Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

2006 Bla
Krawczy
“The nu
to be co
session .
allowed

Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

2006 Black–Halevi
Krawczyk–Krovetz
"The number of m
to be communicat
session . . . should
allowed to approac

Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway
"The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: "The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Extending the Salsa20 nonce

D. J. Bernstein

University of Illinois at Chicago

DES had 64-bit block.
Highly troublesome by 1990s.

AES has 128-bit block.
Becoming troublesome now . . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: "The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Why do they say this?
Answer: Their security proof fails for #messages $\approx 2^{n/2}$ (AES: #messages $\approx 2^{64}$), and becomes quantitatively useless long before that.

So what *should* users do?
No advice from 2006 BHHKKR.

ng the Salsa20 nonce

ernstein

ty of Illinois at Chicago

---

d 64-bit block.
roublesome by 1990s.

s 128-bit block.
ng troublesome now . . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: "The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Why do they say this?
Answer: Their security proof fails for #messages $\approx 2^{n/2}$ (AES: #messages $\approx 2^{64}$), and becomes quantitatively useless long before that.

So what *should* users do?
No advice from 2006 BHHKKR.

Commo

128-bit
produce

First ses
Second s
etc.

Each ses
for limit

Typical
AES-CT
for at m

sa20 nonce

is at Chicago

---

ock.

e by 1990s.

lock.

some now . . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: "The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Why do they say this?

Answer: Their security proof fails for #messages $\approx 2^{n/2}$ (AES: #messages $\approx 2^{64}$), and becomes quantitatively useless long before that.

So what *should* users do?

No advice from 2006 BHHKKR.

Common user resp

128-bit "master" A

produces 128-bit "

First session key:

Second session key

etc.

Each session key *k*

for limited #messa

Typical use of sess

AES-CTR, GCM, e

for at most (e.g.)

e

ago

s.

. . .

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: “The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$.”

Why do they say this?
Answer: Their security proof fails for #messages $\approx 2^{n/2}$ (AES: #messages $\approx 2^{64}$), and becomes quantitatively useless long before that.

So what *should* users do?
No advice from 2006 BHHKKR.

Common user response: Rek

128-bit “master” AES key k
produces 128-bit “session ke

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k($2
etc.

Each session key k' is used
for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks

2006 Black–Halevi–Hevia–Krawczyk–Krovetz–Rogaway: "The number of messages to be communicated in a session . . . should not be allowed to approach $2^{n/2}$."

Why do they say this?
Answer: Their security proof fails for #messages ≈ $2^{n/2}$ (AES: #messages ≈ 2^{64}), and becomes quantitatively useless long before that.

So what *should* users do?
No advice from 2006 BHHKKR.

Common user response: Rekeying.

128-bit "master" AES key k produces 128-bit "session keys".

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks.

ack–Halevi–Hevia–
k–Krovetz–Rogaway:
mber of messages
mmunicated in a
. . should not be
to approach $2^{n/2}$."

they say this?
Their security proof
#messages $\approx 2^{n/2}$
messages $\approx 2^{64}$),
omes quantitatively
ong before that.

should users do?
ce from 2006 BHHKKR.

Common user response: Rekeying.

128-bit "master" AES key k
produces 128-bit "session keys".

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used
for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks.

In other
128-bit
$\text{AES}_{\text{AES}_k}$
$\text{AES}_{\text{AES}_k}$
$\text{AES}_{\text{AES}_k}$
and so o

This is r
$(m, n) \mapsto$
with a d

i–Hevia–
z–Rogaway:
nessages
ed in a
not be
ch $2^{n/2}$."

this?
urity proof
es $\approx 2^{n/2}$
$\approx 2^{64}$),
ntitatively
e that.

sers do?
006 BHHKKR.

Common user response: Rekeying.

128-bit "master" AES key k
produces 128-bit "session keys".

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used
for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks.

In other words:

128-bit AES key k
$\text{AES}_{\text{AES}_k(1)}(1)$, AE
$\text{AES}_{\text{AES}_k(2)}(1)$, AE
$\text{AES}_{\text{AES}_k(3)}(1)$, AE
and so on.

This is really a new
$(m, n) \mapsto \text{AES}_{\text{AES}}$
with a double-size

:

f

KR.

Common user response: Rekeying.

128-bit “master” AES key k produces 128-bit “session keys”.

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks.

In other words:

128-bit AES key k produces
$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}($
$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}($
$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}($
and so on.

This is really a new cipher
$(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$
with a double-size input.

Common user response: Rekeying.

128-bit "master" AES key k
produces 128-bit "session keys".

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used
for limited #messages.

Typical use of session key:
AES-CTR, GCM, etc.
for at most (e.g.) 2^{40} blocks.

In other words:

128-bit AES key k produces
$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, ...;
$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, ...;
$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, ...;
and so on.

This is really a new cipher
$(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$
with a double-size input.

Common user response: Rekeying.

128-bit "master" AES key k produces 128-bit "session keys".

First session key: $\text{AES}_k(1)$.
Second session key: $\text{AES}_k(2)$.
etc.

Each session key k' is used for limited #messages.

Typical use of session key: AES-CTR, GCM, etc. for at most (e.g.) 2^{40} blocks.

In other words:

128-bit AES key k produces
$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, $\ldots$;
$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, $\ldots$;
$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, $\ldots$;
and so on.

This is really a new cipher $(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$ with a double-size input.

Alert: User-designed cipher!
Is this cipher secure?

n user response: Rekeying.

“master” AES k

s 128-bit “session keys”.

sion key: $\text{AES}_k(1)$.

session key: $\text{AES}_k(2)$.

ssion key k' is used

ed #messages.

use of session key:

R, GCM, etc.

ost (e.g.) 2^{40} blocks.

In other words:

128-bit AES key k produces

$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, . . .;

$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, . . .;

$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, . . .;

and so on.

This is really a new cipher

$(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$

with a double-size input.

Alert: User-designed cipher!

Is this cipher secure?

Not real

Collect A

for 2^{40} i

Build 2^4

each cor

iterates

Good ch

$k' = \text{AE}$

Find via

Then tri

AES_{AES}

Current

$<$ 1 year

ponse: Rekeying.

AES key k

"session keys".

$\text{AES}_k(1)$.

y: $\text{AES}_k(2)$.

k' is used

ages.

sion key:

etc.

2^{40} blocks.

In other words:

128-bit AES key k produces
$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, ...;
$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, ...;
$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, ...;
and so on.

This is really a new cipher
$(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$
with a double-size input.

Alert: User-designed cipher!
Is this cipher secure?

Not really. Feasibl

Collect $\text{AES}_{\text{AES}_k(n}$
for 2^{40} inputs $(n,$

Build 2^{40} tiny sear
each computing 2^{4}
iterates of $k' \mapsto$ A
Good chance of co
$k' = \text{AES}_k(n)$ for
Find via distinguis
Then trivially com
$\text{AES}_{\text{AES}_k(n)}(1)$ etc

Current chip techn
< 1 year, $< 10^{10}$

eying.

:

eys".

2).

s.

In other words:

128-bit AES key k produces
$\mathsf{AES}_{\mathsf{AES}_k(1)}(1)$, $\mathsf{AES}_{\mathsf{AES}_k(1)}(2)$, ...;
$\mathsf{AES}_{\mathsf{AES}_k(2)}(1)$, $\mathsf{AES}_{\mathsf{AES}_k(2)}(2)$, ...;
$\mathsf{AES}_{\mathsf{AES}_k(3)}(1)$, $\mathsf{AES}_{\mathsf{AES}_k(3)}(2)$, ...;
and so on.

This is really a new cipher
$(m, n) \mapsto \mathsf{AES}_{\mathsf{AES}_k(m)}(n)$
with a double-size input.

Alert: User-designed cipher!
Is this cipher secure?

Not really. Feasible attack:

Collect $\mathsf{AES}_{\mathsf{AES}_k(n)}(0)$
for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units,
each computing 2^{48}
iterates of $k' \mapsto \mathsf{AES}_{k'}(0)$.
Good chance of collision
$k' = \mathsf{AES}_k(n)$ for some n, k
Find via distinguished point
Then trivially compute
$\mathsf{AES}_{\mathsf{AES}_k(n)}(1)$ etc.

Current chip technology:
< 1 year, $< 10^{10}$ USD.

In other words:

128-bit AES key k produces
$\text{AES}_{\text{AES}_k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, . . .;
$\text{AES}_{\text{AES}_k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, . . .;
$\text{AES}_{\text{AES}_k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, . . .;
and so on.

This is really a new cipher
$(m, n) \mapsto \text{AES}_{\text{AES}_k(m)}(n)$
with a double-size input.

Alert: User-designed cipher!
Is this cipher secure?

Not really. Feasible attack:

Collect $\text{AES}_{\text{AES}_k(n)}(0)$
for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units,
each computing 2^{48}
iterates of $k' \mapsto \text{AES}_{k'}(0)$.
Good chance of collision
$k' = \text{AES}_k(n)$ for some n, k'.
Find via distinguished points.
Then trivially compute
$\text{AES}_{\text{AES}_k(n)}(1)$ etc.

Current chip technology:
< 1 year, $< 10^{10}$ USD.

words:

AES key k produces

$_{k(1)}(1)$, $\text{AES}_{\text{AES}_k(1)}(2)$, . . .;

$_{k(2)}(1)$, $\text{AES}_{\text{AES}_k(2)}(2)$, . . .;

$_{k(3)}(1)$, $\text{AES}_{\text{AES}_k(3)}(2)$, . . .;

on.

eally a new cipher

$\to \text{AES}_{\text{AES}_k(m)}(n)$

ouble-size input.

ser-designed cipher!

pher secure?

Not really. Feasible attack:

Collect $\text{AES}_{\text{AES}_k(n)}(0)$ for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units, each computing 2^{48} iterates of $k' \mapsto \text{AES}_{k'}(0)$.

Good chance of collision $k' = \text{AES}_k(n)$ for some n, k'.

Find via distinguished points.

Then trivially compute $\text{AES}_{\text{AES}_k(n)}(1)$ etc.

Current chip technology: < 1 year, $< 10^{10}$ USD.

Two diff

stopping

1. “Use

Attack r

same inp

by many

. . . but r

leaves m

and raise

: produces

ES$_{\mathsf{AES}_k(1)}(2)$, . . .;

ES$_{\mathsf{AES}_k(2)}(2)$, . . .;

ES$_{\mathsf{AES}_k(3)}(2)$, . . .;

w cipher

$_{k(m)}(n)$

input.

ed cipher!

re?

Not really. Feasible attack:

Collect $\mathsf{AES}_{\mathsf{AES}_k(n)}(0)$ for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units, each computing 2^{48} iterates of $k' \mapsto \mathsf{AES}_{k'}(0)$. Good chance of collision $k' = \mathsf{AES}_k(n)$ for some n, k'. Find via distinguished points. Then trivially compute $\mathsf{AES}_{\mathsf{AES}_k(n)}(1)$ etc.

Current chip technology: < 1 year, $< 10^{10}$ USD.

Two different philo

stopping this type

1. "Use random n

Attack relies critic

same input 0 bein

by many session k

. . . but randomizat

leaves many secur

and raises usability

(2), . . .;
(2), . . .;
(2), . . .;

Not really. Feasible attack:

Collect $\text{AES}_{\text{AES}_k(n)}(0)$ for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units, each computing 2^{48} iterates of $k' \mapsto \text{AES}_{k'}(0)$. Good chance of collision $k' = \text{AES}_k(n)$ for some n, k'. Find via distinguished points. Then trivially compute $\text{AES}_{\text{AES}_k(n)}(1)$ etc.

Current chip technology: < 1 year, $< 10^{10}$ USD.

Two different philosophies fo
stopping this type of attack:

1. "Use random nonces."
Attack relies critically on
same input 0 being encrypte
by many session keys k'.
. . . but randomization still
leaves many security questio
and raises usability question

Not really. Feasible attack:

Collect $\text{AES}_{\text{AES}_k(n)}(0)$ for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units, each computing 2^{48} iterates of $k' \mapsto \text{AES}_{k'}(0)$. Good chance of collision $k' = \text{AES}_k(n)$ for some n, k'. Find via distinguished points. Then trivially compute $\text{AES}_{\text{AES}_k(n)}(1)$ etc.

Current chip technology: < 1 year, $< 10^{10}$ USD.

Two different philosophies for stopping this type of attack:

1. "Use random nonces." Attack relies critically on same input 0 being encrypted by many session keys k'. . . . but randomization still leaves many security questions and raises usability questions.

Not really. Feasible attack:

Collect $\text{AES}_{\text{AES}_k(n)}(0)$ for 2^{40} inputs $(n, 0)$.

Build 2^{40} tiny search units, each computing 2^{48} iterates of $k' \mapsto \text{AES}_{k'}(0)$. Good chance of collision $k' = \text{AES}_k(n)$ for some n, k'. Find via distinguished points. Then trivially compute $\text{AES}_{\text{AES}_k(n)}(1)$ etc.

Current chip technology: < 1 year, $< 10^{10}$ USD.

Two different philosophies for stopping this type of attack:

1. "Use random nonces." Attack relies critically on same input 0 being encrypted by many session keys k'. …but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys." Master key produces 256-bit output block, used as 256-bit session key. We have good 256-bit ciphers!

ly. Feasible attack:

$\mathrm{AES}_{\mathrm{AES}_k(n)}(0)$
nputs $(n, 0)$.

^{0} tiny search units,
mputing 2^{48}
of $k' \mapsto \mathrm{AES}_{k'}(0)$.
ance of collision
$\mathrm{S}_k(n)$ for some n, k'.
distinguished points.
vially compute
$_{k(n)}(1)$ etc.

chip technology:
, $< 10^{10}$ USD.

Two different philosophies for stopping this type of attack:

1. "Use random nonces."
Attack relies critically on same input 0 being encrypted by many session keys k'.
. . . but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys."
Master key produces 256-bit output block, used as 256-bit session key.
We have good 256-bit ciphers!

I'll focus

Could ge
$k' = (\mathrm{A}$
Use k' a

e attack:

$_{n)}(0)$
0).

rch units,
48

ES$_{k'}(0)$.
ollision
some n, k'.
hed points.
pute
c.

nology:
USD.

Two different philosophies for stopping this type of attack:

1. "Use random nonces."
Attack relies critically on same input 0 being encrypted by many session keys k'.
. . . but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys."
Master key produces 256-bit output block, used as 256-bit session key.
We have good 256-bit ciphers!

I'll focus on strate

Could generate 25
$k' = (\text{AES}_k(2n), \text{A}$
Use k' as key for 2

$'$.

s.

Two different philosophies for stopping this type of attack:

1. "Use random nonces."
Attack relies critically on same input 0 being encrypted by many session keys k'.
. . . but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys."
Master key produces 256-bit output block, used as 256-bit session key.
We have good 256-bit ciphers!

I'll focus on strategy #2.

Could generate 256-bit
$k' = (\text{AES}_k(2n), \text{AES}_k(2n +$
Use k' as key for 256-bit AE

Two different philosophies for stopping this type of attack:

1. "Use random nonces."
Attack relies critically on same input 0 being encrypted by many session keys k'.
. . . but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys."
Master key produces 256-bit output block, used as 256-bit session key.
We have good 256-bit ciphers!

I'll focus on strategy #2.

Could generate 256-bit $k' = (\text{AES}_k(2n), \text{AES}_k(2n+1))$.
Use k' as key for 256-bit AES.

Two different philosophies for stopping this type of attack:

1. "Use random nonces." Attack relies critically on same input 0 being encrypted by many session keys k'.
... but randomization still leaves many security questions and raises usability questions.

2. "Use longer keys." Master key produces 256-bit output block, used as 256-bit session key. We have good 256-bit ciphers!

I'll focus on strategy #2.

Could generate 256-bit $k' = (\text{AES}_k(2n), \text{AES}_k(2n+1))$. Use k' as key for 256-bit AES.

But AES isn't a great cipher:

- Small block, so distinguishable.
- Not much security margin.
- Uninspiring key schedule.
- Invites cache-timing attacks.
- Slow on most CPUs.
- Mediocre speed in hardware.
- Even slower with key expansion.

ferent philosophies for
; this type of attack:

random nonces."
elies critically on
put 0 being encrypted
session keys k'.
andomization still
nany security questions
es usability questions.

longer keys."
key produces
output block,
256-bit session key.
e good 256-bit ciphers!

I'll focus on strategy #2.

Could generate 256-bit $k' = (\mathrm{AES}_k(2n), \mathrm{AES}_k(2n+1))$.
Use k' as key for 256-bit AES.

But AES isn't a great cipher:

- Small block, so distinguishable.
- Not much security margin.
- Uninspiring key schedule.
- Invites cache-timing attacks.
- Slow on most CPUs.
- Mediocre speed in hardware.
- Even slower with key expansion.

How abo

- Large
- 150%
- Key at
- Natura
- Fast a
- Better
- No key

Can gen
first 256
using 64
Use k' a

osophies for
of attack:

onces."
ally on
g encrypted
eys k'.
tion still
ity questions
y questions.

ys."
ces
ck,
ssion key.
6-bit ciphers!

I'll focus on strategy #2.

Could generate 256-bit
$k' = (\text{AES}_k(2n), \text{AES}_k(2n+1))$.
Use k' as key for 256-bit AES.

But AES isn't a great cipher:

- Small block, so distinguishable.
- Not much security margin.
- Uninspiring key schedule.
- Invites cache-timing attacks.
- Slow on most CPUs.
- Mediocre speed in hardware.
- Even slower with key expansion.

How about Salsa2

- Large block; aim
- 150% security m
- Key at top, not
- Naturally consta
- Fast across CPU
- Better than AES
- No key expansio

Can generate 256-
first 256 bits of Sa
using 64-bit nonce
Use k' as Salsa20

or
:
ed
ons
s.
ers!

I'll focus on strategy #2.

Could generate 256-bit
$k' = (\text{AES}_k(2n), \text{AES}_k(2n+1))$.
Use k' as key for 256-bit AES.

But AES isn't a great cipher:

- Small block, so distinguishable.
- Not much security margin.
- Uninspiring key schedule.
- Invites cache-timing attacks.
- Slow on most CPUs.
- Mediocre speed in hardware.
- Even slower with key expansion.

How about Salsa20?

- Large block; aims to be P
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardw
- No key expansion.

Can generate 256-bit k' as
first 256 bits of Salsa20 stre
using 64-bit nonce n, key k.
Use k' as Salsa20 session ke

I’ll focus on strategy #2.

Could generate 256-bit
$k' = (\text{AES}_k(2n), \text{AES}_k(2n+1))$.
Use k' as key for 256-bit AES.

But AES isn’t a great cipher:

- Small block, so distinguishable.
- Not much security margin.
- Uninspiring key schedule.
- Invites cache-timing attacks.
- Slow on most CPUs.
- Mediocre speed in hardware.
- Even slower with key expansion.

How about Salsa20?

- Large block; aims to be PRF.
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardware.
- No key expansion.

Can generate 256-bit k' as
first 256 bits of Salsa20 stream
using 64-bit nonce n, key k.
Use k' as Salsa20 session key.

s on strategy #2.

enerate 256-bit

ES$_k(2n)$, AES$_k(2n+1)$).

s key for 256-bit AES.

S isn't a great cipher:

block, so distinguishable.

uch security margin.

piring key schedule.

s cache-timing attacks.

on most CPUs.

cre speed in hardware.

slower with key expansion.

How about Salsa20?

- Large block; aims to be PRF.
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardware.
- No key expansion.

Can generate 256-bit k' as first 256 bits of Salsa20 stream using 64-bit nonce n, key k. Use k' as Salsa20 session key.

Improve

Salsa20

producir

256-bit

Convent

is interp

and 64-b

(so outp

but func

to be fas

giving ra

So allow

Generate

as half o

gy #2.

6-bit

$\mathrm{AES}_k(2n+1)$).

256-bit AES.

reat cipher:

distinguishable.

ity margin.

schedule.

hing attacks.

PUs.

in hardware.

n key expansion.

How about Salsa20?

- Large block; aims to be PRF.
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardware.
- No key expansion.

Can generate 256-bit k' as first 256 bits of Salsa20 stream using 64-bit nonce n, key k. Use k' as Salsa20 session key.

Improvement #1:

Salsa20 is actually
producing 512-bit
256-bit key, 128-b

Conventionally 128
is interpreted as 6
and 64-bit block c
(so output blocks
but function is des
to be fast and sec
giving random acc

So allow 128 bits
Generate 256-bit
as half of 512-bit

$-1))$.

ES.

r:

nable.

.

ks.

re.

nsion.

How about Salsa20?

- Large block; aims to be PRF.
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardware.
- No key expansion.

Can generate 256-bit k' as first 256 bits of Salsa20 stream using 64-bit nonce n, key k. Use k' as Salsa20 session key.

Improvement #1:

Salsa20 is actually a functio producing 512-bit block fror 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonc and 64-bit block counter (so output blocks are a strea but function is designed to be fast and secure giving random access to bloc

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

How about Salsa20?

- Large block; aims to be PRF.
- 150% security margin.
- Key at top, not on side.
- Naturally constant time.
- Fast across CPUs.
- Better than AES in hardware.
- No key expansion.

Can generate 256-bit k' as first 256 bits of Salsa20 stream using 64-bit nonce n, key k. Use k' as Salsa20 session key.

Improvement #1:

Salsa20 is actually a function producing 512-bit block from 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonce and 64-bit block counter (so output blocks are a stream), but function is designed to be fast and secure giving random access to blocks.

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

out Salsa20?
block; aims to be PRF.
security margin.
t top, not on side.
ally constant time.
cross CPUs.
than AES in hardware.
y expansion.

erate 256-bit k' as
bits of Salsa20 stream
-bit nonce n, key k.
s Salsa20 session key.

Improvement #1:

Salsa20 is actually a function producing 512-bit block from 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonce and 64-bit block counter (so output blocks are a stream), but function is designed to be fast and secure giving random access to blocks.

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

Improve

Look mo
at how S
initialize
publicly
adds 256
applies r
adds 256

Take k'
⇒ Skip

Importa
block is
Compare

0?

ns to be PRF.

nargin.

on side.

nt time.

Js.

S in hardware.

n.

-bit k' as

alsa20 stream

e n, key k.

session key.

Improvement #1:

Salsa20 is actually a function producing 512-bit block from 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonce and 64-bit block counter (so output blocks are a stream), but function is designed to be fast and secure giving random access to blocks.

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

Improvement #2:

Look more closely

at how Salsa20 wo

initializes 512-bit

publicly from inpu

adds 256-bit key k

applies many unke

adds 256-bit key k

Take k' as the *oth*

$\Rightarrow$ Skip final k ad

Important here tha

block is much bigg

Compare to Even–

RF.

are.

am

y.

Improvement #1:

Salsa20 is actually a function producing 512-bit block from 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonce and 64-bit block counter (so output blocks are a stream), but function is designed to be fast and secure giving random access to blocks.

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

Improvement #2:

Look more closely at how Salsa20 works: initializes 512-bit block publicly from input n; adds 256-bit key k; applies many unkeyed round adds 256-bit key k.

Take k' as the *other* 256 bit
$\Rightarrow$ Skip final k addition.

Important here that block is much bigger than k
Compare to Even–Mansour

Improvement #1:

Salsa20 is actually a function producing 512-bit block from 256-bit key, 128-bit input.

Conventionally 128-bit input is interpreted as 64-bit nonce and 64-bit block counter (so output blocks are a stream), but function is designed to be fast and secure giving random access to blocks.

So allow 128 bits in n. Generate 256-bit k' as half of 512-bit block.

Improvement #2:

Look more closely at how Salsa20 works: initializes 512-bit block publicly from input n; adds 256-bit key k; applies many unkeyed rounds; adds 256-bit key k.

Take k' as the *other* 256 bits. $\Rightarrow$ Skip final k addition.

Important here that block is much bigger than k. Compare to Even–Mansour etc.

ment #1:

is actually a function
ng 512-bit block from
key, 128-bit input.

ionally 128-bit input
reted as 64-bit nonce
bit block counter
out blocks are a stream),
ction is designed
st and secure
andom access to blocks.

128 bits in n.
e 256-bit k'
of 512-bit block.

Improvement #2:

Look more closely
at how Salsa20 works:
initializes 512-bit block
publicly from input n;
adds 256-bit key k;
applies many unkeyed rounds;
adds 256-bit key k.

Take k' as the *other* 256 bits.
$\Rightarrow$ Skip final k addition.

Important here that
block is much bigger than k.
Compare to Even–Mansour etc.

What ab

Recall fe
Moving
puts atta

Could th

1996 Be
Can con
into sim
attack o
factor $\leq$

Warning
"theorem
Correcte

a function
block from
it input.

8-bit input
4-bit nonce
ounter
are a stream),
signed
ure
ess to blocks.

in n.
k'
block.

Improvement #2:

Look more closely
at how Salsa20 works:
initializes 512-bit block
publicly from input n;
adds 256-bit key k;
applies many unkeyed rounds;
adds 256-bit key k.

Take k' as the *other* 256 bits.
$\Rightarrow$ Skip final k addition.

Important here that
block is much bigger than k.
Compare to Even–Mansour etc.

What about secur

Recall feasible 128
Moving from 128
puts attack very fa

Could there be be

1996 Bellare–Cane
Can convert any q
into similarly effici
attack on original
factor $\leq 2q$ in suc

Warning: FOCS 1
“theorem” omits f
Corrected in 2005

n
m
t
e
am),
cks.

Improvement #2:

Look more closely
at how Salsa20 works:
initializes 512-bit block
publicly from input n;
adds 256-bit key k;
applies many unkeyed rounds;
adds 256-bit key k.

Take k' as the *other* 256 bits.
$\Rightarrow$ Skip final k addition.

Important here that
block is much bigger than k.
Compare to Even–Mansour etc.

What about security?

Recall feasible 128-bit attac
Moving from 128 bits to 25
puts attack very far out of r

Could there be better attack

1996 Bellare–Canetti–Krawc
Can convert any q-query att
into similarly efficient single-
attack on original cipher, los
factor $\leq 2q$ in success proba

Warning: FOCS 1996
"theorem" omits factor q.
Corrected in 2005 online ver

Improvement #2:

Look more closely
at how Salsa20 works:
initializes 512-bit block
publicly from input n;
adds 256-bit key k;
applies many unkeyed rounds;
adds 256-bit key k.

Take k' as the *other* 256 bits.
$\Rightarrow$ Skip final k addition.

Important here that
block is much bigger than k.
Compare to Even–Mansour etc.

What about security?

Recall feasible 128-bit attack.
Moving from 128 bits to 256 bits
puts attack very far out of reach.

Could there be better attacks?

1996 Bellare–Canetti–Krawczyk:
Can convert any q-query attack
into similarly efficient single-key
attack on original cipher, losing
factor $\leq 2q$ in success probability.

Warning: FOCS 1996
"theorem" omits factor q.
Corrected in 2005 online version.

ment #2:

ore closely
Salsa20 works:
s 512-bit block
from input n;
6-bit key k;
many unkeyed rounds;
6-bit key k.

as the *other* 256 bits.
final k addition.

nt here that
much bigger than k.
e to Even–Mansour etc.

What about security?

Recall feasible 128-bit attack.
Moving from 128 bits to 256 bits puts attack very far out of reach.

Could there be better attacks?

1996 Bellare–Canetti–Krawczyk:
Can convert any q-query attack into similarly efficient single-key attack on original cipher, losing factor $\le 2q$ in success probability.

Warning: FOCS 1996 "theorem" omits factor q.
Corrected in 2005 online version.

Better s

1. Loss
$\le (\ell - 1$
Compare

2. Allow
for mast
Attack s
$\le \epsilon$ vs.
$\le \epsilon'$ vs.
$\Rightarrow \le \epsilon +$

Combini
deduce
immedia

orks:
block
t n;
c;
eyed rounds;
c.

her 256 bits.
dition.

at
ger than k.
-Mansour etc.

What about security?

Recall feasible 128-bit attack.
Moving from 128 bits to 256 bits
puts attack very far out of reach.

Could there be better attacks?

1996 Bellare–Canetti–Krawczyk:
Can convert any q-query attack
into similarly efficient single-key
attack on original cipher, losing
factor $\le 2q$ in success probability.

Warning: FOCS 1996
“theorem” omits factor q.
Corrected in 2005 online version.

Better security pro

1. Loss factor $\le q$
$\le (\ell - 1)q + 1$ for
Compare to ℓq fro

2. Allow independ
for master key, ses
Attack success pro
$\le \epsilon$ vs. master cip
$\le \epsilon'$ vs. session cip
$\Rightarrow \le \epsilon + q\epsilon'$ vs. ca

Combining 1 and 2
deduce ℓ-level secu
immediately from

s;

ts.

.

etc.

What about security?

Recall feasible 128-bit attack. Moving from 128 bits to 256 bits puts attack very far out of reach.

Could there be better attacks?

1996 Bellare–Canetti–Krawczyk: Can convert any q-query attack into similarly efficient single-key attack on original cipher, losing factor $\le 2q$ in success probability.

Warning: FOCS 1996 "theorem" omits factor q. Corrected in 2005 online version.

Better security proof, this pa

1. Loss factor $\le q + 1$.
$\le (\ell - 1)q + 1$ for ℓ levels.
Compare to ℓq from 2005 B

2. Allow independent cipher
for master key, session keys.
Attack success probability
$\le \epsilon$ vs. master cipher,
$\le \epsilon'$ vs. session cipher
$\Rightarrow \le \epsilon + q\epsilon'$ vs. cascaded ci

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level sec

What about security?

Recall feasible 128-bit attack.
Moving from 128 bits to 256 bits puts attack very far out of reach.

Could there be better attacks?

1996 Bellare–Canetti–Krawczyk:
Can convert any q-query attack into similarly efficient single-key attack on original cipher, losing factor $\leq 2q$ in success probability.

Warning: FOCS 1996 "theorem" omits factor q.
Corrected in 2005 online version.

Better security proof, this paper:

1. Loss factor $\leq q + 1$.
$\leq (\ell - 1)q + 1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers for master key, session keys.
Attack success probability
$\leq \epsilon$ vs. master cipher,
$\leq \epsilon'$ vs. session cipher
$\Rightarrow \leq \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security immediately from 2-level security.

bout security?

easible 128-bit attack.
from 128 bits to 256 bits
ack very far out of reach.

here be better attacks?

llare–Canetti–Krawczyk:
vert any q-query attack
ilarly efficient single-key
n original cipher, losing
$2q$ in success probability.

: FOCS 1996
m" omits factor q.
ed in 2005 online version.

Better security proof, this paper:

1. Loss factor $\le q + 1$.
$\le (\ell - 1)q + 1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers
for master key, session keys.
Attack success probability
$\le \epsilon$ vs. master cipher,
$\le \epsilon'$ vs. session cipher
$\Rightarrow \le \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level security.

2-level A
2^{40} quer
Is 1-leve

ity?

-bit attack.

bits to 256 bits

ar out of reach.

tter attacks?

etti–Krawczyk:

-query attack

ent single-key

cipher, losing

cess probability.

996

actor q.

online version.

Better security proof, this paper:

1. Loss factor $\le q + 1$.
$\le (\ell - 1)q + 1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers for master key, session keys.
Attack success probability
$\le \epsilon$ vs. master cipher,
$\le \epsilon'$ vs. session cipher
$\Rightarrow \le \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security immediately from 2-level security.

2-level AES is brea

2^{40} queries, space

Is 1-level AES real

k.

6 bits

each.

ks?

zyk:

ack

-key

sing

bility.

sion.

Better security proof, this paper:

1. Loss factor $\leq q+1$.
$\leq (\ell-1)q+1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers
for master key, session keys.
Attack success probability
$\leq \epsilon$ vs. master cipher,
$\leq \epsilon'$ vs. session cipher
$\Rightarrow \leq \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level security.

2-level AES is breakable wit
2^{40} queries, space 2^{40}, time
Is 1-level AES really more se

Better security proof, this paper:

1. Loss factor $\le q + 1$.
$\le (\ell - 1)q + 1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers
for master key, session keys.
Attack success probability
$\le \epsilon$ vs. master cipher,
$\le \epsilon'$ vs. session cipher
$\Rightarrow \le \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level security.

2-level AES is breakable with
2^{40} queries, space 2^{40}, time 2^{48}.
Is 1-level AES really more secure?

Better security proof, this paper:

1. Loss factor $\le q+1$.
$\le (\ell-1)q+1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers
for master key, session keys.
Attack success probability
$\le \epsilon$ vs. master cipher,
$\le \epsilon'$ vs. session cipher
$\Rightarrow \le \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level security.

2-level AES is breakable with
2^{40} queries, space 2^{40}, time 2^{48}.
Is 1-level AES really more secure?
No! 1996 Biham "key collisions"
break 2^{40}-user 1-level AES
in exactly the same way.

Traditional 1-user metric:
Breaking AES using q queries
costs 2^{128} by best attack known.

Biham's multi-user metric:
$2^{128}/q$ by best attack known.

Better security proof, this paper:

1. Loss factor $\leq q+1$.
$\leq (\ell-1)q+1$ for ℓ levels.
Compare to ℓq from 2005 BCK.

2. Allow independent ciphers
for master key, session keys.
Attack success probability
$\leq \epsilon$ vs. master cipher,
$\leq \epsilon'$ vs. session cipher
$\Rightarrow \leq \epsilon + q\epsilon'$ vs. cascaded cipher.

Combining 1 and 2:
deduce ℓ-level security
immediately from 2-level security.

2-level AES is breakable with
2^{40} queries, space 2^{40}, time 2^{48}.
Is 1-level AES really more secure?
No! 1996 Biham "key collisions"
break 2^{40}-user 1-level AES
in exactly the same way.

Traditional 1-user metric:
Breaking AES using q queries
costs 2^{128} by best attack known.

Biham's multi-user metric:
$2^{128}/q$ by best attack known.

Loss factor ≤ 2 between
2-level AES and 1-level AES
in this multi-user metric.

ecurity proof, this paper:

factor $\leq q + 1$.

$)q + 1$ for ℓ levels.

e to ℓq from 2005 BCK.

/ independent ciphers

ter key, session keys.

success probability

master cipher,

session cipher

$- q\epsilon'$ vs. cascaded cipher.

ng 1 and 2:

ℓ-level security

tely from 2-level security.

2-level AES is breakable with 2^{40} queries, space 2^{40}, time 2^{48}.

Is 1-level AES really more secure? No! 1996 Biham "key collisions" break 2^{40}-user 1-level AES in exactly the same way.

Traditional 1-user metric: Breaking AES using q queries costs 2^{128} by best attack known.

Biham's multi-user metric: $2^{128}/q$ by best attack known.

Loss factor ≤ 2 between 2-level AES and 1-level AES in this multi-user metric.

oof, this paper:

$+ 1$.

ℓ levels.

m 2005 BCK.

ent ciphers

ssion keys.

obability

her,

pher

ascaded cipher.

2:

urity

2-level security.

2-level AES is breakable with 2^{40} queries, space 2^{40}, time 2^{48}.
Is 1-level AES really more secure?
No! 1996 Biham "key collisions" break 2^{40}-user 1-level AES in exactly the same way.

Traditional 1-user metric:
Breaking AES using q queries costs 2^{128} by best attack known.

Biham's multi-user metric:
$2^{128}/q$ by best attack known.

Loss factor ≤ 2 between 2-level AES and 1-level AES in this multi-user metric.

aper:

CK.

rs

pher.

curity.

2-level AES is breakable with 2^{40} queries, space 2^{40}, time 2^{48}.

Is 1-level AES really more secure?

No! 1996 Biham "key collisions" break 2^{40}-user 1-level AES in exactly the same way.

Traditional 1-user metric: Breaking AES using q queries costs 2^{128} by best attack known.

Biham's multi-user metric: $2^{128}/q$ by best attack known.

Loss factor ≤ 2 between 2-level AES and 1-level AES in this multi-user metric.

2-level AES is breakable with 2^{40} queries, space 2^{40}, time 2^{48}.
Is 1-level AES really more secure?
No! 1996 Biham "key collisions" break 2^{40}-user 1-level AES in exactly the same way.

Traditional 1-user metric:
Breaking AES using q queries costs 2^{128} by best attack known.

Biham's multi-user metric:
$2^{128}/q$ by best attack known.

Loss factor ≤ 2 between 2-level AES and 1-level AES in this multi-user metric.