On the design of message-authentication codes

D. J. Bernstein
University of Illinois at Chicago

When we design hash functions, stream ciphers, and other secret-key primitives, should we use integer multiplication?

AES uses $32, 32 \to 32$ xor; $32 \to 8$ byte extraction; and $8 \to 32$ inversion box.

IDEA uses $16, 16 \to 16$ xor; $16, 16 \to 16$ addition; and $16, 16 \to 16$ multiplication.

design of
-authentication codes

ernstein
ty of Illinois at Chicago

When we design
hash functions, stream ciphers,
and other secret-key primitives,
should we use
integer multiplication?

AES uses $32, 32 \to 32$ xor;
$32 \to 8$ byte extraction;
and $8 \to 32$ inversion box.

IDEA uses $16, 16 \to 16$ xor;
$16, 16 \to 16$ addition; and
$16, 16 \to 16$ multiplication.

Rabbit u
$32, 32 \to$
$32, 32 \to$
$32, 32 \to$

RC6 use
$32, 32 \to$
$32, 32 \to$
$32, 32 \to$

Salsa20
$32, 32 \to$
$32, 32 \to$

cation codes

is at Chicago

When we design hash functions, stream ciphers, and other secret-key primitives, should we use integer multiplication?

AES uses $32, 32 \to 32$ xor; $32 \to 8$ byte extraction; and $8 \to 32$ inversion box.

IDEA uses $16, 16 \to 16$ xor; $16, 16 \to 16$ addition; and $16, 16 \to 16$ multiplication.

Rabbit uses $32 \to$
$32, 32 \to 32$ additi
$32, 32 \to 32$ xor; a
$32, 32 \to 32, 32$ m

RC6 uses $32, 8 \to$
$32, 32 \to 32$ additi
$32, 32 \to 32$ xor; a
$32, 32 \to 32$ multi

Salsa20 uses $32 \to$
$32, 32 \to 32$ additi
$32, 32 \to 32$ xor.

es

ago

When we design
hash functions, stream ciphers,
and other secret-key primitives,
should we use
integer multiplication?

AES uses $32, 32 \to 32$ xor;
$32 \to 8$ byte extraction;
and $8 \to 32$ inversion box.

IDEA uses $16, 16 \to 16$ xor;
$16, 16 \to 16$ addition; and
$16, 16 \to 16$ multiplication.

Rabbit uses $32 \to 32$ rotatio
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplicatio

RC6 uses $32, 8 \to 32$ rotatio
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotati
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

When we design
hash functions, stream ciphers,
and other secret-key primitives,
should we use
integer multiplication?

AES uses $32, 32 \to 32$ xor;
$32 \to 8$ byte extraction;
and $8 \to 32$ inversion box.

IDEA uses $16, 16 \to 16$ xor;
$16, 16 \to 16$ addition; and
$16, 16 \to 16$ multiplication.

Rabbit uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplication.

RC6 uses $32, 8 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

e design
nctions, stream ciphers,
er secret-key primitives,
ve use
multiplication?

es $32, 32 \to 32$ xor;
byte extraction;
$\to 32$ inversion box.

ses $16, 16 \to 16$ xor;
$\to 16$ addition; and
$\to 16$ multiplication.

Rabbit uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplication.

RC6 uses $32, 8 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

"Multipl
$> 10\times$ a
as additi

Countera
"Multipl
is surpris
Has mar
so CPU
big mult
Typical C
new mul

ream ciphers,
ey primitives,

ion?

→ 32 xor;
ction;
sion box.

→ 16 xor;
ion; and
plication.

Rabbit uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplication.

RC6 uses $32, 8 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

"Multiplication is
$> 10\times$ as many bi
as addition.

Counterargument:
"Multiplication
is surprisingly fast
Has many applicat
so CPU designers
big multiplication
Typical CPUs can
new multiplication

ers,
ves,

Rabbit uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplication.

RC6 uses $32, 8 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

"Multiplication is slow!"
$> 10\times$ as many bit operatio
as addition.

Counterargument:
"Multiplication
is surprisingly fast!"
Has many applications,
so CPU designers include
big multiplication circuits.
Typical CPUs can start a
new multiplication every cyc

Rabbit uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32, 32$ multiplication.

RC6 uses $32, 8 \to 32$ rotation;
$32, 32 \to 32$ addition;
$32, 32 \to 32$ xor; and
$32, 32 \to 32$ multiplication.

Salsa20 uses $32 \to 32$ rotation;
$32, 32 \to 32$ addition; and
$32, 32 \to 32$ xor.

"Multiplication is slow!"
$> 10\times$ as many bit operations
as addition.

Counterargument:
"Multiplication
is surprisingly fast!"
Has many applications,
so CPU designers include
big multiplication circuits.
Typical CPUs can start a
new multiplication every cycle.

uses $32 \to 32$ rotation;
$\to 32$ addition;
$\to 32$ xor; and
$\to 32, 32$ multiplication.

s $32, 8 \to 32$ rotation;
$\to 32$ addition;
$\to 32$ xor; and
$\to 32$ multiplication.

uses $32 \to 32$ rotation;
$\to 32$ addition; and
$\to 32$ xor.

"Multiplication is slow!"
$> 10\times$ as many bit operations
as addition.

Counterargument:
"Multiplication
is surprisingly fast!"
Has many applications,
so CPU designers include
big multiplication circuits.
Typical CPUs can start a
new multiplication every cycle.

"Multipl
scramble
as thoro
several s

"No, it c
Look at
Need ma
to achiev

What if
that mu
the secu

32 rotation;
ion;
nd
ultiplication.

32 rotation;
ion;
nd
plication.

32 rotation;
ion; and

"Multiplication is slow!"
$> 10\times$ as many bit operations as addition.

Counterargument:
"Multiplication is surprisingly fast!"
Has many applications, so CPU designers include big multiplication circuits.
Typical CPUs can start a new multiplication every cycle.

"Multiplication
scrambles its outp
as thoroughly as
several simple ope

"No, it doesn't!
Look at these scar
Need many multip
to achieve confide

What if we can *pr*
that multiplication
the security we ne

on;

on.

on;

on;

"Multiplication is slow!"
$> 10\times$ as many bit operations
as addition.

Counterargument:
"Multiplication
is surprisingly fast!"
Has many applications,
so CPU designers include
big multiplication circuits.
Typical CPUs can start a
new multiplication every cycle.

"Multiplication
scrambles its output
as thoroughly as
several simple operations!"

"No, it doesn't!
Look at these scary attacks.
Need many multiplications
to achieve confidence."

What if we can *prove*
that multiplication provides
the security we need?

“Multiplication is slow!”
$> 10\times$ as many bit operations as addition.

Counterargument:
“Multiplication is surprisingly fast!”
Has many applications, so CPU designers include big multiplication circuits.
Typical CPUs can start a new multiplication every cycle.

“Multiplication scrambles its output as thoroughly as several simple operations!”

“No, it doesn’t!
Look at these scary attacks.
Need many multiplications to achieve confidence.”

What if we can *prove* that multiplication provides the security we need?

ication is slow!"
as many bit operations
ion.

argument:
ication
singly fast!"
ny applications,
designers include
tiplication circuits.
CPUs can start a
ltiplication every cycle.

"Multiplication
scrambles its output
as thoroughly as
several simple operations!"

"No, it doesn't!
Look at these scary attacks.
Need many multiplications
to achieve confidence."

What if we can *prove*
that multiplication provides
the security we need?

An auth

Let's use
to authe

Standard

Sender r
to gener
uniform
$r \in \{0, 1$
$s_1 \in \{0,$
$s_2 \in \{0,$
$\ldots,$
$s_{100} \in \{$

slow!"

it operations

!"

tions,

include

circuits.

start a

every cycle.

"Multiplication scrambles its output as thoroughly as several simple operations!"

"No, it doesn't! Look at these scary attacks. Need many multiplications to achieve confidence."

What if we can *prove* that multiplication provides the security we need?

An authentication

Let's use multiplic

to authenticate m

Standardize a prim

Sender rolls 10-sid

to generate indepe

uniform random s

$r \in \{0, 1, \ldots, 999$

$s_1 \in \{0, 1, \ldots, 99$

$s_2 \in \{0, 1, \ldots, 99$

$\ldots,$

$s_{100} \in \{0, 1, \ldots, 9$

ons

cle.

"Multiplication scrambles its output as thoroughly as several simple operations!"

"No, it doesn't! Look at these scary attacks. Need many multiplications to achieve confidence."

What if we can *prove* that multiplication provides the security we need?

An authentication system

Let's use multiplication to authenticate messages.

Standardize a prime $p = 100$

Sender rolls 10-sided die to generate independent uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

"Multiplication
scrambles its output
as thoroughly as
several simple operations!"

"No, it doesn't!
Look at these scary attacks.
Need many multiplications
to achieve confidence."

What if we can *prove*
that multiplication provides
the security we need?

An authentication system

Let's use multiplication
to authenticate messages.

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die
to generate independent
uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

ication
es its output
ughly as
simple operations!"

doesn't!
these scary attacks.
any multiplications
ve confidence."

we can *prove*
ltiplication provides
rity we need?

An authentication system

Let's use multiplication
to authenticate messages.

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die
to generate independent
uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Sender r
and tells
secrets r

Later: S
100 mes
each hav
$m_n[1], m$
with m_n

Sender t
$m_n[1], m$
together
$(m_n[1]r$
$+ s_n$
and the

ut

rations!”

ry attacks.
lications
nce.”

rove
provides
ed?

An authentication system

Let’s use multiplication
to authenticate messages.

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die
to generate independent
uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Sender meets rece
and tells receiver t
secrets $r, s_1, s_2, \ldots$

Later: Sender war
100 messages $m_1,$
each having 5 com
$m_n[1], m_n[2], m_n$
with $m_n[i] \in \{0, 1$

Sender transmits 3
$m_n[1], m_n[2], m_n$
together with an a
$(m_n[1]r + \cdots + n$
$+ s_n$ mod 1000
and the message n

An authentication system

Let's use multiplication to authenticate messages.

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die to generate independent uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Sender meets receiver in priv
and tells receiver the same
secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send
100 messages $m_1, \ldots, m_{100}$
each having 5 components
$m_n[1], m_n[2], m_n[3], m_n[4]$
with $m_n[i] \in \{0, 1, \ldots, 999$

Sender transmits 30-digit
$m_n[1], m_n[2], m_n[3], m_n[4]$
together with an **authentica**
$(m_n[1]r + \cdots + m_n[5]r^5$ m
$\quad + s_n \bmod 1000000$
and the message number n.

An authentication system

Let's use multiplication to authenticate messages.

Standardize a prime $p = 1000003$.

Sender rolls 10-sided die to generate independent uniform random secrets
$r \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$s_2 \in \{0, 1, \ldots, 999999\}$,
$\ldots$,
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Sender meets receiver in private and tells receiver the same secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each 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**

$$(m_n[1]r + \cdots + m_n[5]r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

entication system

e multiplication

enticate messages.

dize a prime $p = 1000003$.

rolls 10-sided die

ate independent

random secrets

$1, \ldots, 999999\}$,

$1, \ldots, 999999\}$,

$1, \ldots, 999999\}$,

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

Sender meets receiver in private and tells receiver the same secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each 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**

$(m_n[1]r + \cdots + m_n[5]r^5 \bmod p)$
$\quad + s_n \bmod 1000000$

and the message number n.

e.g. $r =$

$m_{10} = 0$

Sender c

$(6r + 7r$

$+ s_{10}$

$(6 \cdot 3141$

mod

$+ 265$

953311 -

218669.

Sender t

authenti

10 000006 000

system

ation

essages.

ne $p = 1000003$.

ed die

endent

ecrets

999},

9999},

9999},

99999}.

Sender meets receiver in private
and tells receiver the same
secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send
100 messages $m_1, \ldots, m_{100}$,
each 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**

$$(m_n[1]r + \cdots + m_n[5]r^5 \bmod p)$$
$$+ s_n \bmod 1000000$$

and the message number n.

e.g. $r = 314159$, s

$m_{10} =$ 000006 000007 00

Sender computes a

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

$+ s_{10}$ mod 1000

$(6 \cdot 314159 + 7 \cdot 3$

mod 1000003)

$+ 265358$ mod

$953311 + 265358$

218669.

Sender transmits

authenticated mes

10 000006 000007 000000 000

00003.

Sender meets receiver in private and tells receiver the same secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each 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** $(m_n[1]r + \cdots + m_n[5]r^5 \bmod p)$
$+ s_n \bmod 1000000$
and the message number n.

e.g. $r = 314159$, $s_{10} = 2653$
$m_{10} =$ 000006 000007 000000 000000 000

Sender computes authentica
$(6r + 7r^2 \bmod p)$
$+ s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\bmod 1000003)$
$+ 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000($
218669.

Sender transmits authenticated message
10 000006 000007 000000 000000 000000 2186

Sender meets receiver in private and tells receiver the same secrets $r, s_1, s_2, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each 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**

$$(m_n[1]r + \cdots + m_n[5]r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

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.

meets receiver in private

s receiver the same

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

ender wants to send

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

ing 5 components

$n_n[2], m_n[3], m_n[4], m_n[5]$

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

ransmits 30-digit

$n_n[2], m_n[3], m_n[4], m_n[5]$

with an **authenticator**

$+ \cdots + m_n[5]r^5 \bmod p)$

mod 1000000

message number n.

e.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.

Speed ar

Notation

To comp

multiply

add m_n

add m_n

add m_n

add m_n

Reduce

Slightly

compute

$(m_n(r)$

iver in private
the same
. . , s_{100}.

nts to send
. . . . , m_{100},
nponents
[3], $m_n[4], m_n[5]$
., . . . , 999999}.

30-digit
[3], $m_n[4], m_n[5]$
authenticator
$n_n[5]r^5$ mod p)
000
number n.

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.

Speed analysis

Notation: $m_n(x)$

To compute $m_n(r$
multiply $m_n[5]$ by
add $m_n[4]$, multip
add $m_n[3]$, multip
add $m_n[2]$, multip
add $m_n[1]$, multip
Reduce mod p aft

Slightly more time
compute authentic
$(m_n(r) \bmod p) +$

vate

l

,

, $m_n[5]$

999}.

, $m_n[5]$

ator

od p)

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.

Speed analysis

Notation: $m_n(x) = \sum m_n[$

To compute $m_n(r) \bmod p$:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each m

Slightly more time to
compute authenticator $a_n =$
$(m_n(r) \bmod p) + s_n \bmod 1($

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.

<u>Speed analysis</u>

Notation: $m_n(x) = \sum m_n[i] x^i$.

To compute $m_n(r) \bmod p$:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each mult.

Slightly more time to
compute authenticator $a_n =$
$(m_n(r) \bmod p) + s_n \bmod 1000000$.

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.

Speed analysis

Notation: $m_n(x) = \sum m_n[i]x^i$.

To compute $m_n(r)$ mod p:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each mult.

Slightly more time to compute authenticator $a_n = (m_n(r) \bmod p) + s_n \bmod 1000000$.

Reducin

e.g., 240

240881

240881(

−72264

−62355

Easily ac

{0, 1, . . .

by addin

(Beware

Speedup

extra p's

subseque

$s_{10} = 265358$,

0000 000000 000000:

authenticator

0000 =

14159^2

1000000 =

mod 1000000 =

ssage

000 000000 218669.

Speed analysis

Notation: $m_n(x) = \sum m_n[i]x^i$.

To compute $m_n(r)$ mod p:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each mult.

Slightly more time to
compute authenticator $a_n =$
$(m_n(r) \bmod p) + s_n \bmod 1000000$.

Reducing mod 100
e.g., 24088109909
$240881 \cdot 1000000$ -
$240881(-3) + 990$
$-722643 + 99091$
-623552.

Easily adjust to ra
$\{0, 1, \ldots, p-1\}$
by adding/subtrac
(Beware timing at

Speedup: Delay th
extra p's won't da
subsequent field o

358,

000:

tor

=

000 =

69.

<u>Speed analysis</u>

Notation: $m_n(x) = \sum m_n[i]x^i$.

To compute $m_n(r)$ mod p:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each mult.

Slightly more time to
compute authenticator $a_n =$
$(m_n(r) \bmod p) + s_n$ mod 1000000.

Reducing mod 1000003 is ea
e.g., $240881099091 =$
$240881 \cdot 1000000 + 99091 \equiv$
$240881(-3) + 99091 =$
$-722643 + 99091 =$
-623552.

Easily adjust to range
$\{0, 1, \ldots, p-1\}$
by adding/subtracting a few
(Beware timing attacks!)

Speedup: Delay the adjustm
extra p's won't damage
subsequent field operations.

Speed analysis

Notation: $m_n(x) = \sum m_n[i]x^i$.

To compute $m_n(r)$ mod p:
multiply $m_n[5]$ by r,
add $m_n[4]$, multiply by r,
add $m_n[3]$, multiply by r,
add $m_n[2]$, multiply by r,
add $m_n[1]$, multiply by r.
Reduce mod p after each mult.

Slightly more time to compute authenticator $a_n = (m_n(r) \text{ mod } p) + s_n$ mod 1000000.

Reducing mod 1000003 is easy:
e.g., $240881099091 =$
$240881 \cdot 1000000 + 99091 \equiv$
$240881(-3) + 99091 =$
$-722643 + 99091 =$
-623552.

Easily adjust to range $\{0, 1, \ldots, p-1\}$ by adding/subtracting a few p's.
(Beware timing attacks!)

Speedup: Delay the adjustment; extra p's won't damage subsequent field operations.

nalysis

n: $m_n(x) = \sum m_n[i]x^i$.

pute $m_n(r)$ mod p:

$m_n[5]$ by r,

[4], multiply by r,

[3], multiply by r,

[2], multiply by r,

[1], multiply by r.

mod p after each mult.

more time to

e authenticator $a_n =$

mod $p) + s_n$ mod 1000000.

Reducing mod 1000003 is easy: e.g., $240881099091 =$ $240881 \cdot 1000000 + 99091 \equiv$ $240881(-3) + 99091 =$ $-722643 + 99091 =$ -623552.

Easily adjust to range $\{0, 1, \ldots, p-1\}$ by adding/subtracting a few p's. (Beware timing attacks!)

Speedup: Delay the adjustment; extra p's won't damage subsequent field operations.

Main wo

For each

have to

of the 6-

into an a

Scaled u

"Poly130

For each

have to

of a 128

into an a

$\approx$ 5 cycl

dependin

$= \sum m_n[i]x^i$.

r) mod p:

r,

ly by r,

ly by r,

ly by r,

ly by r.

er each mult.

to

cator $a_n =$

s_n mod 1000000.

Reducing mod 1000003 is easy: e.g., $240881099091 =$ $240881 \cdot 1000000 + 99091 \equiv$ $240881(-3) + 99091 =$ $-722643 + 99091 =$ -623552.

Easily adjust to range $\{0, 1, \ldots, p-1\}$ by adding/subtracting a few p's. (Beware timing attacks!)

Speedup: Delay the adjustment; extra p's won't damage subsequent field operations.

Main work is mult

For each 6-digit m

have to do one mu

of the 6-digit secre

into an accumulat

Scaled up for serio

"Poly1305" uses p

For each 128-bit m

have to do one mu

of a 128-bit secret

into an accumulat

$\approx$ 5 cycles per me

depending on the

$i]x^i$.

ult.

=

000000.

Reducing mod 1000003 is easy:
e.g., $240881099091 =$
$240881 \cdot 1000000 + 99091 \equiv$
$240881(-3) + 99091 =$
$-722643 + 99091 =$
-623552.

Easily adjust to range
$\{0, 1, \ldots, p - 1\}$
by adding/subtracting a few p's.
(Beware timing attacks!)

Speedup: Delay the adjustment;
extra p's won't damage
subsequent field operations.

Main work is multiplication.
For each 6-digit message ch
have to do one multiplicatio
of the 6-digit secret r
into an accumulator mod p.

Scaled up for serious securit
"Poly1305" uses $p = 2^{130} -$
For each 128-bit message ch
have to do one multiplicatio
of a 128-bit secret r
into an accumulator mod 2^1
$\approx$ 5 cycles per message byte
depending on the CPU.

Reducing mod 1000003 is easy:
e.g., $240881099091 =$
$240881 \cdot 1000000 + 99091 \equiv$
$240881(-3) + 99091 =$
$-722643 + 99091 =$
-623552.

Easily adjust to range
$\{0, 1, \ldots, p - 1\}$
by adding/subtracting a few p's.
(Beware timing attacks!)

Speedup: Delay the adjustment;
extra p's won't damage
subsequent field operations.

Main work is multiplication.
For each 6-digit message chunk,
have to do one multiplication
of the 6-digit secret r
into an accumulator mod p.

Scaled up for serious security:
"Poly1305" uses $p = 2^{130} - 5$.
For each 128-bit message chunk,
have to do one multiplication
of a 128-bit secret r
into an accumulator mod $2^{130} - 5$.
≈ 5 cycles per message byte,
depending on the CPU.

g mod 1000003 is easy:
0881099091 =
1000000 + 99091 ≡
−3) + 99091 =
3 + 99091 =
2.

djust to range
. , $p - 1\}$
ng/subtracting a few p's.
timing attacks!)

o: Delay the adjustment;
s won't damage
ent field operations.

Main work is multiplication.
For each 6-digit message chunk,
have to do one multiplication
of the 6-digit secret r
into an accumulator mod p.

Scaled up for serious security:
"Poly1305" uses $p = 2^{130} - 5$.
For each 128-bit message chunk,
have to do one multiplication
of a 128-bit secret r
into an accumulator mod $2^{130} - 5$.
$\approx$ 5 cycles per message byte,
depending on the CPU.

Security

Attacker
Find n',
$m' \neq m$
$(m'(r)$ r
Here m'

Obvious
Choose a
Choose r
Success

Can repe
Each for
1/10000

00003 is easy:
1 =
$+ 99091 \equiv$
091 =
=

nge

ting a few p's.
tacks!)

ne adjustment;
mage
perations.

Main work is multiplication.
For each 6-digit message chunk,
have to do one multiplication
of the 6-digit secret r
into an accumulator mod p.

Scaled up for serious security:
"Poly1305" uses $p = 2^{130} - 5$.
For each 128-bit message chunk,
have to do one multiplication
of a 128-bit secret r
into an accumulator mod $2^{130} - 5$.
$\approx$ 5 cycles per message byte,
depending on the CPU.

Security analysis

Attacker's goal:
Find n', m', a' suc
$m' \neq m_{n'}$ but a' =
$(m'(r) \bmod p) + s$
Here $m'(x) = \sum_i$

Obvious attack:
Choose any $m' \neq$
Choose uniform ra
Success chance 1/

Can repeat attack
Each forgery has c
1/1000000 of bein

asy:

≡

p's.

ent;

Main work is multiplication.
For each 6-digit message chunk, have to do one multiplication of the 6-digit secret r into an accumulator mod p.

Scaled up for serious security: "Poly1305" uses $p = 2^{130} - 5$.
For each 128-bit message chunk, have to do one multiplication of a 128-bit secret r into an accumulator mod $2^{130} - 5$.
$\approx$ 5 cycles per message byte, depending on the CPU.

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

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 accepte

Main work is multiplication.

For each 6-digit message chunk, have to do one multiplication of the 6-digit secret r into an accumulator mod p.

Scaled up for serious security: "Poly1305" uses $p = 2^{130} - 5$. For each 128-bit message chunk, have to do one multiplication of a 128-bit secret r into an accumulator mod $2^{130} - 5$. $\approx$ 5 cycles per message byte, depending on the CPU.

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.

ork is multiplication.
n 6-digit message chunk,
do one multiplication
-digit secret r
accumulator mod p.

up for serious security:
05" uses $p = 2^{130} - 5$.
n 128-bit message chunk,
do one multiplication
-bit secret r
accumulator mod $2^{130} - 5$.
es per message byte,
ng on the CPU.

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
$x \in \{0,$
modulo

e.g. m_1
$m' = (1$
$m'(x) -$
which ha
$0, 29901$

Success

iplication.
essage chunk,
ultiplication
et r
or mod p.

ous security:
$= 2^{130} - 5$.
message chunk,
ultiplication
r
or mod $2^{130} - 5$.
ssage byte,
CPU.

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

e.g. $m_1 = (100, 0,$
$m' = (125, 1, 0, 0,$
$m'(x) - m_1(x) =$
which has five roo
$0, 299012, 334447,$

Success chance 5/

unk,
n

y:
5.
hunk,
n

$^{30} - 5$.
e,

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 $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$,
$m' = (125, 1, 0, 0, 1)$:
$m'(x) - m_1(x) = x^5 + x^2 +$
which has five roots mod p:
$0, 299012, 334447, 631403, 7$

Success chance 5/1000000.

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 $a' =$

mod 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.

rgery 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

Example

$\in \{1000$

then a fo

$m'(x) =$

also succ

success c

Reason:

$m'(x) -$

Can hav

of $(m'(x$

$(m'(x) -$

$(m'(x) -$

ch that

=

${}_{m'}$ 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

Example: If m_1(3
$\in \{1000000, 10000$
then a forgery (1,
$m'(x) = m_1(x) +$
also succeeds for m
success chance 6/
Reason: 334885 is
$m'(x) - m_1(x) +$

Can have as many
of $(m'(x) - m_1(x$
$(m'(x) - m_1(x) +$
$(m'(x) - m_1(x) -$

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.

Example: If $m_1(334885)$ mo
$\in \{1000000, 1000001, 10000$
then a forgery $(1, m', a_1)$ wi
$m'(x) = m_1(x) + x^5 + x^2 +$
also succeeds for $r = 33488$
success chance 6/1000000.
Reason: 334885 is a root of
$m'(x) - m_1(x) + 1000000.$

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

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 $m_1(334885)$ 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 bette

No. Eas
of $(n', m$
has char
of being

Underlyi
of $(m'(x$
$(m'(x)$ -
$(m'(x)$ -

Warning
the over
$(m_n[1]$ -
$+ s_n$
solve m'

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 $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ 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 varyi

No. Easy to prove

of (n', m', a') with

has chance $\leq 15/$

of being accepted

Underlying fact: $\leq$

of $(m'(x) - m_1(x$

$(m'(x) - m_1(x) -$

$(m'(x) - m_1(x) -$

Warning: very eas

the oversimplified

$(m_n[1] + \cdots + m$

$+ s_n \bmod 1000$

solve $m'(x) - m_1$

(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'?

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

Underlying fact: ≤ 15 roots
of $(m'(x) - m_1(x) - a' + a$
$(m'(x) - m_1(x) - a' + a_1 +$
$(m'(x) - m_1(x) - a' + a_1 -$

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

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.

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 $m_1(334885)$ mod 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
$x) - m_1(x)) \cdot$
$- m_1(x) + 1000000) \cdot$
$- m_1(x) - 1000000)$.

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

Poly1305
with 22
Adds s_n

Assumin
Each for
$\leq 8\lceil L/1$
Probabil

D forger
with pro
$\geq 1 - 8$

e.g. 2^{64}
Pr[all re

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
)) ·
- 1000000) ·
- 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}

Assuming $\leq L$-byt
Each forgery succe
$\leq 8\lceil L/16\rceil$ choice
Probability $\leq 8\lceil L$

D forgeries are all
with probability
$\geq 1 - 8D\lceil L/16\rceil$ /

e.g. 2^{64} forgeries,
Pr[all rejected] $\geq$ (

od p

02}

th

$- 25x$

5;

.

ots

) ·

).

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 securit

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

Assuming $\leq L$-byte message
Each forgery succeeds for
$\leq 8 \lceil L/16 \rceil$ choices of r.
Probability $\leq 8 \lceil L/16 \rceil / 2^{10}$

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] ≥ 0.999999

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
$n', a')$ with $m' \neq m_{n'}$
nce $\leq 15/1000000$
accepted by receiver.

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

: 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

Split stri
maybe w
append
view as
in $\{1, 2,$
Multiply
add next
etc., last
mod 2^{13}

ng a'?

e: Every choice
n $m' \neq m_{n'}$
1000000
by receiver.

≤ 15 roots
$) - a' + a_1) \cdot$
$a' + a_1 + 10^6) \cdot$
$a' + a_1 - 10^6)$.

y to break
authenticator
$n[5]r^4 \bmod p)$
0000:
$(x) = a' - a_1$.

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

Split string into 16
maybe with smalle
append 1 to each
view as little-endia
in $\{1, 2, 3, \ldots, 2^{12}$
Multiply first chun
add next chunk, m
etc., last chunk, m
mod $2^{130} - 5$, add

hoice

n'

er.

$_1) \cdot$

$10^6) \cdot$

$- 10^6)$.

k

ator

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

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

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 chu
maybe with smaller final chu
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
etc., last chunk, multiply by
mod $2^{130} - 5$, add s_n mod

Scaled up for serious security:

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

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

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

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}.

up for serious security:

5 uses 128-bit r's,
bits cleared for speed.
mod 2^{128}.

g $\leq L$-byte messages:
rgery succeeds for
16⌉ choices of r.
ity $\leq 8 \lceil L/16 \rceil / 2^{106}$.

ries are all rejected
bability
$D \lceil L/16 \rceil / 2^{106}$.

forgeries, $L = 1536$:
jected] ≥ 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}.

Reducin

Like the
this auth
has a se

One-tim
L share
to encry

Authent
16 share
to authe

Each ne
new sha
used onl
How to

ous security:

3-bit r's,
ed for speed.
3.

e messages:
eeds for
s of r.
$/16\rceil / 2^{106}$.

rejected

$/2^{106}$.

$L = 1536$:
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}.

Reducing the key

Like the one-time
this authenticatio
has a security gua

One-time pad nee
L shared secret by
to encrypt L mess

Authentication sys
16 shared secret b
to authenticate L

Each new messag
new shared secret
used only once.
How to handle ma

y:

ed.

es:

^{5}.

998.

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}.

<u>Reducing the key length</u>

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message b

Each new message needs
new shared secret bytes,
used only once.
How to handle many messag

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}.

Reducing the key length

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message bytes.

Each new message needs
new shared secret bytes,
used only once.
How to handle many messages?

icator is still secure
ble-length messages,
nt messages are
polynomials mod p.

ing into 16-byte chunks,
vith smaller final chunk;
1 to each chunk;
little-endian integers
$3, \ldots, 2^{129}\}$.
first chunk by r,
t chunk, multiply by r,
t chunk, multiply by r,
$^{0} - 5$, add s_n mod 2^{128}.

Reducing the key length

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message bytes.

Each new message needs
new shared secret bytes,
used only once.
How to handle many messages?

Authent
encrypte

Can repl
with stre

Typical s
AES in c
Sender,
where k
compute

Security
since s_n
but can
attack o
implies a

till secure
messages,
ges are
als mod p.

6-byte chunks,
er final chunk;
chunk;
an integers
$^{9}\}$.
nk by r,
ultiply by r,
ultiply by r,
d s_n mod 2^{128}.

Reducing the key length

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message bytes.

Each new message needs
new shared secret bytes,
used only once.
How to handle many messages?

Authenticator is m
encrypted with on

Can replace one-ti
with stream-cipher

Typical stream cip
AES in counter m
Sender, receiver sh
where k is 16-byte
compute $s_n =$ AE

Security proof brea
since s_n's are dep
but can still prove
attack on authenti
implies attack on

,

.

nks,
unk;

s

r,
r,
2^{128}.

Reducing the key length

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message bytes.

Each new message needs
new shared secret bytes,
used only once.
How to handle many messages?

Authenticator is $m_n(r)$ mod
encrypted with one-time pad

Can replace one-time pad
with stream-cipher output.

Typical stream cipher:
AES in counter mode.
Sender, receiver share (r, k)
where k is 16-byte AES key;
compute $s_n = \text{AES}_k(n)$.

Security proof breaks down
since s_n's are dependent,
but can still prove that
attack on authenticator
implies attack on AES.

Reducing the key length

Like the one-time pad,
this authentication system
has a security guarantee.

One-time pad needs
L shared secret bytes
to encrypt L message bytes.

Authentication system needs
16 shared secret bytes
to authenticate L message bytes.

Each new message needs
new shared secret bytes,
used only once.
How to handle many messages?

Authenticator is $m_n(r) \bmod p$
encrypted with one-time pad s_n.

Can replace one-time pad
with stream-cipher output.

Typical stream cipher:
AES in counter mode.
Sender, receiver share (r, k)
where k is 16-byte AES key;
compute $s_n = \text{AES}_k(n)$.

Security proof breaks down
since s_n's are dependent,
but can still prove that
attack on authenticator
implies attack on AES.

g the key length

one-time pad,
nentication system
curity guarantee.

e pad needs
l secret bytes
pt L message bytes.

ication system needs
d secret bytes
enticate L message bytes.

w message needs
red secret bytes,
y once.
handle many messages?

Authenticator is $m_n(r) \bmod p$ encrypted with one-time pad s_n.

Can replace one-time pad with stream-cipher output.

Typical stream cipher: AES in counter mode. Sender, receiver share (r, k) where k is 16-byte AES key; compute $s_n = \text{AES}_k(n)$.

Security proof breaks down since s_n's are dependent, but can still prove that attack on authenticator implies attack on AES.

```
unsigned in
mpz_class r
for (j = 0;
  rbar += (
mpz_class h
mpz_class p
while (mlen
  mpz_class
  for (j =
    c += ((
  c += ((mp
  m += j; m
  h = ((h +
}
unsigned ch
aes(aeskn,k
for (j = 0;
  h += ((mp
for (j = 0;
  mpz_class
  h >>= 8;
  out[j] =
}
```

length

pad,
n system
rantee.

ds
tes
age bytes.

stem needs
ytes
message bytes.

e needs
bytes,

ny messages?

Authenticator is $m_n(r)$ mod p encrypted with one-time pad s_n.

Can replace one-time pad with stream-cipher output.

Typical stream cipher: AES in counter mode.
Sender, receiver share (r, k) where k is 16-byte AES key; compute $s_n = \text{AES}_k(n)$.

Security proof breaks down since s_n's are dependent, but can still prove that attack on authenticator implies attack on AES.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r
mpz_class h = 0;
mpz_class p = (((mpz_cla
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) &&
    c += ((mpz_class) m[
  c += ((mpz_class) 1) <
  m += j; mlen -= j;
  h = ((h + c) * rbar) %
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aesk
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

s

ytes.

ges?

Authenticator is $m_n(r)$ mod p encrypted with one-time pad s_n.

Can replace one-time pad with stream-cipher output.

Typical stream cipher: AES in counter mode. Sender, receiver share (r, k) where k is 16-byte AES key; compute $s_n = \text{AES}_k(n)$.

Security proof breaks down since s_n's are dependent, but can still prove that attack on authenticator implies attack on AES.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);+
    c += ((mpz_class) m[j]) << (8 * j
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 *
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

Authenticator is $m_n(r)$ mod p encrypted with one-time pad s_n.

Can replace one-time pad with stream-cipher output.

Typical stream cipher: AES in counter mode. Sender, receiver share (r, k) where k is 16-byte AES key; compute $s_n = \text{AES}_k(n)$.

Security proof breaks down since s_n's are dependent, but can still prove that attack on authenticator implies attack on AES.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j);
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130) - 5;
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);++j)
    c += ((mpz_class) m[j]) << (8 * j);
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 * j);
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

icator is $m_n(r)$ mod p
ed with one-time pad s_n.

ace one-time pad
eam-cipher output.

stream cipher:
counter mode.
receiver share (r, k)
is 16-byte AES key;
e $s_n = \text{AES}_k(n)$.

proof breaks down
's are dependent,
still prove that
n authenticator
attack on AES.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j);
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130) - 5;
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);++j)
    c += ((mpz_class) m[j]) << (8 * j);
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 * j);
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

Another
$F_k(n) =$
Somewh

"Hasn't
Distinct
with MD
(2004 W
Still not
$n \mapsto$ MD
We know

Many ot
are unbr

$n_n(r)$ mod p
e-time pad s_n.

me pad
r output.

pher:
ode.
hare (r, k)
e AES key;
$S_k(n)$.

aks down
endent,
that
icator
AES.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j);
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130) - 5;
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);++j)
    c += ((mpz_class) m[j]) << (8 * j);
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 * j);
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

Another stream ci
$F_k(n) = \text{MD5}(k, n$
Somewhat slower

"Hasn't MD5 bee
Distinct $(k, n), (k'$
with $\text{MD5}(k, n) =$
(2004 Wang)
Still not obvious h
$n \mapsto \text{MD5}(k, n)$ fo
We know AES col

Many other strean
are unbroken, fast

d p

d s_n.

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j);
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130) - 5;
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);++j)
    c += ((mpz_class) m[j]) << (8 * j);
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 * j);
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are k
with $\text{MD5}(k, n) = \text{MD5}(k',$
(2004 Wang)
Still not obvious how to pred
$n \mapsto \text{MD5}(k, n)$ for secret k
We know AES collisions too

Many other stream ciphers
are unbroken, faster than AE

```
unsigned int j;
mpz_class rbar = 0;
for (j = 0;j < 16;++j)
  rbar += ((mpz_class) r[j]) << (8 * j);
mpz_class h = 0;
mpz_class p = (((mpz_class) 1) << 130) - 5;
while (mlen > 0) {
  mpz_class c = 0;
  for (j = 0;(j < 16) && (j < mlen);++j)
    c += ((mpz_class) m[j]) << (8 * j);
  c += ((mpz_class) 1) << (8 * j);
  m += j; mlen -= j;
  h = ((h + c) * rbar) % p;
}
unsigned char aeskn[16];
aes(aeskn,k,n);
for (j = 0;j < 16;++j)
  h += ((mpz_class) aeskn[j]) << (8 * j);
for (j = 0;j < 16;++j) {
  mpz_class c = h % 256;
  h >>= 8;
  out[j] = c.get_ui();
}
```

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are known
with $\text{MD5}(k, n) = \text{MD5}(k', n')$.
(2004 Wang)
Still not obvious how to predict
$n \mapsto \text{MD5}(k, n)$ for secret k.
We know AES collisions too!

Many other stream ciphers
are unbroken, faster than AES.

```
t j;
bar = 0;
j < 16;++j)
(mpz_class) r[j]) << (8 * j);
 = 0;
 = (((mpz_class) 1) << 130) - 5;
 > 0) {
 c = 0;
0;(j < 16) && (j < mlen);++j)
mpz_class) m[j]) << (8 * j);
z_class) 1) << (8 * j);
len -= j;
 c) * rbar) % p;

ar aeskn[16];
,n);
j < 16;++j)
z_class) aeskn[j]) << (8 * j);
j < 16;++j) {
 c = h % 256;

c.get_ui();
```

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are known
with $\text{MD5}(k, n) = \text{MD5}(k', n')$.
(2004 Wang)
Still not obvious how to predict
$n \mapsto \text{MD5}(k, n)$ for secret k.
We know AES collisions too!

Many other stream ciphers
are unbroken, faster than AES.

Alternat

Use $\cdots$ e
instead o
No! Des
might al
even if A

Use AES
No! Bro
using $<$
But ok f

Use Sals
Seems to

```
[j]) << (8 * j);

ss) 1) << 130) - 5;

 (j < mlen);++j)
j]) << (8 * j);
< (8 * j);

 p;

n[j]) << (8 * j);
```

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are known with $\text{MD5}(k, n) = \text{MD5}(k', n')$.
(2004 Wang)
Still not obvious how to predict $n \mapsto \text{MD5}(k, n)$ for secret k.
We know AES collisions too!

Many other stream ciphers are unbroken, faster than AES.

Alternatives to +

Use $\cdots \oplus \text{AES}_k(n$
instead of $\cdots + \text{A}$
No! Destroys secu
might allow succes
even if AES is secu

Use $\text{AES}_k(\cdots)$, on
No! Broken by kn
using $< 2^{64}$ authe
But ok for small #

Use Salsa20$(k, n,$
Seems to be mass

```
j);

) - 5;

+j)
);

 j);
```

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are known with $\text{MD5}(k, n) = \text{MD5}(k', n')$.
(2004 Wang)
Still not obvious how to predict $n \mapsto \text{MD5}(k, n)$ for secret k.
We know AES collisions too!

Many other stream ciphers are unbroken, faster than AES.

Alternatives to $+$

Use $\cdots \oplus \text{AES}_k(n)$ instead of $\cdots + \text{AES}_k(n)$?
No! Destroys security analys
might allow successful forge
even if AES is secure.

Use $\text{AES}_k(\cdots)$, omitting n?
No! Broken by known attac
using $< 2^{64}$ authenticators.
But ok for small # message

Use $\text{Salsa20}(k, n, \cdots)$?
Seems to be massive overkil

Another stream cipher:
$F_k(n) = \text{MD5}(k, n)$.
Somewhat slower than AES.

"Hasn't MD5 been broken?"
Distinct $(k, n), (k', n')$ are known with $\text{MD5}(k, n) = \text{MD5}(k', n')$.
(2004 Wang)
Still not obvious how to predict $n \mapsto \text{MD5}(k, n)$ for secret k.
We know AES collisions too!

Many other stream ciphers are unbroken, faster than AES.

<u>Alternatives to $+$</u>

Use $\cdots \oplus \text{AES}_k(n)$ instead of $\cdots + \text{AES}_k(n)$?
No! Destroys security analysis; might allow successful forgeries even if AES is secure.

Use $\text{AES}_k(\cdots)$, omitting n?
No! Broken by known attacks using $< 2^{64}$ authenticators.
But ok for small # messages.

Use $\text{Salsa20}(k, n, \cdots)$?
Seems to be massive overkill.

stream cipher:

$= \text{MD5}(k, n)$.

at slower than AES.

MD5 been broken?”

$(k, n), (k', n')$ are known

D5$(k, n) = \text{MD5}(k', n')$.

/ang)

obvious how to predict

D5(k, n) for secret k.

w AES collisions too!

her stream ciphers

oken, faster than AES.

Alternatives to $+$

Use $\cdots \oplus \text{AES}_k(n)$ instead of $\cdots + \text{AES}_k(n)$?

No! Destroys security analysis; might allow successful forgeries even if AES is secure.

Use $\text{AES}_k(\cdots)$, omitting n?

No! Broken by known attacks using $< 2^{64}$ authenticators.

But ok for small # messages.

Use Salsa20$(k, n, \cdots)$?

Seems to be massive overkill.

Alternat

Notation

$(m(r)$ m

For all d

Pr[Poly1

Poly1

“Small c

For all d

and all 1

Pr[Poly1

Poly1

is very s

“Small c

pher:

n).

than AES.

n broken?"

$', n'$) are known

$\neq$ MD5(k', n').

ow to predict

or secret k.

lisions too!

n ciphers

er than AES.

Alternatives to +

Use $\cdots \oplus$ AES$_k(n)$
instead of $\cdots +$ AES$_k(n)$?
No! Destroys security analysis;
might allow successful forgeries
even if AES is secure.

Use AES$_k(\cdots)$, omitting n?
No! Broken by known attacks
using $< 2^{64}$ authenticators.
But ok for small # messages.

Use Salsa20($k, n, \cdots$)?
Seems to be massive overkill.

Alternatives to Po

Notation: Poly130
($m(r)$ mod 2^{130} −

For all distinct me
Pr[Poly1305$_r(m)$
Poly1305$_r(m')$

"Small collision pr

For all distinct me
and all 16-byte sec
Pr[Poly1305$_r(m)$
Poly1305$_r(m')$
is very small.
"Small differential

’

’

known

$n')$.

dict

.

!

ES.

Alternatives to $+$

Use $\cdots \oplus \text{AES}_k(n)$
instead of $\cdots + \text{AES}_k(n)$?
No! Destroys security analysis;
might allow successful forgeries
even if AES is secure.

Use $\text{AES}_k(\cdots)$, omitting n?
No! Broken by known attacks
using $< 2^{64}$ authenticators.
But ok for small # messages.

Use $\text{Salsa20}(k, n, \cdots)$?
Seems to be massive overkill.

Alternatives to Poly1305

Notation: $\text{Poly1305}_r(m) =$
$(m(r) \bmod 2^{130} - 5) \bmod 2$

For all distinct messages m,
$\Pr[\text{Poly1305}_r(m) =$
$\text{Poly1305}_r(m')]$ is very sm
“Small collision probabilities

For all distinct messages m,
and all 16-byte sequences Δ
$\Pr[\text{Poly1305}_r(m) =$
$\text{Poly1305}_r(m') + \Delta$ mod
is very small.
“Small differential probabilit

Alternatives to $+$

Use $\cdots \oplus \text{AES}_k(n)$
instead of $\cdots + \text{AES}_k(n)$?
No! Destroys security analysis;
might allow successful forgeries
even if AES is secure.

Use $\text{AES}_k(\cdots)$, omitting n?
No! Broken by known attacks
using $< 2^{64}$ authenticators.
But ok for small # messages.

Use Salsa20$(k, n, \cdots)$?
Seems to be massive overkill.

Alternatives to Poly1305

Notation: $\text{Poly1305}_r(m) = (m(r) \bmod 2^{130} - 5) \bmod 2^{128}$.

For all distinct messages m, m':
$\Pr[\text{Poly1305}_r(m) = \text{Poly1305}_r(m')]$ is very small.
"Small collision probabilities."

For all distinct messages m, m'
and all 16-byte sequences Δ:
$\Pr[\text{Poly1305}_r(m) = \text{Poly1305}_r(m') + \Delta \bmod 2^{128}]$
is very small.
"Small differential probabilities."

ives to +

$\oplus$ AES$_k(n)$
of $\cdots +$ AES$_k(n)$?
stroys security analysis;
llow successful forgeries
AES is secure.

S$_k(\cdots)$, omitting n?
oken by known attacks
2^{64} authenticators.
for small # messages.

sa20$(k, n, \cdots)$?
o be massive overkill.

Alternatives to Poly1305

Notation: Poly1305$_r(m) =$ $(m(r) \bmod 2^{130} - 5) \bmod 2^{128}$.

For all distinct messages m, m':
Pr[Poly1305$_r(m) =$
Poly1305$_r(m')$] is very small.
"Small collision probabilities."

For all distinct messages m, m' and all 16-byte sequences Δ:
Pr[Poly1305$_r(m) =$
Poly1305$_r(m') + \Delta \bmod 2^{128}$]
is very small.
"Small differential probabilities."

Easy to
that sati

Embed r
polynom

Use $m \vdash$
r is a ra

Small di
means th
is divisib
when m

(Additio
mod 2^{12}

)

ES$_k(n)$?

urity analysis;

ssful forgeries

ure.

nitting n?

own attacks

nticators.

$\neq$ messages.

$\cdots$)?

ive overkill.

Alternatives to Poly1305

Notation: Poly1305$_r(m) =$
$(m(r)$ mod $2^{130} - 5)$ mod 2^{128}.

For all distinct messages m, m':
Pr[Poly1305$_r(m) =$
Poly1305$_r(m')$] is very small.
"Small collision probabilities."

For all distinct messages m, m'
and all 16-byte sequences Δ:
Pr[Poly1305$_r(m) =$
Poly1305$_r(m') + \Delta$ mod 2^{128}]
is very small.
"Small differential probabilities."

Easy to build othe
that satisfy these

Embed messages a
polynomial ring **Z**

Use $m \mapsto m$ mod
r is a random prin

Small differential
means that $m - n$
is divisible by very
when $m \neq m'$.

(Addition of Δ is
mod 2^{128}; be care

sis;
ries

ks

s.

l.

Alternatives to Poly1305

Notation: $\text{Poly1305}_r(m) =$
$(m(r) \bmod 2^{130} - 5) \bmod 2^{128}$.

For all distinct messages m, m':
$\Pr[\text{Poly1305}_r(m) =$
$\text{Poly1305}_r(m')]$ is very small.
"Small collision probabilities."

For all distinct messages m, m'
and all 16-byte sequences Δ:
$\Pr[\text{Poly1305}_r(m) =$
$\text{Poly1305}_r(m') + \Delta \bmod 2^{128}]$
is very small.
"Small differential probabilities."

Easy to build other function
that satisfy these properties

Embed messages and outpu
polynomial ring $\mathbf{Z}[x_1, x_2, x_3$

Use $m \mapsto m \bmod r$ where
r is a random prime ideal.

Small differential probability
means that $m - m' - \Delta$
is divisible by very few r's
when $m \neq m'$.

(Addition of Δ is
mod 2^{128}; be careful.)

Alternatives to Poly1305

Notation: $\text{Poly1305}_r(m) =$
$(m(r) \bmod 2^{130} - 5) \bmod 2^{128}$.

For all distinct messages m, m':
$\Pr[\text{Poly1305}_r(m) =$
$\quad \text{Poly1305}_r(m')]$ is very small.
"Small collision probabilities."

For all distinct messages m, m'
and all 16-byte sequences Δ:
$\Pr[\text{Poly1305}_r(m) =$
$\quad \text{Poly1305}_r(m') + \Delta \bmod 2^{128}]$
is very small.
"Small differential probabilities."

Easy to build other functions
that satisfy these properties.

Embed messages and outputs into
polynomial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

Use $m \mapsto m \bmod r$ where
r is a random prime ideal.

Small differential probability
means that $m - m' - \Delta$
is divisible by very few r's
when $m \neq m'$.

(Addition of Δ is
mod 2^{128}; be careful.)

ives to Poly1305

n: Poly1305$_r(m)$ =
nod $2^{130} - 5$) mod 2^{128}.

istinct messages m, m':
305$_r(m)$ =
305$_r(m')$] is very small.
collision probabilities."

istinct messages m, m'
6-byte sequences Δ:
305$_r(m)$ =
305$_r(m') + \Delta$ mod 2^{128}]
mall.
lifferential probabilities."

Easy to build other functions that satisfy these properties.

Embed messages and outputs into polynomial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

Use $m \mapsto m$ mod r where r is a random prime ideal.

Small differential probability means that $m - m' - \Delta$ is divisible by very few r's when $m \neq m'$.

(Addition of Δ is mod 2^{128}; be careful.)

Example

View me
specifica
Outputs

Reduce
random
between
(Problen

Low diffe
if $m \neq$
so $m -$
by very

ly1305

$05_r(m) =$
$5)$ mod 2^{128}.

ssages m, m':
$=$
] is very small.
obabilities."

ssages m, m'
quences Δ:
$=$
$+ \Delta$ mod 2^{128}]

probabilities."

Easy to build other functions that satisfy these properties.

Embed messages and outputs into polynomial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

Use $m \mapsto m \bmod r$ where r is a random prime ideal.

Small differential probability means that $m - m' - \Delta$ is divisible by very few r's when $m \neq m'$.

(Addition of Δ is mod 2^{128}; be careful.)

Example: (1981 K

View messages m
specifically multipl
Outputs: $\{0, 1, \ldots$

Reduce m modulo
random prime nun
between 2^{120} and
(Problem: generat

Low differential pr
if $m \neq m'$ then m
so $m - m' - \Delta$ is
by very few prime

^{128}.

m':

mall.

."

m'

:

2^{128}]

ies."

Easy to build other functions that satisfy these properties.

Embed messages and outputs into polynomial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

Use $m \mapsto m \bmod r$ where r is a random prime ideal.

Small differential probability means that $m - m' - \Delta$ is divisible by very few r's when $m \neq m'$.

(Addition of Δ is mod 2^{128}; be careful.)

Example: (1981 Karp Rabin

View messages m as integer
specifically multiples of 2^{128}
Outputs: $\{0, 1, \ldots, 2^{128} - 1$

Reduce m modulo a uniform
random prime number r
between 2^{120} and 2^{128}.
(Problem: generating r is sl

Low differential probability:
if $m \neq m'$ then $m - m' - \Delta$
so $m - m' - \Delta$ is divisible
by very few prime numbers.

Easy to build other functions that satisfy these properties.

Embed messages and outputs into polynomial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

Use $m \mapsto m \bmod r$ where r is a random prime ideal.

Small differential probability means that $m - m' - \Delta$ is divisible by very few r's when $m \neq m'$.

(Addition of Δ is mod 2^{128}; be careful.)

Example: (1981 Karp Rabin)

View messages m as integers, specifically multiples of 2^{128}.
Outputs: $\{0, 1, \ldots, 2^{128} - 1\}$.

Reduce m modulo a uniform random prime number r between 2^{120} and 2^{128}.
(Problem: generating r is slow.)

Low differential probability:
if $m \neq m'$ then $m - m' - \Delta \neq 0$ so $m - m' - \Delta$ is divisible by very few prime numbers.

build other functions
isfy these properties.

messages and outputs into
nial ring $\mathbf{Z}[x_1, x_2, x_3, \ldots]$.

$\to m$ mod r where
ndom prime ideal.

fferential probability
hat $m - m' - \Delta$
le by very few r's
$\neq m'$.

n of Δ is
8; be careful.)

Example: (1981 Karp Rabin)

View messages m as integers,
specifically multiples of 2^{128}.
Outputs: $\{0, 1, \ldots, 2^{128} - 1\}$.

Reduce m modulo a uniform
random prime number r
between 2^{120} and 2^{128}.
(Problem: generating r is slow.)

Low differential probability:
if $m \neq m'$ then $m - m' - \Delta \neq 0$
so $m - m' - \Delta$ is divisible
by very few prime numbers.

Variant

View me
$m_{128} x^{12}$
with eac

Outputs
with eac

Reduce
r is a ur
degree-1
(Problem
typical C
for polyr

er functions
properties.

and outputs into
$[x_1, x_2, x_3, \ldots]$.

r where
ne ideal.

probability
$n' - \Delta$
few r's

ful.)

Example: (1981 Karp Rabin)

View messages m as integers, specifically multiples of 2^{128}.
Outputs: $\{0, 1, \ldots, 2^{128} - 1\}$.

Reduce m modulo a uniform random prime number r between 2^{120} and 2^{128}.
(Problem: generating r is slow.)

Low differential probability:
if $m \neq m'$ then $m - m' - \Delta \neq 0$
so $m - m' - \Delta$ is divisible by very few prime numbers.

Variant that works
View messages m
$m_{128}x^{128} + m_{129}$
with each m_i in $\{$

Outputs: $o_0 + o_1 x$
with each o_i in $\{0$

Reduce m modulo
r is a uniform rand
degree-128 polyno
(Problem: division
typical CPU has n
for polynomial mu

s

ts into

, . . .].

Example: (1981 Karp Rabin)

View messages m as integers, specifically multiples of 2^{128}. Outputs: $\{0, 1, \ldots, 2^{128} - 1\}$.

Reduce m modulo a uniform random prime number r between 2^{120} and 2^{128}. (Problem: generating r is slow.)

Low differential probability: if $m \neq m'$ then $m - m' - \Delta \neq 0$ so $m - m' - \Delta$ is divisible by very few prime numbers.

Variant that works with $\oplus$:

View messages m as polyno
$m_{128}x^{128} + m_{129}x^{129} + \cdots$
with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1 x + \cdots + o_1$
with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ wher
r is a uniform random irredu
degree-128 polynomial over
(Problem: division by r is sl
typical CPU has no big circu
for polynomial multiplicatio

Example: (1981 Karp Rabin)

View messages m as integers, specifically multiples of 2^{128}. Outputs: $\{0, 1, \ldots, 2^{128} - 1\}$.

Reduce m modulo a uniform random prime number r between 2^{120} and 2^{128}. (Problem: generating r is slow.)

Low differential probability: if $m \neq m'$ then $m - m' - \Delta \neq 0$ so $m - m' - \Delta$ is divisible by very few prime numbers.

Variant that works with $\oplus$:

View messages m as polynomials $m_{128}x^{128} + m_{129}x^{129} + \cdots$ with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1 x + \cdots + o_{127}x^{127}$ with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ where r is a uniform random irreducible degree-128 polynomial over $\mathbf{Z}/2$. (Problem: division by r is slow; typical CPU has no big circuit for polynomial multiplication.)

e: (1981 Karp Rabin)

essages m as integers,
lly multiples of 2^{128}.
: $\{0, 1, \ldots, 2^{128} - 1\}$.

m modulo a uniform
prime number r
2^{120} and 2^{128}.
n: generating r is slow.)

erential probability:
n' then $m - m' - \Delta \neq 0$
$m' - \Delta$ is divisible
few prime numbers.

Variant that works with $\oplus$:

View messages m as polynomials $m_{128}x^{128} + m_{129}x^{129} + \cdots$ with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1 x + \cdots + o_{127}x^{127}$ with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ where r is a uniform random irreducible degree-128 polynomial over $\mathbf{Z}/2$. (Problem: division by r is slow; typical CPU has no big circuit for polynomial multiplication.)

Example
MacWill

Choose
View me
polys m
$m_1, m_2,$
Outputs

Reduce
$p, x_1 -$
to $m_1 r_1$
(Problem

Karp Rabin)

as integers,
les of 2^{128}.
$\ldots, 2^{128} - 1\}$.

a uniform
nber r
2^{128}.
ting r is slow.)

obability:
$- m' - \Delta \neq 0$
divisible
numbers.

Variant that works with $\oplus$:

View messages m as polynomials
$m_{128}x^{128} + m_{129}x^{129} + \cdots$
with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1x + \cdots + o_{127}x^{127}$
with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ where
r is a uniform random irreducible
degree-128 polynomial over $\mathbf{Z}/2$.
(Problem: division by r is slow;
typical CPU has no big circuit
for polynomial multiplication.)

Example: (1974 G
MacWilliams Sloa

Choose prime num
View messages m
polys $m_1x_1 + m_2$
$m_1, m_2, m_3 \in \{0,$
Outputs: $\{0, \ldots,$

Reduce m modulo
$p, x_1 - r_1, x_2 - r_2$
to $m_1r_1 + m_2r_2$
(Problem: long m

)

rs,

.

}.

n

ow.)

$\Delta \neq 0$

Variant that works with $\oplus$:

View messages m as polynomials $m_{128}x^{128} + m_{129}x^{129} + \cdots$ with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1 x + \cdots + o_{127} x^{127}$ with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ where r is a uniform random irreducible degree-128 polynomial over $\mathbf{Z}/2$. (Problem: division by r is slow; typical CPU has no big circuit for polynomial multiplication.)

Example: (1974 Gilbert MacWilliams Sloane)

Choose prime number $p \approx 2$

View messages m as linear

polys $m_1x_1 + m_2x_2 + m_3x$

$m_1, m_2, m_3 \in \{0, \ldots, p - 1$

Outputs: $\{0, \ldots, p - 1\}$.

Reduce m modulo

$p, x_1 - r_1, x_2 - r_2, x_3 - r_3$

to $m_1r_1 + m_2r_2 + m_3r_3$ m

(Problem: long m needs lon

Variant that works with $\oplus$:

View messages m as polynomials $m_{128}x^{128} + m_{129}x^{129} + \cdots$ with each m_i in $\{0, 1\}$.

Outputs: $o_0 + o_1 x + \cdots + o_{127}x^{127}$ with each o_i in $\{0, 1\}$.

Reduce m modulo $2, r$ where r is a uniform random irreducible degree-128 polynomial over $\mathbf{Z}/2$. (Problem: division by r is slow; typical CPU has no big circuit for polynomial multiplication.)

Example: (1974 Gilbert MacWilliams Sloane)

Choose prime number $p \approx 2^{128}$. View messages m as linear polys $m_1x_1 + m_2x_2 + m_3x_3$ with $m_1, m_2, m_3 \in \{0, \ldots, p-1\}$. Outputs: $\{0, \ldots, p-1\}$.

Reduce m modulo $p, x_1 - r_1, x_2 - r_2, x_3 - r_3$ to $m_1r_1 + m_2r_2 + m_3r_3 \bmod p$. (Problem: long m needs long r.)

that works with ⊕:

essages m as polynomials
$^{28} + m_{129}x^{129} + \cdots$
ch m_i in $\{0, 1\}$.

: $o_0 + o_1 x + \cdots + o_{127} x^{127}$
ch o_i in $\{0, 1\}$.

m modulo $2, r$ where
niform random irreducible
128 polynomial over $\mathbf{Z}/2$.
m: division by r is slow;
CPU has no big circuit
nomial multiplication.)

Example: (1974 Gilbert
MacWilliams Sloane)

Choose prime number $p \approx 2^{128}$.
View messages m as linear
polys $m_1 x_1 + m_2 x_2 + m_3 x_3$ with
$m_1, m_2, m_3 \in \{0, \ldots, p-1\}$.
Outputs: $\{0, \ldots, p-1\}$.

Reduce m modulo
$p, x_1 - r_1, x_2 - r_2, x_3 - r_3$
to $m_1 r_1 + m_2 r_2 + m_3 r_3 \bmod p$.
(Problem: long m needs long r.)

Example
independ
independ
Johansso

Choose
View me
$m_1 x + $
$m_1, m_2,$
Outputs

Reduce
where r
element
compute

s with $\oplus$:

as polynomials
$x^{129} + \cdots$
$0, 1\}$.

$+ \cdots + o_{127}x^{127}$
$, 1\}$.

$2, r$ where
dom irreducible
mial over $\mathbf{Z}/2$.
by r is slow;
o big circuit
ltiplication.)

Example: (1974 Gilbert MacWilliams Sloane)

Choose prime number $p \approx 2^{128}$.
View messages m as linear polys $m_1x_1 + m_2x_2 + m_3x_3$ with $m_1, m_2, m_3 \in \{0, \ldots, p-1\}$.
Outputs: $\{0, \ldots, p-1\}$.

Reduce m modulo $p, x_1 - r_1, x_2 - r_2, x_3 - r_3$ to $m_1r_1 + m_2r_2 + m_3r_3 \bmod p$.
(Problem: long m needs long r.)

Example: (1993 d
independently 199
independently 199
Johansson Kabati

Choose prime num
View messages m
$m_1x + m_2x^2 + m$
$m_1, m_2, \ldots \in \{0,$
Outputs: $\{0, 1, \ldots$

Reduce m modulo
where r is a unifo
element of $\{0, 1, .$
compute $m_1r + m$

mials

$_{27}x^{127}$

re

ucible

$\mathbf{Z}/2$.

ow;

uit

n.)

Example: (1974 Gilbert MacWilliams Sloane)

Choose prime number $p \approx 2^{128}$. View messages m as linear polys $m_1 x_1 + m_2 x_2 + m_3 x_3$ with $m_1, m_2, m_3 \in \{0, \ldots, p-1\}$. Outputs: $\{0, \ldots, p-1\}$.

Reduce m modulo $p, x_1 - r_1, x_2 - r_2, x_3 - r_3$ to $m_1 r_1 + m_2 r_2 + m_3 r_3 \bmod p$. (Problem: long m needs long r.)

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrau Johansson Kabatianskii Sme

Choose prime number $p \approx 2$ View messages m as polyno $m_1 x + m_2 x^2 + m_3 x^3 + \cdots$ $m_1, m_2, \ldots \in \{0, 1, \ldots, p -$ Outputs: $\{0, 1, \ldots, p-1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p-1\}$ compute $m_1 r + m_2 r^2 + \cdots$ r

Example: (1974 Gilbert MacWilliams Sloane)

Choose prime number $p \approx 2^{128}$. View messages m as linear polys $m_1 x_1 + m_2 x_2 + m_3 x_3$ with $m_1, m_2, m_3 \in \{0, \ldots, p-1\}$. Outputs: $\{0, \ldots, p-1\}$.

Reduce m modulo $p, x_1 - r_1, x_2 - r_2, x_3 - r_3$ to $m_1 r_1 + m_2 r_2 + m_3 r_3 \bmod p$. (Problem: long m needs long r.)

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrauer Johansson Kabatianskii Smeets)

Choose prime number $p \approx 2^{128}$. View messages m as polynomials $m_1 x + m_2 x^2 + m_3 x^3 + \cdots$ with $m_1, m_2, \ldots \in \{0, 1, \ldots, p-1\}$. Outputs: $\{0, 1, \ldots, p-1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p-1\}$; i.e., compute $m_1 r + m_2 r^2 + \cdots \bmod p$.

e: (1974 Gilbert
iams Sloane)

prime number $p \approx 2^{128}$.
essages m as linear
$_1x_1 + m_2x_2 + m_3x_3$ with
$m_3 \in \{0,\ldots,p-1\}$.
: $\{0,\ldots,p-1\}$.

m modulo
$r_1, x_2 - r_2, x_3 - r_3$
$+ m_2r_2 + m_3r_3$ mod p.
n: long m needs long r.)

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrauer Johansson Kabatianskii Smeets)

Choose prime number $p \approx 2^{128}$. View messages m as polynomials $m_1x + m_2x^2 + m_3x^3 + \cdots$ with $m_1, m_2, \ldots \in \{0, 1, \ldots, p-1\}$. Outputs: $\{0, 1, \ldots, p-1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p-1\}$; i.e., compute $m_1r + m_2r^2 + \cdots$ mod p.

“hash12
$p = 2^{127}$

“PolyR”
$p = 2^{64}$
between
to achie
(2000 K

“Poly130
$p = 2^{130}$
fully dev

“CWC”:
(2003 K

ilbert
ne)

ber $p \approx 2^{128}$.
as linear
$x_2 + m_3 x_3$ with
$\ldots, p - 1\}$.
$p - 1\}$.

$_2, x_3 - r_3$
$+ m_3 r_3$ mod p.
needs long r.)

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrauer Johansson Kabatianskii Smeets)

Choose prime number $p \approx 2^{128}$. View messages m as polynomials $m_1 x + m_2 x^2 + m_3 x^3 + \cdots$ with $m_1, m_2, \ldots \in \{0, 1, \ldots, p - 1\}$. Outputs: $\{0, 1, \ldots, p - 1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p - 1\}$; i.e., compute $m_1 r + m_2 r^2 + \cdots$ mod p.

“hash127”: 32-bit
$p = 2^{127} - 1$. (19

“PolyR”: 64-bit m
$p = 2^{64} - 59$; re-e
between p and 2^{64}
to achieve reasona
(2000 Krovetz Rog

“Poly1305”: 128-b
$p = 2^{130} - 5$. (20
fully developed in

“CWC”: 96-bit m
(2003 Kohno Vieg

^{128}.

$_3$ with

}.

od p.

g r.)

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrauer Johansson Kabatianskii Smeets)

Choose prime number $p \approx 2^{128}$. View messages m as polynomials $m_1x + m_2x^2 + m_3x^3 + \cdots$ with $m_1, m_2, \ldots \in \{0, 1, \ldots, p-1\}$. Outputs: $\{0, 1, \ldots, p-1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p-1\}$; i.e., compute $m_1r + m_2r^2 + \cdots \bmod p$.

"hash127": 32-bit m_i's, $p = 2^{127} - 1$. (1999 Bernste

"PolyR": 64-bit m_i's, $p = 2^{64} - 59$; re-encode m_i between p and $2^{64} - 1$; run to achieve reasonable securit (2000 Krovetz Rogaway)

"Poly1305": 128-bit m_i's, $p = 2^{130} - 5$. (2002 Bernste fully developed in 2004–200

"CWC": 96-bit m_i's, $p = 2^1$ (2003 Kohno Viega Whiting

Example: (1993 den Boer; independently 1994 Taylor; independently 1994 Bierbrauer Johansson Kabatianskii Smeets)

Choose prime number $p \approx 2^{128}$. View messages m as polynomials $m_1 x + m_2 x^2 + m_3 x^3 + \cdots$ with $m_1, m_2, \ldots \in \{0, 1, \ldots, p-1\}$. Outputs: $\{0, 1, \ldots, p-1\}$.

Reduce m modulo $p, x - r$ where r is a uniform random element of $\{0, 1, \ldots, p-1\}$; i.e., compute $m_1 r + m_2 r^2 + \cdots \bmod p$.

"hash127": 32-bit m_i's, $p = 2^{127} - 1$. (1999 Bernstein)

"PolyR": 64-bit m_i's, $p = 2^{64} - 59$; re-encode m_i's between p and $2^{64} - 1$; run twice to achieve reasonable security. (2000 Krovetz Rogaway)

"Poly1305": 128-bit m_i's, $p = 2^{130} - 5$. (2002 Bernstein, fully developed in 2004–2005)

"CWC": 96-bit m_i's, $p = 2^{127} - 1$. (2003 Kohno Viega Whiting)

e: (1993 den Boer;
dently 1994 Taylor;
dently 1994 Bierbrauer
on Kabatianskii Smeets)

prime number $p \approx 2^{128}$.
essages m as polynomials
$m_2 x^2 + m_3 x^3 + \cdots$ with
$\ldots \in \{0, 1, \ldots, p-1\}$.
: $\{0, 1, \ldots, p-1\}$.

m modulo $p, x - r$
is a uniform random
of $\{0, 1, \ldots, p-1\}$; i.e.,
e $m_1 r + m_2 r^2 + \cdots \bmod p$.

"hash127": 32-bit m_i's,
$p = 2^{127} - 1$. (1999 Bernstein)

"PolyR": 64-bit m_i's,
$p = 2^{64} - 59$; re-encode m_i's
between p and $2^{64} - 1$; run twice
to achieve reasonable security.
(2000 Krovetz Rogaway)

"Poly1305": 128-bit m_i's,
$p = 2^{130} - 5$. (2002 Bernstein,
fully developed in 2004–2005)

"CWC": 96-bit m_i's, $p = 2^{127} - 1$.
(2003 Kohno Viega Whiting)

There ar
build fu
proven c
different

Example
("CBC"
Conjectu
AES$_r$(A
has sma
True if A

(Much s

en Boer;
4 Taylor;
4 Bierbrauer
anskii Smeets)

nber $p \approx 2^{128}$.
as polynomials
$_3x^3 + \cdots$ with
$1, \ldots, p-1\}$.
$., p-1\}$.

$p, x - r$
rm random
$\ldots, p-1\}$; i.e.,
$_2r^2 + \cdots \bmod p$.

"hash127": 32-bit m_i's,
$p = 2^{127} - 1$. (1999 Bernstein)

"PolyR": 64-bit m_i's,
$p = 2^{64} - 59$; re-encode m_i's
between p and $2^{64} - 1$; run twice
to achieve reasonable security.
(2000 Krovetz Rogaway)

"Poly1305": 128-bit m_i's,
$p = 2^{130} - 5$. (2002 Bernstein,
fully developed in 2004–2005)

"CWC": 96-bit m_i's, $p = 2^{127} - 1$.
(2003 Kohno Viega Whiting)

There are other wa
build functions wit
proven or conjectu
differential probab

Example:
("CBC": "cipher b
Conjecturally m_1,
$\mathrm{AES}_r(\mathrm{AES}_r(\mathrm{AES}_r$
has small different
True if AES is secu

(Much slower than

uer
eets)
2^{128}.
mials
with
1}.

n
; i.e.,
mod p.

"hash127": 32-bit m_i's, $p = 2^{127} - 1$. (1999 Bernstein)

"PolyR": 64-bit m_i's, $p = 2^{64} - 59$; re-encode m_i's between p and $2^{64} - 1$; run twice to achieve reasonable security. (2000 Krovetz Rogaway)

"Poly1305": 128-bit m_i's, $p = 2^{130} - 5$. (2002 Bernstein, fully developed in 2004–2005)

"CWC": 96-bit m_i's, $p = 2^{127} - 1$. (2003 Kohno Viega Whiting)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
("CBC": "cipher block chai
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2$
has small differential probab
True if AES is secure.

(Much slower than Poly1305

“hash127”: 32-bit m_i’s,
$p = 2^{127} - 1$. (1999 Bernstein)

“PolyR”: 64-bit m_i’s,
$p = 2^{64} - 59$; re-encode m_i’s between p and $2^{64} - 1$; run twice to achieve reasonable security. (2000 Krovetz Rogaway)

“Poly1305”: 128-bit m_i’s,
$p = 2^{130} - 5$. (2002 Bernstein, fully developed in 2004–2005)

“CWC”: 96-bit m_i’s, $p = 2^{127} - 1$.
(2003 Kohno Viega Whiting)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
(“CBC”: “cipher block chaining”)
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2) \oplus m_3)$
has small differential probabilities.
True if AES is secure.

(Much slower than Poly1305.)

7": 32-bit m_i's,
$^7 - 1$. (1999 Bernstein)

: 64-bit m_i's,
$- 59$; re-encode m_i's
p and $2^{64} - 1$; run twice
ve reasonable security.
rovetz Rogaway)

05": 128-bit m_i's,
$^0 - 5$. (2002 Bernstein,
veloped in 2004–2005)

96-bit m_i's, $p = 2^{127} - 1$.
ohno Viega Whiting)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
("CBC": "cipher block chaining")
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2) \oplus m_3)$
has small differential probabilities.
True if AES is secure.

(Much slower than Poly1305.)

Example
Conjectu
$\text{AES}_r(1,$
$\text{AES}_r(2,$
$\text{AES}_r(3,$
has sma

(Even sl

Example
is conjec
small co

(Faster t
but not
and "sm

m_i's,
99 Bernstein)

$_i$'s,
ncode m_i's
$^4 - 1$; run twice
ble security.
gaway)

bit m_i's,
02 Bernstein,
2004–2005)

$_i$'s, $p = 2^{127} - 1$.
a Whiting)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
("CBC": "cipher block chaining")
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2) \oplus m_3)$
has small differential probabilities.
True if AES is secure.

(Much slower than Poly1305.)

Example: (1970 Z
Conjecturally m_1,
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small different

(Even slower.)

Example: $m \mapsto$ M
is conjectured to h
small collision prol

(Faster than AES,
but not as fast as
and "small" is deb

ein)

's
twice
ty.

ein,
5)
$^{127}-1$.
)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
("CBC": "cipher block chaining")
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2) \oplus m_3)$
has small differential probabilities.
True if AES is secure.

(Much slower than Poly1305.)

Example: (1970 Zobrist, ada
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small differential probab

(Even slower.)

Example: $m \mapsto \text{MD5}(r, m)$
is conjectured to have
small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

There are other ways to build functions with small proven or conjectured differential probabilities.

Example:
("CBC": "cipher block chaining")
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2) \oplus m_3)$
has small differential probabilities.
True if AES is secure.

(Much slower than Poly1305.)

Example: (1970 Zobrist, adapted)
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small differential probabilities.

(Even slower.)

Example: $m \mapsto \text{MD5}(r, m)$
is conjectured to have small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

re other ways to
nctions with small
or conjectured
ial probabilities.

e:
: "cipher block chaining")
urally $m_1, m_2, m_3 \mapsto$
ES$_r$(AES$_r$(m_1)$\oplus m_2$)$\oplus m_3$)
ll differential probabilities.
AES is secure.

lower than Poly1305.)

Example: (1970 Zobrist, adapted)
Conjecturally $m_1, m_2, m_3 \mapsto$
AES$_r(1, m_1) \oplus$
AES$_r(2, m_2) \oplus$
AES$_r(3, m_3)$
has small differential probabilities.

(Even slower.)

Example: $m \mapsto$ MD5(r, m)
is conjectured to have
small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

How to

1. Choo
$h(m) +$
or $f(h($
or $f(n,$

2. Choo
where th
(+-differ
or collisi
e.g., Pol

3. Choo
that see
from uni

ays to
th small
ured
ilities.

olock chaining")
$m_2, m_3 \mapsto$
$(m_1) \oplus m_2) \oplus m_3)$
ial probabilities.
ure.

n Poly1305.)

Example: (1970 Zobrist, adapted)
Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small differential probabilities.

(Even slower.)

Example: $m \mapsto \text{MD5}(r, m)$
is conjectured to have
small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

How to build your

1. Choose a comb
$h(m) + f(n)$ or h
or $f(h(m))$—wors
or $f(n, h(m))$—bi

2. Choose a rando
where the appropr
(+-differential or
or collision or colli
e.g., Poly1305_r.

3. Choose a rando
that seems indistin
from uniform: e.g.

ning")

$\mapsto$

$)\oplus m_3)$

ilities.

5.)

Example: (1970 Zobrist, adapted)

Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small differential probabilities.

(Even slower.)

Example: $m \mapsto \text{MD5}(r, m)$
is conjectured to have
small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

How to build your own MAC

1. Choose a combination m
$h(m) + f(n)$ or $h(m) \oplus f($
or $f(h(m))$—worse security
or $f(n, h(m))$—bigger f inp

2. Choose a random functio
where the appropriate proba
(+-differential or $\oplus$-different
or collision or collision) is sm
e.g., Poly1305_r.

3. Choose a random functio
that seems indistinguishable
from uniform: e.g., AES_k.

Example: (1970 Zobrist, adapted)

Conjecturally $m_1, m_2, m_3 \mapsto$
$\text{AES}_r(1, m_1) \oplus$
$\text{AES}_r(2, m_2) \oplus$
$\text{AES}_r(3, m_3)$
has small differential probabilities.

(Even slower.)

Example: $m \mapsto \text{MD5}(r, m)$
is conjectured to have
small collision probabilities.

(Faster than AES,
but not as fast as Poly1305,
and "small" is debatable.)

How to build your own MAC

1. Choose a combination method:
$h(m) + f(n)$ or $h(m) \oplus f(n)$
or $f(h(m))$—worse security—
or $f(n, h(m))$—bigger f input.

2. Choose a random function h
where the appropriate probability
($+$-differential or $\oplus$-differential
or collision or collision) is small:
e.g., Poly1305_r.

3. Choose a random function f
that seems indistinguishable
from uniform: e.g., AES_k.

e: (1970 Zobrist, adapted)
urally $m_1, m_2, m_3 \mapsto$
$m_1) \oplus$
$m_2) \oplus$
$m_3)$
ll differential probabilities.

ower.)

e: $m \mapsto$ MD5(r, m)
ctured to have
llision probabilities.

than AES,
as fast as Poly1305,
all" is debatable.)

How to build your own MAC

1. Choose a combination method: $h(m) + f(n)$ or $h(m) \oplus f(n)$ or $f(h(m))$—worse security— or $f(n, h(m))$—bigger f input.

2. Choose a random function h where the appropriate probability (+-differential or $\oplus$-differential or collision or collision) is small: e.g., Poly1305$_r$.

3. Choose a random function f that seems indistinguishable from uniform: e.g., AES$_k$.

4. Optic
Generate
e.g., $k =$
or $k =$ N
many me

5. Choo
for your

6. Put i

7. Publi

obrist, adapted)
$m_2, m_3 \mapsto$
ial probabilities.
D5(r, m)
have
babilities.
Poly1305,
batable.)

How to build your own MAC

1. Choose a combination method: $h(m) + f(n)$ or $h(m) \oplus f(n)$ or $f(h(m))$—worse security— or $f(n, h(m))$—bigger f input.

2. Choose a random function h where the appropriate probability ($+$-differential or $\oplus$-differential or collision or collision) is small: e.g., Poly1305$_r$.

3. Choose a random function f that seems indistinguishable from uniform: e.g., AES$_k$.

4. Optional compl
Generate k, r from
e.g., $k =$ AES$_s(0)$
or $k =$ MD5(s), r
many more possib

5. Choose a Goog
for your MAC.

6. Put it all toget

7. Publish!

apted)

→

ilities.

How to build your own MAC

1. Choose a combination method:
$h(m) + f(n)$ or $h(m) \oplus f(n)$
or $f(h(m))$—worse security—
or $f(n, h(m))$—bigger f input.

2. Choose a random function h
where the appropriate probability
(+-differential or $\oplus$-differential
or collision or collision) is small:
e.g., Poly1305$_r$.

3. Choose a random function f
that seems indistinguishable
from uniform: e.g., AES$_k$.

4. Optional complication:
Generate k, r from a shorter
e.g., $k =$ AES$_s(0)$, $r =$ AES
or $k =$ MD5(s), $r =$ MD5$(s$
many more possibilities.

5. Choose a Googleable nan
for your MAC.

6. Put it all together.

7. Publish!

How to build your own MAC

1. Choose a combination method: $h(m) + f(n)$ or $h(m) \oplus f(n)$ or $f(h(m))$—worse security— or $f(n, h(m))$—bigger f input.

2. Choose a random function h where the appropriate probability (+-differential or $\oplus$-differential or collision or collision) is small: e.g., Poly1305_r.

3. Choose a random function f that seems indistinguishable from uniform: e.g., AES_k.

4. Optional complication: Generate k, r from a shorter key; e.g., $k = \text{AES}_s(0)$, $r = \text{AES}_s(1)$; or $k = \text{MD5}(s)$, $r = \text{MD5}(s \oplus 1)$; many more possibilities.

5. Choose a Googleable name for your MAC.

6. Put it all together.

7. Publish!

build your own MAC

se a combination method:
$f(n)$ or $h(m) \oplus f(n)$
$n)$)—worse security—
$h(m)$)—bigger f input.

se a random function h
ne appropriate probability
rential or $\oplus$-differential
on or collision) is small:
y1305$_r$.

se a random function f
ms indistinguishable
iform: e.g., AES$_k$.

4. Optional complication:
Generate k, r from a shorter key;
e.g., $k =$ AES$_s(0)$, $r =$ AES$_s(1)$;
or $k =$ MD5(s), $r =$ MD5($s \oplus 1$);
many more possibilities.

5. Choose a Googleable name
for your MAC.

6. Put it all together.

7. Publish!

Example

1. Comb
2. Low
AES$_r$
3. Unpr
4. Optio
5. Name
6. EMA
AES$_k$
7. (2000

own MAC

bination method:
$(m) \oplus f(n)$
se security—
igger f input.

om function h
iate probability
$\oplus$-differential
sion) is small:

om function f
nguishable
., AES_k.

4. Optional complication:
Generate k, r from a shorter key;
e.g., $k = \mathrm{AES}_s(0)$, $r = \mathrm{AES}_s(1)$;
or $k = \mathrm{MD5}(s)$, $r = \mathrm{MD5}(s \oplus 1)$;
many more possibilities.

5. Choose a Googleable name
for your MAC.

6. Put it all together.

7. Publish!

Example:

1. Combination: f
2. Low collision pr
$\mathrm{AES}_r(\mathrm{AES}_r(m$
3. Unpredictable:
4. Optional compl
5. Name: "EMAC
6. $\mathrm{EMAC}_{k,r}(m_1, m$
$\mathrm{AES}_k(\mathrm{AES}_r(\mathrm{AE}$
7. (2000 Petrank

C

ethod:

$n)$

out.

on h

bility

tial

nall:

on f

4. Optional complication:
Generate k, r from a shorter key;
e.g., $k = \text{AES}_s(0)$, $r = \text{AES}_s(1)$;
or $k = \text{MD5}(s)$, $r = \text{MD5}(s \oplus 1)$;
many more possibilities.

5. Choose a Googleable name for your MAC.

6. Put it all together.

7. Publish!

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: N
5. Name: “EMAC.”
6. $\text{EMAC}_{k,r}(m_1, m_2) =$ $\text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus$
7. (2000 Petrank Rackoff)

4. Optional complication:
Generate k, r from a shorter key;
e.g., $k = \text{AES}_s(0)$, $r = \text{AES}_s(1)$;
or $k = \text{MD5}(s)$, $r = \text{MD5}(s \oplus 1)$;
many more possibilities.

5. Choose a Googleable name for your MAC.

6. Put it all together.

7. Publish!

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: No.
5. Name: “EMAC.”
6. $\text{EMAC}_{k,r}(m_1, m_2) = \text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2))$.
7. (2000 Petrank Rackoff)

onal complication:
e k, r from a shorter key;
= $\text{AES}_s(0)$, $r = \text{AES}_s(1)$;
MD5(s), $r =$ MD5($s \oplus 1$);
ore possibilities.

se a Googleable name
MAC.

t all together.

sh!

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: No.
5. Name: "EMAC."
6. $\text{EMAC}_{k,r}(m_1, m_2) = \text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2))$.
7. (2000 Petrank Rackoff)

Example
MD5(k,

"HMAC
plus the

(1996 B
claiming
treatmen

Stronger
Stronger
MD5(k,
Wow, I'v
new MA

lication:

n a shorter key;

, $r = \text{AES}_s(1)$;

$= \text{MD5}(s \oplus 1)$;

ilities.

gleable name

her.

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: No.
5. Name: "EMAC."
6. $\text{EMAC}_{k,r}(m_1, m_2) = \text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2))$.
7. (2000 Petrank Rackoff)

Example: "NMAC

$\text{MD5}(k, \text{MD5}(r, m$

"HMAC-MD5" is

plus the optional c

(1996 Bellare Can

claiming "the first

treatment of the s

Stronger: $\text{MD5}(k,$

Stronger and faste

$\text{MD5}(k, n, \text{Poly130}$

Wow, I've just inv

new MACs! Time

r key;

$_s(1)$;

$\oplus 1)$;

ne

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: No.
5. Name: "EMAC."
6. $\text{EMAC}_{k,r}(m_1, m_2) = \text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2))$.
7. (2000 Petrank Rackoff)

Example: "NMAC-MD5" is $\text{MD5}(k, \text{MD5}(r, m))$.

"HMAC-MD5" is NMAC-M

plus the optional complicati

(1996 Bellare Canetti Krawc

claiming "the first rigorous

treatment of the subject")

Stronger: $\text{MD5}(k, n, \text{MD5}(r$

Stronger and faster:

$\text{MD5}(k, n, \text{Poly1305}_r(m))$.

Wow, I've just invented two

new MACs! Time to publish

Example:

1. Combination: $f(h(m))$.
2. Low collision probability: $\text{AES}_r(\text{AES}_r(m_1) \oplus m_2)$.
3. Unpredictable: AES_k.
4. Optional complication: No.
5. Name: “EMAC.”
6. $\text{EMAC}_{k,r}(m_1, m_2) = \text{AES}_k(\text{AES}_r(\text{AES}_r(m_1) \oplus m_2))$.
7. (2000 Petrank Rackoff)

Example: “NMAC-MD5” is $\text{MD5}(k, \text{MD5}(r, m))$.

“HMAC-MD5” is NMAC-MD5 plus the optional complication.

(1996 Bellare Canetti Krawczyk, claiming “the first rigorous treatment of the subject”)

Stronger: $\text{MD5}(k, n, \text{MD5}(r, m))$.
Stronger and faster: $\text{MD5}(k, n, \text{Poly1305}_r(m))$.
Wow, I’ve just invented two new MACs! Time to publish!

e:

bination: $f(h(m))$.

collision probability:

$(\mathrm{AES}_r(m_1) \oplus m_2)$.

edictable: AES_k.

onal complication: No.

e: "EMAC."

$\mathrm{C}_{k,r}(m_1, m_2) =$

$(\mathrm{AES}_r(\mathrm{AES}_r(m_1) \oplus m_2))$.

0 Petrank Rackoff)

Example: "NMAC-MD5" is $\mathrm{MD5}(k, \mathrm{MD5}(r, m))$.

"HMAC-MD5" is NMAC-MD5 plus the optional complication.

(1996 Bellare Canetti Krawczyk, claiming "the first rigorous treatment of the subject")

Stronger: $\mathrm{MD5}(k, n, \mathrm{MD5}(r, m))$.

Stronger and faster: $\mathrm{MD5}(k, n, \mathrm{Poly1305}_r(m))$.

Wow, I've just invented two new MACs! Time to publish!

State-of-

Cycles p

authenti

Ath
Pentium
Pentiur
SPARC
PPC
bytes/

UMAC r

Similar:

$f(h(m))$.

robability:

$_1) \oplus m_2)$.

AES_k.

lication: No.

."

$m_2) =$

$\mathrm{ES}_r(m_1) \oplus m_2))$.

Rackoff)

Example: "NMAC-MD5" is $\mathrm{MD5}(k, \mathrm{MD5}(r, m))$.

"HMAC-MD5" is NMAC-MD5 plus the optional complication.

(1996 Bellare Canetti Krawczyk, claiming "the first rigorous treatment of the subject")

Stronger: $\mathrm{MD5}(k, n, \mathrm{MD5}(r, m))$.
Stronger and faster: $\mathrm{MD5}(k, n, \mathrm{Poly1305}_r(m))$.
Wow, I've just invented two new MACs! Time to publish!

State-of-the-art M

Cycles per byte to authenticate 1024-

	Poly 1305 -AES
Athlon	3.75
Pentium M	4.50
Pentium 4	5.33
SPARC III	5.47
PPC G4	8.27
bytes/key	32

UMAC really likes

Similar: VMAC lik

lo.

$\oplus m_2$)).

Example: "NMAC-MD5" is MD5(k, MD5(r, m)).

"HMAC-MD5" is NMAC-MD5 plus the optional complication.

(1996 Bellare Canetti Krawczyk, claiming "the first rigorous treatment of the subject")

Stronger: MD5(k, n, MD5(r, m)).
Stronger and faster:
MD5(k, n, Poly1305$_r(m)$).
Wow, I've just invented two new MACs! Time to publish!

State-of-the-art MACs

Cycles per byte to authenticate 1024-byte pack

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon

Example: "NMAC-MD5" is MD5(k, MD5(r, m)).

"HMAC-MD5" is NMAC-MD5 plus the optional complication.

(1996 Bellare Canetti Krawczyk, claiming "the first rigorous treatment of the subject")

Stronger: MD5(k, n, MD5(r, m)).
Stronger and faster: MD5(k, n, Poly1305$_r$(m)).
Wow, I've just invented two new MACs! Time to publish!

State-of-the-art MACs

Cycles per byte to authenticate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon 64.

e: “NMAC-MD5” is
MD5(r, m)).

-MD5” is NMAC-MD5
optional complication.

ellare Canetti Krawczyk,
“the first rigorous
nt of the subject”)

r: MD5(k, n, MD5(r, m)).
r and faster:
n, Poly1305$_r$(m)).
ve just invented two
ACs! Time to publish!

State-of-the-art MACs

Cycles per byte to
authenticate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon 64.

Some im

1. Imple
Poly130
split into
convenie
UMAC u
and suff

2. Key a
Poly130
of simult
and rem
keys are
UMAC r

-MD5" is
a)).

NMAC-MD5
complication.

etti Krawczyk,
rigorous
ubject")

n, MD5(r, m)).
er:
$05_r(m)$).
ented two
to publish!

State-of-the-art MACs

Cycles per byte to
authenticate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon 64.

Some important sp

1. Implementor fle
Poly1305 uses 128
split into whatever
convenient for the
UMAC uses P4-siz
and suffers on oth

2. Key agility.
Poly1305 can fit t
of simultaneous ke
and remains fast e
keys are out of cac
UMAC needs big e

D5
on.
zyk,
$, m))$.
n!

State-of-the-art MACs

Cycles per byte to authenticate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon 64.

Some important speed issue

1. Implementor flexibility.
Poly1305 uses 128-bit intege
split into whatever sizes are
convenient for the CPU.
UMAC uses P4-size integers
and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands
of simultaneous keys into ca
and remains fast even when
keys are out of cache.
UMAC needs big expanded

State-of-the-art MACs

Cycles per byte to
authenticate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
Athlon	3.75	7.38
Pentium M	4.50	8.48
Pentium 4	5.33	3.12
SPARC III	5.47	51.06
PPC G4	8.27	21.72
bytes/key	32	1600

UMAC really likes the P4.
Similar: VMAC likes Athlon 64.

Some important speed issues:

1. Implementor flexibility.
Poly1305 uses 128-bit integers, split into whatever sizes are convenient for the CPU.
UMAC uses P4-size integers and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands of simultaneous keys into cache, and remains fast even when keys are out of cache.
UMAC needs big expanded keys.

-the-art MACs

er byte to
cate 1024-byte packet:

	Poly 1305 -AES	UMAC -128
hlon	3.75	7.38
n M	4.50	8.48
m 4	5.33	3.12
C III	5.47	51.06
G4	8.27	21.72
/key	32	1600

really likes the P4.
VMAC likes Athlon 64.

Some important speed issues:

1. Implementor flexibility.
Poly1305 uses 128-bit integers, split into whatever sizes are convenient for the CPU.
UMAC uses P4-size integers and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands of simultaneous keys into cache, and remains fast even when keys are out of cache.
UMAC needs big expanded keys.

3. Numb

den Boe
$(m_1 r +$
Each ch

Gilbert-
$m_1 r_1 +$
Each ch

Winogra
$(m_1 + r$
Each ch

ACs

-byte packet:

UMAC -128
7.38
8.48
3.12
51.06
21.72
1600

the P4.

kes Athlon 64.

Some important speed issues:

1. Implementor flexibility.
Poly1305 uses 128-bit integers, split into whatever sizes are convenient for the CPU.
UMAC uses P4-size integers and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands of simultaneous keys into cache, and remains fast even when keys are out of cache.
UMAC needs big expanded keys.

3. Number of mul

den Boer et al.; P
$(m_1 r + m_2) r + \cdot$
Each chunk: mult

Gilbert-MacWillia
$m_1 r_1 + m_2 r_2 + \cdot$
Each chunk: mult

Winograd; UMAC
$(m_1 + r_1)(m_2 + r$
Each chunk: 0.5 r

ket:

64.

Some important speed issues:

1. Implementor flexibility.
Poly1305 uses 128-bit integers, split into whatever sizes are convenient for the CPU.
UMAC uses P4-size integers and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands of simultaneous keys into cache, and remains fast even when keys are out of cache.
UMAC needs big expanded keys.

3. Number of multiplication

den Boer et al.; Poly1305:
$(m_1 r + m_2) r + \cdots$.
Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:
$m_1 r_1 + m_2 r_2 + \cdots$.
Each chunk: mult, add.

Winograd; UMAC; VMAC:
$(m_1 + r_1)(m_2 + r_2) + \cdots$.
Each chunk: 0.5 mults, 1.5

Some important speed issues:

1. Implementor flexibility.
Poly1305 uses 128-bit integers, split into whatever sizes are convenient for the CPU.
UMAC uses P4-size integers and suffers on other CPUs.

2. Key agility.
Poly1305 can fit thousands of simultaneous keys into cache, and remains fast even when keys are out of cache.
UMAC needs big expanded keys.

3. Number of multiplications.

den Boer et al.; Poly1305:
$(m_1 r + m_2) r + \cdots$.
Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:
$m_1 r_1 + m_2 r_2 + \cdots$.
Each chunk: mult, add.

Winograd; UMAC; VMAC:
$(m_1 + r_1)(m_2 + r_2) + \cdots$.
Each chunk: 0.5 mults, 1.5 adds.

nportant speed issues:

ementor flexibility.
5 uses 128-bit integers,
o whatever sizes are
ent for the CPU.
uses P4-size integers
ers on other CPUs.

agility.
5 can fit thousands
taneous keys into cache,
ains fast even when
out of cache.
needs big expanded keys.

3. Number of multiplications.

den Boer et al.; Poly1305:
$(m_1 r + m_2) r + \cdots$.
Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:
$m_1 r_1 + m_2 r_2 + \cdots$.
Each chunk: mult, add.

Winograd; UMAC; VMAC:
$(m_1 + r_1)(m_2 + r_2) + \cdots$.
Each chunk: 0.5 mults, 1.5 adds.

Does sm
0.5 mult

Yes!

Another
$(((m_1 +$
$(m_3 +$
$((m_5 +$
$(m_7 +$
times a
times r.

"MAC10

peed issues:

exibility.

3-bit integers,

r sizes are

CPU.

ze integers

er CPUs.

housands

eys into cache,

even when

che.

expanded keys.

3. Number of multiplications.

den Boer et al.; Poly1305:

$(m_1 r + m_2)r + \cdots$.

Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:

$m_1 r_1 + m_2 r_2 + \cdots$.

Each chunk: mult, add.

Winograd; UMAC; VMAC:

$(m_1 + r_1)(m_2 + r_2) + \cdots$.

Each chunk: 0.5 mults, 1.5 adds.

Does small key r a

0.5 mults per mes

Yes!

Another old trick

$(((m_1 + r)(m_2 +$

$(m_3 + r))(m_4 +$

$((m_5 + r)(m_6 +$

$(m_7 + r)))(m_8$

times a final nonz

times r.

“MAC1071,” com

s:

ers,

che,

keys.

3. Number of multiplications.

den Boer et al.; Poly1305:
$(m_1 r + m_2)r + \cdots$.
Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:
$m_1 r_1 + m_2 r_2 + \cdots$.
Each chunk: mult, add.

Winograd; UMAC; VMAC:
$(m_1 + r_1)(m_2 + r_2) + \cdots$.
Each chunk: 0.5 mults, 1.5 adds.

Does small key r allow
0.5 mults per message chun

Yes!

Another old trick of Winogr

$$(((m_1 + r)(m_2 + r^2) +$$
$$(m_3 + r))(m_4 + r^4) +$$
$$((m_5 + r)(m_6 + r^2) +$$
$$(m_7 + r)))(m_8 + r^8) + \cdot\cdot$$

times a final nonzero m_n
times r.

"MAC1071," coming soon.

3. Number of multiplications.

den Boer et al.; Poly1305:
$(m_1 r + m_2) r + \cdots$.
Each chunk: mult, add.

Gilbert-MacWilliams-Sloane:
$m_1 r_1 + m_2 r_2 + \cdots$.
Each chunk: mult, add.

Winograd; UMAC; VMAC:
$(m_1 + r_1)(m_2 + r_2) + \cdots$.
Each chunk: 0.5 mults, 1.5 adds.

Does small key r allow
0.5 mults per message chunk?

Yes!

Another old trick of Winograd:

$$(((m_1 + r)(m_2 + r^2) + (m_3 + r))(m_4 + r^4) + ((m_5 + r)(m_6 + r^2) + (m_7 + r)))(m_8 + r^8) + \cdots$$

times a final nonzero m_n
times r.

“MAC1071,” coming soon.