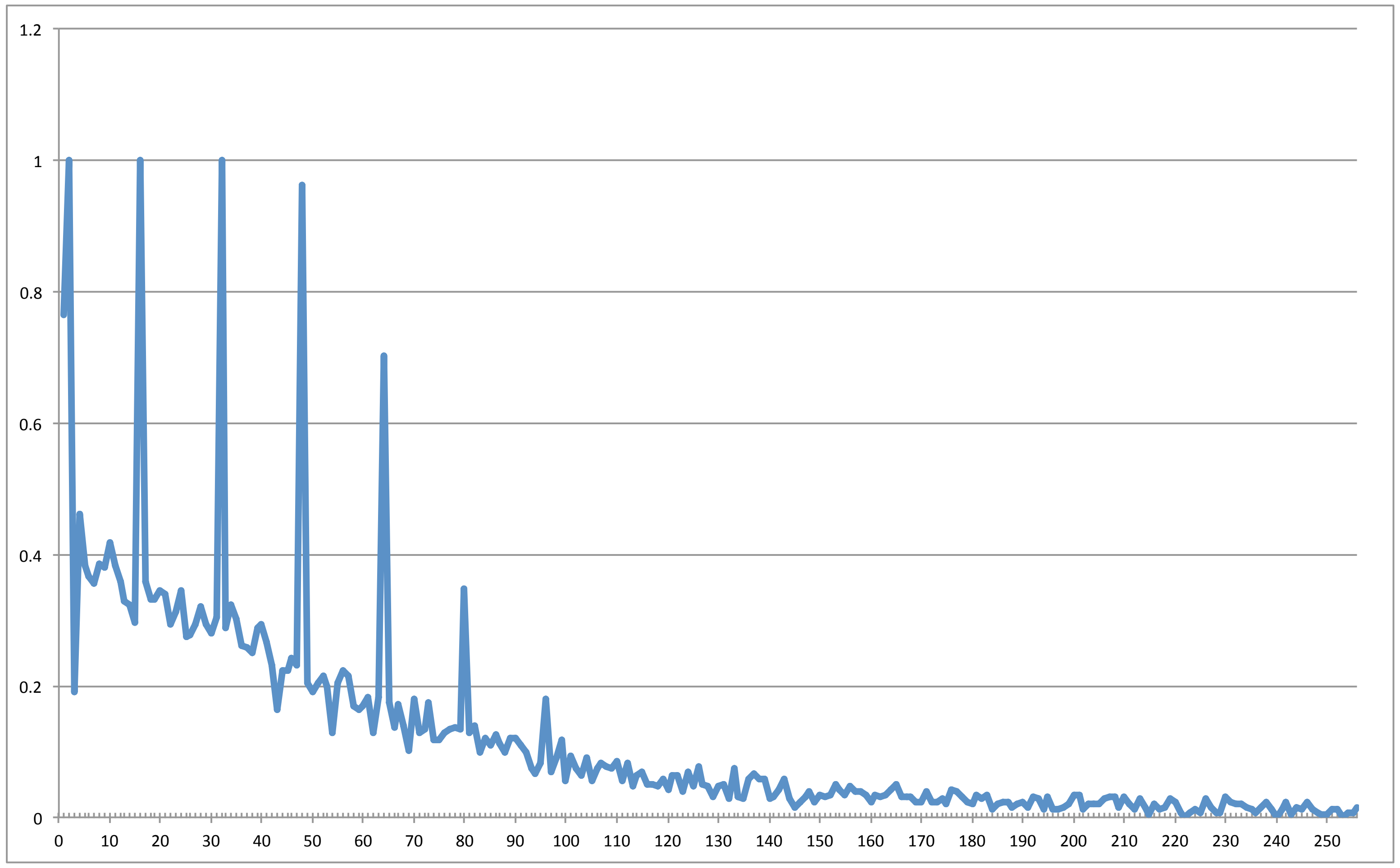


Later bytes: see paper.

Graph of $256\,\Pr[z_{256} = x]$:

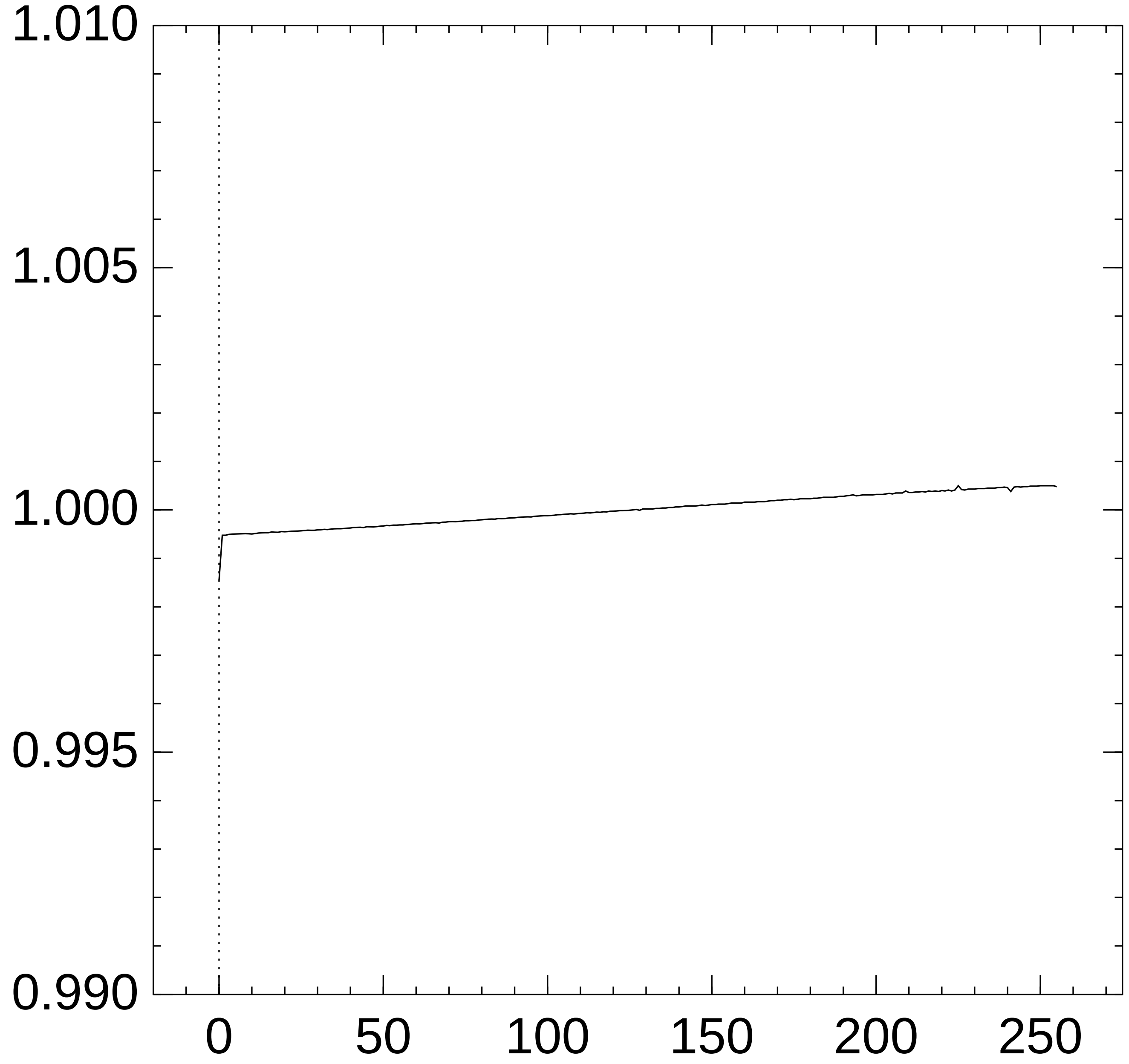


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{24} ciphertexts (with no prior plaintext knowledge):

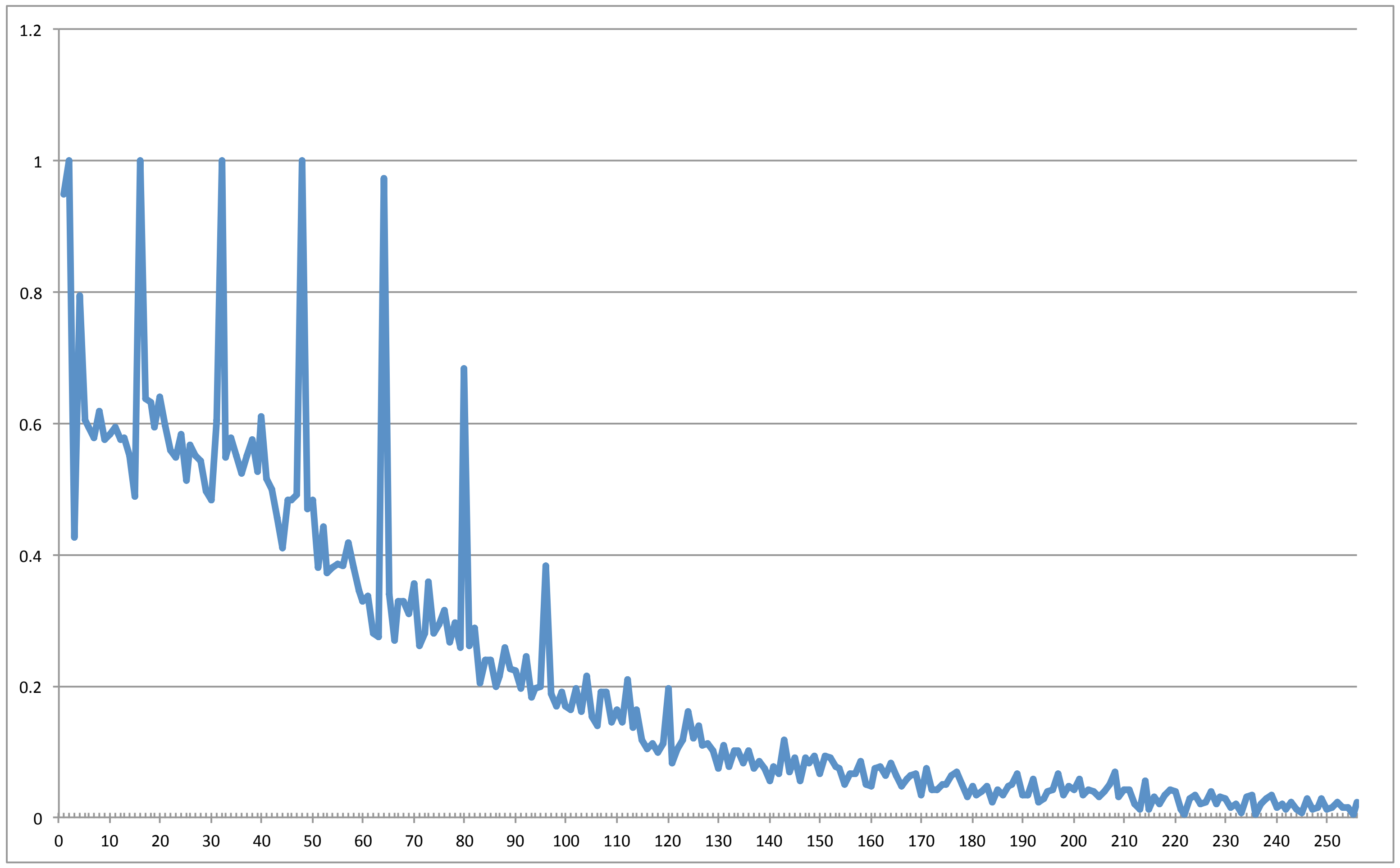


Later bytes: see paper.

Graph of $256\,\Pr[z_{256} = x]$:

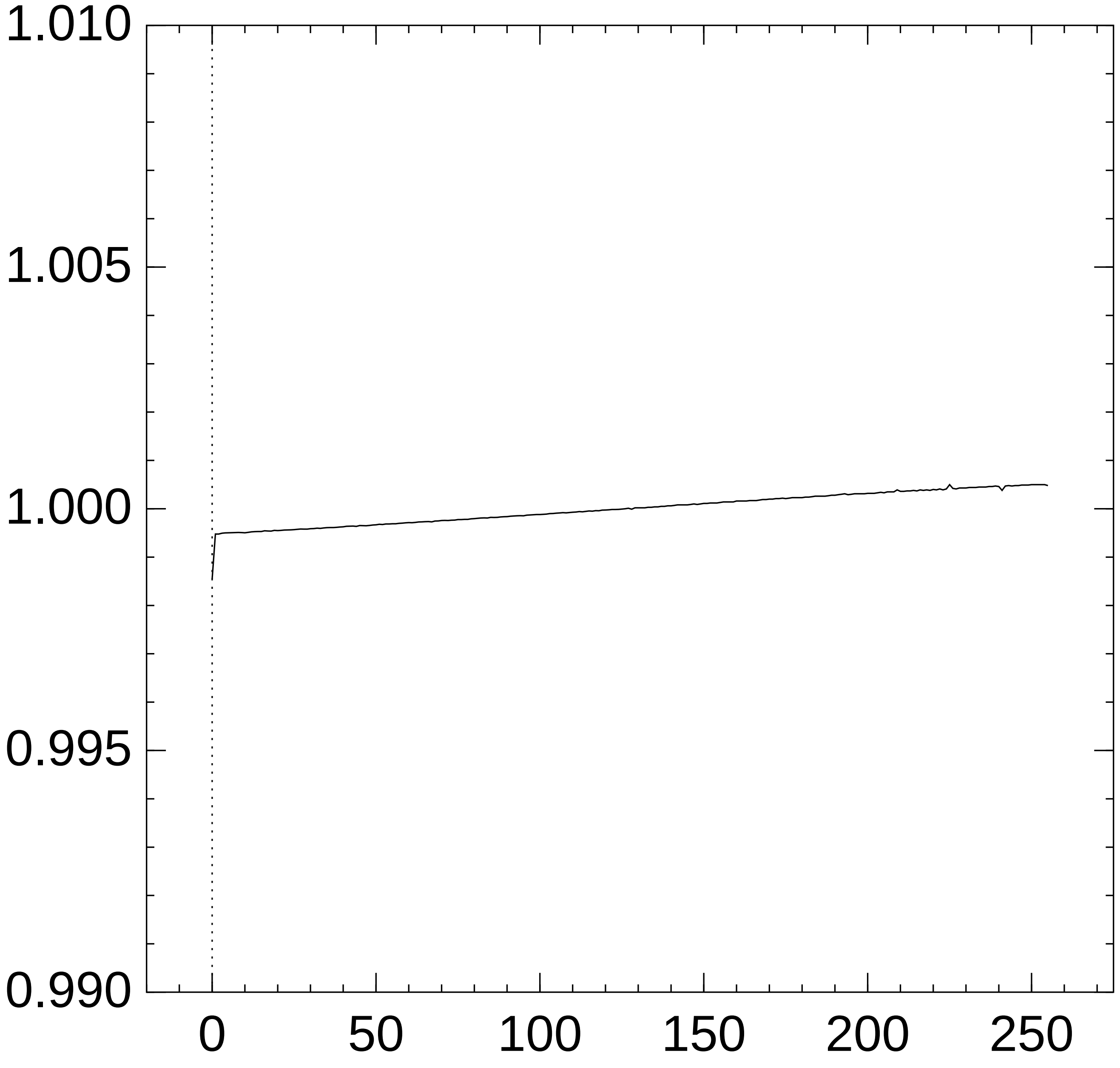


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{25} ciphertexts (with no prior plaintext knowledge):

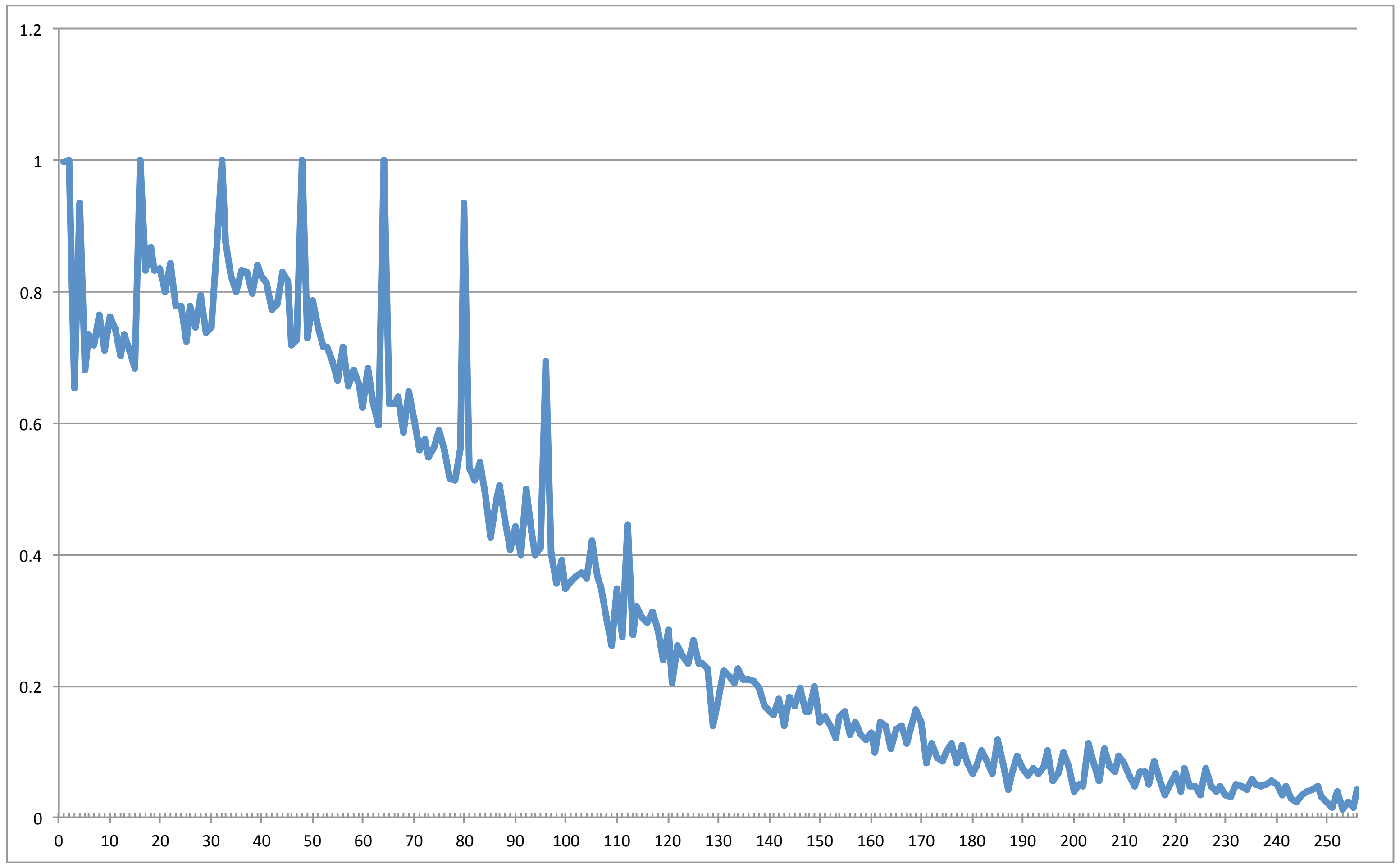


Later bytes: see paper.

Graph of $256 \Pr[z_{256} = x]$:

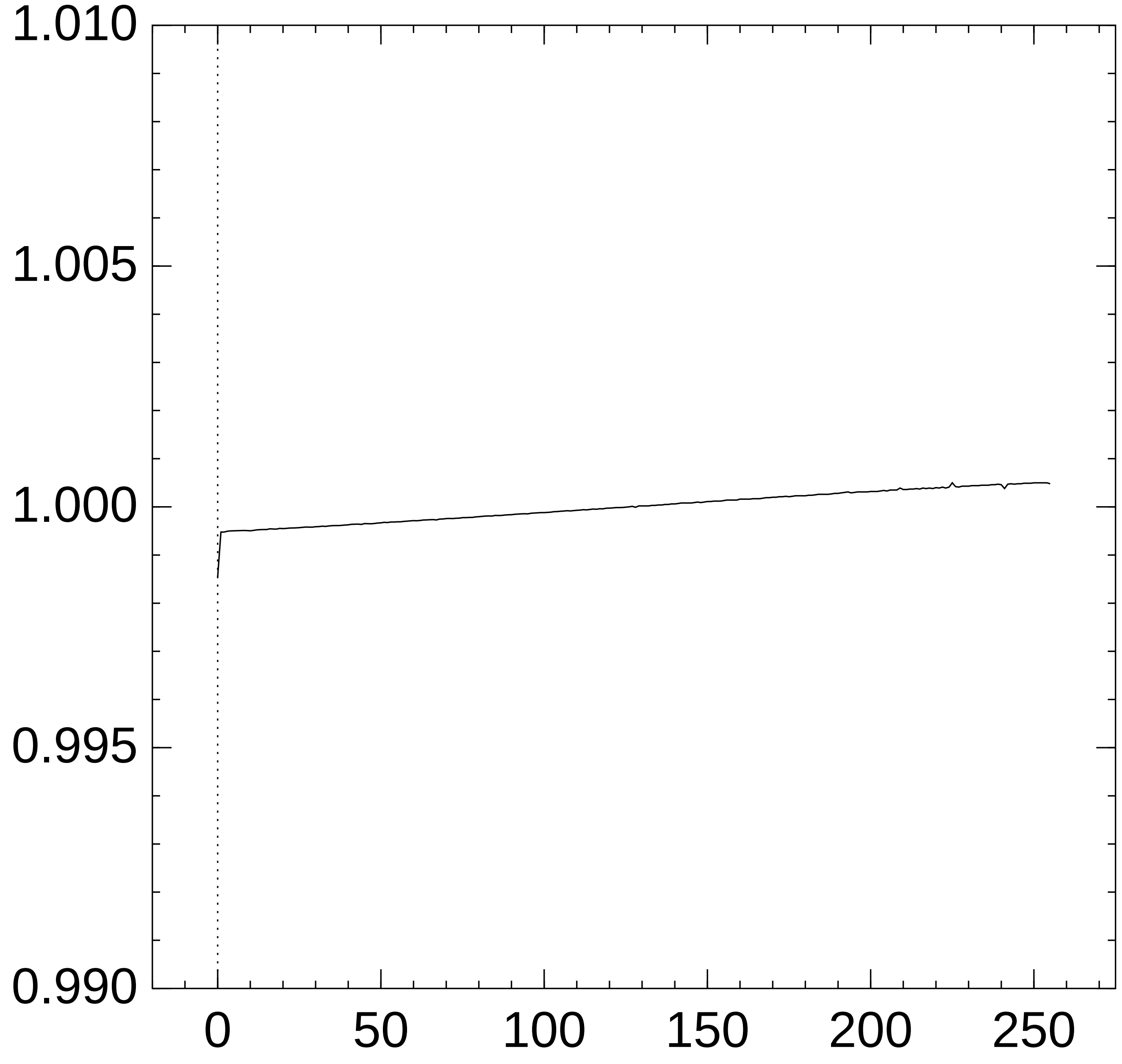


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{26} ciphertexts (with no prior plaintext knowledge):

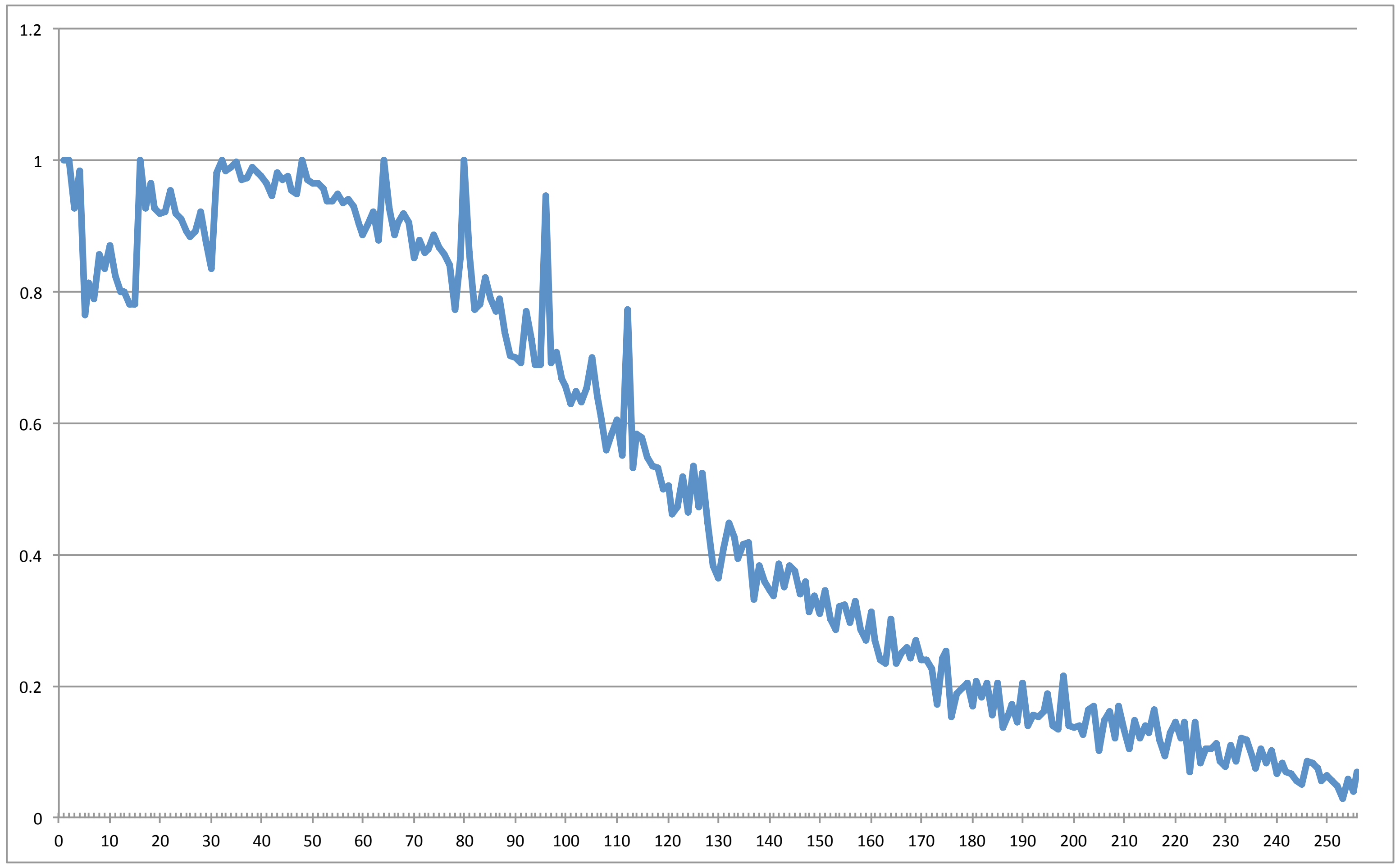


Later bytes: see paper.

Graph of $256 \Pr[z_{256} = x]$:

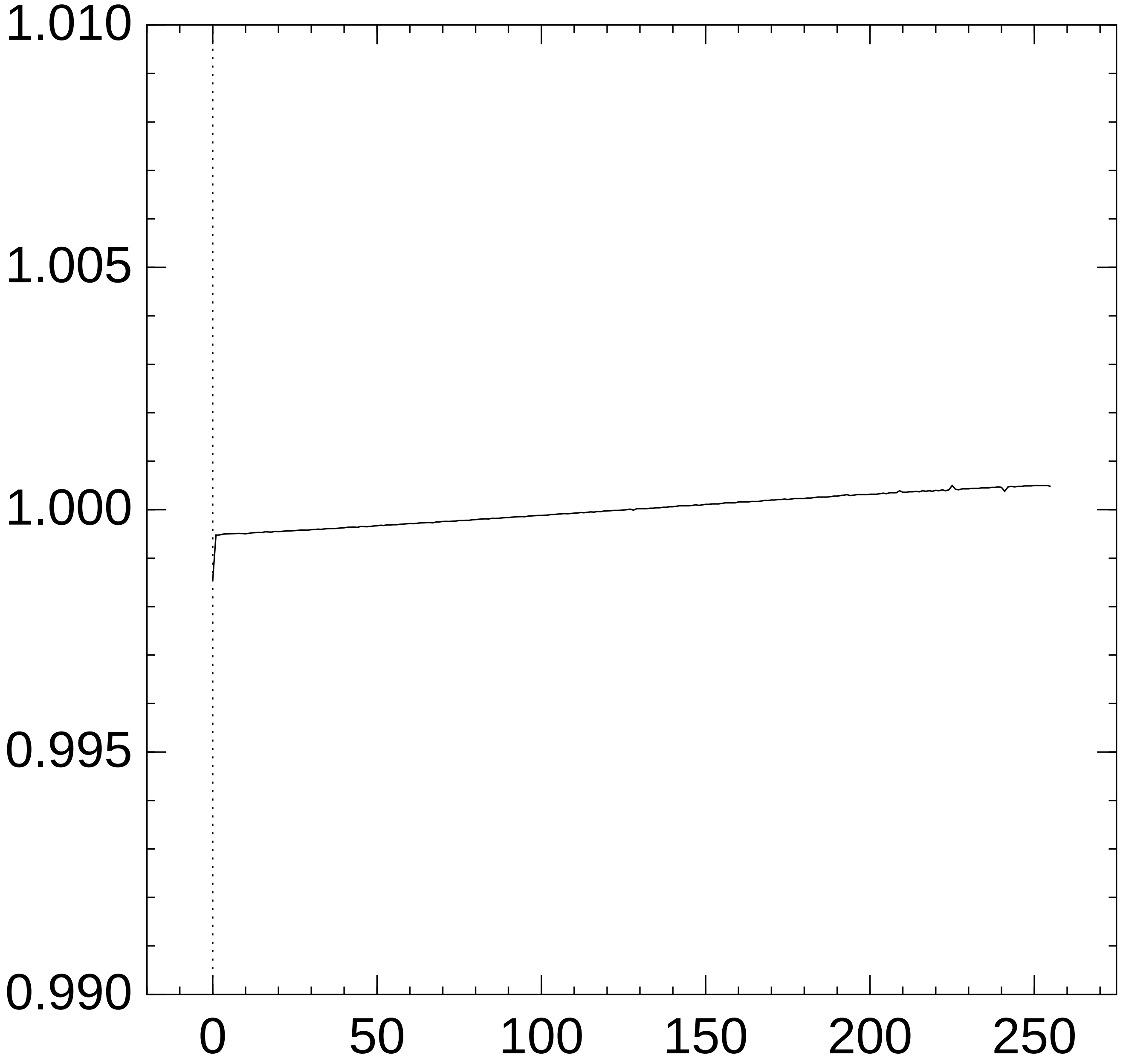


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{27} ciphertexts (with no prior plaintext knowledge):

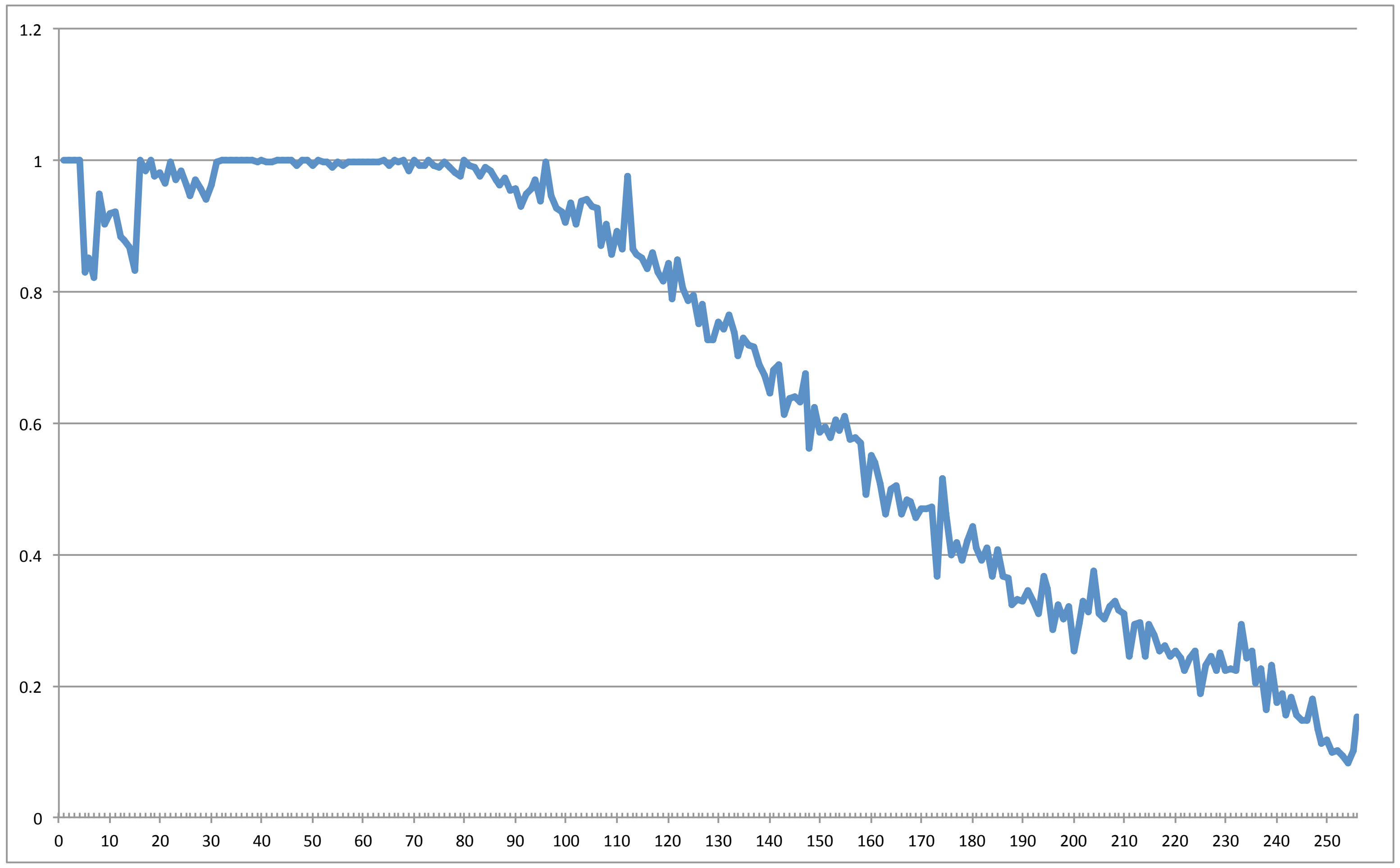


Later bytes: see paper.

Graph of $256 \Pr[z_{256} = x]$:

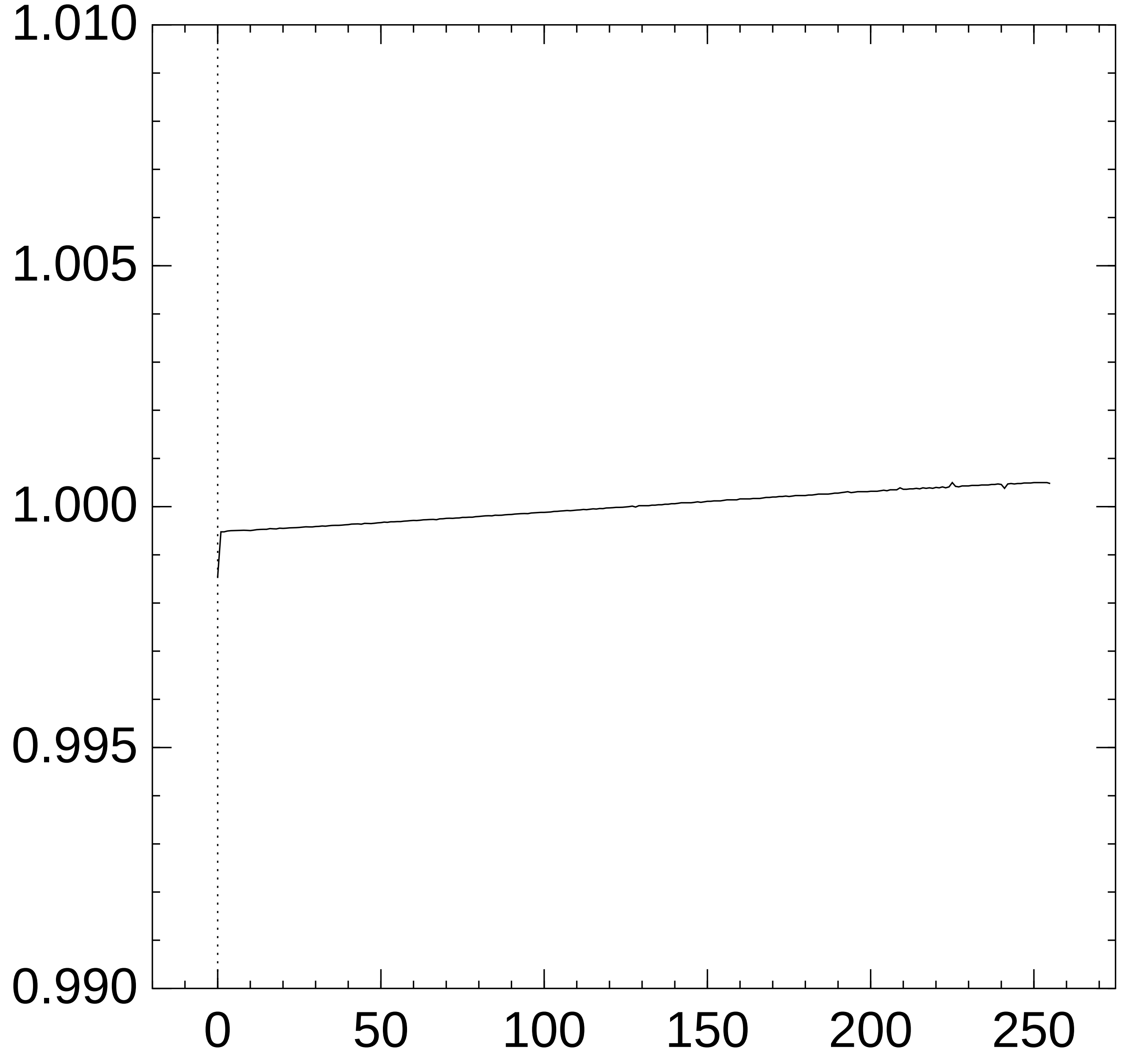


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{28} ciphertexts (with no prior plaintext knowledge):

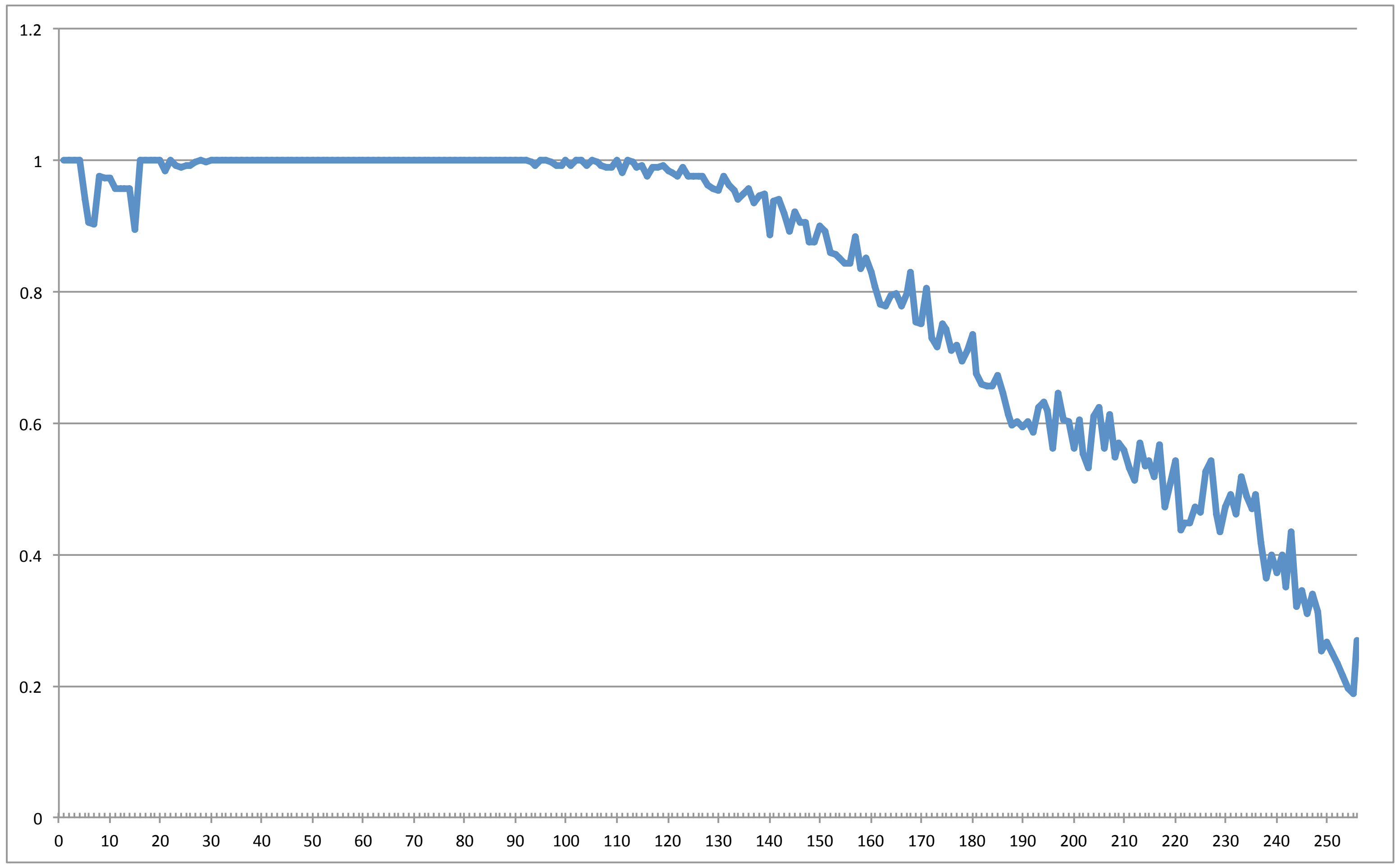


Later bytes: see paper.

Graph of $256\Pr[z_{256} = x]$:

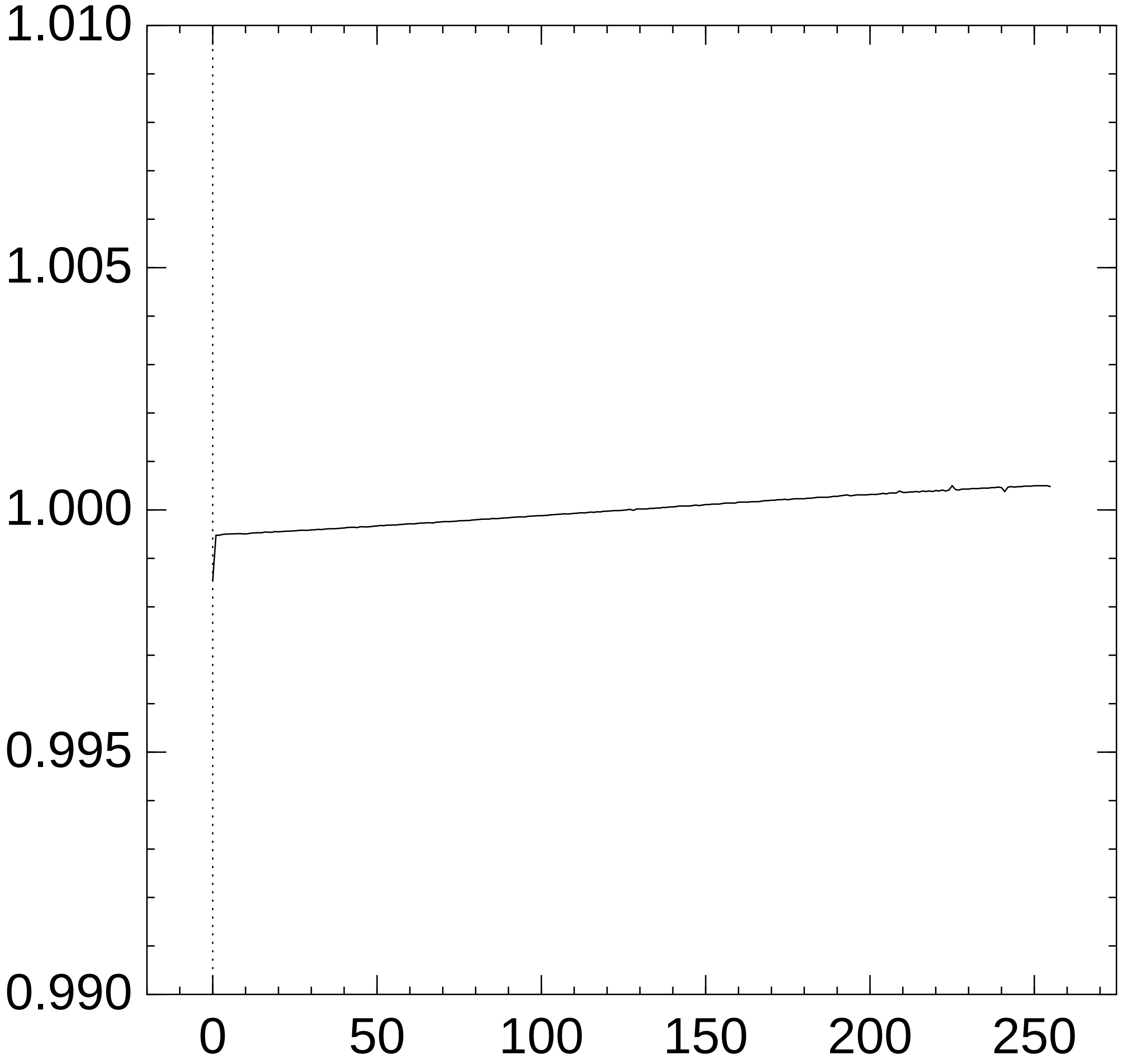


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{29} ciphertexts (with no prior plaintext knowledge):

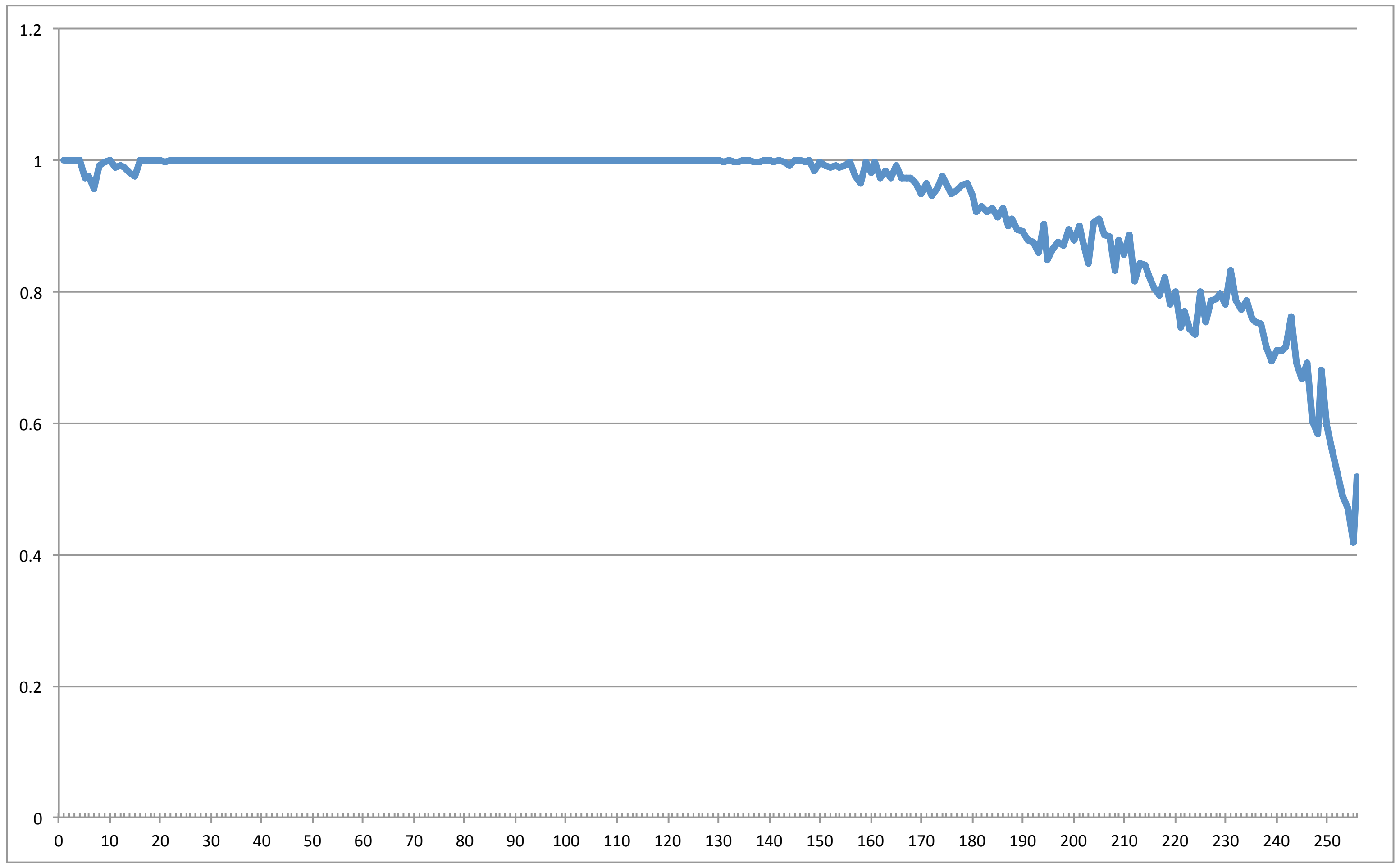


Later bytes: see paper.

Graph of $256\,\Pr[z_{256} = x]$:

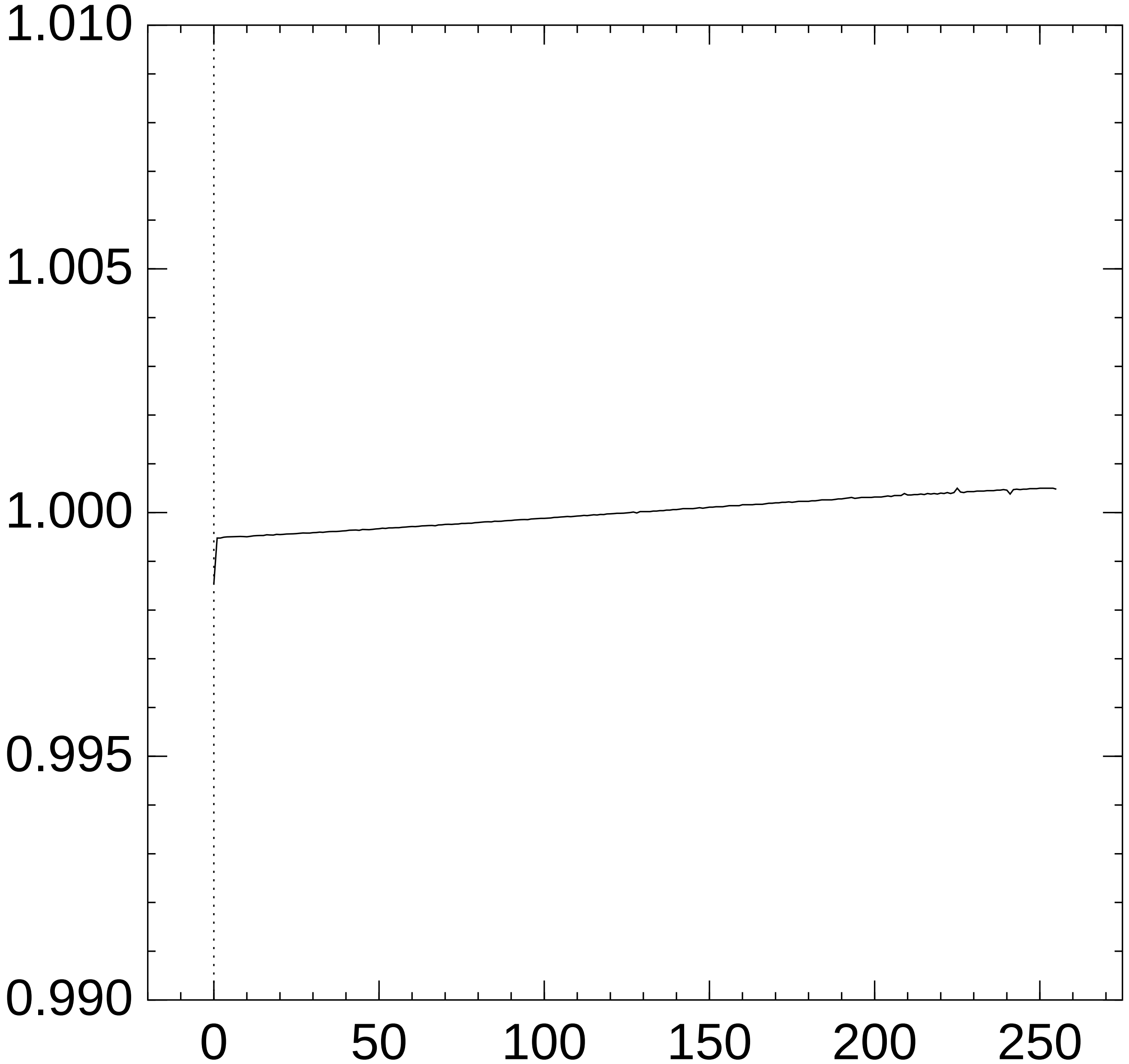


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{30} ciphertexts (with no prior plaintext knowledge):

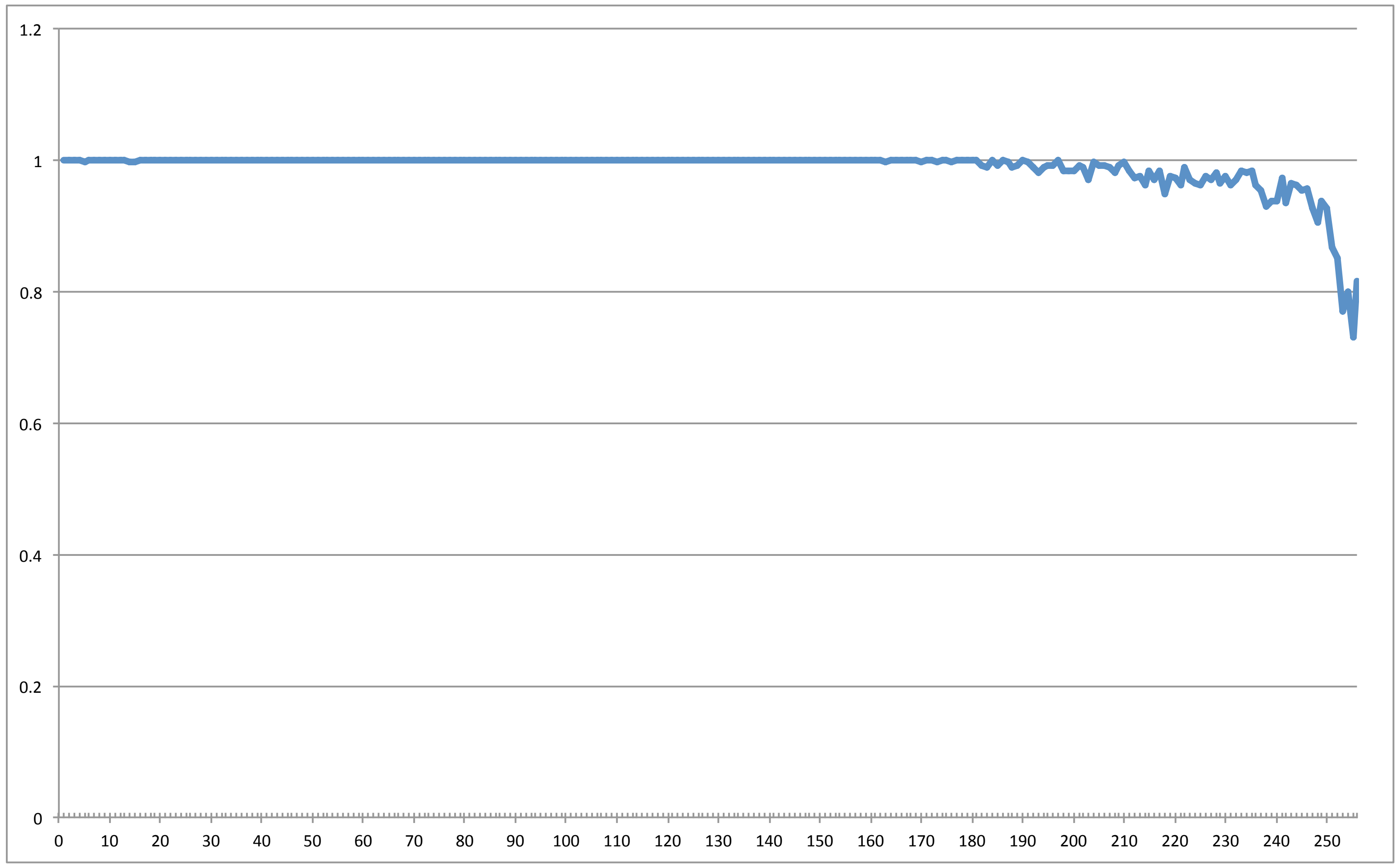


Later bytes: see paper.

Graph of $256 \Pr[z_{256} = x]$:

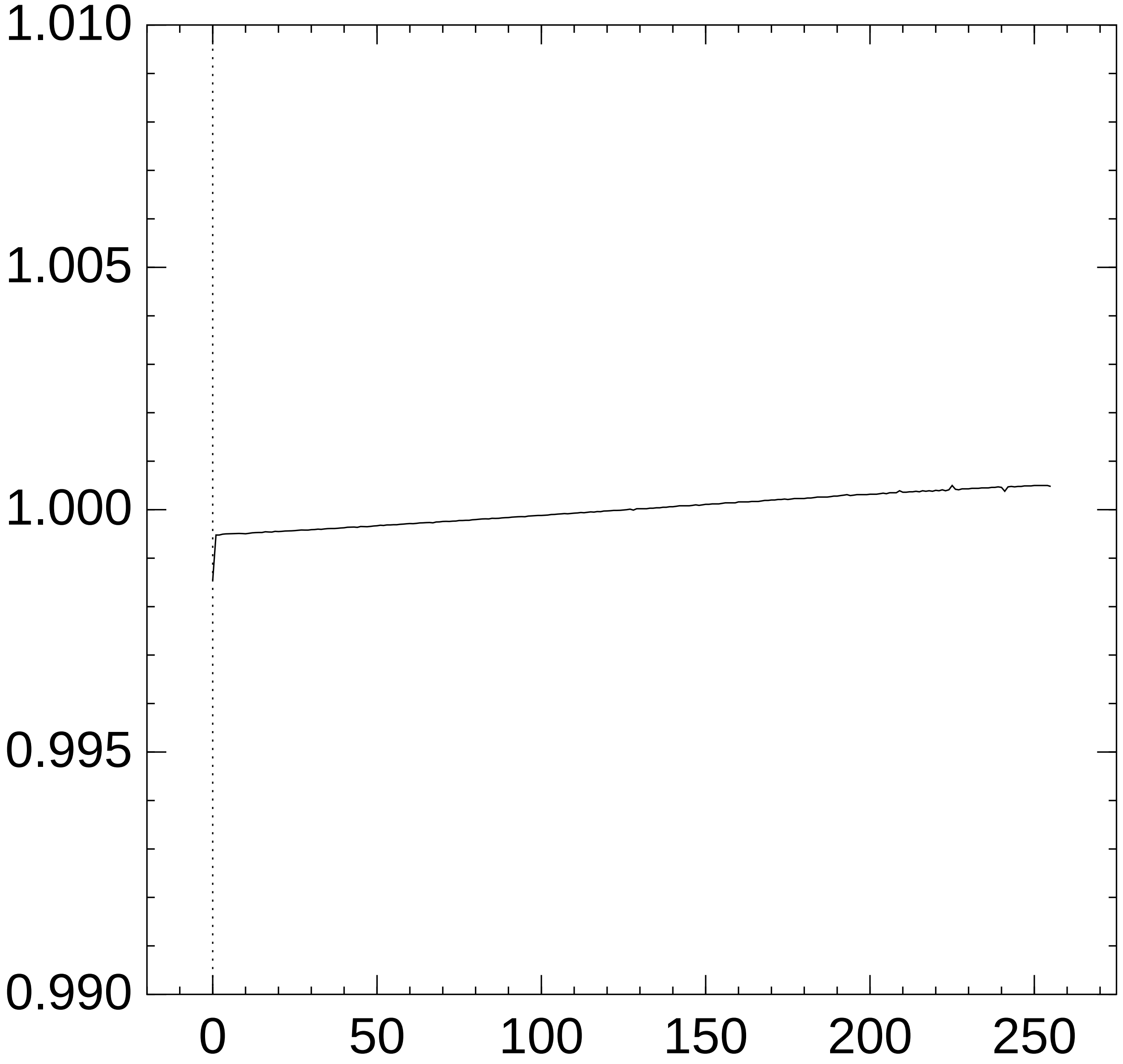


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{31} ciphertexts (with no prior plaintext knowledge):

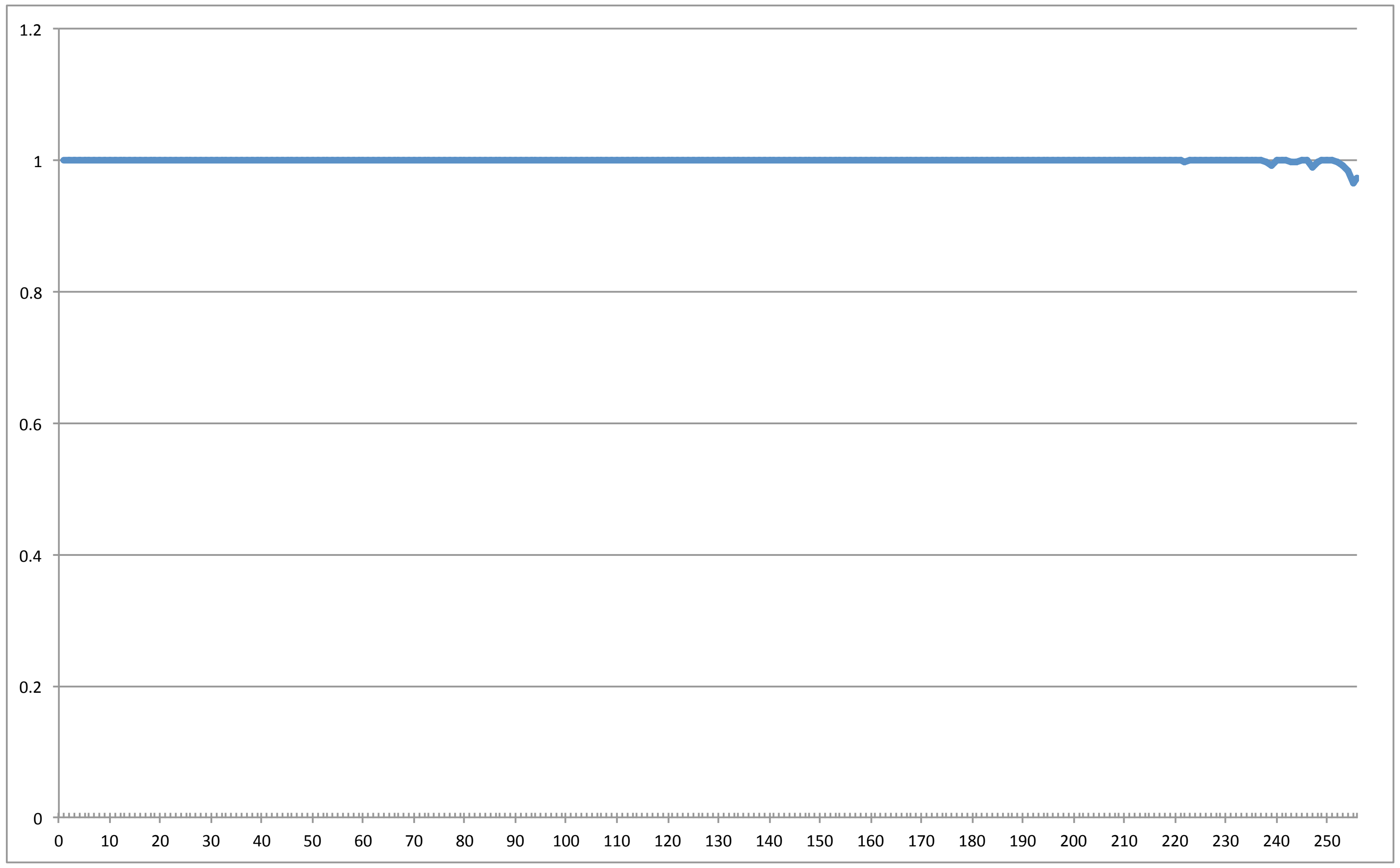


Later bytes: see paper.

Graph of $256\,\Pr[z_{256} = x]$:

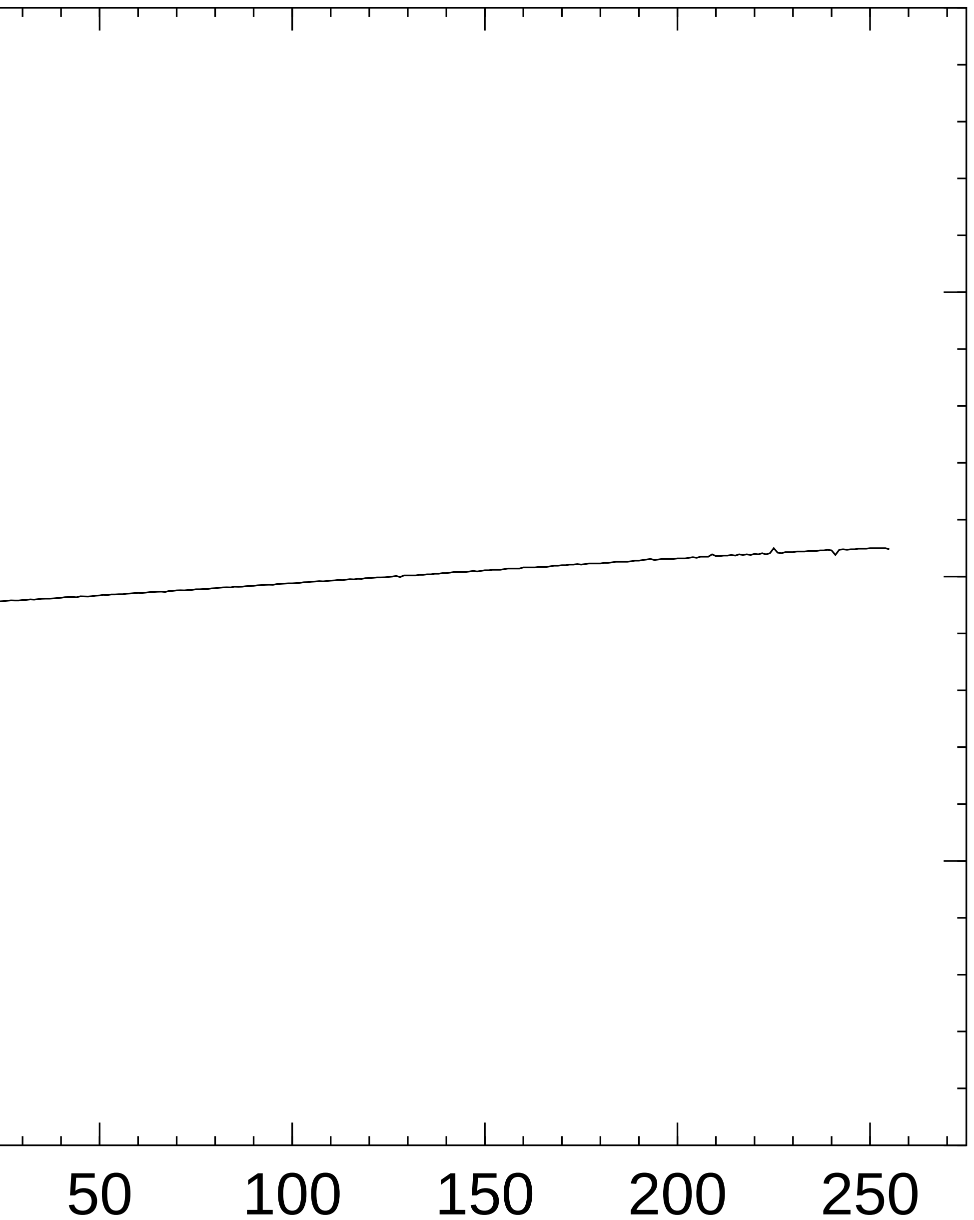


2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{32} ciphertexts (with no prior plaintext knowledge):

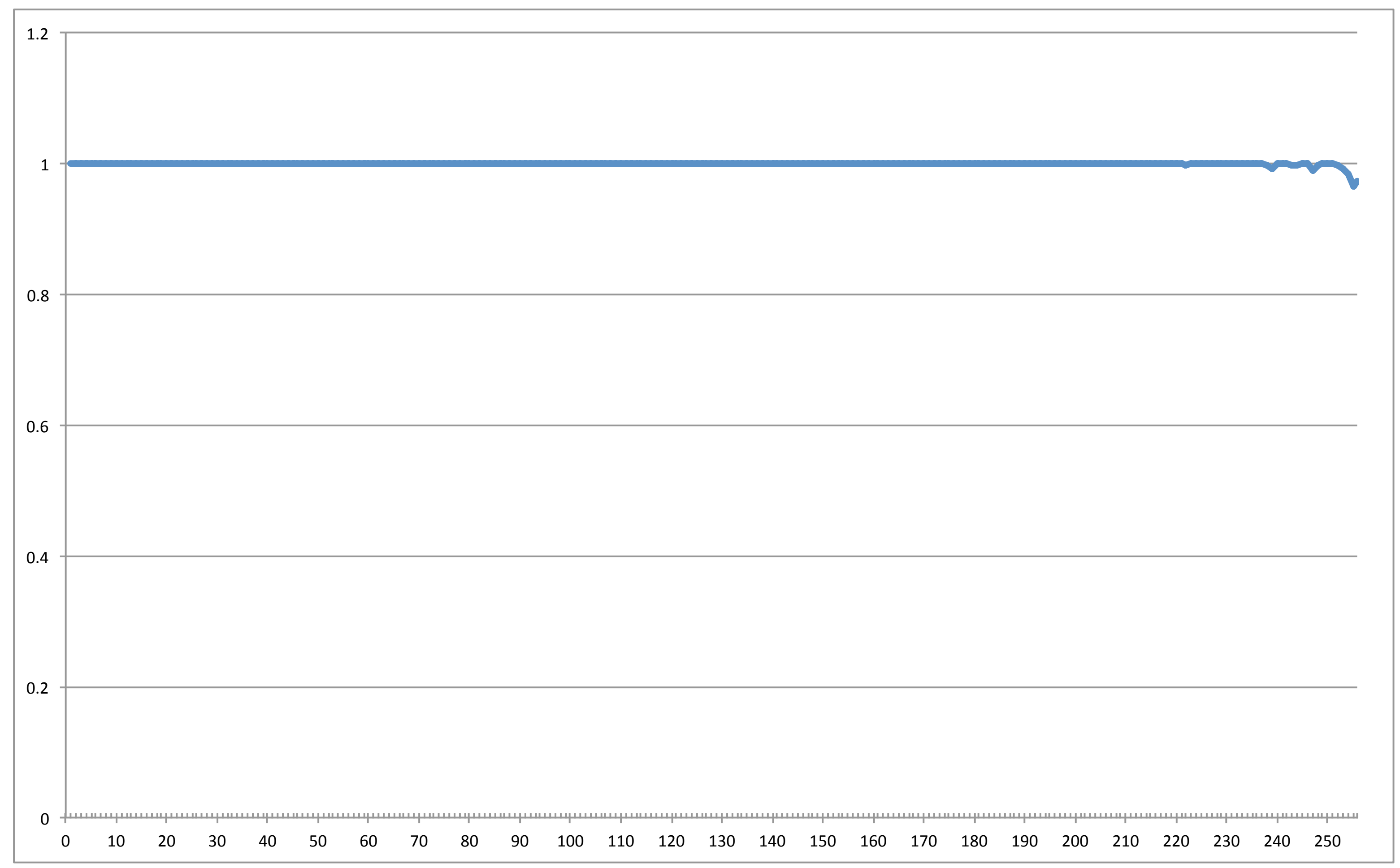


Later bytes: see paper.

f 256 Pr$[z_{256} = x]$:



2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{32} ciphertexts (with no prior plaintext knowledge):



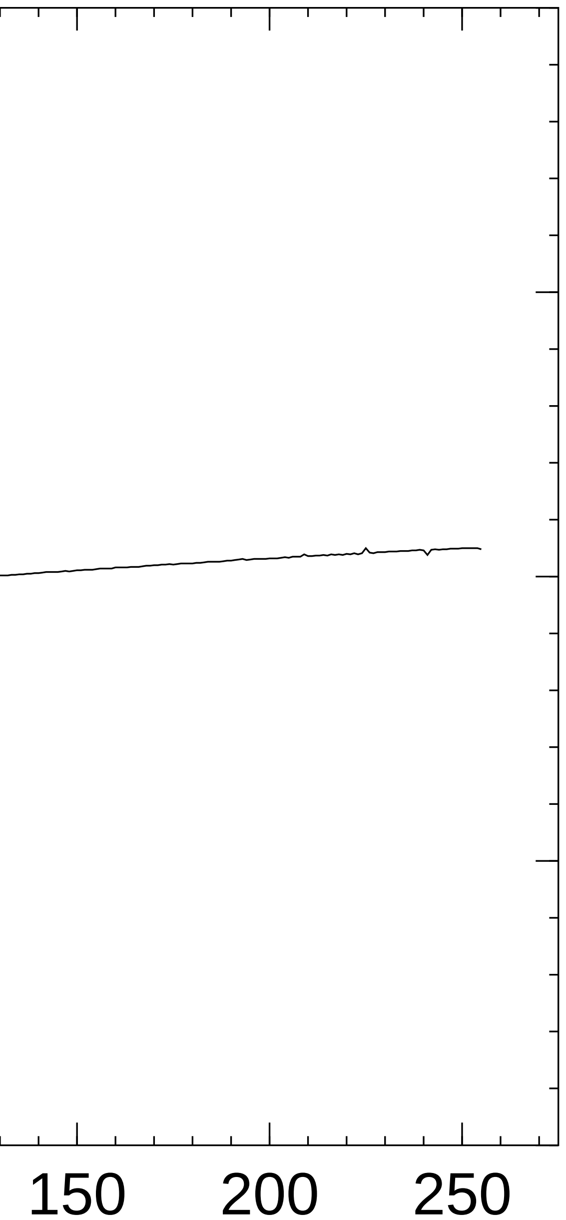
Later bytes: see paper.

Why doe

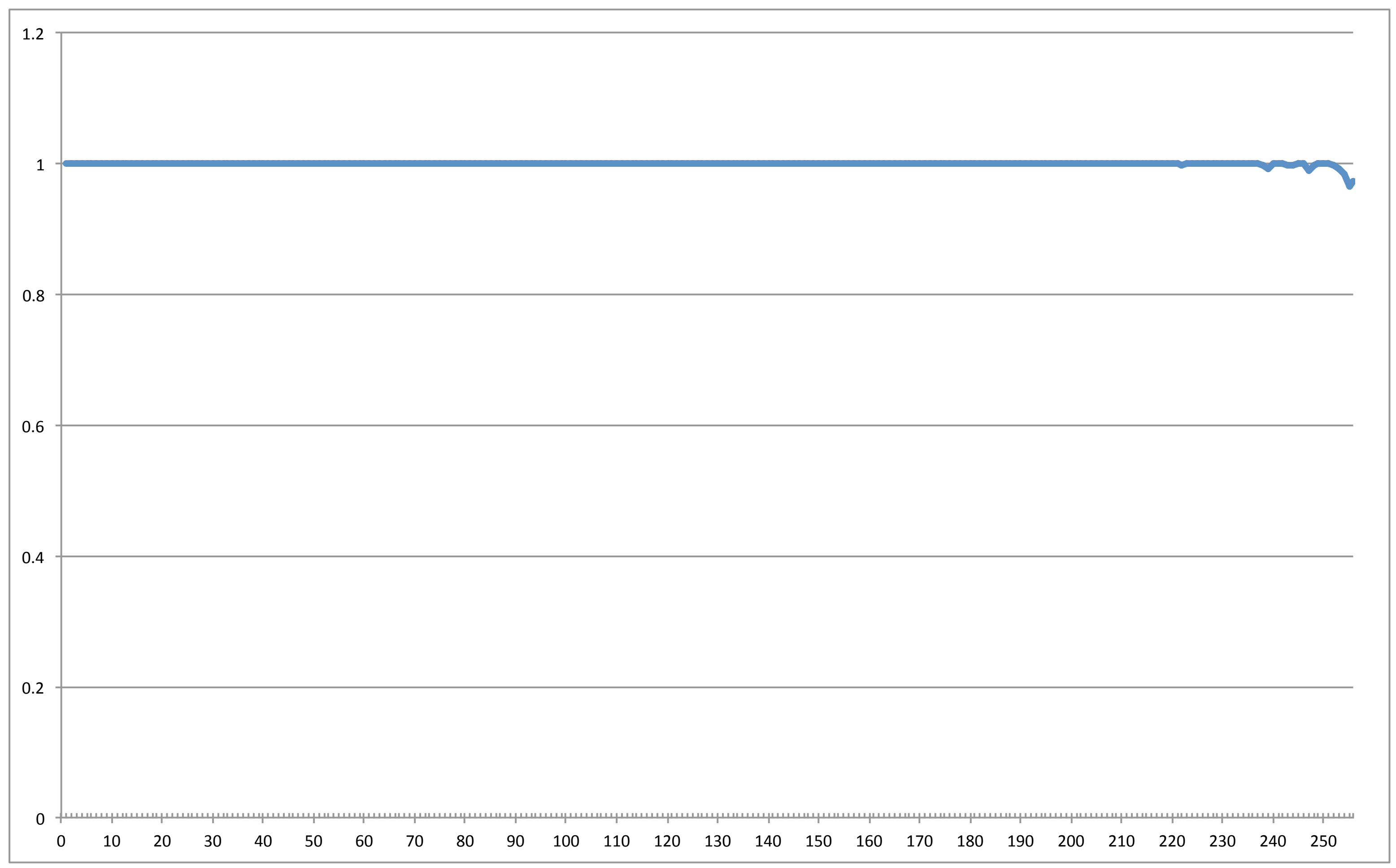
For year
AES-GC
various s

We simp
software
choosing

$256 = x]$:



2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{32} ciphertexts (with
no prior plaintext knowledge):

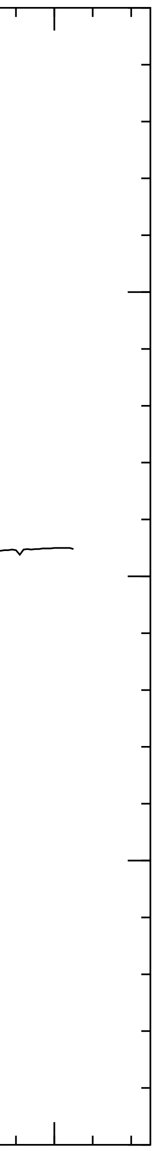


Later bytes: see paper.

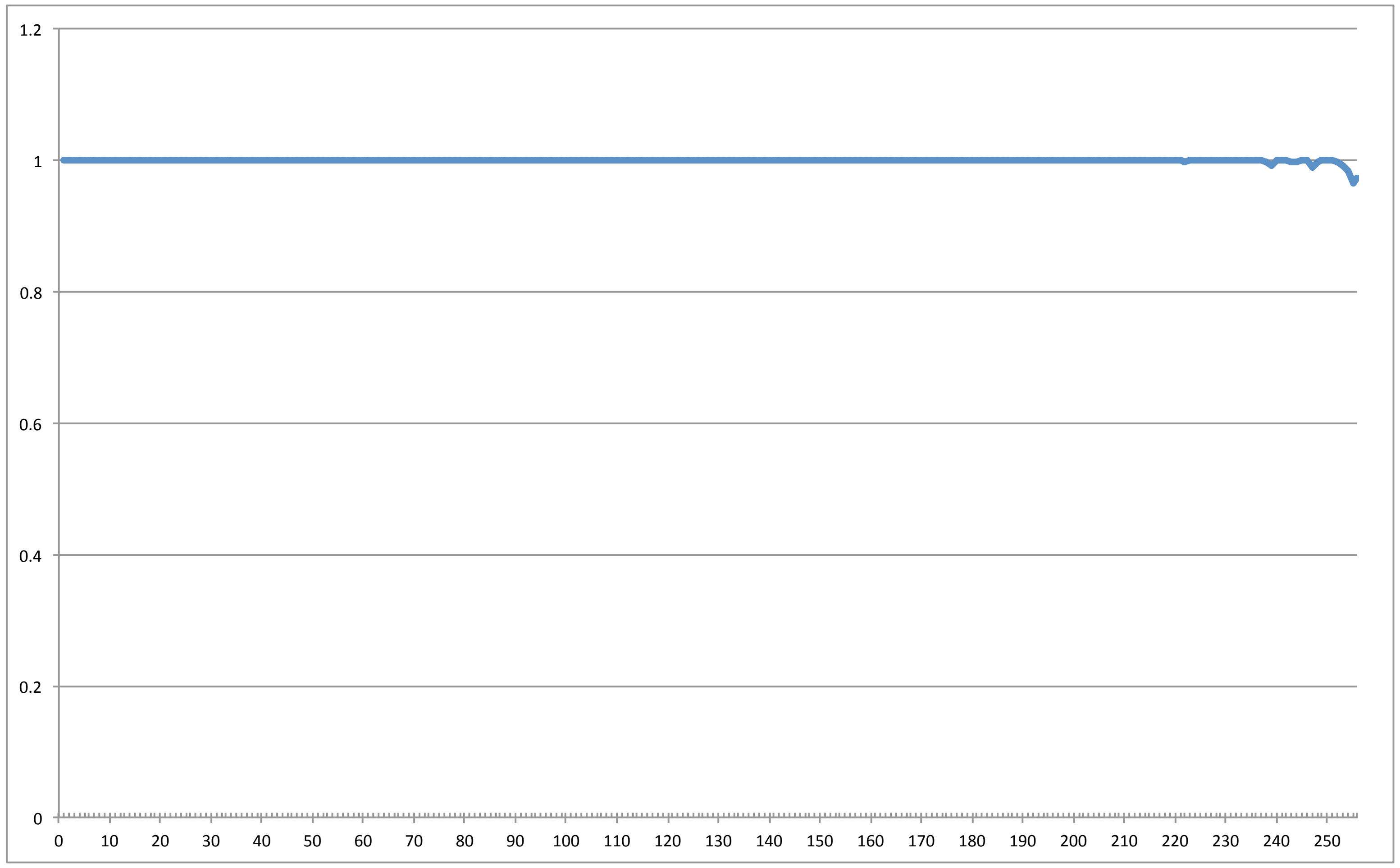
Why does this hap

For years we've ha
AES-GCM; defens
various side-chann

We simply have to
software and hardv
choosing crypto pr



2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{32} ciphertexts (with
no prior plaintext knowledge):



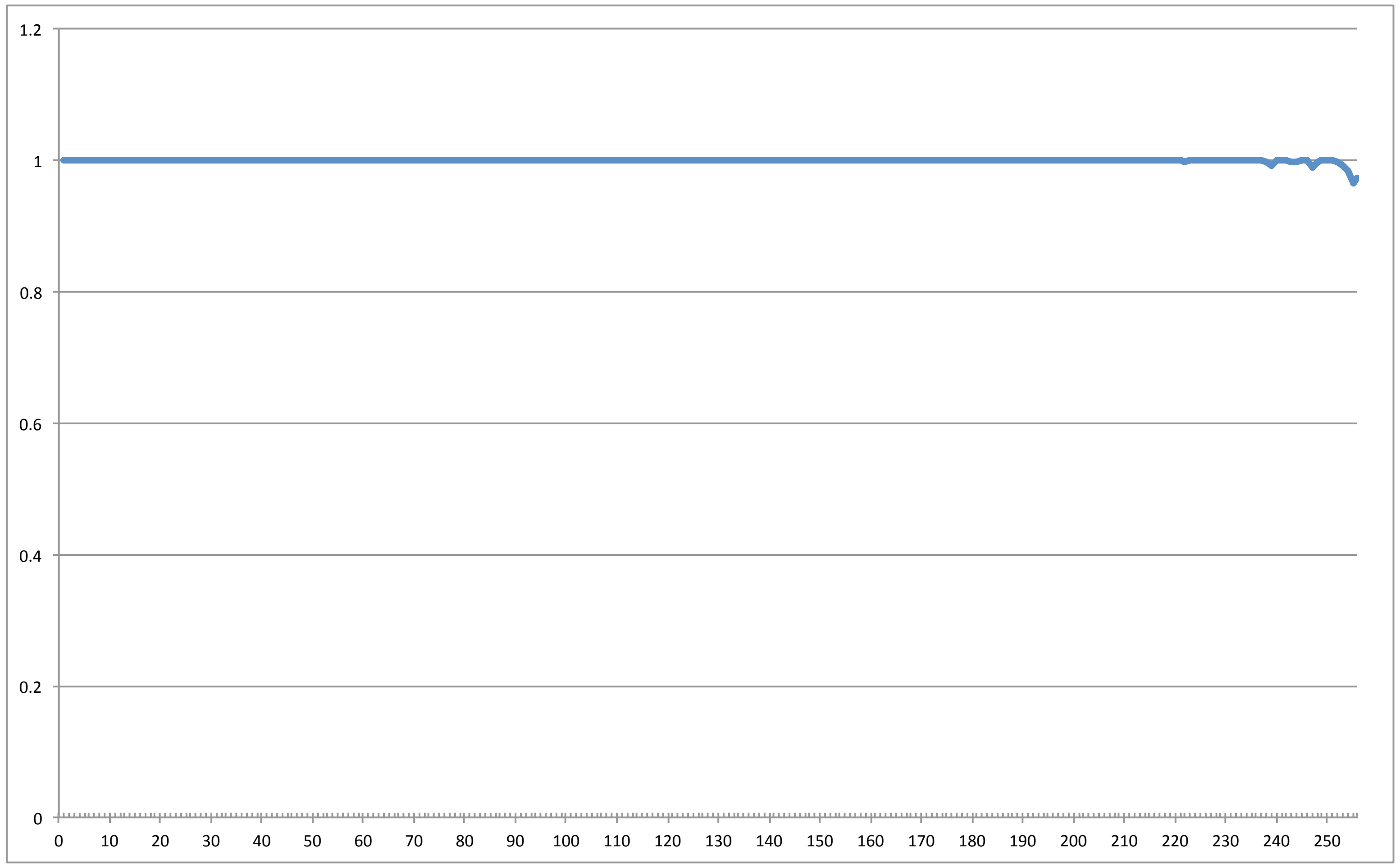
Later bytes: see paper.

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks

We simply have to educate
software and hardware engin
choosing crypto primitives,

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{32} ciphertexts (with no prior plaintext knowledge):



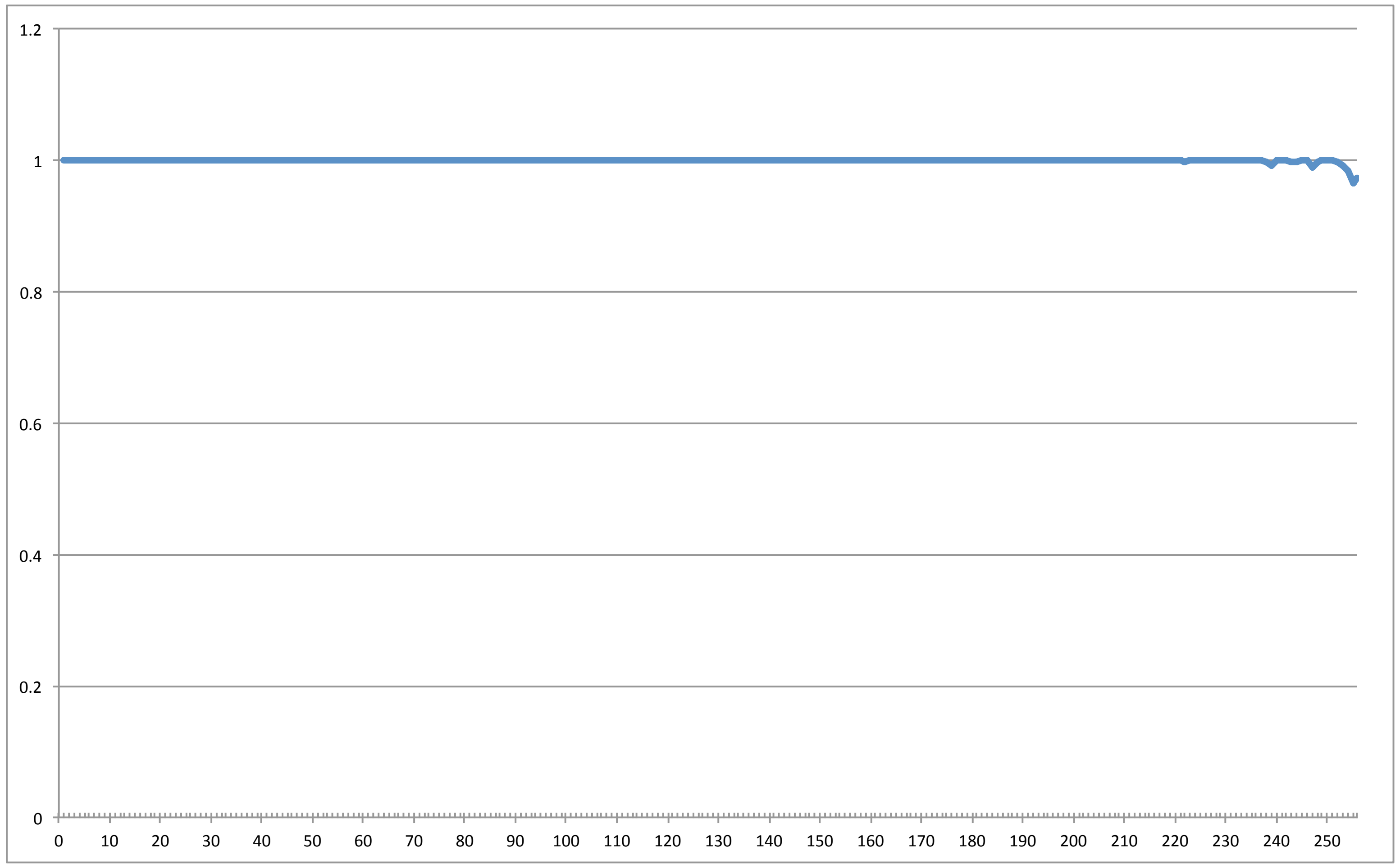
Later bytes: see paper.

Why does this happen?

For years we've had AES; AES-GCM; defenses against various side-channel attacks.

We simply have to educate the software and hardware engineers choosing crypto primitives, right?

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{32} ciphertexts (with no prior plaintext knowledge):



Later bytes: see paper.

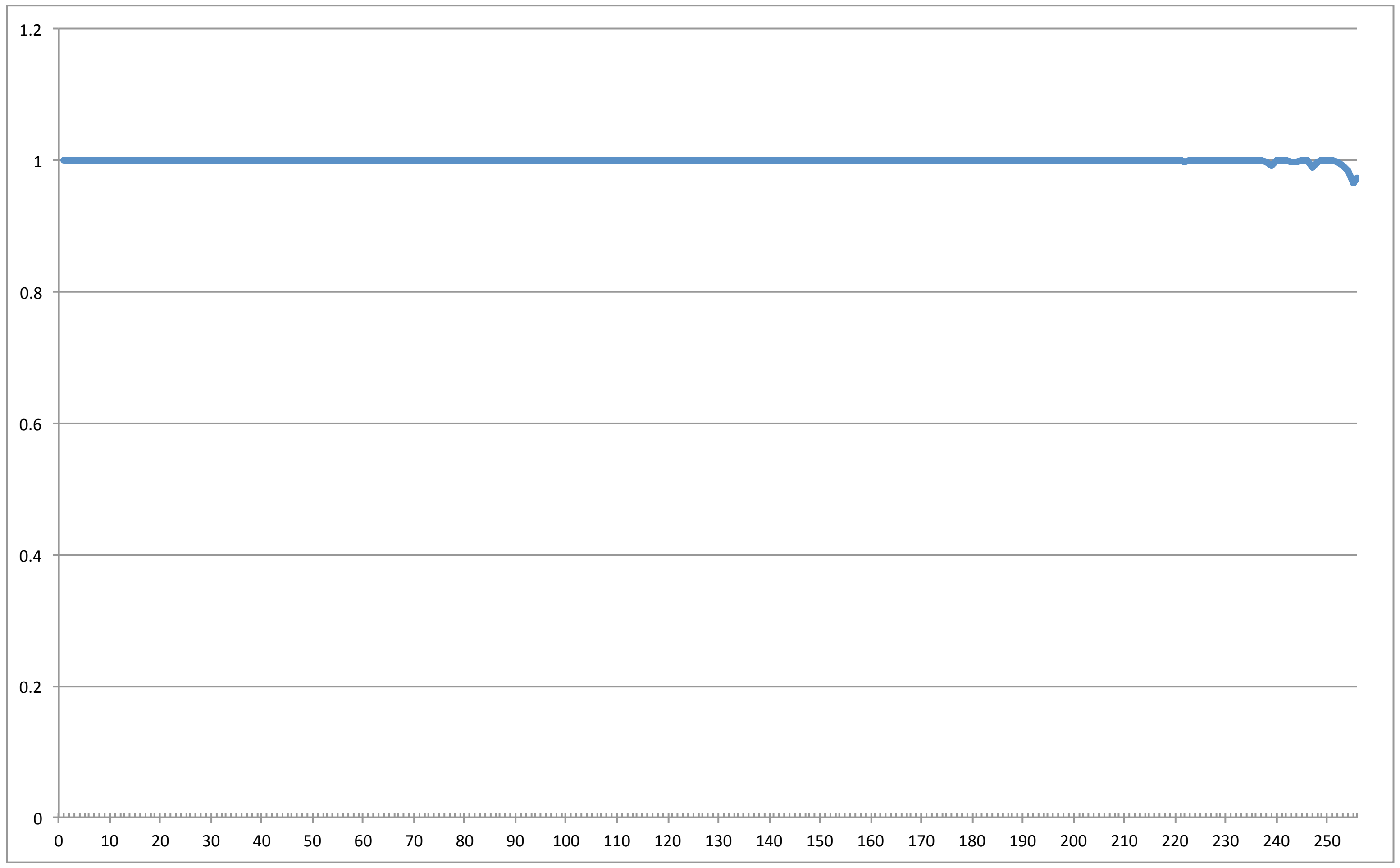
Why does this happen?

For years we've had AES; AES-GCM; defenses against various side-channel attacks.

We simply have to educate the software and hardware engineers choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do what the users need?

2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt success probability (256 trials) for recovering byte x of plaintext from 2^{32} ciphertexts (with no prior plaintext knowledge):



Later bytes: see paper.

<u>Why does this happen?</u>

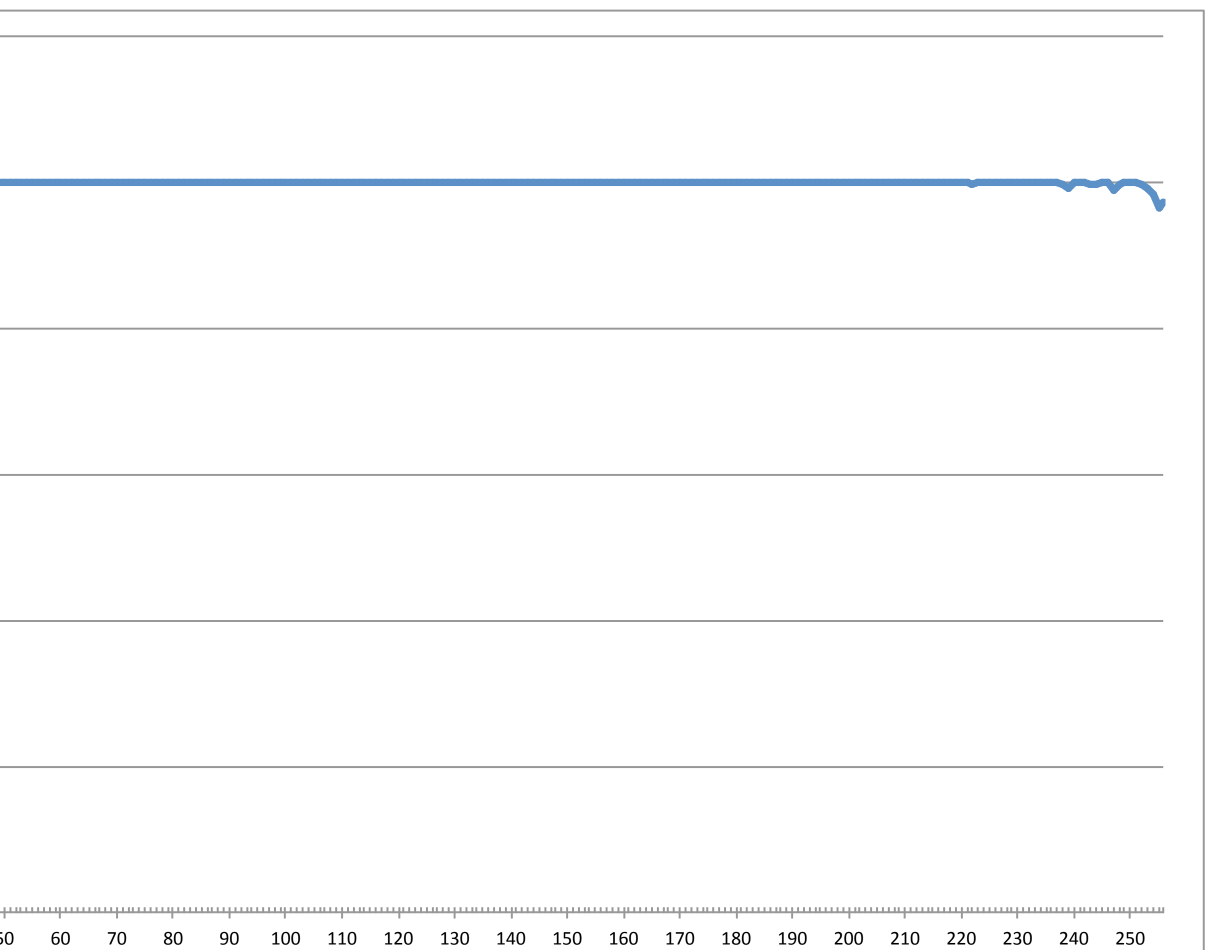
For years we've had AES; AES-GCM; defenses against various side-channel attacks.

We simply have to educate the software and hardware engineers choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do what the users need?

Often it doesn't.
Most obvious issue: performance.

Fardan–Bernstein–
n–Poettering–Schuldt
probability (256 trials)
vering byte x of plaintext
^{2} ciphertexts (with
plaintext knowledge):



tes: see paper.

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do
what the users need?

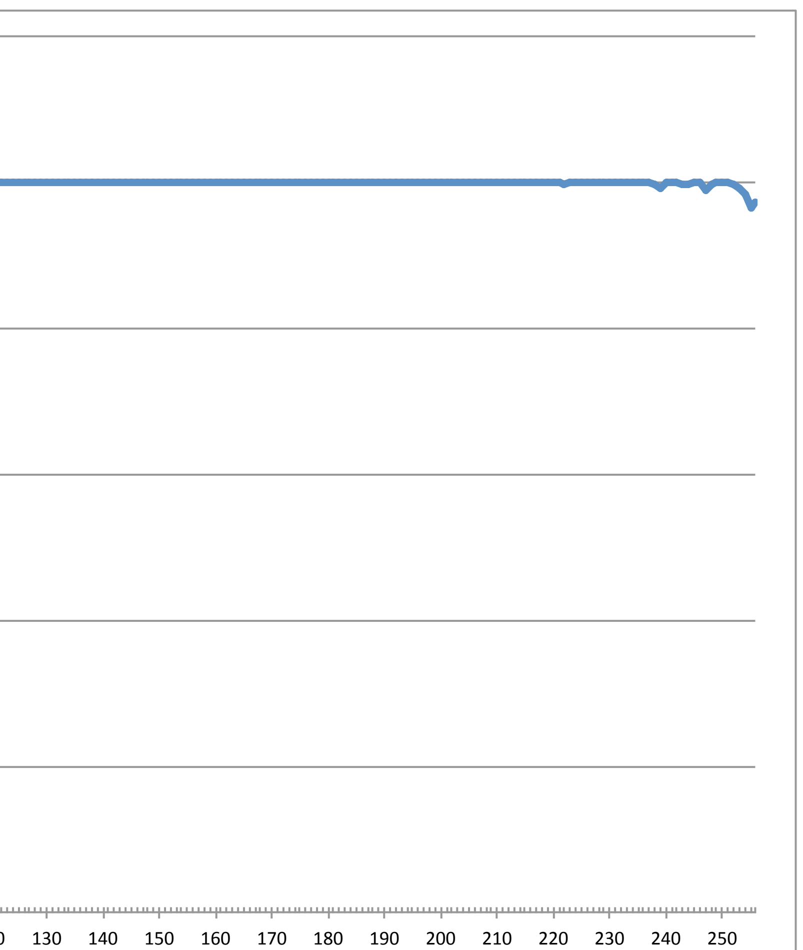
Often it doesn't.
Most obvious issue: performance.

e.g. 200
RC4 is it
and extr
random
likely to
choice fo
embedde

e.g. Ope
based im
for speed
leaking r
2012 We

e.g. RFI

ernstein–
ng–Schuldt
y (256 trials)
e x of plaintext
xts (with
knowledge):



aper.

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do
what the users need?

Often it doesn't.
Most obvious issue: performance.

e.g. 2001 Rivest:
RC4 is its exceptio
and extremely effic
random generator.
likely to remain th
choice for many ap
embedded systems

e.g. OpenSSL still
based implementat
for speed on most
leaking many key
2012 Weiß–Heinz–

e.g. RFIDs need sm

t
ls)
ntext

e):

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do
what the users need?

Often it doesn't.
Most obvious issue: performance.

e.g. 2001 Rivest: "The 'hea
RC4 is its exceptionally simp
and extremely efficient pseu
random generator. . . . RC4
likely to remain the algorith
choice for many application
embedded systems."

e.g. OpenSSL still uses table
based implementations of A
for speed on most CPUs,
leaking many key bits; see, e
2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small cipher

Why does this happen?

For years we've had AES; AES-GCM; defenses against various side-channel attacks.

We simply have to educate the software and hardware engineers choosing crypto primitives, right?

Maybe, maybe not.
Does AES-GCM actually do what the users need?

Often it doesn't.
Most obvious issue: performance.

e.g. 2001 Rivest: "The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., 2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small ciphers.

es this happen?

s we've had AES;
M; defenses against
side-channel attacks.

ly have to educate the
and hardware engineers
crypto primitives, right?

maybe not.
ES-GCM actually do
e users need?

doesn't.
vious issue: performance.

e.g. 2001 Rivest: "The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., 2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small ciphers.

Major re
achieve
than AE
without

Fit into
low area
sometim
minimiz
minimiz

Many di
ASIC m

Many di
precomp

pen?

d AES;
es against
el attacks.

educate the
ware engineers
imitives, right?

t.
ctually do
ed?

e: performance.

e.g. 2001 Rivest: "The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., 2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small ciphers.

Major research dir
achieve better per
than AES-GCM
without sacrificing

Fit into low power
low area (square m
sometimes low late
minimize area$\times$sec
minimize energy (j

Many different CP
ASIC manufacturi

Many different inp
precomputation po

.

the

neers

right?

nance.

e.g. 2001 Rivest: "The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., 2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small ciphers.

Major research direction: achieve better performance than AES-GCM *without* sacrificing security.

Fit into low power (watts), low area (square micrometer sometimes low latency (seco minimize area×seconds/byte minimize energy (joules)/by

Many different CPUs, FPGA ASIC manufacturing technol

Many different input sizes, precomputation possibilities, e.g. 2001 Rivest: "The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., 2012 Weiß–Heinz–Stumpf.

e.g. RFIDs need small ciphers.

Major research direction: achieve better performance than AES-GCM *without* sacrificing security.

Fit into low power (watts), low area (square micrometers), sometimes low latency (seconds); minimize area×seconds/byte; minimize energy (joules)/byte.

Many different CPUs, FPGAs, ASIC manufacturing technologies.

Many different input sizes, precomputation possibilities, etc.

1 Rivest: "The 'heart' of
ts exceptionally simple
emely efficient pseudo-
generator. . . . RC4 is
remain the algorithm of
or many applications and
ed systems."

enSSL still uses table-
nplementations of AES
d on most CPUs,
many key bits; see, e.g.,
eiß–Heinz–Stumpf.

Ds need small ciphers.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area×seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one
in hardw

Some in
Trivium
are "har
but not

"The 'heart' of
onally simple
cient pseudo-
. . . RC4 is
e algorithm of
pplications and
s."

uses table-
tions of AES
CPUs,
bits; see, e.g.,
-Stumpf.

mall ciphers.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area×seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one design do
in hardware *and* s

Some inspirational
Trivium and Kecca
are "hardware" de
but not bad in sof

rt' of
ble
do-
is
m of
s and

e-
ES

e.g.,

rs.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area×seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples
Trivium and Keccak
are "hardware" designs
but not bad in software.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area$\times$seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area×seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

esearch direction:
better performance
S-GCM
sacrificing security.

low power (watts),
(square micrometers),
es low latency (seconds);
e area×seconds/byte;
e energy (joules)/byte.

fferent CPUs, FPGAs,
anufacturing technologies.

fferent input sizes,
utation possibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

Another
achieve
than AE
without

Typical
are start
Limit im
Use larg

ection:
formance

; security.

r (watts),
micrometers),
ency (seconds);
conds/byte;
joules)/byte.

PUs, FPGAs,
ng technologies.

out sizes,
ossibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

Another major res
achieve better secu
than AES-GCM
without sacrificing

Typical 128-bit blo
are starting to feel
Limit impact of co
Use larger blocks?

rs),
onds);
e;
te.

As,
logies.

etc.

Can one design do very well in hardware *and* software?

Some inspirational examples: Trivium and Keccak are "hardware" designs but not bad in software.

Another approach: replace ARX with "ORX". Skein-type mix doesn't work but can imitate Salsa20: compose `a^=((b|c)<<<r)`. Needs a few more rounds, but friendlier to hardware.

Another major research direc
achieve better security
than AES-GCM
without sacrificing performa

Typical 128-bit blocks
are starting to feel too small
Limit impact of collisions?
Use larger blocks?

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Can one design do very well
in hardware *and* software?

Some inspirational examples:
Trivium and Keccak
are "hardware" designs
but not bad in software.

Another approach:
replace ARX with "ORX".
Skein-type mix doesn't work
but can imitate Salsa20:
compose `a^=((b|c)<<<r)`.
Needs a few more rounds,
but friendlier to hardware.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?
(We've started some work.)

design do very well
are *and* software?

spirational examples:
and Keccak
dware" designs
bad in software.

approach:
ARX with "ORX".
pe mix doesn't work
imitate Salsa20:
e `a^=((b|c)<<<r)`.
few more rounds,
dlier to hardware.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?
(We've started some work.)

Allow re

User has
encrypti
will tell

o very well
oftware?

l examples:
ak
signs
ftware.

:
"ORX".
esn't work
alsa20:
c)<<<r).
rounds,
ardware.

Another major research direction: achieve better security than AES-GCM without sacrificing performance.

Typical 128-bit blocks are starting to feel too small. Limit impact of collisions? Use larger blocks?

Typical 128-bit pipe is starting to feel too small. Limit reforgeries? Use wider pipe?

Has anyone tried optimizing 192-bit/256-bit poly hashes? (We've started some work.)

Allow repeated me

User has to expect
encrypting (n, m)
will tell attacker w

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?
(We've started some work.)

Allow repeated message nun

User has to expect that
encrypting (n, m) and $(n, m$
will tell attacker whether m

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?
(We've started some work.)

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

Another major research direction: achieve better security than AES-GCM without sacrificing performance.

Typical 128-bit blocks are starting to feel too small. Limit impact of collisions? Use larger blocks?

Typical 128-bit pipe is starting to feel too small. Limit reforgeries? Use wider pipe?

Has anyone tried optimizing 192-bit/256-bit poly hashes? (We've started some work.)

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.
Limit impact of collisions?
Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.
Limit reforgeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?
(We've started some work.)

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

But user is surprised if repeated
message number leaks more
information, allows forgeries, etc.

2006 Rogaway–Shrimpton:
first authenticate (n, m),
then use the authenticator
as a nonce to encrypt m.

Another major research direction: achieve better security than AES-GCM without sacrificing performance.

Typical 128-bit blocks are starting to feel too small. Limit impact of collisions? Use larger blocks?

Typical 128-bit pipe is starting to feel too small. Limit reforgeries? Use wider pipe?

Has anyone tried optimizing 192-bit/256-bit poly hashes? (We've started some work.)

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

major research direction:
better security
S-GCM
sacrificing performance.

128-bit blocks
ing to feel too small.
pact of collisions?
er blocks?

128-bit pipe
ng to feel too small.
forgeries? Use wider pipe?

one tried optimizing
256-bit poly hashes?
started some work.)

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

But user is surprised if repeated
message number leaks more
information, allows forgeries, etc.

2006 Rogaway–Shrimpton:
first authenticate (n, m),
then use the authenticator
as a nonce to encrypt m.

Is this protection compatible
with fast forgery rejection?

Many ci
"free" m
e.g., AE

Is this co
with rep

earch direction:
urity

performance.

ocks
l too small.
ollisions?

pe
too small.
Use wider pipe?

optimizing
oly hashes?
me work.)

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

Many ciphers integ
"free" message au
e.g., AES-OCB, H

Is this compatible
with repeated mes

ction:

nce.

l.

pipe?

?

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

Many ciphers integrate "free" message authenticati e.g., AES-OCB, Helix, Pheli

Is this compatible with repeated message numl

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

Many ciphers integrate "free" message authentication: e.g., AES-OCB, Helix, Phelix.

Is this compatible with repeated message numbers?

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

Many ciphers integrate "free" message authentication: e.g., AES-OCB, Helix, Phelix.

Is this compatible with repeated message numbers?

Is this compatible with fast forgery rejection?

Allow repeated message numbers?

User has to expect that encrypting (n, m) and (n, m') will tell attacker whether $m = m'$.

But user is surprised if repeated message number leaks more information, allows forgeries, etc.

2006 Rogaway–Shrimpton: first authenticate (n, m), then use the authenticator as a nonce to encrypt m.

Is this protection compatible with fast forgery rejection?

Many ciphers integrate "free" message authentication: e.g., AES-OCB, Helix, Phelix.

Is this compatible with repeated message numbers?

Is this compatible with fast forgery rejection?

One approach: build $HFFH$ Feistel block cipher; reuse first H for fast auth with repeated message numbers; reuse last H for another auth with fast forgery rejection. But this consumes bandwidth.

peated message numbers?

s to expect that
ng (n, m) and (n, m')
attacker whether $m = m'$.

r is surprised if repeated
number leaks more
tion, allows forgeries, etc.

gaway–Shrimpton:
henticate (n, m),
e the authenticator
nce to encrypt m.

rotection compatible
t forgery rejection?

Many ciphers integrate
"free" message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

One approach: build
$HFFH$ Feistel block cipher;
reuse first H for fast auth
with repeated message numbers;
reuse last H for another auth
with fast forgery rejection.
But this consumes bandwidth.

Many m
in authe

AES-GC
the end

Can buil
using sa

Can buil
combine

Can buil
block cip

Can buil
authenti

essage numbers?

t that
and (n, m')
whether $m = m'$.

ed if repeated
eaks more
s forgeries, etc.

rimpton:
(n, m),
enticator
rypt m.

compatible
ejection?

Many ciphers integrate
“free” message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

One approach: build
$HFFH$ Feistel block cipher;
reuse first H for fast auth
with repeated message numbers;
reuse last H for another auth
with fast forgery rejection.
But this consumes bandwidth.

Many more directi
in authenticated c

AES-GCM is clear
the end of the sto

Can build better n
using same MAC,

Can build better M
combine with sam

Can build better
block ciphers, stre

Can build better i
authenticated ciph

mbers?

$n')$

$= m'$.

ated

, etc.

e

Many ciphers integrate
"free" message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

One approach: build
$HFFH$ Feistel block cipher;
reuse first H for fast auth
with repeated message numbers;
reuse last H for another auth
with fast forgery rejection.
But this consumes bandwidth.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers

Can build better integrated
authenticated ciphers.

Many ciphers integrate
"free" message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

One approach: build
$HFFH$ Feistel block cipher;
reuse first H for fast auth
with repeated message numbers;
reuse last H for another auth
with fast forgery rejection.
But this consumes bandwidth.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

phers integrate
message authentication:
S-OCB, Helix, Phelix.

ompatible
eated message numbers?

ompatible
t forgery rejection?

proach: build
Feistel block cipher;
st H for fast auth
eated message numbers;
st H for another auth
t forgery rejection.
consumes bandwidth.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

CAESAR

"Compe
Encrypti
Applicab

competi

Mailing
competi
@google

NIST is
to run a
but has
a $3330
competi

grate
thentication:
elix, Phelix.

ssage numbers?

ejection?

ild
ock cipher;
ast auth
ssage numbers;
nother auth
ejection.
s bandwidth.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

CAESAR

"Competition for
Encryption: Secur
Applicability, and

competitions.cr

Mailing list: cryp
competitions+su
@googlegroups.c

NIST is much too
to run another co
but has generously
a $333099 "Crypt
competitions" gra

on:
x.

bers?

bers;
h

th.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

CAESAR

"Competition for Authentica
Encryption: Security,
Applicability, and Robustnes

`competitions.cr.yp.to`

Mailing list: `crypto-`
`competitions+subscribe`
`@googlegroups.com`

NIST is much too busy
to run another competition
but has generously provided
a $333099 "Cryptographic
competitions" grant to UIC.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

CAESAR

"Competition for Authenticated
Encryption: Security,
Applicability, and Robustness"

`competitions.cr.yp.to`

Mailing list: `crypto-competitions+subscribe@googlegroups.com`

NIST is much too busy
to run another competition
but has generously provided
a \$333099 "Cryptographic
competitions" grant to UIC.