DAENCE: SALSA20 and CHACHA in Deterministic Authenticated Encryption with no noNCEnse

Taylor 'Riastradh' Campbell ⟨campbell+daence@mumble.net⟩

January 22, 2020

Abstract

We present DAENCE, a deterministic authenticated cipher based on a pseudorandom function family and a universal hash family, similar to SIV [17]. We recommend instances with SALSA20 [7] or CHACHA [8], and POLY1305 [6], for high performance, high security, and easy deployment.

1 Introduction

The nonce-based authenticated cipher crypto_secretbox_xsalsa20poly1305 in NaCl [9], and the variant CHACHA/POLY1305 defined by the IETF [15] for TLS [14], are widely available in fast software implementations resistant to timing side channels. The nonce-based authenticated cipher AES-GCM [12] is popular, though only with hardware support is it fast and resistant to timing side channels.

These three nonce-based ciphers fail catastrophically in the face of nonce reuse. They are best suited to protocols that are designed to support sequential message numbers, such as the record number in TLS. Some applications are unable to keep the state needed to maintain a sequential message number, and although they could use an extended nonce like XSALSA20 [10] chosen randomly, some environments may not have a reliable entropy source. The *nonce-misuse-resistant* authenticated cipher AES-GCM-SIV [13] was developed to address these use cases, but it carries with it the performance and side channel costs of AES-GCM—and amplifies the performance cost by deriving fresh keys for each distinct nonce, yet has very narrow security margins.

We propose a deterministic authenticated cipher DAENCE built out of the SALSA20 or CHACHA pseudorandom function family and the POLY1305 universal hash family. The design is based on the SIV construction of Rogaway and Shrimpton [17], with a variable-input-PRF made by composing a universal hash family with a fixed-input PRF [3, §1.5][5, §9, Theorem 9.2]. DAENCE is easily implemented in terms of the primitives available in NaCl and libsodium.

2 Security contract

DAENCE is a deterministic authenticated cipher. This means it consists of two functions:

$c =$ DAENCE-ENCRYPT(k, a, m) takes a key k (96-byte in SALSA20-DAENCE, 64-byte in CHACHA-DAENCE), a string a of fewer than 2^{32} bytes of associated data, and a message m of fewer than 2^{32} bytes.

DAENCE-ENCRYPT returns an authenticated ciphertext c which is 24 bytes longer than m.

$m' =$ DAENCE-DECRYPT(k, a', c') takes a key k, a string a' of fewer than 2^{32} bytes of associated data, and an alleged authenticated ciphertext c' of fewer than $24 + 2^{32}$ bytes.

- If $c' =$ DAENCE-ENCRYPT(k, a', m), DAENCE-DECRYPT returns m.
- Otherwise, DAENCE-DECRYPT reports a forgery with high probability. In the sequel we denote this by the symbol '$\perp$'; in practice, a `crypto_dae_salsa20daence_open` function in the style of NaCl may return an error code or throw an exception.

Responsibilities of the user.

1. You must choose a secret key k uniformly at random and independently of everything else in your application. (You may safely derive the 96-byte key k from a 32-byte key k' by a key derivation function—*e.g.*, $k = \text{HKDF-SHA256}_{k'}(\textit{salt}, \text{'foo'}, 96)$.)
2. If DAENCE-DECRYPT reports a forgery, you must decline to act on the alleged message content except by immediately dropping it on the floor.
3. You must process no more than a total of 2^{80} bytes of data, including associated data, with any single key.

Security guarantee. Under any key k independently:

1. If you repeat an associated data string, then an adversary has no hope of distinguishing the ciphertexts of *distinct* messages from uniform random byte strings of the same length, but can tell when messages are repeated.
2. If you do not repeat an associated data string, then an adversary has no hope of distinguishing the ciphertexts of your messages from uniform random byte strings of the same length.
3. The adversary's probability of succeeding at forgery—even after flooding your system with up to 2^{80} bytes of forgery attempts—is less than 2^{-32}.

3 Definition

Poly1305^2. For 16-byte strings $k_1, \ldots, k_4$, and a byte string m, define

$$\text{Poly1305}^2_{k_1,k_2}(m) := \text{Poly1305}_{k_1}(m) \parallel \text{Poly1305}_{k_2}(m).$$

For byte strings a and m, define

$$\text{Poly1305}^2_{k_1,k_2,k_3,k_4}(a, m) := \text{Poly1305}^2_{k_3,k_4}(h_a \parallel h_m),$$

where $h_a = \text{Poly1305}^2_{k_1,k_2}(a)$ and $h_m = \text{Poly1305}^2_{k_1,k_2}(m)$.

HXSalsa20. For 32-byte key k_0 and 16-byte inputs i and j, define

$$\text{HXSalsa20}_{k_0}(i \parallel j) := \text{HSalsa20}_{\text{HSalsa20}_{k_0}(i)}(j).$$

Note that HXSalsa20 can be defined in terms of XSalsa20 by choosing the appropriate words of the output and subtracting j, just as HSalsa20 can be defined in terms of Salsa20, and has comparable security [10].

Salsa20-Daence. To **encrypt**, given a 96-byte key k, associated data a of fewer than 2^{32} bytes, and a message m of fewer than 2^{32} bytes, compute:

1. $k_0 \parallel k_1 \parallel k_2 \parallel k_3 \parallel k_4 := k$ (32-byte k_0; 16-byte $k_1, \ldots, k_4$)
2. $t \parallel \_ := \text{HXSalsa20}_{k_0}(\text{Poly1305}^2_{k_1,k_2,k_3,k_4}(a, m))$ (24-byte t)
3. $c := m \oplus \big(\text{XSalsa20}_{k_0}(t \parallel 0) \parallel \text{XSalsa20}_{k_0}(t \parallel 1) \parallel \cdots\big)$

The authenticated ciphertext is $t \parallel c$.

To **verify and decrypt**, given a 96-byte key k, associated data a of fewer than 2^{32} bytes, and an alleged authenticated ciphertext $t' \parallel c'$ of 24 to $24 + 2^{32}$ bytes, compute:

1. $k_0 \parallel k_1 \parallel k_2 \parallel k_3 \parallel k_4 := k$ (32-byte k_0; 16-byte $k_1, \ldots, k_4$)
2. $m' := c' \oplus \big(\text{XSalsa20}_{k_0}(t \parallel 0) \parallel \text{XSalsa20}_{k_0}(t \parallel 1) \parallel \cdots\big)$
3. $t' \parallel \_ := \text{HXSalsa20}_{k_0}(\text{Poly1305}^2_{k_1,k_2,k_3,k_4}(a, m'))$ (24-byte t')

If $t' \stackrel{?}{=} t$, return m'; otherwise erase m' and report a forgery.

ChaCha-Daence. Rather than hash associated data separately, implementations of ChaCha/Poly1305 as used in TLS and implemented in libsodium crypto_aead support passing multiple strings into Poly1305, zero-padded as $\underline{a}$ and $\underline{m}$ to multiples of 16 bytes with 8-byte little-endian lengths:

1. $k_0 \parallel k_1 \parallel k_2 := k$ (32-byte k_0, 16-byte k_1, 16-byte k_2)
2. $t \parallel \_ := \text{HXChaCha}_{k_0}(\text{Poly1305}^2_{k_1,k_2}(\underline{a} \parallel \underline{m} \parallel |a|_8 \parallel |m|_8))$ (24-byte t)
3. $c := m \oplus \big(\text{XChaCha20}_{k_0}(t \parallel 0) \parallel \text{XChaCha20}_{k_0}(t \parallel 1) \parallel \cdots\big)$

The corresponding decryption operation is derived similarly.

4 Security notions

Let A be a random decision algorithm with access to an oracle $\mathcal{O}$. Write $\Pr[A(\mathcal{O})]$ for the probability A accepts after making various queries to $\mathcal{O}$:

Encryption queries for a *single* user Given associated data a and message m, return an authenticated ciphertext c.

Decryption queries for a *single* user Given associated data a and alleged authenticated ciphertext c, return a message m, or $\perp$ if the alleged authenticated ciphertext is deemed a forgery.

Encryption queries for *multiple* users Given a user number u, associated data a, and message m, return an authenticated ciphertext c.

Decryption queries for *multiple* users Given a user number u, associated data a, and alleged authenticated ciphertext c, return a message m, or $\perp$ if the alleged ciphertext is deemed a forgery.

Function queries for *multiple* users Given a user number u and an input x, return an output y.

The adversary A is assumed not to repeat queries, nor to submit the answers from encryption queries as decryption queries. We review standard notions of security (*e.g.*, [17, Definition 1][2, §3]), loosely summarized as how well A can tell users of a real cryptosystem—Daence, XSalsa20, *etc.*—from pranksters who just roll dice to answer every query.

Definition 1. *The **pathological deterministic authenticated cipher** U_1 returns an independent uniform random authenticated ciphertext of the appropriate length for each encryption query, and returns $\perp$ for every decryption query. The notion extends naturally to the multi-user setting; call it U.*

Definition 2. *For a deterministic authenticated cipher E_k with random key k, the **multi-user deterministic authenticated encryption advantage** of A against E is the statistical distance from E_k to U measured by A, where by abuse of language k is understood to mean a collection of keys chosen independently by many users and E_k is understood to mean a collection of instances of E for many users keyed by their respective keys:*

$$\mathrm{Adv}_E^{\text{mu-DAE}}(A) := |\Pr[A(E_k)] - \Pr[A(U)]|.$$

Here A may submit encryption and decryption queries for multiple *users.*

The single-user $\mathrm{Adv}_E^{\mathrm{DAE}}(A)$ is defined similarly.

Definition 3. *For a function family ϕ_k with random key k, the **multi-user pseudorandom function advantage** of A against ϕ is the statistical distance from ϕ_k to f measured by A, where f is a a uniform random function of the same domain and codomain (where 'ϕ_k' and 'f' again mean many independent instances):*

$$\mathrm{Adv}_\phi^{\text{mu-PRF}}(A) := |\Pr[A(\phi_k)] - \Pr[A(f)]|.$$

Here A may submit function queries for multiple *users.*

5 Analysis

Theorem 1 (SALSA20-DAENCE). *Let A be a random decision algorithm with encryption and decryption oracles for a set of deterministic authenticated ciphers. Suppose A submits $E(u)$ encryption queries and $D(u)$ decryption queries to the u^{th} user of up to $\ell_a(u)$ bytes of associated data and $\ell_m(u)$-byte messages. Then there is an algorithm A' making $\sum_u \bigl(1+\lceil \ell_m(u)/64\rceil\bigr)\bigl(E(u)+D(u)\bigr)$ oracle queries and having the cost of A plus the cost of evaluating* POLY1305^2 *and* $\oplus$ *on $\sum_u E(u)+D(u)$ different $(\ell_a(u), \ell_m(u))$-byte inputs, such that*

$$\mathrm{Adv}^{\text{mu-DAE}}_{\textsc{Daence}}(A) \le \mathrm{Adv}^{\text{mu-PRF}}_{\textsc{XSalsa20}}(A') + \sum_u \frac{\bigl(E(u)+D(u)\bigr)^2 + D(u) + \binom{E(u)}{2}}{2^{192}} + \varepsilon(\ell_a(u), \ell_m(u)) \cdot \Bigl(D(u)E(u) + \binom{E(u)}{2}\Bigr),$$

where

$$\varepsilon(\ell_a, \ell_m) := \frac{\max\{\lceil \ell_a/16\rceil^2, \lceil \ell_m/16\rceil^2\} + 16}{2^{206}}.$$

Theorem 2 (CHACHA-DAENCE). *Let A be a random decision algorithm with encryption and decryption oracles for a set of deterministic authenticated ciphers. Suppose A submits $E(u)$ encryption queries and $D(u)$ decryption queries to the u^{th} user of up to $\ell_a(u)$ bytes of associated data and $\ell_m(u)$-byte messages. Then there is an algorithm A' making $\sum_u \bigl(1+\lceil \ell_m(u)/64\rceil\bigr)\bigl(E(u)+D(u)\bigr)$ oracle queries and having the cost of A plus the cost of evaluating* POLY1305^2 *and* $\oplus$ *on $\sum_u E(u)+D(u)$ different $(\ell_a(u), \ell_m(u))$-byte inputs, such that*

$$\mathrm{Adv}^{\text{mu-DAE}}_{\textsc{Daence}}(A) \le \mathrm{Adv}^{\text{mu-PRF}}_{\textsc{XChaCha}}(A') + \sum_u \frac{\bigl(E(u)+D(u)\bigr)^2 + D(u) + \binom{E(u)}{2}}{2^{192}} + \frac{\lceil (\ell_a(u)+\ell_m(u)+16)/16\rceil^2}{2^{206}} \cdot \Bigl(D(u)E(u) + \binom{E(u)}{2}\Bigr).$$

Outline of proof.

1. Set a bound on the collision probability of POLY1305^2.
2. Set a bound on DAE advantage against an idealized version of DAENCE.
3. Extend the bound to the multi-user setting.
4. Instantiate the idealization with the actual PRF.

5.1 Collisions under double-hashing with associated data

Lemma 1 (Double-hashing)**.** *Let* k_1, k_2 *be independent* Poly1305 *keys. For any distinct strings* $m \neq m'$ *of at most* ℓ *bytes,*

$$\Pr[\textsc{Poly1305}^2_{k_1,k_2}(m) = \textsc{Poly1305}^2_{k_1,k_2}(m')] \leq \varepsilon(\ell) := \frac{\lceil \ell/16 \rceil^2}{2^{206}}.$$

Proof. By [6, Theorem 3.3],

$$\Pr[\textsc{Poly1305}_{k_1}(m) = \textsc{Poly1305}_{k_1}(m')] \leq \frac{8\lceil \ell/16 \rceil}{2^{106}},$$

and likewise for k_2. Since k_1 and k_2 are independent,

$$\begin{aligned}
&\Pr[\textsc{Poly1305}^2_{k_1,k_2}(m) = \textsc{Poly1305}^2_{k_1,k_2}(m')] \\
&\quad = \Pr[\textsc{Poly1305}_{k_1}(m) = \textsc{Poly1305}_{k_1}(m'), \\
&\qquad\quad \textsc{Poly1305}_{k_2}(m) = \textsc{Poly1305}_{k_2}(m')] \\
&\quad = \Pr[\textsc{Poly1305}_{k_1}(m) = \textsc{Poly1305}_{k_1}(m')] \\
&\qquad \cdot \Pr[\textsc{Poly1305}_{k_2}(m) = \textsc{Poly1305}_{k_2}(m')] \\
&\quad \leq \left(\frac{8\lceil \ell/16 \rceil}{2^{106}}\right)^2 = \frac{64\lceil \ell/16 \rceil^2}{2^{212}} = \frac{\lceil \ell/16 \rceil^2}{2^{206}}.
\end{aligned}$$

□

Lemma 2 (Hashing tuples)**.** *Let* $k_1, \ldots, k_4$ *be independent* Poly1305 *keys. For strings* a, a' *up to* ℓ_a *bytes and* m, m' *up to* ℓ_m *bytes, if* $(a, m) \neq (a', m')$ *then*

$$\begin{aligned}
&\Pr[\textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a, m) = \textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a', m')] \\
&\qquad \leq \varepsilon(\ell_a, \ell_m) := \frac{\max\{\lceil \ell_a/16 \rceil^2, \lceil \ell_m/16 \rceil^2\} + 16}{2^{206}}.
\end{aligned}$$

Proof. Write $H := \textsc{Poly1305}^2_{k_1,k_2}$ and $H^* := \textsc{Poly1305}^2_{k_3,k_4}$. Let $h_x := H(x)$. We must have either $a \neq a'$ or $m \neq m'$, or both. If $a \neq a'$, then by Lemma 1, $\Pr[(h_a, h_m) = (h_{a'}, h_{m'})] \leq \Pr[h_a = h_{a'}] \leq \varepsilon(\ell_a)$. Similarly, if $m \neq m'$, the probability is bounded by $\varepsilon(\ell_m)$, so in either case,

$$\Pr[(h_a, h_m) = (h_{a'}, h_{m'})] \leq \max\{\varepsilon(\ell_a), \varepsilon(\ell_m)\}.$$

Finally, since H^* is independent of H and thus of the h_x:

$$\begin{aligned}
&\Pr[H^*(h_a \parallel h_m) = H^*(h_{a'} \parallel h_{m'})] \\
&\quad \leq \Pr[(h_a, h_m) = (h_{a'}, h_{m'})] \\
&\qquad + \Pr[H^*(h_a \parallel h_m) = H^*(h_{a'} \parallel h_{m'}) \mid (h_a, h_m) \neq (h_{a'}, h_{m'})] \\
&\quad \leq \max\{\varepsilon(\ell_a), \varepsilon(\ell_m)\} + \varepsilon(64) = \frac{\max\{\lceil \ell_a/16 \rceil^2, \lceil \ell_m/16 \rceil^2\} + 4^2}{2^{206}}.
\end{aligned}$$

□

5.2 Idealizing the cipher

For a function ϕ yielding bit strings, denote by ϕ_{192} the first 192 bits of ϕ; denote by $\hat{\phi}$ the transformation so that if $\phi = \text{XSALSA20}$ then $\hat{\phi} = \text{HXSALSA20}$; and denote by $\phi_*(t)$ the concatenation $\phi(t \parallel 0) \parallel \phi(t \parallel 1) \parallel \cdots$, with input counter encoded in little-endian, and with output understood to be as long as is needed. Let $f, g\colon \{0,1\}^{256} \to \{0,1\}^{512}$ be uniform random. Note that $\hat{f}$, $\hat{f}_{192}$, f_*, and g_* are uniformly distributed.

Define **Deuce** to be as Daence, but with f substituted for XSALSA20_{k_0}:

1. $h := \text{POLY1305}^2_{k_1,k_2,k_3,k_4}(a, m)$ (32-byte h)
2. $t := \hat{f}_{192}(h)$ (24-byte t)
3. $c := m \oplus f_*(t)$

Define **Dice** to be as Deuce, but with g substituted for f in the pad:

1. $h := \text{POLY1305}^2_{k_1,k_2,k_3,k_4}(a, m)$ (32-byte h)
2. $t := \hat{f}_{192}(h)$ (24-byte t)
3. $c := m \oplus g_*(t)$

Finally, define **Dunce** to be as Dice, but with the decryption oracle modified to return $\perp$ for every input.

Lemma 3 (Single-user idealized security)**.** *Let A be a random decision algorithm with encryption and decryption oracles for a deterministic authenticated cipher. Suppose A makes E encryption queries and D decryption queries of up to ℓ_a bytes of associated data and ℓ_m-byte messages. Then*

$$\text{Adv}^{\text{DAE}}_{\text{DEUCE}}(A) \leq \frac{(E+D)^2 + D + \binom{E}{2}}{2^{192}} + \varepsilon(\ell_a, \ell_m) \cdot \left(DE + \binom{E}{2}\right).$$

Distinguishing Deuce from Dice: independent tags and pads. Dice is simply Deuce with two independent functions for deriving a tag t from a hash h for deriving a pad from a tag. Only in the event C_{th} of a collision between one of the values of h and one of the values of $t \parallel j$ do Dice and Deuce have different distributions, since if the h do not overlap with the $t \| j$, then the $f(t \| j)$ and $g(t \parallel j)$ are both uniform random strings independent of the $\hat{f}_{192}(h)$. And since $f(t \| j)$ does not figure into Dice, a collision between a value of h and a value of $t \parallel j$ value cannot affect the outcome, so $\Pr[A(\text{DICE}) \mid \neg C_{th}] = \Pr[A(\text{DICE})]$. Thus,

$$\begin{aligned}
\Pr[A(\text{DEUCE})] &= \Pr[A(\text{DEUCE}) \mid C_{th}] \Pr[C_{th}] \\
&\quad + \Pr[A(\text{DEUCE}) \mid \neg C_{th}] \Pr[\neg C_{th}] \\
&\leq \Pr[C_{th}] + \Pr[A(\text{DEUCE}) \mid \neg C_{th}] \\
&= \Pr[C_{th}] + \Pr[A(\text{DICE}) \mid \neg C_{th}] \\
&= \Pr[C_{th}] + \Pr[A(\text{DICE})].
\end{aligned}$$

How often do collisions occur? It suffices to consider the event $C_{th_{192}}$ of a collision between a t and just the upper 192 bits of an h; call it h_{192}. There are at most $E + D$ distinct values of h and thus of h_{192}. The values of t are independent uniform 192-bit strings, and there are up to $E + D$ of them, so

$$\Pr[C_{th}] \leq \Pr[C_{th_{192}}] \leq \sum_{i=1}^{E+D} \frac{E+D}{2^{192}} = \frac{(E+D)^2}{2^{192}}.$$

Finally, since A was arbitrary and could be replaced by $\neg A$, we have

$$|\Pr[A(\textsc{Deuce})] - \Pr[A(\textsc{Dice})]| \leq \Pr[C_{th}] \leq \frac{(E+D)^2}{2^{192}}.$$

Distinguishing Dice from Dunce: bounding forgery probability. The decryption oracle of Dice returns anything other than $\perp$ only in the event F that A forges a ciphertext not returned by the encryption oracle. Conditional on $\neg F$, Dice and Dunce otherwise have the same distribution—the encryption oracle is the same and the decryption oracle always returns $\perp$ in both cases—so

$$\begin{aligned}\Pr[A(\textsc{Dice})] &= \Pr[A(\textsc{Dice}) \mid F] \Pr[F] + \Pr[A(\textsc{Dice}) \mid \neg F] \Pr[\neg F] \\ &\leq \Pr[F] + \Pr[A(\textsc{Dice}) \mid \neg F] \\ &= \Pr[F] + \Pr[A(\textsc{Dunce})],\end{aligned}$$

and thus

$$|\Pr[A(\textsc{Dice})] - \Pr[A(\textsc{Dunce})]| \leq \Pr[F].$$

Assume A halts after the first forgery attempt—an adversary that makes $D > 1$ forgery attempts can be broken into one that halts after the first forgery attempt, and another one that simulates forgery failure for the first attempt before making $D - 1$ more attempts. If the first succeeded with probability at most p, then by induction the second succeeds with probability at most $(D-1)p$, so that the original succeeds with probability at most $p + (D - 1)p = Dp$.

Suppose A submits distinct encryption queries $(a_1, m_1), \ldots, (a_E, m_E)$ giving answers $t_1 \| c_1, \ldots, t_E \| c_E$ via hashes $h_i = \textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a_i, m_i)$, so that $t_i = f(h_i)$ and $c_i = m_i \oplus g_*(t_i)$. Suppose A attempts a forgery (a', t', c') with $t' \| c' \neq t_i \| c_i$ for all i. Let $m' = c' \oplus g_*(t')$ and $h' = \textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a', m')$. The forgery succeeds if $t' = f(h')$. Note that $(a', m') \neq (a_i, m_i)$, for if $(a', m') = (a_i, m_i)$, then we would have $t' \| c' = t_i \| c_i$, since—for fixed f, g, and $k_1, \ldots, k_4$—$t' \| c'$ is a deterministic function of (a', m').

The string $f(h')$ is a uniform random 192-bit string independent of the t_i and c_i, except in the event $C_{h'}$ that h' coincides with one of the h_i:

$$\begin{aligned}\Pr[C_{h'}] &= \Pr[\exists i\colon h' = h_i] \leq \sum_i \Pr[h' = h_i] \\ &= \sum_i \Pr[\textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a', m') = \textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a_i, m_i)] \\ &\leq \sum_i \varepsilon(\ell_a, \ell_m) = E \cdot \varepsilon(\ell_a, \ell_m).\end{aligned}$$

Hence

$$\begin{aligned}
|\Pr[A(\textsc{Dice})] - \Pr[A(\textsc{Dunce})]| &\le \Pr[F] \\
&\le D \cdot \Pr\bigl[t' = f(h') \mid t_i = f(h_i), c_i = m_i \oplus g_*(t_i)\bigr] \\
&\le D \cdot \bigl(\Pr[t' = f(h') \mid \neg C_{h'}] + \Pr[C_{h'}]\bigr) \\
&\le \frac{D}{2^{192}} + DE \cdot \varepsilon(\ell_a, \ell_m).
\end{aligned}$$

Distinguishing Dunce from uniform random. Let U_1 be a pathological 'DAE' whose encryption oracle returns an independent uniform random tag and ciphertext for each distinct input, and whose decryption oracle returns $\perp$ for every input. Suppose A submits distinct encryption queries $(a_1, m_1), \dots, (a_E, m_E)$ giving answers $t_1 \parallel c_1, \dots, t_E \parallel c_E$. If the oracle is Dunce, then all the strings $t_i = f(h_i)$ and $c_i = m_i \oplus g_*(t_i)$ are independent uniform random—and thus the oracle is indistinguishable from U_1—except in the event C_h of a collision $h_i = h_j$ for some $i \ne j$, or in the event C_t of a collision $t_i = t_j$ for some $i \ne j$:

$$\begin{aligned}
\Pr[C_h] = \Pr[\exists i < j \colon h_i = h_j] &\le \sum_{i<j} \Pr[h_i = h_j] \\
&= \sum_{i<j} \Pr[\textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a_i, m_i) = \textsc{Poly1305}^2_{k_1,k_2,k_3,k_4}(a_j, m_j)] \\
&\le \sum_{i<j} \varepsilon(\ell_a, \ell_m) = \binom{E}{2} \varepsilon(\ell_a, \ell_m).
\end{aligned}$$

If the h_i are all distinct, there is a collision among the t_i only with probability

$$\begin{aligned}
\Pr[C_t \mid \neg C_h] = \Pr[\exists i < j \colon t_i = t_j \mid \neg C_h] &\le \sum_{i<j} \Pr[t_i = t_j \mid \neg C_h] \\
&= \sum_{i<j} \Pr[f(h_i) = f(h_j) \mid h_i \ne h_j] \le \sum_{i<j} \frac{1}{2^{192}} = \binom{E}{2} \frac{1}{2^{192}}.
\end{aligned}$$

Thus,

$$\begin{aligned}
\Pr[A(\textsc{Dunce})] &\le \Pr\bigl[A(\textsc{Dunce}) \mid \neg(C_h \text{ or } C_t)\bigr] + \Pr[C_h \text{ or } C_t] \\
&= \Pr[A(U_1)] + \Pr[C_h \text{ or } C_t] \\
&\le \Pr[A(U_1)] + \Pr[C_h] + \Pr[C_t \mid \neg C_h] \\
&\le \Pr[A(U_1)] + \binom{E}{2} \varepsilon(\ell_a, \ell_m) + \binom{E}{2} \frac{1}{2^{192}},
\end{aligned}$$

so that

$$|\Pr[A(\textsc{Dunce})] - \Pr[A(U_1)]| \le \binom{E}{2} \left(\varepsilon(\ell_a, \ell_m) + \frac{1}{2^{192}} \right).$$

Summing it up.

$$
\begin{aligned}
&|\Pr[A(\textsc{Deuce})] - \Pr[A(U_1)]| \\
&\quad \le |\Pr[A(\textsc{Deuce})] - \Pr[A(\textsc{Dice})]| + |\Pr[A(\textsc{Dice})] - \Pr[A(\textsc{Dunce})]| \\
&\qquad + |\Pr[A(\textsc{Dunce})] - \Pr[A(U_1)]| \\
&\quad \le \frac{(E+D)^2}{2^{192}} + \frac{D}{2^{192}} + DE \cdot \varepsilon(\ell_a, \ell_m) + \binom{E}{2}\left(\varepsilon(\ell_a, \ell_m) + \frac{1}{2^{192}}\right) \\
&\quad = \frac{(E+D)^2 + D + \binom{E}{2}}{2^{192}} + \varepsilon(\ell_a, \ell_m) \cdot \left(DE + \binom{E}{2}\right). \qquad \square
\end{aligned}
$$

5.3 Multi-user security

Lemma 4 (Multi-user idealized security)**.** *Let A be a random decision algorithm with encryption and decryption oracles for a* set *of deterministic authenticated ciphers. Suppose A makes $E(u)$ encryption queries and $D(u)$ decryption queries to the u^{th} user of up to $\ell_a(u)$ bytes of associated data and $\ell_m(u)$-byte messages. Then*

$$
\mathrm{Adv}^{\text{mu-DAE}}_{\textsc{Deuce}}(A) \le \sum_u \frac{\big(E(u) + D(u)\big)^2 + D(u) + \binom{E(u)}{2}}{2^{192}} + \varepsilon(\ell_a(u), \ell_m(u)) \cdot \left(D(u)E(u) + \binom{E(u)}{2}\right).
$$

Proof. In the foregoing analysis of the idealized cipher in the single-user setting (Lemma 3), the probabilities of the critical events—C_{th}, F, $C_{h'}$, C_h, and C_t—can be straightforwardly seen to sum over the users independently. For example, the event C_{th} of a collision in *any one* of the users each with their own independent function f and keys $k_1, \dots, k_4$ is the sum over all users of C_{th} for each user separately. Consequently, the multi-user DAE advantage against Deuce—that is, the statistical distance under A from Deuce to a pathological 'DAE' collection U—is at most the sum of the single-user DAE advantages. □

5.4 Instantiating the idealized cipher

Lemma 5 (Multi-user instantiation)**.** *Let A be a random decision algorithm with encryption and decryption oracles for a* set *of deterministic authenticated ciphers. Suppose A makes $E(u)$ encryption queries and $D(u)$ decryption queries to the u^{th} user of up to $\ell_a(u)$ bytes of associated data and $\ell_m(u)$-byte messages. Then there is an algorithm A' making $\sum_u \big(1 + \lceil \ell_m(u)/64 \rceil\big)\big(E(u) + D(u)\big)$ oracle queries and having the cost of A plus the cost of evaluating* Poly1305^2 *and $\oplus$ on $\sum_u E(u) + D(u)$ different $(\ell_a(u), \ell_m(u))$-byte inputs, such that*

$$
|\Pr[A(\textsc{Daence})] - \Pr[A(\textsc{Deuce})]| = \mathrm{Adv}^{\text{mu-PRF}}_{\textsc{XSalsa20}}(A').
$$

Proof. Deuce is simply Daence with a uniform random function f substituted for $\textsc{XSalsa20}_{k_0}$, so if A can distinguish a collection of Daence users from a collection of DEUCE users then it can be used in an algorithm A' to distinguish a collection of XSALSA20_{k_0} users from a collection of f users (recall $f\colon \{0,1\}^{256} \to \{0,1\}^{512}$ is a uniform random function).

If $\mathcal{O}(u,x)$ is an oracle for a collection of function instances indexed by u, define $A'(\mathcal{O})$ to run A with oracles for the following DAE under independent uniform random keys $k_1, \dots, k_4$ for the u^{th} user, decryption being defined the obvious way:

1. $h := \text{POLY1305}^2_{k_1,k_2,k_3,k_4}(a,m)$ (32-byte h)
2. $t := \hat{\mathcal{O}}_{192}(u,h)$ (24-byte t)
3. $c := m \oplus \mathcal{O}_*(u,t)$

Then $A'(\text{XSALSA20}_{k_0}) = A(\text{DAENCE})$ and $A'(f) = A(\text{DEUCE})$ (with 'k_0' and 'f' understood to mean a collection of independent keys/functions), so

$$\begin{aligned}|\Pr[A(\text{DAENCE})] - \Pr[A(\text{DEUCE})]| &= |\Pr[A'(\text{XSALSA20}_{k_0})] - \Pr[A'(f)]| \\ &= \text{Adv}^{\text{mu-PRF}}_{\text{XSALSA20}}(A'). \qquad \square\end{aligned}$$

5.5 Tying the room together

Proof of Theorem 1. By stringing all the inequlities together, we complete the proof:

$$\begin{aligned}\text{Adv}^{\text{mu-DAE}}_{\text{DAENCE}}(A) &= |\Pr[A(\text{DAENCE})] - \Pr[A(U)]| \\ &\le |\Pr[A(\text{DAENCE})] - \Pr[A(\text{DEUCE})]| \\ &\quad + |\Pr[A(\text{DEUCE})] - \Pr[A(U)]| \\ &\le \text{Adv}^{\text{mu-PRF}}_{\text{XSALSA20}}(A') \\ &\quad + \sum_u \frac{\big(E(u)+D(u)\big)^2 + D(u) + \binom{E(u)}{2}}{2^{192}} \\ &\qquad + \varepsilon(\ell_a(u), \ell_m(u)) \cdot \Big(D(u)E(u) + \binom{E(u)}{2}\Big). \qquad \square\end{aligned}$$

Proof of Theorem 2. The CHACHA version of DAENCE uses

$$\text{POLY1305}^2_{k_1,k_2}(a \,\|\, m \,\|\, |a|_8 \,\|\, |m|_8) \qquad \text{instead of} \qquad \text{POLY1305}^2_{k_1,k_2,k_3,k_4}(a,m).$$

By Lemma 1, the collision probability is bounded by $\varepsilon(\ell_a + \ell_m + 16)$ rather than by $\varepsilon(\ell_a, \ell_m)$; the rest of the analysis carries over identically, with XCHACHA in the place of XSALSA20. □

6 Shaving IAQs—Infrequently Asked Questions

Why not just use the Caesar competition winner? In our estimation, the Caesar competition[1] was too broad and dragged on for too long, and in the end failed to gain the traction it needed. While NaCl, IETF ChaCha/Poly1305, and AES-GCM are ubiquitous today, with a variety of high-quality implementations available in many programming languages, the benefits of the Caesar winners do not seem to justify the engineering effort to make the novel cryptographic primitives as ubiquitous. In contrast, Daence requires negligible engineering effort on top of NaCl or libsodium.

Why not just use AES-GCM-SIV? AES-GCM-SIV is optimized for applications that can *guarantee* hardware support for the primitives—otherwise it may be subject to severe performance degradation and/or timing side channel attacks. This may be reasonable when the engineer can control everything about the hardware and software stack, and audit the software stack all the way down to the hardware to ensure safety, but it's less appealing for a general-purpose tool.

The AES-GCM-SIV security guarantee requires unusually detailed safe usage limits [13, §9], and users are advised to choose nonces at random—which is exactly the opposite of the advice for nonce-based ciphers like AES-GCM and NaCl crypto_secretbox_xsalsa20poly1305. This limits AES-GCM-SIV's value as a drop-in replacement for nonce-based ciphers with a safety net for Tarsnap-style accidental nonce reuse bugs [16] and virtual machine rollbacks, and makes applications *more* vulnerable to broken entropy sources than Daence would.

Why not a nonce-misuse-resistant authenticated cipher? If you have a nonce—say a message sequence number—then you can use it in the associated data. So out of Daence you can build your own nonce-misuse-resistant authenticated cipher: just prefix a fixed-size nonce to the associated data!

But there is a conceptual cost to *requiring* a nonce parameter *and* associated data—what are you supposed to put in the nonce and what are you supposed to put in the associated data? The terms 'nonce' and 'nonce-misuse-resistant authenticated cipher' are confusing to non-experts (not to mention the British).

The security bounds could be better if we derived a fresh key for each distinct nonce like AES-GCM-SIV does. But AES-GCM-SIV has weaker security bounds to begin with. At the level of security that Daence provides, there is little reason to pay the interface complexity cost.

[1] Caesar: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar.html

Why bother with SALSA20? Why not just CHACHA?
The NaCl crypto_onetimeauth_poly1305 interface is widely available in many languages, particularly via TweetNaCl. It is easy to implement SALSA20-DAENCE in terms of this interface, but not CHACHA-DAENCE.

On the other hand, CHACHA/POLY1305 is seeing wider use after IETF standardization in TLS—and systems with CHACHA/POLY1305 will generally have the parts needed to implement CHACHA-DAENCE just as easily.

There is a small security advantage to the hashing in SALSA20-DAENCE; there may be a small performance advantage to the hashing in CHACHA-DAENCE; but the primary advantage to defining both alternatives is to reduce the engineering costs to adopting either one in environments that already use NaCl or already implement CHACHA/POLY1305.

Why not use AUTH256? The AUTH256 [11] message authentication code is based on a universal hash family with collision probability bounded by $1/2^{255}$ using a key as long as the message. This bound obviously seems better than the $\approx \ell^2/2^{206}$ bound for POLY1305^2, so why not reach for it?

With DAENCE, even if messages were the maximum length, 2^{33} (2^{32} bytes of associated data and 2^{32} bytes of message), and you exposed hundreds of trillions of such messages to the adversary ($2^{33} \cdot 2^{48} = 2^{81}$ bytes of data), and even if the adversary attempted about a quintillion forgeries (2^{60}, representing an unimaginable 2^{93} bytes of data altogether being processed by the application), the probability of *one* forgery would stay below 2^{-32}. So there is little *security* motivation to replace POLY1305 by a larger hash.

What about performance? Software performance of AUTH256 is not better than software performance of POLY1305. Maybe AUTH256 would improve on POLY1305^2—it's not clear, *a priori*, and since AUTH256 requires a message-length key the cost would have to figure in key generation.

But most importantly, neither AUTH256 nor any other ≈256-bit universal hash family, whether in a binary field or large prime field, is widely implemented and deployed the way POLY1305 is. So—even if there may be a slim performance improvement—switching from POLY1305 would substantially raise the engineering costs of adopting DAENCE.

How fast is it? The main cost over crypto_secretbox_xsalsa20poly1305 is evaluating POLY1305 twice rather than once, and completing it before starting XSALSA20, so SALSA20-DAENCE should cost about 1–2x what crypto_secretbox_xsalsa20poly1305 costs. If the analysis survives some scrutiny, we will submit DAENCE to SUPERCOP[2] for reliable, fair measurements across a variety of machines.

But if you must see rough numbers first, on our Intel Kaby Lake i7, SUPERCOP measures ≈2.5 cpb for crypto_secretbox_xsalsa20poly1305, ≈1.9 cpb for hardware-accelerated AES-GCM, and ≈3.6 cpb for SALSA20-DAENCE.

[2] https://bench.cr.yp.to/

There is room for improvement: this naïve code makes no attempt to compute the two Poly1305's in parallel, which may be faster than computing them serially—especially for messages exceeding the CPU cache.

Why a 96-byte key (or 64-byte for ChaCha-Daence)? For ChaCha-Daence, in addition to ChaCha we use two independent Poly1305 instances—each requiring 16-byte keys, for a total of $32 + 2 \cdot 16 = 64$ bytes of key material—in order to provide high security without nonces.

For Salsa20-Daence, in addition to Salsa20 we use *four* independent Poly1305 instances—each requiring 16-byte keys, for a total of $32+4\cdot16 = 96$ bytes of key material—primarily to support incorporating associated data without relying an incremental-update API for Poly1305. This way, Salsa20-Daence in order to take advantage of the existing deployed NaCl library which can only compute Poly1305 on an entire message.

We could start from a 32-byte master key and then derive subkeys from it. However, doing this with the obvious tool at hand—either Salsa20 or ChaCha—would add hundreds of cycles to the cost of processing each message. Since many applications already use trees of key derivation for various purposes, we feel that the cost of deriving another few dozen bytes of key material—which can be cached and reused by the application—is not worth the cost of hundreds of additional cycles per message.

Does Daence support streaming? If you want to stream large files, break them into bite-size pieces, and include the piece number and an end-of-stream flag in the associated data for each piece as a separate message. Ensure the pieces are no larger than the amount of memory you are willing to let an adversary waste with a forgery in a denial of service attack.

How is the 'ae' in Daence pronounced? Like the 'a' in 'data'.

We do not anticipate that Daence will replace ChaCha/Poly1305 in major protocols such as TLS designed by world-class cryptographers, which can easily take advantage of a message number guaranteed not to repeat. We do hope that Daence will find its way into the repertoire of general-purpose application engineers who need to store messages safe from eavesdropping and forgery in diverse software environments—*without* auditing the software stack all the way down to machine instructions to ensure their use of AES-GCM-SIV is safe from timing side channels, *without* adopting unusual cryptographic primitives or an entirely new cryptography library, and *without* teetering on the brink of catastrophe from nonce reuse in crypto_secretbox_xsalsa20poly1305 or AES-GCM.

References

[1] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. Cryptology ePrint Archive, Report 2016/564, 2016. URL https://eprint.iacr.org/2016/564. ePrint of [2], last updated 2017-11-27. Cited on p. 15.

[2] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology—CRYPTO 2016, 36th Annual International Cryptology Conference, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 247–276, August 2016. ISBN 978-3-662-53017-7. DOI 10.1007/978-3-662-53018-4_10. URL https://link.springer.com/chapter/10.1007/978-3-662-53018-4_10. Paywalled version of [1]. Cited on pp. 4 and 14.

[3] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security, 1996. URL https://cseweb.ucsd.edu/~mihir/papers/cascade.html. Full version of [4]. Cited on p. 1.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: the cascade construction and its concrete security. In *Proceedings of the 37th Conference on Foundations of Computer Science—FOSC 1996*, pages 514–523, October 1996. DOI 10.1109/SFCS.1996.548510. URL https://ieeexplore.ieee.org/document/548510. Cited on p. 15.

[5] Daniel J. Bernstein. Floating-point arithmetic and message authentication. Manuscript, last updated 2004-09-18, April 2000. URL https://cr.yp.to/papers.html#hash127. Cited on p. 1.

[6] Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption—FSE 2005, 12th International Workshop*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer-Verlag, February 2005. ISBN 3-540-26541-4. DOI 10.1007/11502760_3. URL https://cr.yp.to/papers.html#poly1305. Cited on pp. 1 and 6.

[7] Daniel J. Bernstein. The Salsa20 family of stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer-Verlag, 2008. ISBN 978-3-540-68350-6. DOI 10.1007/978-3-540-68351-3_8. URL https://cr.yp.to/papers.html#salsafamily. Cited on p. 1.

[8] Daniel J. Bernstein. ChaCha, a variant of Salsa20. Workshop Record of SASC 2008: The State of the Art in Stream Ciphers, January 2008. URL https://cr.yp.to/papers.html#chacha. Cited on p. 1.

[9] Daniel J. Bernstein. Cryptography in NaCl. March 2009. URL https://cr.yp.to/papers.html#naclcrypto. Cited on p. 1.

[10] Daniel J. Bernstein. Extending the Salsa20 nonce. Workshop Record of Symmetric Key Encryption Workshop, February 2011. URL https://cr.yp.to/papers.html#xsalsa. Cited on pp. 1 and 3.

[11] Daniel J. Bernstein and Tung Chou. Faster binary-field multiplication and faster binary-field MACs. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography—SAC 2014, 21st International Conference, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 92–111. Springer, August 2014. ISBN 978-3-319-13050-7. DOI 10.1007/978-3-319-13051-4_6. URL https://cr.yp.to/papers.html#auth256. Cited on p. 13.

[12] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, National Institute of Standards and Technology, U.S. Department of Commerce, November 2007. URL https://csrc.nist.gov/publications/detail/sp/800-38d/final. NIST Special Publication 800–38D. Cited on p. 1.

[13] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. RFC 8452, April 2019. URL https://rfc-editor.org/rfc/rfc8452.txt. Cited on pp. 1 and 12.

[14] Adam Langley, Wan-Teh Chang, Nikos Mavrogiannopoulos, Joachim Strombergson, and Simon Josefsson. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905, June 2016. URL https://rfc-editor.org/rfc/rfc7905.txt. Cited on p. 1.

[15] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015. URL https://rfc-editor.org/rfc/rfc7539.txt. Cited on p. 1.

[16] Colin Percival. Tarsnap critical security bug. Daemonic Dispatches blog, January 2011. URL https://www.daemonology.net/blog/2011-01-18-tarsnap-critical-security-bug.html. Cited on p. 12.

[17] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem, 2006. URL https://web.cs.ucdavis.edu/~rogaway/papers/keywrap.html. Full version of [18], last updated 2007-08. Cited on pp. 1 and 4.

[18] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology—EUROCRYPT 2006, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer-Verlag, May 2006. ISBN 978-3-540-34546-6. DOI 10.1007/11761679_23. URL https://link.springer.com/chapter/10.1007/11761679_23. Cited on p. 16.

A Tweetable implementation

```
#include "tweetdaence.h"    /* declares prototypes */
#include "tweetnacl.h"
#define FOR(i,n) for (i = 0;i < n;++i)
typedef unsigned char u8;
typedef unsigned long long u64;
static const u8 sigma[] = "expand 32-byte k";

static void prf(u8 *t,const u8 *m,u64 mlen,
    const u8 *a,u64 alen,const u8 *k)
{
  u8 k1[32],k2[32],ham[64],h[32],i;
  FOR(i,16) { k1[i] = k[32 + i]; k1[16 + i] = 0; }
  FOR(i,16) { k2[i] = k[48 + i]; k2[16 + i] = 0; }
  crypto_onetimeauth_poly1305(ham,a,alen,k1);
  crypto_onetimeauth_poly1305(ham + 16,a,alen,k2);
  crypto_onetimeauth_poly1305(ham + 32,m,mlen,k1);
  crypto_onetimeauth_poly1305(ham + 48,m,mlen,k2);
  FOR(i,16) { k1[i] = k[64 + i]; k1[16 + i] = 0; }
  FOR(i,16) { k2[i] = k[80 + i]; k2[16 + i] = 0; }
  crypto_onetimeauth_poly1305(h,ham,64,k1);
  crypto_onetimeauth_poly1305(h + 16,ham,64,k2);
  crypto_core_hsalsa20(t,h,k,sigma);
  crypto_core_hsalsa20(t,h + 16,t,sigma);
}
void crypto_dae_salsa20daence(u8 *c,const u8 *m,
    u64 mlen,const u8 *a,u64 alen,const u8 *k)
{
  u8 t[32],i;
  prf(t,m,mlen,a,alen,k);
  FOR(i,24) c[i] = t[i];
  crypto_stream_xsalsa20_xor(c + 24,m,mlen,c,k);
}
int crypto_dae_salsa20daence_open(u8 *m,const u8 *c,
    u64 mlen,const u8 *a,u64 alen,const u8 *k)
{
  u8 t[32],t_[32];
  u64 i;
  crypto_stream_xsalsa20_xor(m,c + 24,mlen,c,k);
  prf(t,m,mlen,a,alen,k);
  FOR(i,24) t_[i] = c[i];
  FOR(i,8) t[24 + i] = t_[24 + i] = 0;
  if (crypto_verify_32(t,t_)) {
    FOR(i,mlen) m[i] = 0;
    return -1;
  }
  return 0;
}
```

B Reference implementation

```
#include <string.h>

#include <sodium/crypto_core_hsalsa20.h>
#include <sodium/crypto_onetimeauth_poly1305.h>
#include <sodium/crypto_stream_xsalsa20.h>
#include <sodium/crypto_verify_32.h>
#include <sodium/utils.h>

static const unsigned char sigma[16] = "expand 32-byte k";

static void
compressauth(unsigned char t[static 24],
#ifdef DAENCE_GENERATE_KAT
    unsigned char v_ham[static restrict 64],
    unsigned char v_h[static restrict 32],
    unsigned char v_u[static restrict 32],
#endif
    const unsigned char *m, unsigned long long mlen,
    const unsigned char *a, unsigned long long alen,
    const unsigned char k[static 96])
{
    const unsigned char *k0 = k;    /* k0 := k[0..32] */
    unsigned char k1[32], k2[32], k3[32], k4[32], ham[64];
    unsigned char *ha1 = ham +  0, *ha2 = ham + 16;
    unsigned char *hm1 = ham + 32, *hm2 = ham + 48;
    unsigned char h[32], *h3 = h, *h4 = h + 16;
    unsigned char u[32];

    /* Poly1305: Set evaluation point; zero addend.  */
    memcpy(k1, k + 32, 16); memset(k1 + 16, 0, 16);
    memcpy(k2, k + 48, 16); memset(k2 + 16, 0, 16);
    memcpy(k3, k + 64, 16); memset(k3 + 16, 0, 16);
    memcpy(k4, k + 80, 16); memset(k4 + 16, 0, 16);

    /*
     * Message compression:
     *  ha := Poly1305^2_{k1,k2}(a)
     *  hm := Poly1305^2_{k1,k2}(m)
     *  h := Poly1305^2_{k3,k4}(ha || hm)
     */
    crypto_onetimeauth_poly1305(ha1, a, alen, k1);
    crypto_onetimeauth_poly1305(ha2, a, alen, k2);
    crypto_onetimeauth_poly1305(hm1, m, mlen, k1);
    crypto_onetimeauth_poly1305(hm2, m, mlen, k2);
    crypto_onetimeauth_poly1305(h3, ham, 64, k3);
    crypto_onetimeauth_poly1305(h4, ham, 64, k4);
```

```
    /* Tag generation: t, _ := HXSalsa20_k0(h3 || h4) */
    crypto_core_hsalsa20(u, h3, k0, sigma);
#ifdef DAENCE_GENERATE_KAT
    memcpy(v_ham, ham, sizeof ham);
    memcpy(v_h, h, sizeof h);
    memcpy(v_u, u, 32);
#endif
    crypto_core_hsalsa20(u, h4, u, sigma);
    memcpy(t, u, 24);

    /* paranoia */
    sodium_memzero(k1, sizeof k1);
    sodium_memzero(k2, sizeof k2);
    sodium_memzero(k3, sizeof k3);
    sodium_memzero(k4, sizeof k4);
    sodium_memzero(ham, sizeof ham);
    sodium_memzero(h, sizeof h);
    sodium_memzero(u, sizeof u);
}

void
crypto_dae_salsa20daence_test(unsigned char *c,
#ifdef DAENCE_GENERATE_KAT
    unsigned char v_ham[static restrict 64],
    unsigned char v_h[static restrict 32],
    unsigned char v_u[static restrict 32],
#endif
    const unsigned char *m, unsigned long long mlen,
    const unsigned char *a, unsigned long long alen,
    const unsigned char k[static 96])
{
    const unsigned char *k0 = k;    /* k0 := k[0..32] */

    /* c[0..24] := HXSalsa20_k0(Poly1305^2(a,m)) */
    compressauth(c,
#ifdef DAENCE_GENERATE_KAT
        v_ham, v_h, v_u,
#endif
        m, mlen, a, alen, k);

    /*
     * Stream cipher:
     *  c[24..24+mlen] := m[0..mlen]
     *      ^ XSalsa20_k0(t @ c[0..24])
     */
    crypto_stream_xsalsa20_xor(c + 24, m, mlen, c, k0);
}
```

```
int
crypto_dae_salsa20daence_open(unsigned char *m,
    const unsigned char *c, unsigned long long mlen,
    const unsigned char *a, unsigned long long alen,
    const unsigned char k[static 96])
{
    const unsigned char *k0 = k;    /* k0 := k[0..32] */
#ifdef DAENCE_GENERATE_KAT
    unsigned char v_ham[64], v_h[32], v_u[32];
#endif
    unsigned char t[32], t_[32];
    int ret;

    /*
     * Stream cipher:
     *  m[0..mlen] := c[24..24+mlen]
     *      ^ XSalsa20_k0(t' @ c[0..24])
     */
    crypto_stream_xsalsa20_xor(m, c + 24, mlen, c, k0);

    /* t := HXSalsa20_k0(Poly1305^2(a,m)) */
    compressauth(t,
#ifdef DAENCE_GENERATE_KAT
        v_ham, v_h, v_u,
#endif
        m, mlen, a, alen, k);

    /* Verify tag: c[0..24] ?= t (no crypto_verify_24) */
    memcpy(t_, c, 24);
    memset(t + 24, 0, 8);
    memset(t_ + 24, 0, 8);
    ret = crypto_verify_32(t_, t);
    if (ret)
        sodium_memzero(m, mlen); /* paranoia */

    /* paranoia */
    sodium_memzero(t, sizeof t);
    sodium_memzero(t_, sizeof t_);
#ifdef DAENCE_GENERATE_KAT
    sodium_memzero(v_ham, sizeof v_ham);
    sodium_memzero(v_h, sizeof v_h);
    sodium_memzero(v_u, sizeof v_u);
#endif

    return ret;
}
```

```
#ifdef DAENCE_GENERATE_KAT

static const unsigned char k[96] = {
    0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
    0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
    0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
    0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
    0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,
    0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
    0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
    0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
    0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,
    0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
    0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,
    0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
};

static const unsigned char a[16] = {
    0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,
    0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
};

static const unsigned char m[33] = {
    0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,
    0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
    0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,
    0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f, 0x90,
};

static void
show(const char *name, const unsigned char *buf, size_t len)
{
    size_t i;

    printf("%s=", name);
    for (i = 0; i < len; i++) {
        printf("%02hhx", buf[i]);
        if (i + 1 < len && ((i + 1) % 24) == 0)
            printf("\n%*s", (int)strlen(name) + 1, "");
    }
    printf("\n");
}
```

```
int
main(void)
{
    unsigned char ham[64], h[32], u[32];
    unsigned char c[24 + sizeof m], m_[sizeof m];
    unsigned i;
    int ret = 0;

    for (i = 0; i <= sizeof m; i++) {
        /* paranoia */
        memset(ham, 0, sizeof ham);
        memset(h, 0, sizeof h);
        memset(u, 0, sizeof u);
        memset(c, 0, sizeof c);
        memset(m_, 0, sizeof m_);

        /* test */
        crypto_dae_salsa20daence_test(c,
            ham,h,u, m, i, a, sizeof a, k);
        if (crypto_dae_salsa20daence_open(m_, c,
            i, a, sizeof a, k) != 0)
            ret = 1;
        if (memcmp(m, m_, i) != 0)
            ret = 2;

        /* show */
        printf("mlen=%u\n", i);
        printf("alen=%zu\n", sizeof a);
        show("m", m, i);
        show("m_", m_, i);
        show("k", k, sizeof k);
        show("a", a, sizeof a);
        show("h_a", ham, 32);
        show("h_m", ham + 32, 32);
        show("h", h, sizeof h);
        show("u", u, sizeof u);
        show("c", c, 24 + i);
        printf("\n");
    }

    fflush(stdout);
    if (ferror(stdout))
        ret = 3;

    return ret;
}

#endif  /* DAENCE_GENERATE_KAT */
```

C Test vectors

```
mlen=0
alen=16
m=
m_=
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=000000000000000000000000000000000000000000000000
    0000000000000000
h=da2fa7a11e4476ed9dc3384f196c7afd52bc3543af53eca1
  d498781dc8e91246
u=cb9fdd8b7d715b47fa571f1c27aebda1ed556f4b55a68033
  f3350fdac10b43f4
c=762709b9b287e7bd12351f2b0371509cc923f6c2ae612e61

mlen=1
alen=16
m=70
m_=70
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=199e0f81c463d5468a299b0c50ef60d219b52698c47aec5d
    8a40b223500678e9
h=b860cc41353a31d18f44bea79c237bd3921531ed61ea2b28
  0c5fdc236b02cca7
u=ca0397f120d48a492d901f2254ce82ca0892d0c7a61179a4
  f7db29838d39e92d
c=844c482d0cfb1b658bbdd74ba1404eeeb5206bde14c70ccf
  ec

mlen=2
alen=16
m=7071
m_=7071
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=62b9ad904528391cd1b3c4a75c3f50337bd0dbbe5c3f674a
    e8caf2d573567e61
h=f8ea8c4aa43831a1ad45a54b68a1aeab81a0a20bf4042b53
  10174c2262119869
u=f6e114983789324bcfad887a532a53ccc31816434c942a7a
  3adcea440cf0879c
c=6dce0f262deddde10ba72249e4bddb454119cb8580c6ef76
  3d1e

mlen=3
alen=16
m=707172
m_=707172
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=5113e94f77ac21a5cc0177fa2157cc4f614727a5b5da5ffa
    0a30b54f60850aa5
h=b690dd3a1edf2489de743ed1e3c6bd93047b87ac06ef22d9
  0ef93731c6ab048d
u=ac4b3e835031edaa2bbe03156726bda452350f9195fcd458
  0acfbd13674c6268
c=b3e66c3fc5ac774608a2f3d255b6738f0ee28d8b3eda60ac
  2fed3f

mlen=4
alen=16
m=70717273
m_=70717273
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=f0ff55ab5700a9b17a23ccd49d46eff70139b120cd65fe26
    f088214a13ac446d
h=bcf61c535749d2709ac96245db02e538df398cc0cc666403
  418d7eb728b64b98
u=f156f947c26c6498443d838be3c571bab1d9b02c95c80f03
  d806605f3261e5bb
c=de839ad762740fe5f3284d596109089e45fdc578795299cb
```

```
  824c3cca

mlen=5
alen=16
m=7071727374
m_=7071727374
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=79543c2bd3011f3cabf61331a0eb082686a6b0bd78aebbc8
    50a3b0bd4598a5b2
h=58bb88ebb8615e4689155387a657dbff76d173d0cb60c3de
  b51fe5604a84e468
u=d0959a8374902d61a27814e8b63aebab36f4cb303caebf5e
  af4a7f9ec788586b
c=91850eb531e6162198ead702942debbc59c1616e30ebe629
  f2348d130b

mlen=6
alen=16
m=707172737475
m_=707172737475
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=bf54490b669d41d4384b0c9f0316d769eab6eaca288a35b8
    f5370083c002cb4d
h=9873c5599810462e9e2939580058437ec020d073ee6903e0
  f5a6f268bc75f10e
u=2fc7f16f864fdd0208031a802bb9a2523df3457be9819df4
  cceb71bda5592a33
c=14b840b3392558f62dfd41747acb52cbc5e812a5d8715eaa
  9b86a776e3af

mlen=7
alen=16
m=70717273747576
m_=70717273747576
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=4baac0d03f43fd17f3db84bc97802961b82986d1434d4163
    17e0c807bc846dac
h=45b41d51774e9eadca4ff16242742946213bd2d0632e9073
  8bd46aa3f7e79b47
u=3d82843a4d745c237a5414734f348bbccacf8265494e9307
  77b8128dfa5ea3bc
c=81c5243427c8cc5d0ef985b9ab15384f32efc6ea454c7d88
  7b182646883f1e

mlen=8
alen=16
m=7071727374757677
m_=7071727374757677
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=732824bfbd16b6f357b81859da3a9bdb36fd06f81676179f
    f333740476b6f686
h=8fdf405b620bbaa19a70fd64961e50a24f3ccc2a7ba51c58
  2cafd33c5fe7fbc8
u=d0cacdbffca0825141e8567e017fcccd2a6c0fbce190ccf0
  88dd5aa306d57835
c=8200c3085ba9c392bc384a276f67f8ff988c1dcdb5ab660e
  02c5d2b9ef498849

mlen=9
alen=16
m=707172737475767778
m_=707172737475767778
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=7b049730234d83bf533e17f09aa27b548a42e386c8154e88
    5f41fab2a6a55e17
h=b366d74801bb445de175ab9d1c695e5d381c6ef5b8fdbd52
  1901625f176aa6a4
u=f2e26553420d4c882507e0e7d7b278592b5582303e597a35
  e9679bd235e48275
c=7c2772be0bf9e5aa90fbf9041af3948caead0daa0287943c
```

```
  f6688b6180def2ed91

mlen=10
alen=16
m=70717273747576777879
m_=70717273747576777879
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=9b812698a2297286325abe103587085bc81ff06b6552bacc
    5bdd386b580a83b5
h=2b235e4fec7aa0d7df12a7ae0d497677e98f773c1a403cf7
  89bc9c12e333384f
u=11ec8d3f3c97bfe56496b22b0305757dceab0e05adebb1de
  72b251d78edfe232
c=1f0151626829065e7b8e79c8da1a56a5be6f02240e96671b
  f668b3d121e631384206

mlen=11
alen=16
m=707172737475767778797a
m_=707172737475767778797a
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=97aa15dbfeb6c52df34bfad87724110d5166534053fd8105
    aeec04db42bf1b0f
h=daf8b916e1772628577fbfd2ff15375ae356c0f874094b6f
  eade4936a03074fe
u=19b4e3bf1762d7da9dd2017284d5e89a61bd42b860215882
  63a572926551ba9c
c=d7e39253df23a93cf06af34ca9b58f3ff92fd591faf4ef1d
  23fddea6bf3e5ead731c52

mlen=12
alen=16
m=707172737475767778797a7b
m_=707172737475767778797a7b
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=2092473cf50761f85206ff346189b13985f3a229eff33a58
    b34c27d7e3dbd3db
h=1f75b7be8c1a9a19e21c99318e032ece6ea76339c146b5c5
  72c90512938829c9
u=4dfcb2f5a9f19c4631e398e7c6c3cb63deceb1da65ce631d
  6065a7b641d3da82
c=ddcc71b2d2b801a973f182bf57706e6b441bff52ba19415e
  4ee562aaf5ea9ccc75b40388

mlen=13
alen=16
m=70717273747576777879 7a7b7c
m_=707172737475767778797a7b7c
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=84cd1e77c8b1a6a1941eb34cdd9338dd85e833825e573a1f
    911e950c10ae2297
h=c5063b361676c732800afc74e417e2f586de76c028d22e12
  9984a697063b4856
u=2fdd46ce74bffd164dbfec9f237433abda5d8b964cf446c4
  81b77d5656f243f4
c=75554db41fb176c7c73623d43057d3dc48a350aa2cb89261
  c5a567c3f88f343f708df4fda9

mlen=14
alen=16
m=707172737475767778797a7b7c7d
m_=707172737475767778797a7b7c7d
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=569f083e0cf703dc4bd783fa07306486756aeb16c04c6527
    66873388580a2618
h=c9f1d84061c8673aeda2bdcbad3cf7b40b4b0ce911ad78e7
  46aed4dadf74d91c
u=231947fc4f1ddbcab4fd0919c32803916003a680c7204b2a
  b95d8194e428e1cc
c=f3b9e3e78fc196d85489e653f058e99543219469b6e382a5
```

```
  1aaa95bf3d4a7b7eb8ee9d57a561

mlen=15
alen=16
m=707172737475767778797a7b7c7d7e
m_=707172737475767778797a7b7c7d7e
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=957347d6c243bbec7a9cb383af6d20d3965cee9058b7e019
    776a27f3e165b24c
h=6005e5089f73d7ecbae8a06ab59fd2b01d3c91a6fd802d24
  a7980c2080de9cb9
u=6d8fb42fe180aaa6edf25ce8fa63d68be70b4fecea549ff1
  ed67369800d883a8
c=30dad0b473cd10d3080513d104098fcee23961ffa01c574b
  a76b428a928658905d10d4e8825f4c

mlen=16
alen=16
m=707172737475767778797a7b7c7d7e7f
m_=707172737475767778797a7b7c7d7e7f
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=5c9c1f8209ea10163fc0862af10e71af537be4149553541b
    3184187219fd2009
h=52b326c9caa73391488ff7f75c9cdfe99927c3f19c8f4799
  bef333b22ea79621
u=2218ddf34a39e717202134869b79668906966b85a8809b36
  200a07a1ddbbfdf2
c=75236be4a3d3df0614d2bd8f2ceb6b12c4e986e918e513fa
  41a90081283be2ba2273c376dd08c3b2

mlen=17
alen=16
m=707172737475767778797a7b7c7d7e7f80
m_=707172737475767778797a7b7c7d7e7f80
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=0b70e14933fb764a55520ac2afec3572d1cbf1ef0f8a819f
    1649c51557164e6b
h=e910f931eb3c59e7d741b55c11d8eb0bead5711f6c4a40a0
  a36ac9b677a1bf67
u=0b6e71f1f1e5e015cb836a7ec20c17b78380844100216693
  37af799296e13ca2
c=35ed97cde04e867d6646d7206c9a1afc86751a075cc6bdae
  baf7d0d042ca68c6de99aae3bfa0cd67e9

mlen=18
alen=16
m=707172737475767778797a7b7c7d7e7f8081
m_=707172737475767778797a7b7c7d7e7f8081
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=858c917be6ff2c820e5dc6ff6efdf7b564e8b939db8e4fef
    e753996b2e2728c7
h=3a19c87bdb3e044189e053805d9b3410c4ce03ec3a40a59f
  08626b0ad21fd44a
u=41001223ba749e7bd560537f6e1c5d30bc34ccd417097e44
  2e582c003ae82a68
c=ce5361c1f3b8799acf51abc5ae105dea99db455fc4d0c338
  79e5f83ec7f12d163155717e3ab8da8b1e29

mlen=19
alen=16
m=707172737475767778797a7b7c7d7e7f808182
m_=707172737475767778797a7b7c7d7e7f808182
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=1116ce4c3ab6555d6c1df9e4d6c734a527900633575d88f2
    6d2cdc78be0975de
h=bda5c578c4b5a386b6ef33b1a121cfe2b5817b7f2bb246cd
  91470e02d226e5fa
u=4ce0c457c93746ed1ec98b95472c91444ca4220565391b9c
  9e31c536481fc2e3
c=bd16fdd2ebd7e29e9dfbd07bf73ae8fdaff99fb3809a2031
```

```
  9cad7fae72443de660047138211797af03970f

mlen=20
alen=16
m=707172737475767778797a7b7c7d7e7f80818283
m_=707172737475767778797a7b7c7d7e7f80818283
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=349e6aa92c2c0faa6ca1c03fe5590a0e905ec1af810b5a5f
    a6e8bbf304d46267
h=0e2a40a2f9aa792125d56602045193883dc5cc1f0e57d6bd
  39009ec5ee29ef44
u=b0931de0261ab7aa0eea763f8e8d04859cfad53f049b419e
  0ec802258f30e68d
c=b6eecfc2d50fa56f845e0d757a9c59d8298a59ee2a54148b
  1476eec2432b2e5b50905d9a11910a068bfcaf98

mlen=21
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384
m_=707172737475767778797a7b7c7d7e7f8081828384
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=2d7aec58a93fa766ddc66a0e6891c6eec5949d7d2e673a34
    4756aedab7536760
h=34857e128f04c06a7b6ace6ffce9f2996876088828b72814
  a92bc8ced645763d
u=e71ee727c6eabb06a028d98521e6a4f3611db8e024c654c8
  0607fbb36ea5747b
c=03b637d04e5b0d77c6c140b99716627ce4f04c8d2c49b036
  ed01e154d16ed3c70e7dd3e25668e260a8bfb3b7aa

mlen=22
alen=16
m=707172737475767778797a7b7c7d7e7f808182838485
m_=707172737475767778797a7b7c7d7e7f808182838485
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=54eb80d46bdcdb209d5bb5de3d3c27d50a56a0670f44c746
    1f2b5103a63e209f
h=bba9d7bda795f7b9705990a2f44ab0a496c0bcfc2feea777
  e066ddf50f59d014
u=822c587bfaa63e39617c15b798ea68a5622c7db4fd640a5e
  c4028527a0e6808a
c=8993e03339201e7c6fcb17c4e19b4fa8412c617a5bd91bbd
  265346e345c611ff858141ad1ec442ce9072e38772c4

mlen=23
alen=16
m=707172737475767778797a7b7c7d7e7f80818283848586
m_=707172737475767778797a7b7c7d7e7f80818283848586
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=2d206921e1b19876791e6e4e3419fa5e60a9ec360738d404
    64065adb04344391
h=fddb200234601cd95c556ef3d3979d41bf5d6accadc28a33
  dc3aa4ea35c0a411
u=566529a2b8e279121a0b3d28b95c8fafa58a13f3f31a9eca
  81a592403df31cb4
c=2cfd1fe271bc8dc6aa8ff86f8cd4d29250fa5720fcd9f9c0
  77d9a8b5062a9a23e27b79e12d22c2eeececccea61351c

mlen=24
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
m_=707172737475767778797a7b7c7d7e7f8081828384858687
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=89e9ab80e6208153f01c342bc935de5957094e0ea33ddb41
    537d381812c93fec
h=c9ae25475fb25057f722b9a1394fcc9969dc7116a6299049
  11a41d7c0c73d8bb
u=2194349372c64bb783442227c8dbf29cab22793b31c8c90a
  db281969ae1ff511
c=99aee961eb465caab356c31ceb087a28ed0d0cca2b732d97
```

```
  a7938e0f1446c2c16ddd3a39e0f2e2dd9a64f0529460ada4

mlen=25
alen=16
m=707172737475767778797a7b7c7d7e7f808182838485868788
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=b6fc69d1bcdee94eedb454f4c9ef204510c7b67d05a6ee5b
    c09de1f9820b0bf0
h=654ccb34c608e4b36d61a7fcf7cc220073ad2191a02a3331
  1c97969a4b1619e7
u=da8296265b3a1cce01522540ff3e7796707ac265ae4940e3
  e2f0a765140b8bde
c=e55749c4765d25bb1229b7b884a7be1873bfdc92bf28bef0
  b359b145b2b2e779c33c4ded678ca4c22aeeff33155cad79
  ad

mlen=26
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  8889
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   8889
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=679a30841b2c60b1fbd10d37961400aedf043cef8293237d
    ed3a33d567b082f1
h=d408232105058a6ce09277fb806738c95dd29a3c096e2103
  d459824384134c44
u=5114f35c9352f760c846ae23759e88262607c58d1f1db681
  911ed551783f1a79
c=c1ed1e6aa0245740cc4ecb1533d948495d2ae9d0e0a8bc8a
  fc67949cea590a5623ad2f80a8bdcc378c32545c1cf19c0e
  327e

mlen=27
alen=16
```

```
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f2021222324252627282292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=565240fec29824e057f34a11fbe348b2a5dbff3bfd1e9c7e
    1c7b005875985b9e
h=fea6bfb210a797ba08a2fc0be8d23b71293756b6dada8139
  42b45cd57b819dcf
u=5a1bbbe3817dbf943274b687d101e74c4d6ce1e7e031ee5f
  b12553e35cef03be
c=e134baf6bdf660b2d5657d54de1a36d8f183ac062a292ec2
  55d838206503b7a9fadfc67ff10c855325c1d2bf2b8f42c1
  55a530

mlen=28
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f2021222324252627282292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=cd340180f0349f1b3f497f6ef66a1b1ffda4df8511a23582
    eab7535529d846ab
h=e8a07cf78ccf0a0667dd13c676d7acfb23a5bfa1f5aa2a76
  e1830e48beb0778a
u=2c277f60b1515a3c638335a12f6ef0af704963d8278ebaee
  6a3f12220bb679cf
c=1d58a6548207fb20d8307a69335985f0c8e381979f277d00
  fdf744610626f784a3dd3ab4b1dd6ccd55f23e372f76c320
  1dc8fd55

mlen=29
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b8c
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b8c
```

```
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=0157d349e41530a4f1e8ceb57387c4f40dc2ac6ee17dc129
    79529ebb56bdb899
h=7a869e2cd476f1b5fe91194f376271528e66c2ccbff3b1b4
  f516d17e6baae995
u=8ec07f6f0e219056f9d683ebb3d6e535f6719b42730274a1
  10bf5cd4374df038
c=bafc2b4212a6066bc1b2b0d0839a54743219336df76d147d
  757d3c16173a5475fc0df2a85ab88e0dd870823319a61942
  6566194506

mlen=30
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b8c8d
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b8c8d
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=0ff9a30bb77bc429881227ffcd2402c03e548c3fd3d364be
    2e6c0514d01acf3d
h=8de51dadb41a72e46dd560797c01316b01bf5692e0084d95
  ad509e4fbabcf8cf
u=cdbc732c78db41e7bdf6037b8d3e160d6df1f628c5c4fd2e
  9cda4d77996b883d
c=1dbd0743b8dcd590cf438b397c3333d99a2babedb9ed48e6
  f79b8364d9e4aaaf5d6ef8a0f9f237b03af582d446f240c9
  e43f032d83e5

mlen=31
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b8c8d8e
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b8c8d8e
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
```

```
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=fb0bb38a68579c7102b7c70f1192bf1e47869fe1a7ce4029
    cc2faa4736a75c85
h=64e0185330a119153e3496237ef007895b0415794bcaac79
  dc4466935aa4d3f3
u=c84559604fbba0f5ee5e977a3fa350aa491ce769fb365b2a
  d9a4da95ec138aed
c=dd460078fbac7bf63d24a12ca00013c78611b6372b8d54b9
  9e98cb12976c62c528805f13fd3cc3246e06f2777bb0c9bc
  28b3769f6a829c

mlen=32
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b8c8d8e8f
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b8c8d8e8f
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=56dfc90696f880bbfd257a27dace3ffce290d5750ca7448b
    fed57b77361bfc42
h=9bd930763a03e95602c96482d30a0da7da2c8e8237547a79
  4c753f0969ab0c7c
u=99e99122d1bd10e40dcc736ae2d57b3bcd150170712d5599
  007cf041338bcd4e
c=4f79fba5c6821587611154c7a386ca5df87865d5774c73ed
  d9ec09f9412d41ceaf2dfb7638a86b2e0958a3e68a63a4cb
  0691a9ae350a5eae

mlen=33
alen=16
m=707172737475767778797a7b7c7d7e7f8081828384858687
  88898a8b8c8d8e8f90
m_=707172737475767778797a7b7c7d7e7f8081828384858687
   88898a8b8c8d8e8f90
k=000102030405060708090a0b0c0d0e0f1011121314151617
  18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f
  303132333435363738393a3b3c3d3e3f4041424344454647
  48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
a=606162636465666768696a6b6c6d6e6f
h_a=59192594ab20e001cbf05f30a779940f7c699bd828bc9579
    6fa724ab421b385d
h_m=0285637b57b6dde5cb69ccc05b7a96968715651951b52560
```

```
    33ffef1311bc1e95
h=9c344e839ead36cd4c258b44448a869cffae9223b6a17314
  d957eef7460a829d
u=c31b08fdfaecf5c7384a43322da3ceb3fc82bcd41caceada
  029845b7bd122d8c
c=a5096e6cd6564131dcfbd186cb1e13728e2b6719b0bf7194
  14fb8f328fca052acd4327d13712679619355663185538 71
  b90cc90829a9d960f9
```