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 $2^{n/2}$.”
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 $\#\text{messages} \approx 2^{n/2}$ (AES: $\#\text{messages} \approx 2^{64}$), and becomes quantitatively useless long before that.

So what should users do? No advice from 2006 BHHHKKR.
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

$AES_{AES_k(1)}(1)$, $AES_{AES_k(1)}(2)$, $\ldots$;

$AES_{AES_k(2)}(1)$, $AES_{AES_k(2)}(2)$, $\ldots$;

$AES_{AES_k(3)}(1)$, $AES_{AES_k(3)}(2)$, $\ldots$;

and so on.

This is really a new cipher

$\langle m, n \rangle \mapsto AES_{AES_k(m)}(n)$

with a double-size input.
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?
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.
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
\[ k' = (AES_k(2n), AES_k(2n + 1)). \]
Use \( k' \) as key for 256-bit AES.
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.
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. ⇒ 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.
Better security proof, this paper:

1. Loss factor \( \leq q + 1 \).
   \[ \leq (\ell - 1)q + 1 \text{ for } \ell \text{ levels.} \]
   Compare to \( \ell q \) from 2005 BCK.

2. Allow independent ciphers for master key, session keys.
   Attack success probability
   \[ \leq \varepsilon \text{ vs. master cipher,} \]
   \[ \leq \varepsilon' \text{ vs. session cipher} \]
   \[ \Rightarrow \leq \varepsilon + q\varepsilon' \text{ vs. cascaded cipher.} \]

Combining 1 and 2:
   deduce \( \ell \)-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?
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.
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 $\leq 2$ between 2-level AES and 1-level AES in this multi-user metric.