# Cycle counts for authenticated encryption

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago, Chicago, IL 60607–7045
djb@crypt.com

<table>
<thead>
<tr>
<th>System</th>
<th>Cipher key bits</th>
<th>Cipher</th>
<th>MAC</th>
<th>Total key bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc-v3-poly1305</td>
<td>128</td>
<td>ABC v3</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>aes-128-poly1305</td>
<td>128</td>
<td>10-round AES</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>aes-256-poly1305</td>
<td>256</td>
<td>14-round AES</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>cryptmt-v3-poly1305</td>
<td>256</td>
<td>CryptMT 3</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>dicing-p2-poly1305</td>
<td>256</td>
<td>DICING P2</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>dragon-poly1305</td>
<td>256</td>
<td>Dragon</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>grain-128-poly1305</td>
<td>128</td>
<td>Grain-128</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>grain-v1-poly1305</td>
<td>80</td>
<td>Grainv1</td>
<td>Poly1305</td>
<td>208</td>
</tr>
<tr>
<td>hc-128-poly1305</td>
<td>128</td>
<td>HC-128</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>hc-256-poly1305</td>
<td>256</td>
<td>HC-256</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>lex-v1-poly1305</td>
<td>128</td>
<td>LEX v1</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>mickey-128-2-poly1305</td>
<td>128</td>
<td>MICKEY-128 2.0</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>nls-ae</td>
<td>128</td>
<td>NLS</td>
<td>built-in</td>
<td>128</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>128</td>
<td>NLS</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>phelix</td>
<td>256</td>
<td>Phelix</td>
<td>built-in</td>
<td>256</td>
</tr>
<tr>
<td>py6-poly1305</td>
<td>256</td>
<td>Py6</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>py-poly1305</td>
<td>256</td>
<td>Py</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>pypy-poly1305</td>
<td>256</td>
<td>Pypy</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>rabbit-poly1305</td>
<td>128</td>
<td>Rabbit</td>
<td>Poly1305</td>
<td>256</td>
</tr>
<tr>
<td>rc4-poly1305</td>
<td>256</td>
<td>RC4</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>salsa20-8-poly1305</td>
<td>256</td>
<td>Salsa20/8</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>salsa20-12-poly1305</td>
<td>256</td>
<td>Salsa20/12</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>salsa20-poly1305</td>
<td>256</td>
<td>Salsa20</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>snow-2.0-poly1305</td>
<td>256</td>
<td>SNOW 2.0</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>sosemunuk-poly1305</td>
<td>256</td>
<td>SOSEMANUK</td>
<td>Poly1305</td>
<td>384</td>
</tr>
<tr>
<td>trivium-poly1305</td>
<td>80</td>
<td>TRIVIUM</td>
<td>Poly1305</td>
<td>208</td>
</tr>
</tbody>
</table>

* Date of this document: 2007.01.13. Permanent ID of this document: be6b4df07e1bea67aba9338991b78388.
Abstract. How much time is needed to encrypt, authenticate, verify, and decrypt a packet? The answer depends on the machine (most importantly, but not solely, the CPU), on the choice of authenticated-encryption function, on the packet length, on the level of competition for the instruction cache, on the number of keys handled in parallel, et al. This paper reports, in graphical and tabular form, measurements of the speeds of a wide variety of authenticated-encryption functions on a wide variety of CPUs.

This paper reports speed measurements for the secret-key authenticated-encryption systems listed on the first page.

I included all of the “software focus” ciphers (Dragon, HC, LEX, Phelix, Py, Salsa20, SOSEMANUK) in phase 2 of eSTREAM, the ECRYPT Stream Cipher Project; all of the “hardware focus” ciphers (Grain, MICKEY, Phelix, Trivium); the remaining “software” ciphers, except for Polar Bear, which I couldn’t make work; and the “benchmark” ciphers (AES, RC4, SNOW 2.0) for comparison.

I did not exclude ciphers for which there are claims of attacks: ABC, NLS, Py, and RC4. For LEX, I chose version 1 (for which there is a claim of an attack) rather than version 2 (for which there are no such claims) because I’m not aware of functioning software for version 2 of LEX; my impression is that the versions will have similar speeds, but speculation is no substitute for measurement.

Non-authenticating stream ciphers

Most of the stream ciphers do not include message authentication. I converted each non-authenticating stream cipher into an authenticated-encryption system by combining it in a standard way with Poly1305, a state-of-the-art message-authentication code.

Here are the details: The key for the authenticated-encryption system is $(r, k)$ where $r$ a 16-byte Poly1305 key and $k$ is a key for the non-authenticating stream cipher $F$. The authenticated encryption of a message $m$ with nonce $n$ is $(\text{Poly1305}_r(c, s), c)$ where $(s, c) = F_k(n) \oplus (0, m)$, both $s$ and 0 having 16 bytes. Here $F_k(n)$ is the “keystream” produced by $F$ using key $k$ and nonce $n$, and $\oplus$ xors its inputs after truncating the longer input to the same length as the shorter input.

Previous eSTREAM benchmarks did not include separate authenticators; they simply reported encryption timings for non-authenticating ciphers along with encryption timings for authenticating ciphers. The reality is that users need authenticated encryption, not just encryption, so they need to combine non-authenticating ciphers with message-authentication codes, slowing down those ciphers. How quickly do these combined systems handle legitimate packets, and how quickly do they reject forged packets? Are they faster than ciphers with built-in authentication? To compare the speeds of authenticating ciphers and non-authenticating ciphers from the user’s perspective, benchmarks must take the extra authentication time into account.
“Isn’t this a purely academic question?” one might ask. “Haven’t all the authenticating ciphers been broken? Frogbit flunks a simple IV-diffusion test. Courtois broke SFINKS. Cho and Piperzyk broke both versions of NLS. Wu and Preneel broke Phelix. Okay, okay, VEST is untouched, but it’s much too expensive for anyone to want to use.” The simplest response is that, in fact, Phelix has not been broken. (The Wu-Preneel “attack” ignores both the concept of a nonce and the standard definition of cipher security; the “attack” assumes that senders repeat nonces. The same silly assumption easily “breaks” every eSTREAM submission.) Phelix remains one of the top eSTREAM submissions.

I’m planning future work to extend my database of timings to cover other authenticated-encryption systems. I plan to include more ciphers, for example; I plan to include other modes of use of Poly1305; and I plan to include UMAC, VMAC, CBC-MAC, and HMAC-SHA-1 as alternatives to Poly1305. I will also endeavor to incorporate improved implementations of systems already covered: for example, I’m planning a 64-bit implementation of Poly1305. But the existing data should already be useful in comparing eSTREAM candidates.

“Why is it necessary to time authenticated encryption?” one might ask. “If you want a table of authenticated-encryption timings, why not simply add a table of authentication timings to a table of encryption timings?” Response: The existing tables are deficient. This paper’s timings are much more comprehensive than previous encryption timings. This paper systematically measures all packet lengths in a wide range, for example, and systematically measures multiple-key cache-miss costs. Furthermore, adding all the contributing times isn’t as easy as it sounds; for example, if the authentication software uses more than half of the code cache, and the encryption software uses more than half of the code cache, authenticated encryption will need time for code-cache misses. Component benchmarks can be interesting and informative, but whole-function benchmarks are the simplest way to ensure that no components are forgotten.

API for authenticated-encryption systems

What does a secret-key authenticated-encryption system do for the user? It takes keys; it encrypts and authenticates each outgoing packet; it verifies and decrypts each incoming packet. So I specified an authenticated-encryption API with three functions: $\textit{makekey}$ to generate a key (and an “expanded key,” the output of any desired precomputation); $\textit{encrypt}$ to authenticate and encrypt an outgoing packet; and $\textit{decrypt}$ to verify and decrypt an incoming packet.

The $\textit{encrypt}$ function includes an authenticator in its encrypted output packet. The $\textit{decrypt}$ function is given an encrypted packet allegedly produced by $\textit{encrypt}$; it rejects the packet if the authenticator is wrong. Many systems can limit their decryption work for long packets when the authenticator is wrong. In particular, for the Poly1305 combination described above, an authenticator can be checked as soon as 16 bytes of keystream have been generated; if the authenticator is wrong then one can skip the work of generating the remaining bytes of keystream.
In contrast, in the official eSTREAM stream-cipher API, both `encrypt` and `decrypt` put an authenticator somewhere else. It is the responsibility of the `decrypt` user to verify authenticators. Having `decrypt` write an authenticator, rather than read it, means that rejection of forged packets is necessarily just as slow as decryption of legitimate packets. This doesn’t seem to have been a problem for the authenticating stream ciphers submitted to eSTREAM, but it unnecessarily slows down other authenticated-encryption systems.

There are many other details of the new API, but this paper can be read without regard to those details. Example: `encrypt` and `decrypt` receive lengths as 64-bit integers (`long long` in C). On many CPUs, using fewer bits for lengths would save a few cycles, marginally shifting the graphs in this paper.

**Tools for benchmarking**

Previous eSTREAM speed reports use the official eSTREAM benchmarking toolkit. The toolkit includes (1) software written by Christophe de Cannière to measure the speeds of stream-cipher implementations that follow the official eSTREAM stream-cipher API and (2) stream-cipher implementations collected from cipher authors.

To collect the timings reported in this paper I wrote a new benchmarking toolkit, `ciphercycles`, available from [http://cr.yp.to/streamciphers.html](http://cr.yp.to/streamciphers.html). I wrote a separate tool to convert stream ciphers from the official eSTREAM stream-cipher API to my new API (and in particular to add authentication to the non-authenticating stream ciphers); the resulting implementations are included in the toolkit. Subsequent updates to the implementations in the official eSTREAM benchmarking toolkit will be easy to reflect in `ciphercycles`.

Many portions of `ciphercycles` are derived from BATMAN (Benchmarking of Asymmetric Tools on Multiple Architectures, Non-Interactively), a public-key benchmarking toolkit that I wrote for eBATS (ECRYPT Benchmarking of Asymmetric Systems). The new speed reports produced by `ciphercycles`, like the eBATS speed reports, are in a simple format designed for easy computer processing. I’m planning future work to integrate benchmarking projects.

The timings collected by `ciphercycles` include (authenticated) encryption, (verified) decryption of legitimately encrypted packets, and rejection of forged packets. Decryption times are usually almost identical to encryption times, but rejection times are often much smaller, for the reasons discussed above. The official eSTREAM timings include only encryption times.

The timings collected by `ciphercycles` systematically cover each packet length between 0 bytes and 8192 bytes. By superimposing graphs one can easily see the packet-length cutoffs between different ciphers. The official eSTREAM timings include only a few selected lengths (40 bytes, 576 bytes, 1500 bytes, long), hiding block-size penalties and many other length-dependent effects.

The timings collected by `ciphercycles` include benchmarks for encryption of short packets bouncing between multiple keys: for example, when there are 1024 active keys, how many cycles are used for encryption of a 775-byte packet under a random choice of key; including the cache misses needed to access the
The official eSTREAM timings include one fuzzy “agility” number for each cipher but are otherwise dedicated to single-key benchmarks.

The timings collected by ciphercycles also include makekey timings, but those timings are not reported in this paper.

Graphs

The sample graph on the left below shows timings for the abc-v3-poly1305 system on a 900MHz AMD Athlon (622) computer named thoth.

The horizontal axis is packet length, between 0 bytes and 8192 bytes. The vertical axis is time, between 0 cycles and 98304 cycles. The diagonal from the lower left corner of the graph to the upper right corner is 12 cycles per byte.

The two main lines visible on the graph are (1) roughly 7 cycles per byte for encryption and decryption and (2) roughly 3 cycles per byte for rejection. Faint lines are visible above the main lines; there are 15 timings for each packet length, and initial timings are slightly slower because of cache misses. There is also a short curve up the left side of the graph for encrypting packets of ≤ 1024 bytes using a random key from a pool of 1024 active keys. Also plotted, and faintly visible, are packet lengths of ≤ 960 bytes for 512 active keys, packet lengths of ≤ 896 bytes for 256 active keys, etc.

The sample graph on the right shows timings for the pypy-poly1305 system on a 2137MHz Intel Core 2 Duo (6f6) computer named katana. The spreading line shows variance in Pypy’s stream-generation time, perhaps from cache-timing effects. Note also the large cost of handling small packets.
Graphs for grain-128-poly1305, grain-v1-poly1305, mickey-128-2-poly1305 are omitted. All three systems are very slow.
Here are the machines used (in order) for the above graphs:

- a 900MHz AMD Athlon (622) named *thoth*;
- an 800MHz Pentium M (6d8) named *atlas*;
- a 900MHz Sun UltraSPARC III named *wessel*;
- a 2137MHz Intel Core 2 Duo (6f6) named *katana*;
- a 2000MHz AMD Athlon 64 X2 (15,75,2) named *mace*; and
- a 533MHz Motorola PowerPC G4 7410 named *gggg*.

**Tables**

The following table shows median cycle counts for authenticated encryption as a function of cipher and packet length. All timings are from a 900MHz AMD Athlon (622) named *thoth*. All timings are for a single active key.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>40</th>
<th>402</th>
<th>576</th>
<th>1500</th>
<th>8192</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc-v3-poly1305</td>
<td>946</td>
<td>1635</td>
<td>4341</td>
<td>5525</td>
<td>12654</td>
<td>64314</td>
</tr>
<tr>
<td>aes-128-poly1305</td>
<td>957</td>
<td>3330</td>
<td>20623</td>
<td>28023</td>
<td>71473</td>
<td>393518</td>
</tr>
<tr>
<td>aes-256-poly1305</td>
<td>987</td>
<td>3442</td>
<td>21013</td>
<td>28627</td>
<td>73239</td>
<td>402759</td>
</tr>
<tr>
<td>cryptmt-v3-poly1305</td>
<td>21983</td>
<td>22678</td>
<td>27482</td>
<td>29679</td>
<td>33653</td>
<td>121893</td>
</tr>
<tr>
<td>dicen-p2-poly1305</td>
<td>2454</td>
<td>3446</td>
<td>9865</td>
<td>12581</td>
<td>28885</td>
<td>145143</td>
</tr>
<tr>
<td>dragon-poly1305</td>
<td>3229</td>
<td>4747</td>
<td>17031</td>
<td>22634</td>
<td>53741</td>
<td>277644</td>
</tr>
<tr>
<td>grain-128-poly1305</td>
<td>2280</td>
<td>4263</td>
<td>20739</td>
<td>28413</td>
<td>33653</td>
<td>121893</td>
</tr>
<tr>
<td>grain-v1-poly1305</td>
<td>2399</td>
<td>5149</td>
<td>27917</td>
<td>38740</td>
<td>96765</td>
<td>526804</td>
</tr>
<tr>
<td>hc-128-poly1305</td>
<td>33936</td>
<td>34635</td>
<td>37388</td>
<td>38666</td>
<td>45759</td>
<td>96712</td>
</tr>
<tr>
<td>hc-256-poly1305</td>
<td>90598</td>
<td>95181</td>
<td>97045</td>
<td>106890</td>
<td>179098</td>
<td></td>
</tr>
<tr>
<td>lex-v1-poly1305</td>
<td>1195</td>
<td>2269</td>
<td>7795</td>
<td>10270</td>
<td>24266</td>
<td>124971</td>
</tr>
<tr>
<td>nls-ae</td>
<td>2300</td>
<td>3740</td>
<td>9410</td>
<td>11043</td>
<td>23220</td>
<td>118608</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>1374</td>
<td>2539</td>
<td>6001</td>
<td>6811</td>
<td>14226</td>
<td>70396</td>
</tr>
<tr>
<td>phelix</td>
<td>743</td>
<td>1025</td>
<td>3326</td>
<td>4356</td>
<td>10109</td>
<td>51716</td>
</tr>
<tr>
<td>py6-poly1305</td>
<td>2866</td>
<td>3749</td>
<td>6518</td>
<td>7724</td>
<td>14674</td>
<td>64486</td>
</tr>
<tr>
<td>py-poly1305</td>
<td>7274</td>
<td>9181</td>
<td>11946</td>
<td>13187</td>
<td>20124</td>
<td>69881</td>
</tr>
<tr>
<td>pypy-poly1305</td>
<td>8561</td>
<td>10658</td>
<td>14231</td>
<td>15825</td>
<td>25065</td>
<td>91266</td>
</tr>
<tr>
<td>rabbit-poly1305</td>
<td>697</td>
<td>1310</td>
<td>4552</td>
<td>5905</td>
<td>14302</td>
<td>73521</td>
</tr>
<tr>
<td>rc4-poly1305</td>
<td>14085</td>
<td>14999</td>
<td>21367</td>
<td>24316</td>
<td>40761</td>
<td>158469</td>
</tr>
<tr>
<td>salsa20-8-poly1305</td>
<td>662</td>
<td>1216</td>
<td>4227</td>
<td>5511</td>
<td>12716</td>
<td>64672</td>
</tr>
<tr>
<td>salsa20-12-poly1305</td>
<td>781</td>
<td>1342</td>
<td>5071</td>
<td>6708</td>
<td>15547</td>
<td>79788</td>
</tr>
<tr>
<td>salsa20-12-poly1305</td>
<td>1007</td>
<td>1563</td>
<td>6693</td>
<td>9016</td>
<td>21078</td>
<td>109477</td>
</tr>
<tr>
<td>snow-2.0-poly1305</td>
<td>1512</td>
<td>2197</td>
<td>5416</td>
<td>6887</td>
<td>15032</td>
<td>74002</td>
</tr>
<tr>
<td>sosemank-poly1305</td>
<td>1405</td>
<td>2037</td>
<td>5992</td>
<td>7554</td>
<td>16703</td>
<td>87773</td>
</tr>
<tr>
<td>trivium-poly1305</td>
<td>1217</td>
<td>1755</td>
<td>4919</td>
<td>6258</td>
<td>14328</td>
<td>71471</td>
</tr>
</tbody>
</table>

The packet lengths I selected are 40 bytes, 576 bytes, and 1500 bytes from the official eSTREAM timings; 0 bytes; 8192 bytes; and 402 bytes, an approximation to the average Internet packet length.
The following table shows median cycle counts for authenticated encryption as a function of cipher and the number of active keys. All timings are from a 900MHz AMD Athlon (622) named thoth. All timings are for 576-byte packets.

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>16</th>
<th>64</th>
<th>256</th>
<th>1024</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5652</td>
<td>6437</td>
<td>20967</td>
<td>26962</td>
<td>29155</td>
<td>28836</td>
<td>abc-v3-poly1305</td>
</tr>
<tr>
<td>28066</td>
<td>28040</td>
<td>28126</td>
<td>28129</td>
<td>28409</td>
<td>28559</td>
<td>aes-128-poly1305</td>
</tr>
<tr>
<td>28783</td>
<td>28636</td>
<td>28662</td>
<td>28788</td>
<td>29031</td>
<td>29426</td>
<td>aes-256-poly1305</td>
</tr>
<tr>
<td>29703</td>
<td>29703</td>
<td>31326</td>
<td>35937</td>
<td>37842</td>
<td>38501</td>
<td>cryptmt-v3-poly1305</td>
</tr>
<tr>
<td>12538</td>
<td>12538</td>
<td>14951</td>
<td>14723</td>
<td>18080</td>
<td>18992</td>
<td>dicing-p2-poly1305</td>
</tr>
<tr>
<td>22619</td>
<td>22582</td>
<td>21335</td>
<td>22441</td>
<td>22413</td>
<td>22920</td>
<td>dragon-poly1305</td>
</tr>
<tr>
<td>28563</td>
<td>28519</td>
<td>28629</td>
<td>29458</td>
<td>31558</td>
<td>32564</td>
<td>grain-128-poly1305</td>
</tr>
<tr>
<td>38722</td>
<td>38766</td>
<td>38693</td>
<td>39587</td>
<td>41223</td>
<td>42613</td>
<td>grain-v1-poly1305</td>
</tr>
<tr>
<td>38619</td>
<td>38681</td>
<td>40787</td>
<td>40823</td>
<td>46322</td>
<td>48922</td>
<td>hc-128-poly1305</td>
</tr>
<tr>
<td>97370</td>
<td>97290</td>
<td>103092</td>
<td>108715</td>
<td>115247</td>
<td>119621</td>
<td>hc-256-poly1305</td>
</tr>
<tr>
<td>10338</td>
<td>10326</td>
<td>10316</td>
<td>10298</td>
<td>10569</td>
<td>10926</td>
<td>lex-v1-poly1305</td>
</tr>
<tr>
<td>11030</td>
<td>11086</td>
<td>11033</td>
<td>11149</td>
<td>11139</td>
<td>11429</td>
<td>nls-ae</td>
</tr>
<tr>
<td>6859</td>
<td>6855</td>
<td>6857</td>
<td>6944</td>
<td>6919</td>
<td>7120</td>
<td>nls-poly1305</td>
</tr>
<tr>
<td>4357</td>
<td>4357</td>
<td>4412</td>
<td>4386</td>
<td>4514</td>
<td>4415</td>
<td>phelix</td>
</tr>
<tr>
<td>7731</td>
<td>7727</td>
<td>7787</td>
<td>8095</td>
<td>8372</td>
<td>10195</td>
<td>py6-poly1305</td>
</tr>
<tr>
<td>13371</td>
<td>13380</td>
<td>14560</td>
<td>17433</td>
<td>20995</td>
<td>23702</td>
<td>py-poly1305</td>
</tr>
<tr>
<td>15804</td>
<td>17163</td>
<td>16907</td>
<td>18252</td>
<td>25976</td>
<td>26964</td>
<td>pypy-poly1305</td>
</tr>
<tr>
<td>5906</td>
<td>5921</td>
<td>5921</td>
<td>5985</td>
<td>6071</td>
<td>6217</td>
<td>rabbit-poly1305</td>
</tr>
<tr>
<td>24322</td>
<td>24280</td>
<td>24369</td>
<td>24792</td>
<td>26147</td>
<td>26456</td>
<td>rc4-poly1305</td>
</tr>
<tr>
<td>5521</td>
<td>5530</td>
<td>5530</td>
<td>5583</td>
<td>5599</td>
<td>5622</td>
<td>salsa20-8-poly1305</td>
</tr>
<tr>
<td>6729</td>
<td>6708</td>
<td>6709</td>
<td>6708</td>
<td>6942</td>
<td>6973</td>
<td>salsa20-12-poly1305</td>
</tr>
<tr>
<td>9026</td>
<td>9019</td>
<td>9020</td>
<td>9066</td>
<td>9303</td>
<td>9311</td>
<td>salsa20-poly1305</td>
</tr>
<tr>
<td>6871</td>
<td>6884</td>
<td>7008</td>
<td>7104</td>
<td>7004</td>
<td>7355</td>
<td>snow-2.0-poly1305</td>
</tr>
<tr>
<td>7602</td>
<td>8230</td>
<td>8765</td>
<td>8791</td>
<td>9392</td>
<td>9706</td>
<td>sosemanuk-poly1305</td>
</tr>
<tr>
<td>6237</td>
<td>6242</td>
<td>6276</td>
<td>6273</td>
<td>6359</td>
<td>6466</td>
<td>trivium-poly1305</td>
</tr>
</tbody>
</table>

The “bytes” column in the above table indicates the number of bytes in an expanded key. The penalty for handling 1024 active keys, compared to just 1, is usually around 2 cycles for each expanded-key byte, presumably reflecting this machine’s cache-load bandwidth. Grain shows a smaller penalty compared to its expanded-key size; presumably Grain does not access the entire expanded key for a 576-byte packet.
The following table shows median cycle counts for verified decryption as a function of cipher and machine. All timings are for 576-byte packets. All timings are for a single active key.

<table>
<thead>
<tr>
<th>cipher</th>
<th>thoth</th>
<th>atlas</th>
<th>wessel</th>
<th>katana</th>
<th>mace</th>
<th>gggg</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc-v3-poly1305</td>
<td>5493</td>
<td>5453</td>
<td>7275</td>
<td>5656</td>
<td>6058</td>
<td>11920</td>
</tr>
<tr>
<td>aes-128-poly1305</td>
<td>28114</td>
<td>22903</td>
<td>37371</td>
<td>14560</td>
<td>15388</td>
<td>25520</td>
</tr>
<tr>
<td>aes-256-poly1305</td>
<td>28594</td>
<td>20964</td>
<td>42039</td>
<td>17840</td>
<td>18526</td>
<td>29424</td>
</tr>
<tr>
<td>cryptmt-v3-poly1305</td>
<td>30090</td>
<td>29323</td>
<td>41915</td>
<td>13032</td>
<td>18157</td>
<td>39408</td>
</tr>
<tr>
<td>dicing-p2-poly1305</td>
<td>12546</td>
<td>11854</td>
<td>14582</td>
<td>9384</td>
<td>10745</td>
<td>19472</td>
</tr>
<tr>
<td>dragon-poly1305</td>
<td>22372</td>
<td>19234</td>
<td>29724</td>
<td>15768</td>
<td>18838</td>
<td>25776</td>
</tr>
<tr>
<td>grain-128-poly1305</td>
<td>28497</td>
<td>22713</td>
<td>48608</td>
<td>50912</td>
<td>grain-v1-poly1305</td>
<td></td>
</tr>
<tr>
<td>grain-v1-poly1305</td>
<td>38660</td>
<td>35248</td>
<td>52004</td>
<td>26000</td>
<td>28001</td>
<td>43216</td>
</tr>
<tr>
<td>hc-128-poly1305</td>
<td>38302</td>
<td>36708</td>
<td>52004</td>
<td>26000</td>
<td>28001</td>
<td>43216</td>
</tr>
<tr>
<td>hc-256-poly1305</td>
<td>97446</td>
<td>83339</td>
<td>119769</td>
<td>58760</td>
<td>78089</td>
<td>94480</td>
</tr>
<tr>
<td>lex-v1-poly1305</td>
<td>10348</td>
<td>9400</td>
<td>14095</td>
<td>9088</td>
<td>9929</td>
<td>16192</td>
</tr>
<tr>
<td>nls-aepoly1305</td>
<td>10866</td>
<td>10683</td>
<td>15724</td>
<td>7688</td>
<td>9346</td>
<td>10960</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6750</td>
<td>7453</td>
<td>8989</td>
<td>7936</td>
<td>7828</td>
<td>13584</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>4371</td>
<td>4773</td>
<td>12908</td>
<td>6152</td>
<td>5718</td>
<td>10032</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>7742</td>
<td>7688</td>
<td>11775</td>
<td>7920</td>
<td>9475</td>
<td>14800</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>13262</td>
<td>11051</td>
<td>21369</td>
<td>10608</td>
<td>13563</td>
<td>21472</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>15882</td>
<td>16548</td>
<td>22560</td>
<td>13792</td>
<td>16305</td>
<td>22080</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6045</td>
<td>6623</td>
<td>11051</td>
<td>7456</td>
<td>7081</td>
<td>19696</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>24418</td>
<td>18112</td>
<td>31312</td>
<td>11912</td>
<td>25761</td>
<td>24384</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>5570</td>
<td>6543</td>
<td>8001</td>
<td>6224</td>
<td>6304</td>
<td>16496</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6734</td>
<td>7667</td>
<td>9708</td>
<td>7064</td>
<td>7236</td>
<td>10880</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>9045</td>
<td>10633</td>
<td>12984</td>
<td>8904</td>
<td>9019</td>
<td>11808</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6892</td>
<td>7185</td>
<td>10773</td>
<td>7160</td>
<td>8098</td>
<td>14416</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>7379</td>
<td>8161</td>
<td>11537</td>
<td>8632</td>
<td>8357</td>
<td>15216</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6333</td>
<td>6536</td>
<td>8531</td>
<td>6672</td>
<td>7034</td>
<td>18656</td>
</tr>
<tr>
<td>nls-poly1305</td>
<td>6333</td>
<td>6536</td>
<td>8531</td>
<td>6672</td>
<td>7034</td>
<td>18656</td>
</tr>
</tbody>
</table>
The following table shows median cycle counts for rejection of a forged packet as a function of cipher and machine. All timings are for 576-byte packets. All timings are for a single active key.

<table>
<thead>
<tr>
<th>thoth</th>
<th>atlas</th>
<th>wessel</th>
<th>katana</th>
<th>mace</th>
<th>gggg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2854</td>
<td>3206</td>
<td>3873</td>
<td>4184</td>
<td>4212</td>
<td>9168</td>
</tr>
<tr>
<td>2858</td>
<td>3109</td>
<td>4107</td>
<td>4024</td>
<td>4210</td>
<td>9088</td>
</tr>
<tr>
<td>2919</td>
<td>3068</td>
<td>4241</td>
<td>4104</td>
<td>4052</td>
<td>9216</td>
</tr>
<tr>
<td>23882</td>
<td>22687</td>
<td>37791</td>
<td>11232</td>
<td>15721</td>
<td>34800</td>
</tr>
<tr>
<td>4377</td>
<td>4223</td>
<td>5340</td>
<td>4848</td>
<td>5573</td>
<td>10640</td>
</tr>
<tr>
<td>5137</td>
<td>5017</td>
<td>5564</td>
<td>5008</td>
<td>5258</td>
<td>10336</td>
</tr>
<tr>
<td>4196</td>
<td>4191</td>
<td>11712</td>
<td>11200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35466</td>
<td>34312</td>
<td>47536</td>
<td>24472</td>
<td>26002</td>
<td>39984</td>
</tr>
<tr>
<td>92778</td>
<td>79852</td>
<td>113603</td>
<td>56240</td>
<td>74321</td>
<td>90176</td>
</tr>
<tr>
<td>3156</td>
<td>3353</td>
<td>4498</td>
<td>4352</td>
<td>4465</td>
<td>9488</td>
</tr>
<tr>
<td>10876</td>
<td>10648</td>
<td>15650</td>
<td>7712</td>
<td>9347</td>
<td>10944</td>
</tr>
<tr>
<td>3287</td>
<td>3605</td>
<td>4444</td>
<td>4448</td>
<td>4766</td>
<td>9440</td>
</tr>
<tr>
<td>4368</td>
<td>4774</td>
<td>12914</td>
<td>6144</td>
<td>5717</td>
<td>10032</td>
</tr>
<tr>
<td>4778</td>
<td>4935</td>
<td>6539</td>
<td>5824</td>
<td>5694</td>
<td>11040</td>
</tr>
<tr>
<td>9194</td>
<td>7858</td>
<td>14172</td>
<td>8256</td>
<td>9295</td>
<td>16208</td>
</tr>
<tr>
<td>10466</td>
<td>11023</td>
<td>12858</td>
<td>9256</td>
<td>9873</td>
<td>15424</td>
</tr>
<tr>
<td>2650</td>
<td>3112</td>
<td>3987</td>
<td>4160</td>
<td>4024</td>
<td>9792</td>
</tr>
<tr>
<td>15984</td>
<td>10584</td>
<td>21629</td>
<td>11096</td>
<td>17582</td>
<td>17808</td>
</tr>
<tr>
<td>2567</td>
<td>3057</td>
<td>3749</td>
<td>4048</td>
<td>4018</td>
<td>9536</td>
</tr>
<tr>
<td>2683</td>
<td>3205</td>
<td>3901</td>
<td>4168</td>
<td>4216</td>
<td>8880</td>
</tr>
<tr>
<td>2917</td>
<td>3488</td>
<td>4225</td>
<td>4312</td>
<td>4288</td>
<td>8944</td>
</tr>
<tr>
<td>3374</td>
<td>3781</td>
<td>4667</td>
<td>4552</td>
<td>4792</td>
<td>9888</td>
</tr>
<tr>
<td>3242</td>
<td>3785</td>
<td>4769</td>
<td>4664</td>
<td>4695</td>
<td>9824</td>
</tr>
<tr>
<td>3128</td>
<td>3447</td>
<td>4019</td>
<td>4376</td>
<td>4418</td>
<td>10688</td>
</tr>
</tbody>
</table>

abc-v3-poly1305
aes-128-poly1305
aes-256-poly1305
cryptmt-v3-poly1305
dicing-p2-poly1305
dragon-poly1305
grain-128-poly1305
grain-v1-poly1305
hc-128-poly1305
hc-256-poly1305
lex-v1-poly1305
nls-ae
nls-poly1305
phelix
py6-poly1305
py-poly1305
pypy-poly1305
rabbit-poly1305
rc4-poly1305
salsa20-8-poly1305
salsa20-12-poly1305
salsa20-poly1305
sosemanuk-poly1305
trivium-poly1305
Appendix: Tunings

A cipher in the official eSTREAM benchmarking toolkit can have several tunings: several implementations in separate subdirectories of the cipher directory, and several “variants” of each implementation.

The new toolkit automatically tries encrypting several 1536-byte packets under each tuning. It then selects the tuning producing the smallest median cycle count, and uses that tuning for subsequent timings. The following table lists the selected tunings.

<table>
<thead>
<tr>
<th>thoth</th>
<th>atlas</th>
<th>wessel</th>
<th>katana</th>
<th>mace</th>
<th>gggg</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>abc-v3-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>aes-128-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>aes-256-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>cryptmt-v3-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>dicing-p2-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>dragon-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>grain-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>hc-128-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>hc-256-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>lex-poly1305</td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>nls-aes</td>
<td></td>
</tr>
<tr>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>nls-poly1305</td>
<td></td>
</tr>
<tr>
<td>i386</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>phelix</td>
</tr>
<tr>
<td>opt/3</td>
<td>opt/4</td>
<td>opt/3</td>
<td>opt/4</td>
<td>opt/3</td>
<td>rabbit-poly1305</td>
</tr>
<tr>
<td>x86-athlon</td>
<td>x86-3</td>
<td>sparc</td>
<td>ami64-3</td>
<td>ami64-3</td>
<td>ref</td>
</tr>
<tr>
<td>x86-athlon</td>
<td>x86-pm</td>
<td>sparc</td>
<td>ami64-3</td>
<td>ami64-3</td>
<td>ppc-altivec</td>
</tr>
<tr>
<td>x86-athlon</td>
<td>x86-pm</td>
<td>sparc</td>
<td>ami64-3</td>
<td>ami64-3</td>
<td>ppc-altivec</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>rsal20-8-poly1305</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>rsal20-12-poly1305</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>rsal20-poly1305</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>sosemunuk-poly1305</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>trivium-poly1305</td>
</tr>
</tbody>
</table>

I’m surprised by some of the selections of “ref” in this table; I’m investigating.