Introduction to symmetric crypto

D. J. Bernstein

How HTTPS protects connection:

- Public-key encryption system encrypts *one* secret message: a random 256-bit session key.
- Public-key signature system stops NSAITM attacks.
- Fast **authenticated cipher** uses the 256-bit session key to protect further messages.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

Introduction to symmetric crypto

D. J. Bernstein

How HTTPS protects connection:

- Public-key encryption system encrypts *one* secret message: a random 256-bit session key.
- Public-key signature system stops NSAITM attacks.
- Fast **authenticated cipher** uses the 256-bit session key to protect further messages.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

Introduction to symmetric crypto

D. J. Bernstein

How HTTPS protects connection:

- Public-key encryption system encrypts *one* secret message: a random 256-bit session key.
- Public-key signature system stops NSAITM attacks.
- Fast **authenticated cipher** uses the 256-bit session key to protect further messages.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

tion to symmetric crypto

ernstein

TTPS protects connection:

-key encryption system
ts *one* secret message:
om 256-bit session key.

-key signature system
NSAITM attacks.

uthenticated cipher
ne 256-bit session key
tect further messages.

Some cipher history

1973, and again in 1974:
U.S. National Bureau of
Standards solicits proposals
for a Data Encryption Standard.

1975: NBS publishes IBM DES
proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and
Hellman to discuss criticism.
Claims "somewhere over
$400,000,000" to break a DES
key; "I don't think you can tell
any Congressman what's going to
be secure 25 years from now."

1977: D

1977: D
publish c
$20,000,
hundred

mmetric crypto

---

ects connection:

ption system
cret message:
it session key.

ture system
attacks.

ted cipher
session key
er messages.

<u>Some cipher history</u>

1973, and again in 1974:
U.S. National Bureau of
Standards solicits proposals
for a Data Encryption Standard.

1975: NBS publishes IBM DES
proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and
Hellman to discuss criticism.
Claims "somewhere over
$400,000,000" to break a DES
key; "I don't think you can tell
any Congressman what's going to
be secure 25 years from now."

1977: DES is stan

1977: Diffie and H
publish detailed de
$20,000,000 mach
hundreds of DES

rypto

ection:

em

ge:

key.

m

er

ey

es.

Some cipher history

1973, and again in 1974:
U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to bre
hundreds of DES keys per y

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

pher history

nd again in 1974:
tional Bureau of
ds solicits proposals
ta Encryption Standard.

BS publishes IBM DES
. 64-bit block, 56-bit key.

SA meets Diffie and
to discuss criticism.
'somewhere over
0,000" to break a DES
lon't think you can tell
gressman what's going to
e 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U
of Stand
(NIST, f
for prop
Encrypti
block, 1

ry

n 1974:
eau of
proposals
tion Standard.

hes IBM DES
lock, 56-bit key.

Diffie and
s criticism.
re over
break a DES
k you can tell
what's going to
. from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. Nation
of Standards and T
(NIST, formerly N
for proposals for A
Encryption Standa
block, 128/192/25

lard.

DES
it key.

.

ES
tell
ing to
/.”

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes “NSA convinced IBM that a reduced key size was sufficient”.

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institut
of Standards and Technolog
(NIST, formerly NBS) calls
for proposals for Advanced
Encryption Standard. 128-b
block, 128/192/256-bit key.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of \$20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under \$250000 to break hundreds of DES keys per year.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of \$20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under \$250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

ES is standardized.

iffie and Hellman
detailed design of
,000 machine to break
s of DES keys per year.

ongressional investigation
A influence concludes
onvinced IBM that a
key size was sufficient".

988, 1993: Government
s DES standard.

hers publish new cipher
s and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: N
selects R

"Securit
factor in

ndardized.

Hellman
esign of
ine to break
keys per year.

nal investigation
e conclusions
BM that a
was sufficient".

Government
ndard.

h new cipher
urity analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advis
selects Rijndael as

"Security was the
factor in the evalu

ak
ear.

gation
es
n
ent".

ent

her
sis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NS
selects Rijndael as AES.

"Security was the most imp
factor in the evaluation"—R

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. ... Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

.S. National Institute
lards and Technology
formerly NBS) calls
osals for Advanced
on Standard. 128-bit
28/192/256-bit key.

5 AES proposals.

FF builds "Deep Crack"
r $250000 to break
s of DES keys per year.

IST selects five
alists: MARS, RC6,
, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main op
add rour
apply **su**
$x \mapsto x^{25}$
to each
linearly r

nal Institute
Technology
BS) calls
Advanced
ard. 128-bit
56-bit key.

posals.

"Deep Crack"
to break
keys per year.

ts five
RS, RC6,
Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in
add round key to
apply **substitution**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in blo
linearly mix bits ac

e

y

it

ack”

ear.

2000: NIST, advised by NSA, selects Rijndael as AES.

“Security was the most important factor in the evaluation”—Really?

“Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin.”

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box** $x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box** $x \mapsto x^{254}$ in $\mathbf{F}_{256}$ to each byte in block;
linearly mix bits across block.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box** $x \mapsto x^{254}$ in $\mathbf{F}_{256}$ to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world, no serious threats to AES-256 in a strong security model, "multi-target SPRP security".

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box** $x \mapsto x^{254}$ in $\mathbf{F}_{256}$ to each byte in block;
linearly mix bits across block.

Extensive security analysis. Even in a post-quantum world, no serious threats to AES-256 in a strong security model, "multi-target SPRP security".

So why isn't AES-256 the end of the symmetric-crypto story?

IST, advised by NSA,
Rijndael as AES.

y was the most important
the evaluation"—Really?

el appears to offer an
e security margin. . . .
appears to offer a
urity margin."

008: eSTREAM
tion for stream ciphers.
012: SHA-3 competition.
019: CAESAR competition.
ow: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

So why isn't AES-256 the end
of the symmetric-crypto story?

Goog

The latest ne
on the Interne

Speedin
HTTPS
Android
April 24, 20

Posted by Eli

Earlier this
Chrome tha
GCM on de

sed by NSA,
AES.

most important
ation"—Really?

to offer an
margin. . . .
o offer a
gin."

EAM
ream ciphers.
3 competition.
SAR competition.
WC competition.

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

So why isn't AES-256 the end
of the symmetric-crypto story?

Google Secu

The latest news and insights fro
on the Internet

Speeding up and st
HTTPS connection
Android

April 24, 2014

Posted by Elie Bursztein, Anti-Ab

Earlier this year, we deploy
Chrome that operates thre
GCM on devices that don't

A,

ortant
eally?

n

. .

ers.
tion.
etition.
etition.

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

So why isn't AES-256 the end
of the symmetric-crypto story?

Google Security Blog

The latest news and insights from Google on secu
on the Internet

Speeding up and strengthenin
HTTPS connections for Chron
Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lea

Earlier this year, we deployed a new TLS ci
Chrome that operates three times faster t
GCM on devices that don't have AES hardv

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

So why isn't AES-256 the end
of the symmetric-crypto story?

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware

erations in AES:
nd key to block;
bstitution box
^{4} in $\mathbf{F}_{256}$
byte in block;
mix bits across block.

e security analysis.
a post-quantum world,
us threats to AES-256
ong security model,
arget SPRP security".

isn't AES-256 the end
ymmetric-crypto story?

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware

acceleratio
wearable d
computers
latency and
amount of

To make th
Ben Laurie
-- ChaCha 2
for authent
2013. It wa
implementi
order to su
Associated
AEAD enab
happen co
optimize th
CBC. More
also promp

The benefit

n AES:
block;
n box

ock;
cross block.

analysis.
antum world,
to AES-256
y model,
P security".

256 the end
crypto story?

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware

acceleration, including mo
wearable devices such as
computers. This improves
latency and saving battery
amount of time spent encr

To make this happen, Ada
Ben Laurie and I began imp
-- ChaCha 20 for symmetri
for authentication -- in Ope
2013. It was a complex eff
implementing a new abstra
order to support the Authe
Associated Data (AEAD) e
AEAD enables encryption
happen concurrently, maki
optimize than older, comm
CBC. Moreover, recent atta
also prompted us to make

The benefits of this new ci

k.

rld,
56

”.

nd
ry?

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware

acceleration, including most Android phon
wearable devices such as Google Glass ar
computers. This improves user experience
latency and saving battery life by cutting d
amount of time spent encrypting and decr

To make this happen, Adam Langley, Wan-
Ben Laurie and I began implementing new
-- ChaCha 20 for symmetric encryption an
for authentication -- in OpenSSL and NSS i
2013. It was a complex effort that require
implementing a new abstraction layer in O
order to support the Authenticated Encryp
Associated Data (AEAD) encryption mode
AEAD enables encryption and authenticati
happen concurrently, making it easier to u
optimize than older, commonly-used mod
CBC. Moreover, recent attacks against RC
also prompted us to make this change.

The benefits of this new cipher suite inclu

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware

acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

le Security Blog

ws and insights from Google on security and safety
et

g up and strengthening connections for Chrome on

14

e Bursztein, Anti-Abuse Research Lead

year, we deployed a new TLS cipher suite in
at operates three times faster than AES-
vices that don't have AES hardware

acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

```
Date:
Message
[Downloa

From: Er

Hi all,

(Please
it to be

It was c
encrypti
storage
"Androic
these de
have to
Cryptogr

As we ex
challeng
the very
suitable
Speck, i
has a la

Therefor
encrypti
ChaCha s
paper he
```

rity Blog

m Google on security and safety

trengthening
s for Chrome on

ouse Research Lead

ed a new TLS cipher suite in
e times faster than AES-
have AES hardware

acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

```
Date:        201
Message-ID:  201
[Download message

From: Eric Biggers

Hi all,

(Please note that
it to be merged qu

It was officially
encryption [1].  W
storage encryption
"Android Go" devic
these devices stil
have to use older
Cryptography Exten

As we explained in
challenging proble
the very strict pe
suitable for pract
Speck, in this day
has a large politi

Therefore, we (wel
encryption mode, H
ChaCha stream ciph
paper here: https:
```

rity and safety

ng
ne on

d

pher suite in
han AES-
ware

7

acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

8

```
Date:       2018-08-06 2
Message-ID: 20180806223
[Download message RAW]

From: Eric Biggers &lt;ebiggers

Hi all,

(Please note that this patch
it to be merged quite yet!)

It was officially decided to
encryption [1].  We've been 
storage encryption to entry-
"Android Go" devices sold in
these devices still ship wit
have to use older CPUs like 
Cryptography Extensions, mak

As we explained in detail ea
challenging problem due to t
the very strict performance 
suitable for practical use i
Speck, in this day and age t
has a large political elemen

Therefore, we (well, Paul Cr
encryption mode, HPolyC.  In
ChaCha stream cipher for dis
paper here: https://eprint.i
```

8

acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

```
Date:        2018-08-06 22:32:51
Message-ID:  20180806223300.11389
[Download message RAW]

From: Eric Biggers <ebiggers@google.co

Hi all,

(Please note that this patchset is a t
it to be merged quite yet!)

It was officially decided to *not* all
encryption [1].  We've been working to
storage encryption to entry-level Andr
"Android Go" devices sold in developin
these devices still ship with no encry
have to use older CPUs like ARM Cortex
Cryptography Extensions, making AES-XT

As we explained in detail earlier, e.g
challenging problem due to the lack of
the very strict performance requiremen
suitable for practical use in dm-crypt
Speck, in this day and age the choice
has a large political element, restric

Therefore, we (well, Paul Crowley did
encryption mode, HPolyC.  In essence,
ChaCha stream cipher for disk encrypti
paper here: https://eprint.iacr.org/20
```

8

n, including most Android phones,
evices such as Google Glass and older
. This improves user experience, reducing
l saving battery life by cutting down the
time spent encrypting and decrypting data.

is happen, Adam Langley, Wan-Teh Chang,
and I began implementing new algorithms
20 for symmetric encryption and Poly1305
ication -- in OpenSSL and NSS in March
s a complex effort that required
ng a new abstraction layer in OpenSSL in
pport the Authenticated Encryption with
l Data (AEAD) encryption mode properly.
les encryption and authentication to
ncurrently, making it easier to use and
an older, commonly-used modes such as
over, recent attacks against RC4 and CBC
pted us to make this change.

s of this new cipher suite include:

```
Date:        2018-08-06 22:32:51
Message-ID:  20180806223300.113891-1-ebigg
[Download message RAW]

From: Eric Biggers <ebiggers@google.com>

Hi all,

(Please note that this patchset is a true RFC, i.
it to be merged quite yet!)

It was officially decided to *not* allow Android
encryption [1].  We've been working to find an al
storage encryption to entry-level Android devices
"Android Go" devices sold in developing countries
these devices still ship with no encryption, sinc
have to use older CPUs like ARM Cortex-A7; and th
Cryptography Extensions, making AES-XTS much too

As we explained in detail earlier, e.g. in [2], t
challenging problem due to the lack of encryption
the very strict performance requirements, while s
suitable for practical use in dm-crypt and fscryp
Speck, in this day and age the choice of cryptogr
has a large political element, restricting the op

Therefore, we (well, Paul Crowley did the real wo
encryption mode, HPolyC.  In essence, HPolyC make
ChaCha stream cipher for disk encryption.  HPolyC
paper here: https://eprint.iacr.org/2018/720.pdf
```

st Android phones,
Google Glass and older
user experience, reducing
life by cutting down the
ypting and decrypting data.

m Langley, Wan-Teh Chang,
plementing new algorithms
c encryption and Poly1305
nSSL and NSS in March
ort that required
action layer in OpenSSL in
nticated Encryption with
ncryption mode properly.
and authentication to
ng it easier to use and
only-used modes such as
cks against RC4 and CBC
this change.

pher suite include:

8

Date: 2018-08-06 22:32:51
Message-ID: 20180806223300.113891-1-ebiggers () ke
[Download message RAW]

From: Eric Biggers <ebiggers@google.com>

Hi all,

(Please note that this patchset is a true RFC, i.e. we're r
it to be merged quite yet!)

It was officially decided to *not* allow Android devices to
encryption [1]. We've been working to find an alternative
storage encryption to entry-level Android devices like the
"Android Go" devices sold in developing countries. Unfortu
these devices still ship with no encryption, since for cost
have to use older CPUs like ARM Cortex-A7; and these CPUs l
Cryptography Extensions, making AES-XTS much too slow.

As we explained in detail earlier, e.g. in [2], this is a v
challenging problem due to the lack of encryption algorithm
the very strict performance requirements, while still being
suitable for practical use in dm-crypt and fscrypt. And as
Speck, in this day and age the choice of cryptographic prim
has a large political element, restricting the options eve

Therefore, we (well, Paul Crowley did the real work) design
encryption mode, HPolyC. In essence, HPolyC makes it secur
ChaCha stream cipher for disk encryption. HPolyC is specif
paper here: https://eprint.iacr.org/2018/720.pdf ("HPolyC:

es,
nd older
e, reducing
lown the
ypting data.

-Teh Chang,
algorithms
d Poly1305
n March
d
penSSL in
tion with
properly.
ion to
se and
es such as
4 and CBC

de:

8

Date: 2018-08-06 22:32:51
Message-ID: 20180806223300.113891-1-ebiggers () kernel ! o
[Download message RAW]

From: Eric Biggers <ebiggers@google.com>

Hi all,

(Please note that this patchset is a true RFC, i.e. we're not ready f
it to be merged quite yet!)

It was officially decided to *not* allow Android devices to use Speck
encryption [1]. We've been working to find an alternative way to bri
storage encryption to entry-level Android devices like the inexpensiv
"Android Go" devices sold in developing countries. Unfortunately, of
these devices still ship with no encryption, since for cost reasons t
have to use older CPUs like ARM Cortex-A7; and these CPUs lack the AR
Cryptography Extensions, making AES-XTS much too slow.

As we explained in detail earlier, e.g. in [2], this is a very
challenging problem due to the lack of encryption algorithms that mee
the very strict performance requirements, while still being secure an
suitable for practical use in dm-crypt and fscrypt. And as we saw wi
Speck, in this day and age the choice of cryptographic primitives als
has a large political element, restricting the options even further.

Therefore, we (well, Paul Crowley did the real work) designed a new
encryption mode, HPolyC. In essence, HPolyC makes it secure to use t
ChaCha stream cipher for disk encryption. HPolyC is specified by our
paper here: https://eprint.iacr.org/2018/720.pdf ("HPolyC:

Date: 2018-08-06 22:32:51 10
Message-ID: 20180806223300.113891-1-ebiggers () kernel ! org
[Download message RAW]

From: Eric Biggers <ebiggers@google.com>

Hi all,

(Please note that this patchset is a true RFC, i.e. we're not ready for it to be merged quite yet!)

It was officially decided to *not* allow Android devices to use Speck encryption [1]. We've been working to find an alternative way to bring storage encryption to entry-level Android devices like the inexpensive "Android Go" devices sold in developing countries. Unfortunately, often these devices still ship with no encryption, since for cost reasons they have to use older CPUs like ARM Cortex-A7; and these CPUs lack the ARMv8 Cryptography Extensions, making AES-XTS much too slow.

As we explained in detail earlier, e.g. in [2], this is a very challenging problem due to the lack of encryption algorithms that meet the very strict performance requirements, while still being secure and suitable for practical use in dm-crypt and fscrypt. And as we saw with Speck, in this day and age the choice of cryptographic primitives also has a large political element, restricting the options even further.

Therefore, we (well, Paul Crowley did the real work) designed a new encryption mode, HPolyC. In essence, HPolyC makes it secure to use the ChaCha stream cipher for disk encryption. HPolyC is specified by our paper here: https://eprint.iacr.org/2018/720.pdf ("HPolyC:

2018-08-06 22:32:51
e-ID: 20180806223300.113891-1-ebiggers () kernel ! org
ad message RAW]

ric Biggers <ebiggers@google.com>

note that this patchset is a true RFC, i.e. we're not ready for
e merged quite yet!)

officially decided to *not* allow Android devices to use Speck
ion [1]. We've been working to find an alternative way to bring
encryption to entry-level Android devices like the inexpensive
d Go" devices sold in developing countries. Unfortunately, often
evices still ship with no encryption, since for cost reasons they
use older CPUs like ARM Cortex-A7; and these CPUs lack the ARMv8
raphy Extensions, making AES-XTS much too slow.

xplained in detail earlier, e.g. in [2], this is a very
ging problem due to the lack of encryption algorithms that meet
y strict performance requirements, while still being secure and
e for practical use in dm-crypt and fscrypt. And as we saw with
in this day and age the choice of cryptographic primitives also
arge political element, restricting the options even further.

re, we (well, Paul Crowley did the real work) designed a new
ion mode, HPolyC. In essence, HPolyC makes it secure to use the
stream cipher for disk encryption. HPolyC is specified by our
ere: https://eprint.iacr.org/2018/720.pdf ("HPolyC:

10

Goog

The latest new
on the Interne

Introduc
the Next
February 7,

Posted by Pau
Privacy Team

Storage enc

8-08-06 22:32:51
8080622300.113891-1-ebiggers () kernel ! org
RAW]

: <ebiggers@google.com>

this patchset is a true RFC, i.e. we're not ready for
ite yet!)

decided to *not* allow Android devices to use Speck
le've been working to find an alternative way to bring
to entry-level Android devices like the inexpensive
es sold in developing countries. Unfortunately, often
l ship with no encryption, since for cost reasons they
CPUs like ARM Cortex-A7; and these CPUs lack the ARMv8
sions, making AES-XTS much too slow.

detail earlier, e.g. in [2], this is a very
m due to the lack of encryption algorithms that meet
rformance requirements, while still being secure and
ical use in dm-crypt and fscrypt. And as we saw with
and age the choice of cryptographic primitives also
cal element, restricting the options even further.

l, Paul Crowley did the real work) designed a new
PolyC. In essence, HPolyC makes it secure to use the
er for disk encryption. HPolyC is specified by our
//eprint.iacr.org/2018/720.pdf ("HPolyC:

10

Google Secu

The latest news and insights fro
on the Internet

Introducing Adiant
the Next Billion Use
February 7, 2019

Posted by Paul Crowley and Eric
Privacy Team

Storage encryption protect

2:32:51
300.113891-1-ebiggers () kernel ! org

@google.com>

set is a true RFC, i.e. we're not ready for

not allow Android devices to use Speck
working to find an alternative way to bring
level Android devices like the inexpensive
developing countries. Unfortunately, often
h no encryption, since for cost reasons they
ARM Cortex-A7; and these CPUs lack the ARMv8
ing AES-XTS much too slow.

rlier, e.g. in [2], this is a very
he lack of encryption algorithms that meet
requirements, while still being secure and
n dm-crypt and fscrypt. And as we saw with
he choice of cryptographic primitives also
t, restricting the options even further.

owley did the real work) designed a new
essence, HPolyC makes it secure to use the
k encryption. HPolyC is specified by our
acr.org/2018/720.pdf ("HPolyC:

Google Security Blog

The latest news and insights from Google on secur
on the Internet

Introducing Adiantum: Encryp
the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android S
Privacy Team

Storage encryption protects your data if yo

1-1-ebiggers () kernel ! org

m>

rue RFC, i.e. we're not ready for

ow Android devices to use Speck
find an alternative way to bring
oid devices like the inexpensive
g countries. Unfortunately, often
ption, since for cost reasons they
-A7; and these CPUs lack the ARMv8
S much too slow.

. in [2], this is a very
encryption algorithms that meet
ts, while still being secure and
and fscrypt. And as we saw with
of cryptographic primitives also
ting the options even further.

the real work) designed a new
HPolyC makes it secure to use the
on. HPolyC is specified by our
18/720.pdf ("HPolyC:

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone ers () kernel ! org

.e. we're not ready for

devices to use Speck
lternative way to bring
s like the inexpensive
s. Unfortunately, often
ce for cost reasons they
hese CPUs lack the ARMv8
slow.

this is a very
n algorithms that meet
still being secure and
ot. And as we saw with
raphic primitives also
ptions even further.

ork) designed a new
es it secure to use the
C is specified by our
("HPolyC:

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone

ilesystem
Where AES
encryption
operation, v
Android su
encryption
However, w
is no widely
performan

To solve thi
encryption
us to use th
preserving
proposals f
HCTR and
encryption
about 10.6
AES-256-XT

ernel ! org

not ready for

o use Speck
 way to bring
 inexpensive
unately, often
t reasons they
lack the ARMv8

very
ms that meet
g secure and
s we saw with
mitives also
n further.

ned a new
re to use the
fied by our

10

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone

11

filesystem design.

Where AES is used, the cor
encryption is to use the XT
operation, which are length
Android supports AES-128
encryption and AES-256-XT
However, when AES perfor
is no widely accepted alter
performance on lower-end

To solve this problem, we h
encryption mode called Ad
us to use the ChaCha strea
preserving mode, by adapt
proposals for length-prese
HCTR and HCH. On ARM C
encryption and decryption
about 10.6 cycles per byte,
AES-256-XTS.

rg

or

ng
ve
ten
hey
Mv8

t
d
th
o

he

10

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone

11

filesystem design.

Where AES is used, the conventional solut
encryption is to use the XTS or CBC-ESSIV
operation, which are length-preserving. Cu
Android supports AES-128-CBC-ESSIV for
encryption and AES-256-XTS for file-based
However, when AES performance is insuffi
is no widely accepted alternative that has
performance on lower-end ARM processo

To solve this problem, we have designed a
encryption mode called Adiantum. Adiantu
us to use the ChaCha stream cipher in a le
preserving mode, by adapting ideas from A
proposals for length-preserving encryption
HCTR and HCH. On ARM Cortex-A7, Adian
encryption and decryption on 4096-byte se
about 10.6 cycles per byte, around 5x faste
AES-256-XTS.

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

le Security Blog

ws and insights from Google on security and safety
et

cing Adiantum: Encryption for
t Billion Users

2019

ul Crowley and Eric Biggers, Android Security &

cryption protects your data if your phone

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES per
in both
by small
heavy S-

rity Blog

m Google on security and safety

um: Encryption for
ers

: Biggers, Android Security &

:s your data if your phone

⋮

ilesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance
in both hardware a
by small 128-bit b
heavy S-box desig

rity and safety

tion for

Security &

our phone

⋮

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance seems lim
in both hardware and softwa
by small 128-bit block size,
heavy S-box design strategy.

⋮

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

⋮

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

⋮

filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits “PRF” security. Workarounds are hard to audit.

⋮

design.

is used, the conventional solution for disk is to use the XTS or CBC-ESSIV modes of which are length-preserving. Currently pports AES-128-CBC-ESSIV for full-disk and AES-256-XTS for file-based encryption. hen AES performance is insufficient there accepted alternative that has sufficient ce on lower-end ARM processors.

is problem, we have designed a new mode called Adiantum. Adiantum allows he ChaCha stream cipher in a length-mode, by adapting ideas from AES-based or length-preserving encryption such as HCH. On ARM Cortex-A7, Adiantum and decryption on 4096-byte sectors is cycles per byte, around 5x faster than TS.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits “PRF” security. Workarounds are hard to audit.

ChaCha with mu

nventional solution for disk
'S or CBC-ESSIV modes of
-preserving. Currently
-CBC-ESSIV for full-disk
'S for file-based encryption.
mance is insufficient there
native that has sufficient
ARM processors.

ave designed a new
iantum. Adiantum allows
m cipher in a length-
ing ideas from AES-based
ving encryption such as
ortex-A7, Adiantum
on 4096-byte sectors is
around 5x faster than

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits "PRF" security. Workarounds are hard to audit.

ChaCha creates sa
with much less wo

ion for disk
/ modes of
rrently
full-disk
d encryption.
icient there
sufficient
rs.

n new
um allows
ength-
AES-based
n such as
tum
ectors is
er than

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits "PRF" security. Workarounds are hard to audit.

ChaCha creates safe system
with much less work than A

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits "PRF" security. Workarounds are hard to audit.

ChaCha creates safe systems with much less work than AES.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits "PRF" security. Workarounds are hard to audit.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

rformance seems limited
hardware and software
128-bit block size,
-box design strategy.

tware ecosystem is
ated and dangerous.
tware implementations
S-box often leak
through timing.

is worse for high-security
cated ciphers. 128-bit
ze limits "PRF" security.
unds are hard to audit.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Authent

Standard

Assume
uniform
$r_1 \in \{0,$
$r_2 \in \{0,$
$\vdots$
$r_5 \in \{0,$
$s_1 \in \{0,$
$\vdots$
$s_{100} \in \{$

seems limited
and software
lock size,
n strategy.

system is
dangerous.
ementations
n leak
ming.

or high-security
ners. 128-bit
PRF" security.
hard to audit.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Authentication det

Standardize a prim

Assume sender kn
uniform random se
$r_1 \in \{0, 1, \ldots, 999$
$r_2 \in \{0, 1, \ldots, 999$
$\vdots$
$r_5 \in \{0, 1, \ldots, 999$
$s_1 \in \{0, 1, \ldots, 999$
$\vdots$
$s_{100} \in \{0, 1, \ldots, 9$

ited
are

ons

urity
bit
urity.
dit.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Authentication details

Standardize a prime $p = 100$

Assume sender knows indep
uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets

$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

creates safe systems
ch less work than AES.

amples of how symmetric
es have been improving
implicity, security:

NT is better than DES.

s better than
nd Speck.

BLAKE2, Ascon
er than MD5, SHA-0,
SHA-256, SHA-512.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume
secrets

...fe systems
...rk than AES.

... how symmetric
...en improving
...security:

...er than DES.

...an

... Ascon
...D5, SHA-0,
... SHA-512.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver k...
secrets $r_1, r_2, \ldots,$ ...

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets

$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the s

secrets $r_1, r_2, \ldots, r_5, s_1, \ldots,$

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets

$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**
$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p)$
$\quad + s_n \bmod 1000000$
and the message number n.

ication details

dize a prime $p = 1000003$.

sender knows independent

random secrets

$1, \ldots, 999999\}$,

$1, \ldots, 999999\}$,

$1, \ldots, 999999\}$,

$1, \ldots, 999999\}$,

$0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p)$

$+ s_n \bmod 1000000$

and the message number n.

e.g. $r_1 =$

$r_3 = 979$

$r_5 = 338$

$m_{10} = 00$

tails

ne $p = 1000003$.

ows independent

ecrets

999},

999},

999},

999},

99999}.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$,

$r_3 = 979323$, $r_4 =$

$r_5 = 338327$, $s_{10} =$

$m_{10} =$ 000006 000007 00

00003.

endent

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$, $r_2 = 2653$
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$, $r_4 = 846264$, $r_5 = 338327$, $s_{10} = 950288$, $m_{10} =$ 000006 000007 000000 000000 000000:

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$, $r_4 = 846264$, $r_5 = 338327$, $s_{10} = 950288$, $m_{10} = 000006\ 000007\ 000000\ 000000\ 000000$:

Sender computes authenticator

$$(6r_1 + 7r_2 \bmod p) + s_{10} \bmod 1000000 =$$

$$(6 \cdot 314159 + 7 \cdot 265358 \bmod 1000003) + 950288 \bmod 1000000 =$$

$$742451 + 950288 \bmod 1000000 =$$

$$692739.$$

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator** $(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p)$ $+ s_n \bmod 1000000$ and the message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$, $r_4 = 846264$, $r_5 = 338327$, $s_{10} = 950288$, $m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator $(6r_1 + 7r_2 \bmod p)$ $+ s_{10} \bmod 1000000 =$ $(6 \cdot 314159 + 7 \cdot 265358$ $\bmod 1000003)$ $+ 950288 \bmod 1000000 =$ $742451 + 950288 \bmod 1000000 =$ 692739.

Sender transmits

10 000006 000007 000000 000000 000000 692739.

receiver knows the same

$_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

ender wants to send

sages $m_1, \ldots, m_{100}$,

having 5 components

$_{,2}, m_{n,3}, m_{n,4}, m_{n,5}$

$_{,i} \in \{0, 1, \ldots, 999999\}$.

ransmits 30-digit

$_{,2}, m_{n,3}, m_{n,4}, m_{n,5}$

with an **authenticator**

$+ \cdots + m_{n,5} r_5 \bmod p)$

mod 1000000

message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$,

$r_3 = 979323$, $r_4 = 846264$,

$r_5 = 338327$, $s_{10} = 950288$,

$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6 r_1 + 7 r_2 \bmod p)$

$+ s_{10} \bmod 1000000 =$

$(6 \cdot 314159 + 7 \cdot 265358$

$\bmod 1000003)$

$+ 950288 \bmod 1000000 =$

$742451 + 950288 \bmod 1000000 =$

692739.

Sender transmits

10 000006 000007 000000 000000 000000 692739.

A MAC

Instead

$r_1, r_2, \ldots$

choose r

nows the same

$r_5, s_1, \ldots, s_{100}$.

nts to send

$\ldots, m_{100}$,

components

$m_{n,4}, m_{n,5}$

$\ldots, 999999\}$.

30-digit

$m_{n,4}, m_{n,5}$

authenticator

$_{n,5} r_5 \bmod p)$

000

number n.

e.g. $r_1 = 314159$, $r_2 = 265358$,

$r_3 = 979323$, $r_4 = 846264$,

$r_5 = 338327$, $s_{10} = 950288$,

$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r_1 + 7r_2 \bmod p)$

$+ s_{10} \bmod 1000000 =$

$(6 \cdot 314159 + 7 \cdot 265358$

$\bmod 1000003)$

$+ 950288 \bmod 1000000 =$

$742451 + 950288 \bmod 1000000 =$

692739.

Sender transmits

10 000006 000007 000000 000000 000000 692739.

A MAC using fewe

Instead of choosin

$r_1, r_2, \ldots, r_5, s_1, \ldots$

choose $r, s_1, s_2, \ldots$

same

s_{100}.

l

nts

9}.

ator

$p)$

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r_1 + 7r_2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003)$
$\quad + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

Sender transmits
10 000006 000007 000000 000000 000000 692739.

A MAC using fewer secrets

Instead of choosing indepen

$r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$,
choose $r, s_1, s_2, \ldots, s_{100}$.

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$(6r_1 + 7r_2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\bmod 1000003)$
$+ 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

Sender transmits
10 000006 000007 000000 000000 000000 692739.

A MAC using fewer secrets

Instead of choosing independent
$r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$,
choose $r, s_1, s_2, \ldots, s_{100}$.

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r_1 + 7r_2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003)$
$\quad + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

Sender transmits
10 000006 000007 000000 000000 000000 692739.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p)$
$\quad + s_n \bmod 1000000$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p)$
$\quad + s_n \bmod 1000000$.

$= 314159$, $r_2 = 265358$,

9323, $r_4 = 846264$,

3327, $s_{10} = 950288$,

00006 000007 000000 000000 000000:

computes authenticator

$r_2 \bmod p)$

$\bmod 1000000 =$

$59 + 7 \cdot 265358$

$1000003)$

$0288 \bmod 1000000 =$

$+ 950288 \bmod 1000000 =$

transmits

0007 000000 000000 000000 692739.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r =$

$m_{10} =$ 00

$r_2 = 265358$,

846264,

= 950288,

0000 000000 000000:

authenticator

0000 =

65358

1000000 =

mod 1000000 =

000 000000 692739.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r = 314159$, s

$m_{10} =$ 000006 000007 00

58,

000:

tor

=

000 =

739.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r = 314159$, $s_{10} = 2653$

$m_{10} =$ 000006 000007 000000 000000 000

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$$(m_{n,1} r + \cdots + m_{n,5} r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r = 314159$, $s_{10} = 265358$,

$m_{10} =$ 000006 000007 000000 000000 000000:

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator $(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p)$ $+ s_n \bmod 1000000$ and the message number n.

i.e.: take $r_i = r^i$ in previous $(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p)$ $+ s_n \bmod 1000000$.

e.g. $r = 314159$, $s_{10} = 265358$, $m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r + 7r^2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\bmod 1000003)$
$+ 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator
$$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p) + s_n \bmod 1000000$$
and the message number n.

i.e.: take $r_i = r^i$ in previous
$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r = 314159$, $s_{10} = 265358$, $m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$$(6r + 7r^2 \bmod p) + s_{10} \bmod 1000000 =$$
$$(6 \cdot 314159 + 7 \cdot 314159^2 \bmod 1000003) + 265358 \bmod 1000000 =$$
$$953311 + 265358 \bmod 1000000 =$$
218669.

Sender transmits authenticated message
10 000006 000007 000000 000000 000000 218669.

using fewer secrets

of choosing independent

$\ldots, r_5, s_1, \ldots, s_{100},$

$r, s_1, s_2, \ldots, s_{100}.$

transmits 30-digit

$_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$

with an authenticator

$+ \cdots + m_{n,5} r^5 \bmod p)$

mod 1000000

message number n.

e $r_i = r^i$ in previous

$+ \cdots + m_{n,5} r_5 \bmod p)$

mod 1000000.

e.g. $r = 314159$, $s_{10} = 265358$,

$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r + 7r^2 \bmod p)$

$\quad + s_{10} \bmod 1000000 =$

$(6 \cdot 314159 + 7 \cdot 314159^2$

$\quad \bmod 1000003)$

$\quad + 265358 \bmod 1000000 =$

$953311 + 265358 \bmod 1000000 =$

218669.

Sender transmits

authenticated message

10 000006 000007 000000 000000 000000 218669.

Security

Attacker

Find n',

$m' \neq m_n$

$(m'(r)$ m

Here m'

er secrets

g independent

. . , s_{100},

. , s_{100}.

30-digit

$n_{n,4}, m_{n,5}$

uthenticator

$_{,5} r^5$ mod p)

000

umber n.

n previous

$_{n,5} r_5$ mod p)

000.

e.g. $r = 314159$, $s_{10} = 265358$,

$m_{10} = 000006\ 000007\ 000000\ 000000\ 000000$:

Sender computes authenticator

$(6r + 7r^2 \bmod p)$

$\quad + s_{10} \bmod 1000000 =$

$(6 \cdot 314159 + 7 \cdot 314159^2$

$\quad \bmod 1000003)$

$\quad + 265358 \bmod 1000000 =$

$953311 + 265358 \bmod 1000000 =$

218669.

Sender transmits

authenticated message

10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:

Find n', m', a' such

$m' \neq m_{n'}$ but $a' =$

$(m'(r) \bmod p) + s$

Here $m'(x) = \sum_i$

dent

tor

$p)$

$p)$

e.g. $r = 314159$, $s_{10} = 265358$,

$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r + 7r^2 \bmod p)$

$+ s_{10} \bmod 1000000 =$

$(6 \cdot 314159 + 7 \cdot 314159^2$

$\bmod 1000003)$

$+ 265358 \bmod 1000000 =$

$953311 + 265358 \bmod 1000000 =$

218669.

Sender transmits

authenticated message

10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:

Find n', m', a' such that

$m' \neq m_{n'}$ but $a' =$

$(m'(r) \bmod p) + s_{n'} \bmod 10$

Here $m'(x) = \sum_i m'[i] x^i$.

e.g. $r = 314159$, $s_{10} = 265358$,
$m_{10} = 000006\ 000007\ 000000\ 000000\ 000000$:

Sender computes authenticator
$(6r + 7r^2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\bmod 1000003)$
$+ 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

Sender transmits
authenticated message
10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

e.g. $r = 314159$, $s_{10} = 265358$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$(6r + 7r^2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\bmod 1000003)$
$+ 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

Sender transmits
authenticated message
10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

e.g. $r = 314159$, $s_{10} = 265358$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$(6r + 7r^2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\bmod 1000003)$
$+ 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

Sender transmits
authenticated message
10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000.$
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

314159, $s_{10} = 265358$,

00006 000007 000000 000000 000000:

computes authenticator

^{2} mod p)

mod 1000000 =

159 + 7 · 314159^2

1000003)

5358 mod 1000000 =

+ 265358 mod 1000000 =

transmits

cated message

0007 000000 000000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance 1/1000000.

Can repeat attack.
Each forgery has chance
1/1000000 of being accepted.

More su

Choose

the poly

has 5 di

$x \in \{0, 1$

modulo

$_{10} = 265358$,

0000 000000 000000:

authenticator

0000 =

14159^2

1000000 =

mod 1000000 =

sage

000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

More subtle attac
Choose $m' \neq m_1$ s
the polynomial m'
has 5 distinct root
$x \in \{0, 1, \ldots, 999$
modulo p. Choose

58,

000:

tor

=

000 =

69.

Security analysis

Attacker's goal:

Find n', m', a' such that $m' \neq m_{n'}$ but $a' = (m'(r) \bmod p) + s_{n'} \bmod 1000000$.

Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:

Choose any $m' \neq m_1$.

Choose uniform random a'.

Success chance $1/1000000$.

Can repeat attack.

Each forgery has chance $1/1000000$ of being accepted.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1($

has 5 distinct roots

$x \in \{0, 1, \ldots, 999999\}$

modulo p. Choose $a' = a$.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

More subtle attack:
Choose $m' \neq m_1$ so that
the polynomial $m'(x) - m_1(x)$
has 5 distinct roots
$x \in \{0, 1, \ldots, 999999\}$
modulo p. Choose $a' = a$.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000.$
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

More subtle attack:
Choose $m' \neq m_1$ so that
the polynomial $m'(x) - m_1(x)$
has 5 distinct roots
$x \in \{0, 1, \ldots, 999999\}$
modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144.$

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

More subtle attack:
Choose $m' \neq m_1$ so that
the polynomial $m'(x) - m_1(x)$
has 5 distinct roots
$x \in \{0, 1, \ldots, 999999\}$
modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

analysis

's goal:

m', a' such that

$_{n'}$ but $a' =$

nod $p) + s_{n'}$ mod 1000000.

$(x) = \sum_i m'[i]x^i$.

attack:

any $m' \neq m_1$.

uniform random a'.

chance $1/1000000$.

eat attack.

gery has chance

00 of being accepted.

More subtle attack:

Choose $m' \neq m_1$ so that
the polynomial $m'(x) - m_1(x)$
has 5 distinct roots
$x \in \{0, 1, \ldots, 999999\}$
modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually

can be a

n that

=

$_{n'}$ mod 1000000.

$m'[i]x^i$.

m_1.

ndom a'.

/1000000.

.

chance

ng accepted.

More subtle attack:

Choose $m' \neq m_1$ so that
the polynomial $m'(x) - m_1(x)$
has 5 distinct roots
$x \in \{0, 1, \ldots, 999999\}$
modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success c
can be above 5/10

00000.

d.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success chance can be above $5/1000000$.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$, $m' = (125, 1, 0, 0, 1)$: $m'(x) - m_1(x) = x^5 + x^2 + 25x$ which has five roots mod p: $0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success chance can be above $5/1000000$.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 + 25x$
which has five roots mod p:
$0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success chance can be above $5/1000000$.

Example: If $m_1(334885)$ mod p $\in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance $6/1000000$.
Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$, $m' = (125, 1, 0, 0, 1)$: $m'(x) - m_1(x) = x^5 + x^2 + 25x$ which has five roots mod p: $0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success chance can be above $5/1000000$.

Example: If $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance $6/1000000$. Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot (m'(x) - m_1(x) + 1000000) \cdot (m'(x) - m_1(x) - 1000000)$.

btle attack:
$m' \neq m_1$ so that
nomial $m'(x) - m_1(x)$
stinct roots
$1, \ldots, 999999\}$
p. Choose $a' = a$.

$= (100, 0, 0, 0, 0)$,
$25, 1, 0, 0, 1)$:
$m_1(x) = x^5 + x^2 + 25x$
as five roots mod p:
$2, 334447, 631403, 735144$.

chance $5/1000000$.

Actually, success chance
can be above $5/1000000$.

Example: If $m_1(334885) \bmod p$
$\in \{1000000, 1000001, 1000002\}$
then a forgery $(1, m', a_1)$ with
$m'(x) = m_1(x) + x^5 + x^2 + 25x$
also succeeds for $r = 334885$;
success chance $6/1000000$.
Reason: 334885 is a root of
$m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots
of $(m'(x) - m_1(x)) \cdot$
$(m'(x) - m_1(x) + 1000000) \cdot$
$(m'(x) - m_1(x) - 1000000)$.

Do bette

k:

so that

$(x) - m_1(x)$

ts

999}

e $a' = a$.

$0, 0, 0)$,

$1)$:

$x^5 + x^2 + 25x$

ts mod p:

$631403, 735144$.

/1000000.

Actually, success chance can be above 5/1000000.

Example: If $m_1(334885)$ mod p $\in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance 6/1000000. Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot (m'(x) - m_1(x) + 1000000) \cdot (m'(x) - m_1(x) - 1000000)$.

Do better by varyi

$x)$

$25x$

35144.

Actually, success chance can be above 5/1000000.

Example: If $m_1(334885) \bmod p$
$\in \{1000000, 1000001, 1000002\}$
then a forgery $(1, m', a_1)$ with
$m'(x) = m_1(x) + x^5 + x^2 + 25x$
also succeeds for $r = 334885$;
success chance 6/1000000.
Reason: 334885 is a root of
$m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot$
$(m'(x) - m_1(x) + 1000000) \cdot$
$(m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

Actually, success chance can be above 5/1000000.

Example: If $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance 6/1000000.
Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot (m'(x) - m_1(x) + 1000000) \cdot (m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

Actually, success chance can be above $5/1000000$.

Example: If $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance $6/1000000$. Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot (m'(x) - m_1(x) + 1000000) \cdot (m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Actually, success chance can be above 5/1000000.

Example: If $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance 6/1000000. Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot$ $(m'(x) - m_1(x) + 1000000) \cdot$ $(m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$ $(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$ $(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Actually, success chance can be above 5/1000000.

Example: If $m_1(334885) \bmod p$ $\in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance 6/1000000. Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot$ $(m'(x) - m_1(x) + 1000000) \cdot$ $(m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$ $(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$ $(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator $(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$ $+ s_n \bmod 1000000$: solve $m'(x) - m_1(x) = a' - a_1$.

, success chance
above $5/1000000$.

e: If $m_1(334885) \bmod p$
$000, 1000001, 1000002\}$
orgery $(1, m', a_1)$ with
$m_1(x) + x^5 + x^2 + 25x$
ceeds for $r = 334885$;
chance $6/1000000$.
334885 is a root of
$m_1(x) + 1000000$.

e as many as 15 roots
$) - m_1(x)) \cdot$
$- m_1(x) + 1000000) \cdot$
$- m_1(x) - 1000000)$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator
$(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$\quad + s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled u

Poly130
with 22
Adds s_n

chance
000000.

34885) mod p
$001, 1000002\}$
m', a_1) with
$x^5 + x^2 + 25x$
$r = 334885$;
1000000.
s a root of
1000000.

as 15 roots
$)) \cdot$
1000000) $\cdot$
1000000).

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot (m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot (m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator $(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$ $+ s_n \bmod 1000000$: solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serio

Poly1305 uses 128
with 22 bits cleare
Adds s_n mod 2^{128}

od p
02}
th
25x
5;
ts

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator
$(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$\quad + s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious securit

Poly1305 uses 128-bit r's,
with 22 bits cleared for spee
Adds $s_n \bmod 2^{128}$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator
$(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$\quad + s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds $s_n \bmod 2^{128}$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot (m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot (m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator $(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$ $+ s_n \bmod 1000000$: solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds $s_n \bmod 2^{128}$.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16 \rceil$ choices of r. Probability $\leq 8\lceil L/16 \rceil / 2^{106}$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator $(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$+ s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds $s_n \bmod 2^{128}$.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8 \lceil L/16 \rceil$ choices of r. Probability $\leq 8 \lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected with probability $\geq 1 - 8D \lceil L/16 \rceil / 2^{106}$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator
$(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$+ s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds $s_n \bmod 2^{128}$.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16 \rceil$ choices of r. Probability $\leq 8\lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected with probability $\geq 1 - 8D\lceil L/16 \rceil / 2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$: Pr[all rejected] ≥ 0.9999999998.

er by varying a'?

y to prove: Every choice
$', a')$ with $m' \neq m_{n'}$
nce $\leq 15/1000000$
accepted by receiver.

ng fact: ≤ 15 roots
$) - m_1(x) - a' + a_1) \cdot$
$- m_1(x) - a' + a_1 + 10^6) \cdot$
$- m_1(x) - a' + a_1 - 10^6)$.

: very easy to break
simplified authenticator
$+ \cdots + m_n[5]r^4 \bmod p)$
mod 1000000:
$(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's,
with 22 bits cleared for speed.
Adds $s_n \bmod 2^{128}$.

Assuming $\leq L$-byte messages:
Each forgery succeeds for
$\leq 8 \lceil L/16 \rceil$ choices of r.
Probability $\leq 8 \lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected
with probability
$\geq 1 - 8D \lceil L/16 \rceil / 2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$:
Pr[all rejected] ≥ 0.9999999998.

Authent
for varia
if differe
different

ng a'?

: Every choice

$m' \neq m_{n'}$

1000000

by receiver.

≤ 15 roots

$) - a' + a_1) \cdot$

$a' + a_1 + 10^6) \cdot$

$a' + a_1 - 10^6)$.

y to break

authenticator

$[5]r^4 \bmod p)$

000:

$x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's,
with 22 bits cleared for speed.
Adds $s_n \bmod 2^{128}$.

Assuming $\leq L$-byte messages:
Each forgery succeeds for
$\leq 8 \lceil L/16 \rceil$ choices of r.
Probability $\leq 8 \lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected
with probability
$\geq 1 - 8D \lceil L/16 \rceil / 2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$:
Pr[all rejected] ≥ 0.9999999998.

Authenticator is st
for variable-length
if different messag
different polynomi

hoice

er.

) ·

10^6) ·

10^6).

k

ator

p)

a_1.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8 \lceil L/16 \rceil$ choices of r. Probability $\leq 8 \lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected with probability $\geq 1 - 8D \lceil L/16 \rceil / 2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$: Pr[all rejected] ≥ 0.9999999998.

Authenticator is still secure for variable-length messages if different messages are different polynomials mod p

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16\rceil$ choices of r. Probability $\leq 8\lceil L/16\rceil/2^{106}$.

D forgeries are all rejected with probability $\geq 1 - 8D\lceil L/16\rceil/2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$: Pr[all rejected] ≥ 0.9999999998.

Authenticator is still secure for variable-length messages, if different messages are different polynomials mod p.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16\rceil$ choices of r. Probability $\leq 8\lceil L/16\rceil/2^{106}$.

D forgeries are all rejected with probability $\geq 1-8D\lceil L/16\rceil/2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$: Pr[all rejected] ≥ 0.9999999998.

Authenticator is still secure for variable-length messages, if different messages are different polynomials mod p.

Split string into 16-byte chunks, maybe with smaller final chunk; append 1 to each chunk; view as little-endian integers in $\{1, 2, 3, \ldots, 2^{129}\}$. Multiply first chunk by r, add next chunk, multiply by r, etc., last chunk, multiply by r, mod $2^{130}-5$, add s_n mod 2^{128}.