Some challenges in heavyweight cipher design

Daniel J. Bernstein

University of Illinois at Chicago & Technische Universiteit Eindhoven

Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

GET / HTTP/1.1

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

Can attacker deduce $\text{AES}_k(20)$?

We constantly tell people: “No! AES is secure! This is all safe!”
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:
GET / HTTP/1.1

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

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: “No! AES is secure! This is all safe!”

Attacker learns $\text{AES}_k(1)$ for, say, $2^{40}$ user keys $k$.
Attacker finds some user key using feasible $2^{88}$ computation.
Attacker decrypts, maybe forges, data for that user.
Is this $2^{128}$ “security”?
See 2002 Biham “key collisions”. 
Some challenges in heavyweight cipher design

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

Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$,
$m_2$ as $\text{AES}_k(2) \oplus m_2$,
$m_3$ as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block $m_1$ is predictable:
GET / HTTP/1.1
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: “No! AES is secure! This is all safe!”

Attacker learns AES $k$ for, say, $2^{40}$ user keys.
Attacker finds some user key using feasible $2^{88}$ computation.
Attacker decrypts, maybe forges, data for that user.
Is this $2^{128}$ “security”?
See 2002 Biham “key collisions”
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $AES_{k}(1) \oplus m_1$, $m_2$ as $AES_{k}(2) \oplus m_2$, $m_3$ as $AES_{k}(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

Attacker learns $AES_{k}(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

Can attacker deduce $AES_{k}(20)$? We constantly tell people: “No! AES is secure! This is all safe!”
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

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

Can attacker deduce $\text{AES}_k(20)$?

We constantly tell people: “No! AES is secure! This is all safe!”

Attacker learns $\text{AES}_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

```
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$,
$m_2$ as $\text{AES}_k(2) \oplus m_2$,
$m_3$ as $\text{AES}_k(3) \oplus m_3$,

etc. Also authenticates.

First block $m_1$ is predictable:

GET / HTTP/1.1\r\n
Attacker learns AES$_k$(1).

Can attacker deduce AES$_k$(20)?
We constantly tell people: “No! AES is secure! This is all safe!”

Attacker learns AES$_k$(1) for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation.
Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”?
See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

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

Can attacker deduce $\text{AES}_k(20)$? We constantly tell people: “No! AES is secure! This is all safe!”

Attacker learns $\text{AES}_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation.

Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys.

(Side discussion: Is 192 enough?)
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$,
message block $m_2$ as $\text{AES}_k(2) \oplus m_2$,
message block $m_3$ as $\text{AES}_k(3) \oplus m_3$,
and so on.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

Attacker learns AES$_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key $k$ using feasible $2^{88}$ computation.

Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”?

See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys.

(Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys:
quantum computing.

Grover finds $k$ from AES$_k(1)$ using $2^{64}$ iterations on a small quantum computer.

Another reason to be concerned about 128-bit cipher keys: quantum computing.
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

Attacker learns AES$_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”?

See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys.

(Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from AES$_k(1)$ using $2^{64}$ iterations on a small quantum processor.
Protocol generates new AES-128 key $k$.

Protocol encrypts message block $m_1$ as $\text{AES}_k(1) \oplus m_1$, $m_2$ as $\text{AES}_k(2) \oplus m_2$, $m_3$ as $\text{AES}_k(3) \oplus m_3$, etc. Also authenticates.

First block $m_1$ is predictable:

```
GET / HTTP/1.1
```

Attacker learns $\text{AES}_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation.

Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ "security"?

See 2002 Biham "key collisions".

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys.

(Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $\text{AES}_k(1)$ using $2^{64}$ iterations on a small quantum processor.
Attacker learns $AES_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys. (Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $AES_k(1)$ using $2^{64}$ iterations on a small quantum processor.
Attacker learns AES$_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys. (Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from AES$_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.
Attacker learns $AES_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation. Attacker decrypts, maybe forges, data for that user.

Is this $2^{128}$ “security”? See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys. (Side discussion: Is 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $AES_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Attacker learns $AES_k(1)$ for, say, $2^{40}$ user keys $k$.

Attacker finds some user key using feasible $2^{88}$ computation.

Attacker decrypts, maybe forges, data for that user.

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $AES_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization.


Should MACs have nonces?

To authenticate $m_1; m_2; m_3; m_4$:

Compute function with small differential probabilities. e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4$, where $r$ is secret.

Generate a one-time key $s_n = AES_n(k)$ from master key $k$.

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.
Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $AES_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization.


Should MACs have nonces?

To authenticate $(m_1, m_2, m_3, m_4)$:

Compute function with small differential probability, e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where $r$ is secret.

Generate a one-time key $s_n = AES_k(n)$ from master key $k$.

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.
Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $\text{AES}_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.


Should MACs have nonces?

To authenticate $(m_1, m_2, m_3, m_4)$:

Compute function with small differential probabilities. e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$ where $r$ is secret.

Generate a one-time key $s_n = \text{AES}_k(n)$ from master key $k$.

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.
Another reason to be concerned about 128-bit cipher keys:
quantum computing.

Grover finds $k$ from $\text{AES}_k(1)$
using $2^{64}$ iterations
on a small quantum processor.

Parallelize: $N^2$ processors,
each running $2^{64}/N$ iterations.
1999 Zalka claims this is optimal.

Multiple targets should allow
much better parallelization.
Related algos: 2009 Bernstein;

Should MACs have nonces?
To authenticate $(m_1, m_2, m_3, m_4)$:
Compute function with small
differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$,
where $r$ is secret.

Generate a **one-time** key
$s_n = \text{AES}_k(n)$ from master key $k$.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed:
consider, e.g., GCM.
Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from $AES_k(1)$ using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, running $2^{64}/N$ iterations. Zalka claims this is optimal.

Multiple targets should allow much better parallelization.


Should MACs have nonces?

To authenticate $(m_1, m_2, m_3, m_4)$:

Compute function with small differential probabilities.

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4, \]

where $r$ is secret.

Generate a **one-time** key

\[ s_n = AES_k(n) \]

from master key $k$.

Add to obtain MAC:

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4 + s_n. \]

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": twice in GCM $\Rightarrow$ repeated $s_n \Rightarrow$ attacker figures out $r$, can easily forge messages.
Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from AES$_k(1)$ using $2^{64}$ iterations on a small quantum processor. Parallelize: $N$ processors, each running $2^{64} = N$ iterations. 1999 Zalka claims this is optimal. Multiple targets should allow much better parallelization. Related algos: 2009 Bernstein; 2004 Grover–Radhakrishnan.

Should MACs have nonces?

To authenticate $(m_1, m_2, m_3, m_4)$:

Compute function with small differential probabilities. e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where $r$ is secret.

Generate a **one-time** key $s_n = $ AES$_k(n)$ from master key $k$.

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": not twice in GCM $\Rightarrow$ repeated $s_n \Rightarrow$ attacker figures out $r$, can easily forge messages.
Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds $k$ from AES using $2^{64}$ iterations on a small quantum processor.

Parallelize: $N^2$ processors, each running $2^{64} = N$ iterations.

1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization.


Should MACs have nonces?

To authenticate $(m_1, m_2, m_3, m_4)$:

Compute function with small differential probabilities.

E.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where $r$ is secret.

Generate a **one-time** key

$s_n = AES_k(n)$ from master key $k$.

Add to obtain MAC:

$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux “forbidden attack”: in GCM ⇒ repeated

⇒ attacker figures out $r$, can easily forge messages.
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

Compute function with small differential probabilities.

E.g., \(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4\),

where \(r\) is secret.

Generate a **one-time** key

\(s_n = \text{AES}_k(n)\) from master key \(k\).

Add to obtain MAC:

\(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n\).

Widely deployed for speed:

Consider, e.g., GCM.

---

2006 Joux “forbidden attack”:

- Twice in GCM ⇒ repeated \(s_n\)
- ⇒ Attacker figures out \(r\), can easily forge messages.
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

Compute function with small differential probabilities.

\[
e.g., r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4,
\]

where \(r\) is secret.

Generate a **one-time** key

\[
s_n = AES_k(n)
\]

from master key \(k\).

Add to obtain MAC:

\[
r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n.
\]

Widely deployed for speed:

consider, e.g., GCM.

2006 Joux “forbidden attack”:

\[\text{ntwice in GCM} \Rightarrow \text{repeated } s_n \Rightarrow \text{attacker figures out } r, \]

\[\text{can easily forge messages.}\]

Joux’s suggested response:

\[
AES_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)
\]

“seems a safe option”. (Also suggested and analyzed in, e.g.,

2000 Bernstein; earlier refs?)
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

Compute function with small differential probabilities.

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4, \]

where \(r\) is secret.

Generate a **one-time** key

\[ s_n = \text{AES}_k(n) \]

from master key \(k\).

Add to obtain MAC:

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4 + s_n. \]

Widely deployed for speed:

consider, e.g., GCM.

2006 Joux “forbidden attack”:

\(n\)twice in GCM \(\Rightarrow\) repeated \(s_n\)

\(\Rightarrow\) attacker figures out \(r\),

\(\Rightarrow\) can easily forge messages.

Joux’s suggested response:

\[ \text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4) \]

“seems a safe option”. (Also suggested and analyzed in, e.g.,

2000 Bernstein; earlier refs?)

Is this \(2^{128}\) “security”?
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

Compute function with small differential probabilities.

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4, \]

where \(r\) is secret.

Generate a **one-time** key

\[ s_n = \text{AES}_k(n) \]

from master key \(k\).

Add to obtain MAC:

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4 + s_n. \]

Widely deployed for speed:

consider, e.g., GCM.

2006 Joux “forbidden attack”:

\(\text{ntwice in GCM} \Rightarrow \text{repeated } s_n \Rightarrow \text{attacker figures out } r,\)

\(\text{can easily forge messages.}\)

Joux’s suggested response:

\[ \text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4) \]

“seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this \(2^{128}\) “security”?

Forgery chance \(\leq \delta + \epsilon\) where \(\epsilon\) is AES PRF insecurity and \n
\[ \delta \approx q^2 L / 2^{128} \]

for message lengths \(\leq L\).
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

1. Compute function with small differential probabilities.
   
   \[ m_1 + r^3 m_2 + r^2 m_3 + rm_4, \]

   where \(r\) is secret.

2. Generate a one-time key \(s_n = AES_k(n)\) from master key \(k\).

3. Obtain MAC:
   
   \[ r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4 + s_n. \]

Widely deployed for speed: consider, e.g., GCM.

2006 Joux “forbidden attack”:

twice in GCM \(\Rightarrow\) repeated \(s_n\)
\(\Rightarrow\) attacker figures out \(r\),
can easily forge messages.

Joux’s suggested response:

\(AES_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4)\)

“seems a safe option”. (Also suggested and analyzed in, e.g.,
2000 Bernstein; earlier refs?)

Is this \(2^{128}\) “security”?

Forgery chance \(\leq \delta + \epsilon\) where
\(\epsilon\) is AES PRF insecurity and
\(\delta \approx q^2 L / 2^{128}\)

for message lengths \(\leq L\).
Should MACs have nonces?

To authenticate \((m_1, m_2, m_3, m_4)\):

1. Compute function with small
differential probabilities.
   
   \[ r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4, \]

2. Generate a one-time
   key

   \[ s_n = AES_k(n) \]

   from master key \(k\).

3. Add to obtain MAC:
   
   \[ r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4 + s_n. \]

Widely deployed for speed:
consider, e.g., GCM.

2006 Joux “forbidden attack”:
ntwice in GCM \(\Rightarrow\) repeated \(s_n\)
\(\Rightarrow\) attacker figures out \(r\),
can easily forge messages.

Joux’s suggested response:
\[ AES_k(r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4) \]
“seems a safe option”. (Also
suggested and analyzed in, e.g.,
2000 Bernstein; earlier refs?)

\(\epsilon\) is at least \(q(q - 1) = 2^{129}\).

Solution: better PRP/PRF switch
(2005 Bernstein), ok for \(q \approx 2^{64}\).
Should MACs have nonces?

To authenticate \((m_1; m_2; m_3; m_4)\):

- Compute function with small differential probabilities.
- e.g., \(r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4\), where \(r\) is secret.
- Generate a one-time key \(s_n = \text{AES}_k(n)\) from master key \(k\).
- Add to obtain MAC: \(r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4 + s_n\).

Widely deployed for speed: consider, e.g., GCM.

### 2006 Joux “forbidden attack”:

- Twice in GCM \(\Rightarrow\) repeated \(s_n\)
- \(\Rightarrow\) attacker figures out \(r\), can easily forge messages.

Joux’s suggested response:

- \(\text{AES}_k(r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4)\)
- “seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

### Is this \(2^{128}\) “security”?

- Forgery chance \(\leq \delta + \epsilon\) where \(\epsilon\) is AES PRF insecurity and \(\delta \approx q^2 L / 2^{128}\)
- \(\epsilon\) is at least \(q(q - 1)/2^{129}\).
- Solution: better PRP/PRF switch (2005 Bernstein), ok for \(q \approx 2^{64}\).
2006 Joux “forbidden attack”: ntwice in GCM ⇒ repeated $s_n$ ⇒ attacker figures out $r$, can easily forge messages.

Joux’s suggested response: $\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + rm_4)$ “seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this $2^{128}$ “security”? Forgery chance $\leq \delta + \epsilon$ where $\epsilon$ is AES PRF insecurity and $\delta \approx q^2 L / 2^{128}$ for message lengths $\leq L$.

\[ \epsilon \text{ is at least } q(q - 1)/2^{129}. \]

Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$. \[ \delta \] is at least $q (q - 1) = 2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.\]
2006 Joux “forbidden attack”:  
ntwice in GCM ⇒ repeated $s_n$  
⇒ attacker figures out $r$,  
can easily forge messages.

Joux’s suggested response:  
$\text{AES}_k(r^4m_1 + r^3m_2 + r^2m_3 + rm_4)$  
“seems a safe option”. (Also suggested and analyzed in, e.g.,  
2000 Bernstein; earlier refs?)

Is this $2^{128}$ “security”?  
Forgery chance $\leq \delta + \epsilon$ where  
$\epsilon$ is AES PRF insecurity and  
$\delta \approx q^2L/2^{128}$  
for message lengths $\leq L$.  

$\epsilon$ is at least $q(q - 1)/2^{129}$.  
Solution: better PRP/PRF switch  
(2005 Bernstein), ok for $q \approx 2^{64}$.  
$\delta$ is still unacceptably large.  
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)
2006 Joux “forbidden attack”: ntwice in GCM ⇒ repeated \( s_n \) ⇒ attacker figures out \( r \), can easily forge messages.

Joux’s suggested response: 
\[
\text{AES}_k (r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)
\]
“seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this \( 2^{128} \) “security”?

Forgery chance \( \leq \delta + \epsilon \) where
\( \epsilon \) is AES PRF insecurity and
\( \delta \approx q^2 L / 2^{128} \)
for message lengths \( \leq L \).

\( \epsilon \) is at least \( q(q - 1)/2^{129} \).
Solution: better PRP/PRF switch (2005 Bernstein), ok for \( q \approx 2^{64} \).

\( \delta \) is still unacceptably large.
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: “Switch keys!”
2006 Joux “forbidden attack”: ntwice in GCM ⇒ repeated $s_n$ ⇒ attacker figures out $r$, can easily forge messages.

Joux’s suggested response:

$$AES_k(r^4m_1 + r^3m_2 + r^2m_3 + rm_4)$$

“seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this $2^{128}$ “security”?

Forgery chance $\leq \delta + \epsilon$ where

- $\epsilon$ is AES PRF insecurity and
- $\delta \approx q^2L/2^{128}$

for message lengths $\leq L$.

$\epsilon$ is at least $q(q - 1)/2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

$\delta$ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: “Switch keys!”

Much simpler: 256-bit blocks.

2014 Bernstein–Chou “Auth256”:

- 29 bit ops/message bit for differential probability $< 2^{-255}$
- Or try EHC from 2013 Nandi?
Joux “forbidden attack”: \[ \text{in GCM} \Rightarrow \text{repeated } s_n \Rightarrow \text{attacker figures out } r, \quad \text{can easily forge messages.} \]

Suggested response:

\[ r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 \]

“seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

\[ \epsilon \] is at least \( q(q - 1)/2^{129} \).

Solution: better PRP/PRF switch (2005 Bernstein), ok for \( q \approx 2^{64} \).

\( \delta \) is still unacceptably large.

(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: “Switch keys!”

Much simpler: 256-bit blocks.

2014 Bernstein–Chou “Auth256”:

29 bit ops/message bit for differential probability \(< 2^{-255} \).

Or try EHC from 2013 Nandi?

Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
2006 Joux “forbidden attack”:
repeated $s_n$ ⇒ attacker figures out $r$,
can easily forge messages.
Joux’s suggested response:
$AES(k^4 + r^3m_2 + r^2m_3 + rm_4)$
"seems a safe option". (Also analyzed in, e.g., 2000 Bernstein; earlier refs?) Is this $2^{128}$ “security”?
$\delta + \epsilon$ where $\delta$ is AES PRF insecurity and $\epsilon$ is at least $q(q - 1)/2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.
$\delta$ is still unacceptably large.
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)
Fragile solution: “Switch keys!”
Much simpler: 256-bit blocks.
Or try EHC from 2013 Nandi?
Improving Tor
Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.
(But current cipher is a disaster, so can consider compromises.)
Also: secure chaining from each block to the next.
Tor is considering deployment of AEZ or HHFHFH in 2016.
See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
2006 Joux “forbidden attack”:
- $ntw \rightarrow s_n \rightarrow$ attacker figures out $r$,
can easily forge messages.
- Joux’s suggested response:
  - AES $k(r_4 m_1 + r_3 m_2 + r_2 m_3 + r m_4)$
  - “seems a safe option”. (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this $2^{128}$ “security”?
- Forgery chance $\leq \epsilon + \delta$ where
  - $\epsilon$ is AES PRF insecurity and
  - $\delta = q^2 2^{128}$ for message lengths $\leq L$.

$\delta$ is at least $q(q - 1) = 2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

$\delta$ is still unacceptably large.
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)
Fragile solution: “Switch keys!”

Much simpler: 256-bit blocks.

2014 Bernstein–Chou “Auth256”:
- 29 bit ops/message bit for differential probability $< 2^{-255}$.
- Or try EHC from 2013 Nandi?

Improving Tor
Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.
(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.
Tor is considering deployment of AEZ or HHFHFH in 2016.
See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
\[ \epsilon \text{ is at least } q(q - 1)/2^{129}. \]
Solution: better PRP/PRF switch (2005 Bernstein), ok for \( q \approx 2^{64} \).
\[ \delta \text{ is still unacceptably large.} \]
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)
Fragile solution: “Switch keys!”
Much simpler: 256-bit blocks.
2014 Bernstein–Chou “Auth256”: 29 bit ops/message bit for differential probability < \( 2^{-255} \).
Or try EHC from 2013 Nandi?

**Improving Tor**
Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.
(But current cipher is a disaster, so can consider compromises.)
Also: secure chaining from each block to the next.
Tor is considering deployment of AEZ or HHFHFH in 2016.
See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
least $q(q - 1)/2^{129}$.

Solution: better PRP/PRF switch (Bernstein), ok for $q \approx 2^{64}$.

is still unacceptably large.

(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: “Switch keys!”

Solution: “Switch keys!”

Simpler: 256-bit blocks.

Bernstein–Chou “Auth256”:

29 bit ops/message bit for differential probability $< 2^{-255}$.

EHC from 2013 Nandi?

Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patentencumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.

Tor wants “fast, proven, secure, easy-to-implement, non-patentencumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
is at least $q(q-1) = 2^{129}$.

Practical PRP/PRF switch look for $q \approx 2^{64}$.

Possibly large.

(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

“Switch keys!”

Call for 56-bit blocks.

Fragile solution: “Switch keys!”

“Auth256”:

256-bit blocks.

29 bit ops/message bit for differential probability $< 2^{-255}$.

Or try EHC from 2013 Nandi?

Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.

---

Feistel

CTR

OFB

 ↙ ↙

NR

CMC

EME

XCB

HCTR

PEP

HCH

TET

HEH

iHCH

HOH

EMME

↓ ↓

stream cipher (strong PRF)

Feistel

SCTES

HHFHFH

↘ ↘

block cipher (strong SPRP)
Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
Improving Tor

Tor wants “fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free” 509-byte block cipher.

The current cipher is a disaster, so can consider compromises.

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte block cipher.

But current cipher is a disaster, so can consider compromises.

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent encumbered, side-channel-free" 509-byte block cipher.

(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each block to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).

$H$ is purely combinatorial;

$F$ is a stream cipher.

Ingredients: 4-round Feistel;

$H$ at top (1996 Lucks),

bottom (1997 Naor–Reingold);

$H_1, H_2, H_3, H_4$ allow one-block nonces;

$H_1, H_2$ are stretched by 0-pad;


Allow one $H_1, H_2, H_3, H_4$ key;

unify $H_1, H_2$ hypotheses;

unify $H_3, H_4$ hypotheses.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar). $H$ is purely combinatorial; $F$ is a stream cipher.

Ingredients: 4-round Feistel; $H$ at top (1996 Lucks), bottom (1997 Naor–Reingold); $H_2, H_3$ allow one-block nonces; $H_1, H_4$ are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one $H_1, H_2, H_3, H_4$ key; unify $H_1, H_2$ hypotheses; unify $H_3, H_4$ hypotheses.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).

$H$ is purely combinatorial; $F$ is a stream cipher.

Ingredients: 4-round Feistel; $H$ at top (1996 Lucks), bottom (1997 Naor–Reingold); $H_2, H_3$ allow one-block nonces; $H_1, H_4$ are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one $H_1, H_2, H_3, H_4$ key; unify $H_1, H_2$ hypotheses; unify $H_3, H_4$ hypotheses.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar). 
$H$ is purely combinatorial; 
$F$ is a stream cipher.

Ingredients: 4-round Feistel; 
$H$ at top (1996 Lucks), 
bottom (1997 Naor–Reingold); 
$H_2, H_3$ allow one-block nonces; 
$H_1, H_4$ are stretched by 0-pad; 
XCB/HCTR-style tweak, faster 

Allow one $H_1, H_2, H_3, H_4$ key; 
unify $H_1, H_2$ hypotheses; 
unify $H_3, H_4$ hypotheses.
Previous slide: HHFHFH
(Bernstein–Nandi–Sarkar).
$H$ is purely combinatorial;
$F$ is a stream cipher.

Ingredients: 4-round Feistel;
$H$ at top (1996 Lucks),
bottom (1997 Naor–Reingold);
$H_2, H_3$ allow one-block nonces;
$H_1, H_4$ are stretched by 0-pad;
XCB/HCTR-style tweak, faster

Allow one $H_1, H_2, H_3, H_4$ key;
unify $H_1, H_2$ hypotheses;
unify $H_3, H_4$ hypotheses.

One possibility for $F$:
permutation in EM in CTR.
Full-width permutation output
beats squeezing for long output;
and CTR is highly parallel.
Also choose highly parallel $H$.
We’re still optimizing choices.
Use single-block tweak $w$.
“chopTC”: chain by choosing
$w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array
twice, writes each bit once.
Something I’m working on now:
more locality inside permutation.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).

$H$ is purely combinatorial; $F$ is a stream cipher.

Ingredients: 4-round Feistel; $H$ at top (1996 Lucks), bottom (1997 Naor–Reingold); $H_2, H_3$ allow one-block nonces; $H_1, H_4$ are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one $H_1, H_2, H_3, H_4$ key; unify $H_1, H_2$ hypotheses; unify $H_3, H_4$ hypotheses.

One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$ as truncation of $P \oplus C$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).

$H$ is purely combinatorial; $F$ is a stream cipher.

Ingredients: 4-round Feistel; $H$ at top (1996 Lucks), bottom (1997 Naor–Reingold); $H_2, H_3$ allow one-block nonces; $H_1, H_4$ are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one $H_1, H_2, H_3, H_4$ key; unify $H_1, H_2$ hypotheses; unify $H_3, H_4$ hypotheses.

One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$.

We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.
Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).

$H$ is purely combinatorial;
$F$ is a stream cipher.

Ingredients: 4-round Feistel;
$H$ at top (1996 Lucks),
bottom (1997 Naor–Reingold);
$H_2, H_3$ allow one-block nonces;
$H_1, H_4$ are stretched by 0-pad;
XCB/HCTR-style tweak, faster

Allow one $H_1, H_2, H_3, H_4$ key;
unify $H_1, H_2$ hypotheses;
unify $H_3, H_4$ hypotheses.

One possibility for $F$:
permutation in EM in CTR.

Full-width permutation output
beats squeezing for long output;
and CTR is highly parallel.

Also choose highly parallel $H$.
We’re still optimizing choices.

Use single-block tweak $w$.
“chopTC”: chain by choosing
$w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array
twice, writes each bit once.
Something I’m working on now:
more locality inside permutation.
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$.

We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2 = 2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”? 
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?

One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$. We’re still optimizing choices.

Use single-block tweak $w$. “chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$. 
One possibility for $F$: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel $H$.

We’re still optimizing choices.

Use single-block tweak $w$.

“chopTC”: chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$:
basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit $n$ safe in ChaCha?
One possibility for $F$: permutation in EM in CTR. With permutation output squeezing for long output; CTR is highly parallel.

Choose highly parallel $H$. Still optimizing choices.

Single-block tweak $w$. CHFHFH reads each bit in array twice, writes each bit once.

Something I’m working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”? Fragile fix: “beyond-birthday-bound security.” Complicates implementation, security analysis.

Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit $n$ safe in ChaCha?

Heavyweight ciphers
Interesting cipher -design space:
$\geq 256$ bits for all pipes,
$\geq 256$-bit keys,
$\geq 256$-bit outputs,
$\geq 256$-bit subkeys, etc.
One possibility for \( F \): permutation in EM in CTR. Full-width permutation output beats squeezing for long output; and CTR is highly parallel. Also choose highly parallel \( H \). We're still optimizing choices.

Use single-block tweak \( w \). "chopTC": chain by choosing \( w \) as truncation of \( P \oplus C \).

HHFHFH reads each bit in array twice, writes each bit once.

Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of \( F \): basically \( q^2/2^{128} \), assuming 128-bit blocks and typical choice of \( H \).

Is this \( 2^{128} \) “security”? Fragile fix: “beyond-birthday-bound security.” Complicates implementation, security analysis.

Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security \( q^2/2^{256} \).

Is 256-bit \( n \) safe in ChaCha?

Heavyweight ciphers

Interesting cipher design space:

- \( \geq 256 \) bits for all pipes.
- \( \geq 256 \)-bit keys, \( \geq 256 \)-bit outputs,
- \( \geq 256 \)-bit subkeys, etc.
One possibility for $F$: permutation in EM in CTR. Full-width permutation output beats squeezing for long output; and CTR is highly parallel. Also choose highly parallel $H$. We're still optimizing choices.

Use single-block tweak $w$. "chopTC": chain by choosing $w$ as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.

Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ "security"?


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit $n$ safe in ChaCha?

Heavyweight ciphers

Interesting cipher-design space:

$\geq 256$ bits for all pipes.
$\geq 256$-bit keys, $\geq 256$-bit outputs,
$\geq 256$-bit subkeys, etc.
Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit $n$ safe in ChaCha?

Heavyweight ciphers

Interesting cipher-design space:

≥256 bits for all pipes.

≥256-bit keys, ≥256-bit outputs, ≥256-bit subkeys, etc.
Security loss of mode compared to security of $F$: basically $q^2 / 2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2 / 2^{256}$.

Is 256-bit $n$ safe in ChaCha?

---

Heavyweight ciphers

Interesting **cipher**-design space:

- $\geq 256$ bits for all pipes.
- $\geq 256$-bit keys, $\geq 256$-bit outputs,
- $\geq 256$-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, . . . . This needs far more attention, optimization.

Hash designs are usually overkill.
Security loss of mode compared to security of $F$: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”? 


Simpler fix: “bigger-birthday-bound security.” Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit $n$ safe in ChaCha?

Heavyweight ciphers

Interesting cipher-design space: $\geq 256$ bits for all pipes. $\geq 256$-bit keys, $\geq 256$-bit outputs, $\geq 256$-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, .... This needs far more attention, optimization.

Hash designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!
Security loss of mode compared to security of $F$: 

$$q^2/2^{128},$$

assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ “security”?


Simpler fix: “bigger-birthday-bound security.” Use $2^{256}$-bit blocks, security $q^2 = 2^{256}$.

Is 256-bit $n$ safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space:

- $\geq 256$ bits for all pipes.
- $\geq 256$-bit keys, $\geq 256$-bit outputs,
- $\geq 256$-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization.

**Hash** designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

Another optimization target: 

PRF inside EdDSA signatures.

EdDSA generates per-signature random number mod $2^{256}$ as truncated hash: $H(s;m)$ mod $2^{256}$.

$H$ is SHA-512; $s$ is subkey.

2015 Bellare–Bernstein–Tessaro: truncated prefixed MD hash is a high-security multi-user MAC.

Even with the constraint of reusing preimage-resistant hash, surely can build better design in both software and hardware.
Security loss of mode compared to security of $F$:
basically $q^2 = 2^{128}$,
assuming 128-bit blocks and typical choice of $H$.

Is this $2^{128}$ "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2 = 2^{256}$.

Is 256-bit safe in ChaCha?

Heavyweight ciphers

Interesting cipher-design space:
$\geq 256$ bits for all pipes.
$\geq 256$-bit keys, $\geq 256$-bit outputs,
$\geq 256$-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, .....

This needs far more attention, optimization.

Hash designs are usually overkill.

Is 256 fundamentally much slower,
or much less energy-efficient,
than 128? My guess: No!

Another optimization target:
PRF inside EdDSA signatures.

EdDSA generates a random number mod 256-bit 'as truncated hash: $H(s;m)$ mod $2^{256}$.$H$ is SHA-512; $s$ is subkey.

2015 Bellare–Bernstein–Tessaro: truncated prefixed MD hash is a high-security multi-user MAC.

Even with the constraint of reusing preimage-resistant hash,
surely can build better design in both software and hardware.
Heavyweight ciphers

Interesting **cipher**-design space:
≥ 256 bits for all pipes.
≥ 256-bit keys, ≥ 256-bit outputs,
≥ 256-bit subkeys, etc.

Occasional designs: Rijndael,
OMD (SHA-2), Keccak, BLAKE2,
NORX, Simpира, . . . . This needs
far more attention, optimization.

**Hash** designs are usually overkill.

Is 256 fundamentally much slower,
or much less energy-efficient,
than 128? My guess: No!

Another optimization target:
PRF inside EdDSA signatures.

EdDSA generates per-signature
random number mod 256-bit
truncated hash: $H(s, m) \mod 2^{256}$.
$H$ is SHA-512; $s$ is subkey.

2015 Bellare–Bernstein–Tessaro:
truncated prefixed MD hash
high-security multi-user MAC.

Even with the constraint of
reusing preimage-resistant hash,
surely can build better design
in both software and hardware.
Heavyweight ciphers

Interesting **cipher**-design space:

≥ 256 bits for all pipes.
≥ 256-bit keys, ≥ 256-bit outputs,
≥ 256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpира, . . . . This needs far more attention, optimization.

**Hash** designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

Another optimization target:

PRF inside EdDSA signatures.

EdDSA generates per-signature random number mod 256-bit $\ell$ as truncated hash: $H(s, m) \mod \ell$. $H$ is SHA-512; $s$ is subkey.

2015 Bellare–Bernstein–Tessaro: truncated prefixed MD hash is a high-security multi-user MAC.

Even with the constraint of reusing preimage-resistant hash, surely can build better design in both software and hardware.