The Poly1305-AES message-authentication code

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

The Poly1305-AES function

Given byte sequence m,
16-byte sequence n,
16-byte sequence k,
16-byte sequence r
with certain bits cleared,
Poly1305-AES produces
16-byte sequence
$\text{Poly1305}_r(m, \text{AES}_k(n))$.

Very simple definition
using polynomial evaluation
modulo the prime $2^{130} - 5$.

Poly1305-AES authenticators

Sender, receiver share secret uniform random k, r.

Sender attaches authenticator $a = \text{Poly1305}_r(m, \text{AES}_k(n))$ to message m with nonce n.

(The usual nonce requirement: never use the same nonce for two different messages.)

Receiver rejects n', m', a' if $a' \neq \text{Poly1305}_r(m', \text{AES}_k(n'))$.

Poly1305-AES security guarantee

Attacker adaptively chooses $C \leq 2^{64}$ messages, sees their authenticators, attempts D forgeries; all messages $\leq L$ bytes.

Define δ as attacker's chance of breaking AES, i.e., distinguishing AES_k from uniform random permutation using $C + D$ queries.

Then Pr[all forgeries rejected] $\geq 1 - \delta - 14D \lceil L/16 \rceil / 2^{106}$.

Example: Say $L = 1536$; $\delta \le 2^{-40}$; see 2^{64} authenticators; attempt 2^{64} forgeries. Then $\Pr[\text{all rejected}] \ge 0.999999999998$.

For comparison, that much effort easily breaks many other 16-byte MACs: CBC-AES, HMAC-MD5, DMAC-AES, etc.

Those MACs have guarantees too! How can they possibly be broken? Answer: Look at the numbers. e.g. "$8LC^2/2^{128}$" is not small.

Do nonces require "additional message expansion overhead"? No!

Consider TCP connection transmitting (e.g.) 2^{64} bytes $x_0, x_1, \ldots, x_{12345678901}, \ldots$.

Message $(x_i, x_{i+1}, \ldots, x_j)$ has nonce (i, j) known to both sides. (TCP sequence number is bottom 32 bits of i, but both sides know top bits too.)

Using this nonce for cryptography does not take any extra bandwidth.

Poly1305-AES speed

Fast public-domain software now available: `cr.yp.to/mac.html`.

CPU cycles for ℓ-byte message with all data aligned in L1 cache:

ℓ	16	128	1024
Athlon	712	1055	3843
Pentium III	746	1247	5361
PowerPC Sstar	910	1459	5905
UltraSPARC III	854	1383	5601

Bottom line: Faster than MD5.
Much faster than CBC-AES etc.

Unaligned messages

Some applications can easily guarantee alignment; some can't.

CPU cycles for ℓ-byte message with all data unaligned:

ℓ	43	127	1025
Athlon	890	1152	4060
Pentium III	970	1383	5316
PowerPC Sstar	1159	1560	6083
UltraSPARC III	1075	1444	5742

Many more situations covered in comprehensive speed tables:

`cr.yp.to/mac/speed.html`

The art of benchmarking

Many deceptive timings in the cryptographic literature:

- Bait-and-switch timings.
- Guesses reported as timings.
- My-favorite-CPU timings.
- Long-message timings.
- Timings after precomputation.

Consequence: In the real world, these functions are often much slower than advertised.

In contrast, Poly1305-AES provides *consistent* high speed.

Bait-and-switch timings

Deception strategy: Create two versions of your function, a small Fun-Breakable and a big Fun-Slow. Report timings for Fun-Breakable.

Example in literature:
"More than 1 Gbit/sec
on a 200 MHz Pentium Pro"
. . . if you switch to a
silly 4-byte authenticator.

The honest alternative:
Focus on *one* function.
Poly1305-AES is strong *and* fast.

Guesses reported as timings

Deception strategy: Measure only part of the computation. Estimate the other parts.

Example in literature: "achieves 2.2 clock cycles per byte" . . . if the unimplemented parts are as fast as various estimates.

The honest alternative: Measure exactly the function call `verify(a,kr,n,m,mlen)` that applications will use.

My-favorite-CPU timings

Deception strategy: Choose CPU where function is very fast. Ignore all other CPUs.

Example in literature: "All speeds were measured on a Pentium 4" . . . because other chips take many more cycles per byte for this particular computation.

The honest alternative: Measure every CPU you can find. If reader doesn't care about a particular chip, he can ignore it.

Long-message timings

Deception strategy: Report time only for long messages.
Ignore per-message overhead.
Ignore applications that handle short messages.

Example in literature:
"2 cycles per byte"
... plus 2000 cycles per message.

The honest alternative:
Report times for n-byte messages for each $n \in \{0, 1, 2, \ldots, 8192\}$.

Timings after precomputation

Deception strategy: Report time to compute authenticator *after* a big key-dependent table has been precomputed and loaded into L1 cache. Ignore applications that handle many simultaneous keys.

I'm guilty of this! In April 1999, I broke the MD5 speed barrier, but only by ignoring the cost of handling big key-dependent tables. Many newer functions: same issue.

The honest alternative:
Measure precomputation time;
measure time to load inputs
that weren't already in cache.

My Poly1305-AES timings
include AES key expansion and
all necessary r computations.
Cache effects: see `speed.html`.

Poly1305-AES offers much higher
key agility than hash127-AES etc.

Crucial detail: $2^{130} - 5$
allows 128-bit coefficients.