T-79.515 Cryptography: Special Topics

# Poly1305-AES MAC

Sami Vaarala

Helsinki University of Technology

sami.vaarala@iki.fi

# Background

Security of MD5 and SHA1 is dubious, so a MAC with a security proof relative to a block cipher would be nice. Poly1305-AES provides such a MAC.

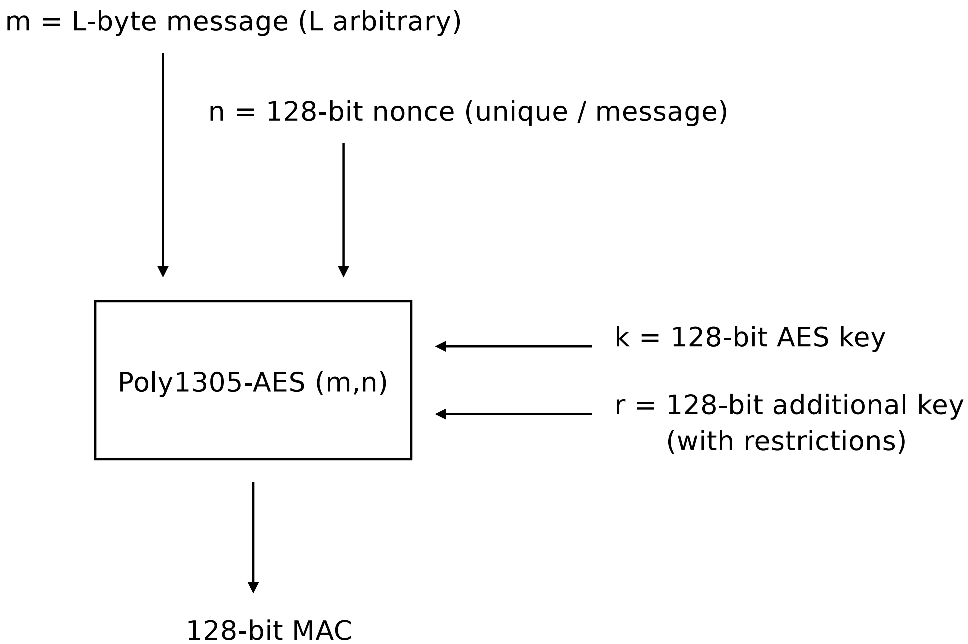
This presentation is based on the following papers:

- Daniel J. Bernstein: *The Poly1305-AES Message Authentication Code*, Fast Software Encryption (FSE) 2005.
- Daniel J. Bernstein: *Stronger security bounds for Wegman-Carter-Shoup authenticators*.

# Poly1305-AES description

# Poly1305-AES in a nutshell

$$\boxed{\text{Poly1305-AES}_{(k,r)}(n,m) = h_r(m) + \text{AES}_k(n) \pmod{2^{128}}}$$




- $h_r(m)$ is a polynomial defined by message $m$, evaluated at *additional key* $r$, modulo $2^{130} - 5$.
- $AES_k(n)$ computed using a 128-bit key $k$ with a (guaranteed to be unique) nonce $n$, result interpreted as an integer modulo $2^{128}$.
- The two terms are finally summed modulo $2^{128}$, yielding a 128-bit result.

# Intuition

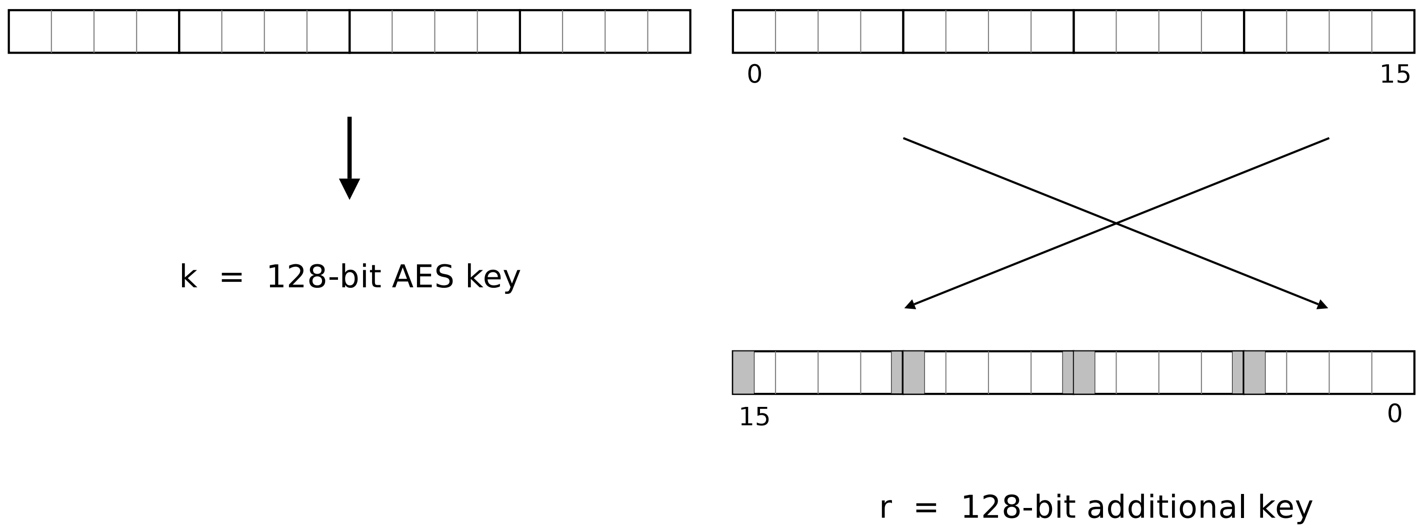
We don't want to expose the I/O relationship of $h_r(m)$, so we mask the term with a uniform random injective function evaluated at a (guaranteed to be unique) nonce, resulting in a random "masking value" which never repeats.

An actual uniform random injective function is impractical, so we use AES to simulate one, relying on AES to be indistinguishable from a true uniform random injective function. The resulting key $(k, r)$ has a fixed size (256 bits). The AES indistinguishability assumption is dealt with in the security proof.

The crux of Poly1305-AES description is in the details of the function $h_r(m)$, especially how an $L$-byte message is broken up into a polynomial (modulo $2^{130} - 5$).

# Key format

The 256-bit key $(k, r)$ consists of a 128-bit AES key, $k$, and an additional key, $r$. The AES-key is straightforward, but the additional key has some restrictions, yielding a key length of $128 + 106 = 234$ bits.



# Key format...

The additional key, $r$, is a little endian interpretation $r = r[0] + 2^8 r[1] + ... + 2^{120} r[15]$ with special bit restrictions to optimize implementation (actual key size 106 bits):

- $r[3]$, $r[7]$, $r[11]$, $r[15]$ are required to have their top four bits clear.
- $r[4]$, $r[8]$, $r[12]$ are required to have their two bottom bits clear.

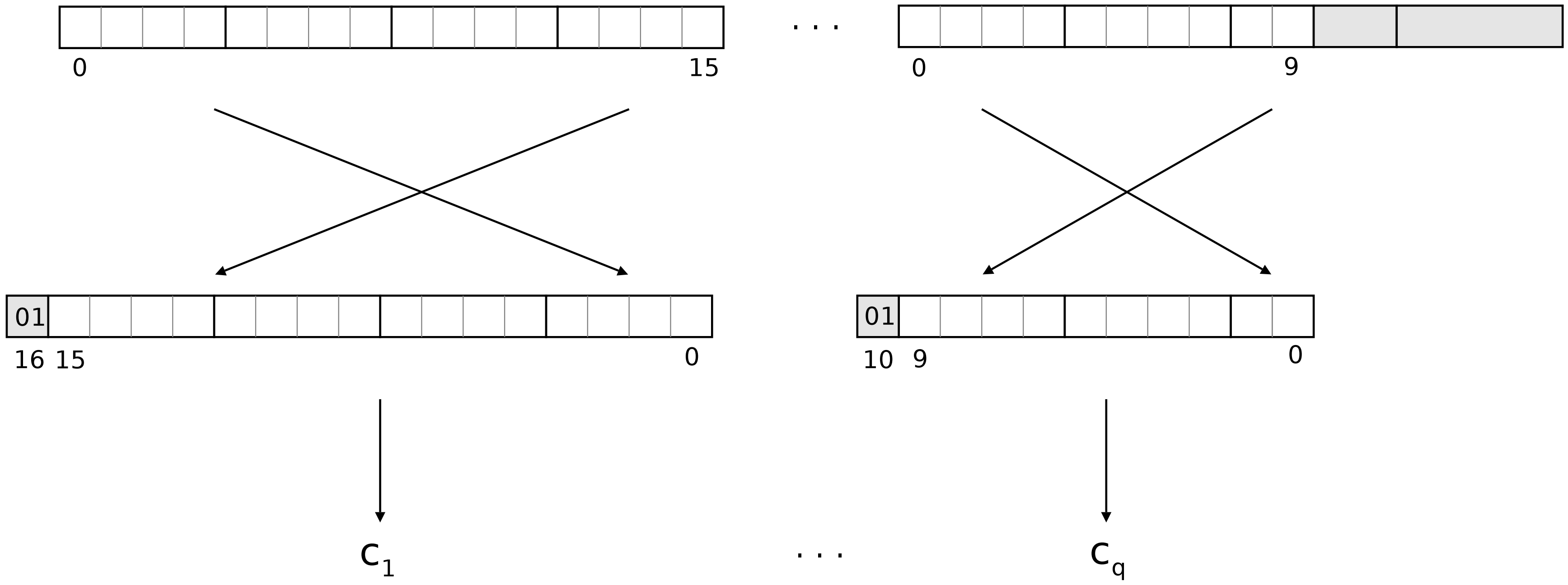
The implementation (which uses floating point arithmetic) represents a large integer as $x = x_0 + x_1 + x_2 + x_3$. The bit restrictions for $r$ ensure that carries can be propagated conveniently in this representation. The restrictions don't seem to have a security reason.

# Input padding

Input message $m$ of $L$ bytes is processed in $q = \lceil L/16 \rceil$ 16-byte chunks, with possible last partial chunk having special treatment. The chunks are interpreted as little endian integers and referred to as $c_1, ..., c_q$:

1. Append 1 (0x01) to the $i$th chunk.
2. Given a partial chunk, append the chunk with zeros to 17 byte length.
3. Interpret the 17-element array as an unsigned little endian integer, $c_i$.

# Input padding...



# Input as a polynomial

Construct polynomial $f$ from chunks $c_1, ..., c_q$:

$$f(x) = c_1 x^q + ... + c_q x^1 \pmod{2^{130} - 5},$$

which is easy to evaluate incrementally. Initialize accumulator $h_0 = 0$; for $i = 1, ..., q$, update $h_i = (h_{i-1} + c_i)x$, reducing intermediate results modulo $2^{130} - 5$, resulting in:

$$\begin{aligned} h_0 &= 0 \\ h_1 &= c_1 x^1 \\ h_2 &= c_1 x^2 + c_2 x^1 \\ &\ldots \\ h_q &= c_1 x^q + ... + c_q x^1 \end{aligned}$$

Final value $h_q$ is $f(x)$.

# Definition of $h_r(m)$

The $h_r(m)$ term in

$$\text{Poly1305-AES}_{(k,r)}(n, m) = h_r(m) + \text{AES}_k(n) \pmod{2^{128}}$$

is computed quite simply by:

1. converting the input message $m$ into the chunk values $c_1, ..., c_q$;
2. generating the corresponding polynomial $f(x)$; and
3. evaluating the polynomial $f(x)$ at $r$, the additional key, resulting in $h_r(m) = f(r)$.

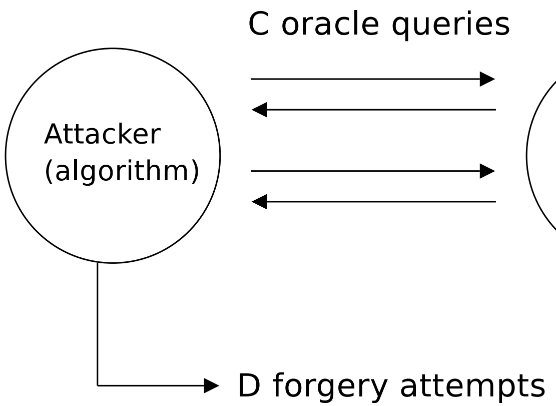
# Completing the computation

The $h_r(m)$ term is reduced modulo $2^{128}$ and added to the 128-bit AES term. The result is reduced again modulo $2^{128}$, and finally converted into a little endian representation.

This results in a 16-byte (128-bit) final authenticator value.

# Poly1305-AES security proof

# Attack model




$$S(n,m) = h(m) + f(n)$$
$$S(n,m) = h_r(m) + AES_k(n)$$

- Attacker performs $C$ (adaptive) queries $(n_i, m_i) \to S(n_i, m_i) = a_i$ from oracle S, with restriction $m_i \neq m_j \Rightarrow n_i \neq n_j$. (Duplicate nonces not allowed unless message also duplicate.)
- Attacker prints out $D$ forgery attempts $(n'_i, m'_i, a'_i)$.
- Attack successful if at least one forgery attempt has $a'_i = S(n'_i, m'_i)$ and $n'_i, m'_i$ is a fresh pair.
- I.e. forged nonce/message pair is new, and accepted as authentic.

# Preliminaries - Interpolation probability

Let $f : N \to G$ be random (not necessarily uniform). **Maximum $k$-interpolation probability of $f$** is the maximum, for all $x_1, ..., x_k \in G$ and all distinct $n_1, ..., n_k \in N$ of the probability that $(f(n_1), ..., f(n_k)) = (x_1, ..., x_k)$.

In other words: consider all input-output vectors and compute the probability of that input-output combination **over distribution of f**. Take the maximum. This is useful as a bound for the probability of a certain input-output combination given that $f$ has some random distribution, and is used in the security proof for $f$ (ultimately, AES).

# Preliminaries - Interpolation probability

**Uniform random function, N and G finite,** $\#N \le \#G$. Then maximum $k$-interpolation probability of $f$ is $1/\#G^k$.

Proof: $(f(n_1), ..., f(n_k)) = (x_1, ..., x_k)$ with probability $1/\#G^k$. Note that each selection independent because $n_i$ are distinct.

**Uniform random injective function, N and G finite,** $\#N \le \#G$. Then maximum $k$-interpolation probability of $f$ is $(1-(k-1)\#G)^{-k/2}/\#G^k$.

Proof: Fix $x_i$ and (distinct) $n_i$. If $x_i = x_j$ for some $i \ne j$ (collision), probability is 0. If no collisions, $P[f(n_1) = x_1] = 1/\#G$, $P[f(n_2) = x_2] = 1/(\#G - 1)$ (conditional), etc. Total probability $(1/\#G)...(1/(\#G-k+1)) = ... = (1-(k-1)\#G)^{-k/2}/\#G^k$, independent of particular $x_i$, $n_i$ (when $x_i$ don't collide).

# Preliminaries - Differential probability

Let $h : M \rightarrow G$ be random (not necessarily uniform), $M$ a finite set, and $G$ a commutative group. Assume for all $g \in G$ and all distinct $m, m' \in M$ that $P[h(m) = h(m') + g] \leq \epsilon$ (over distribution of $h$). Then $h$ is said to have a differential probability of $\epsilon$.

In other words: when considering certain two distinct inputs (messages) $m, m'$ what bound can be placed on the probability that their output difference $h(m) - h(m')$ is exactly equal to some specific value $g$? Note that the probability is computed over $h$, the polynomial, which is not assumed to be uniform in the main proof.

# Statement of main theorem

**Assumptions**

- Let $h : M \to G$ be random, $M$ nonempty, $G$ finite commutative group. Let $f : N \to G$ be random, $N$ finite, $h$ and $f$ independent.
- Let $C$ (# oracle queries) and $D$ (# forgery attempts) be positive integers. Assume $C + 1 \le \#N \le \#G$.
- Assume maximum differential probability of $h$ to be at most $\epsilon$.
- Assume maximum $C$-interpolation probability of $f$ to be at most $\delta/\#G^C$, and maximum $C+1$-interpolation probability to be at most $\delta\epsilon/\#G^C$.

**Then** any attack with at most $C$ oracle queries and at most $D$ forgery attempts succeeds against $(n, m) \to h(m) + f(n)$ with probability at most $D\delta\epsilon$.

# Proof of main theorem

## Simplifications

- Suffices to show that probability of one successful forgery attempt is $\delta\epsilon$.
- Assume all $C$ queries are distinct.
- $\Rightarrow$ We're trying to bound the probability of one successful forgery attempt, given $C$ distinct queries.

## Naming

- $(n_i, m_i)$ is the $i$th oracle query with response $a_i = h(m_i) + f(n_i)$, $n_i$ distinct.
- $(n', m', a')$ is the attempted forgery, where $n'$ may be one of $n_i$.

# Proof of main theorem ...

All outputs of the attack (algorithm) are functions of (1) coin flips $b$ and (2) oracle responses $a_i$. In particular:

- $n_1, ..., n_C, m_1, ..., m_C, n', m', a'$ are all functions evaluated at $b, a_1, a_2, ..., a_C$.
- Furthermore, $a_i = h(m_i) + f(n_i) \Rightarrow f(n_i) = a_i - h(m_i)$ is a function of $h, b, a_1, ..., a_C$.

Fix $\bar{g} = (g_1, g_2, ..., g_C) \in G^C$, and let $\bar{a} = (a_1, ..., a_C)$. Consider the event that $\bar{a} = \bar{g}$ and $(n', m', a')$ is a successful forgery. If we can prove that the probability for this is at most $\delta\epsilon/\#G^C$ (for arbitrary $\bar{g}$), then the probability of a successful forgery (regardless of particular $\bar{a}$) is at most $\delta\epsilon$ (regardless of distribution of $\bar{a}$).

# Proof of main theorem ...

The proof is split into two sub-cases: (1) $n'$ is fresh; and (2) $n' = n_i$ for some $i$. More formally: let $p$ the unknown probability (case 1) that $\bar{a} = \bar{g} \Rightarrow n' \notin \{n_1, ..., n_C\}$. Since $\bar{g}$ fixed, $p$ depends only on $b$.

**Case 1**. By assumptions, $\#\{n_1, ..., n_C, n'\} = C + 1$, and $f(n_1), ..., f(n_C), f(n')$ are various functions evaluated at $h, b, \bar{g}$, and $f$, $h$, and $b$ are independent, $\bar{g}$ fixed. The conditional probability of $f$ interpolating these $C + 1$ values is at most $\delta\epsilon/\#G^C$ (assumption on $f$'s interpolation probability). (Note that we first compute the *required values* for $f$ and then the probability of $f$ taking on the values.)

# Proof of main theorem ...

**Case 2**. By assumptions, $\#\{n_1, ..., n_C, n'\} = C$, and $n' = n_i$ for a unique $i$. We must have $m' \neq m_i$ (otherwise not a forgery), $a_i = h(m_i) + f(n_i)$ and $a' = h(m') + f(n') = h(m') + f(n_i)$. Then $h(m_i) - h(m') = a_i - a'$. The inputs $m_i, m'$ and output $a_i - a'$ are various functions evaluated at $b, \bar{g}$, and thus independent of $h$. By assumption on $h$'s differential probabilities, $P[h(m_i) - h(m') = a_i - a'] \leq \epsilon$. Furthermore, the probability that $f$ interpolates the required $C$ values $f(n_1), ..., f(n_C)$ is at most $\delta/\#G^C$.

**Wrap-up**. Total probability of success is at most $p(\delta\epsilon/\#G^C) + (1-p)(\epsilon)(\delta/\#G^C) = \delta\epsilon/\#G^C$. Final probability is $D\delta\epsilon$. We're done.

# Derivatives of the main theorem

Note that we didn't assume any particular distributions for $f$ and $h$. By strengthening the assumptions on $f$ we get more specific results. The following we'll need in the Poly1305-AES security proof (we skip the proof):

- $h$ random (not necessarily uniform) with maximum differential probability $\epsilon$, $f$ **uniform random injective** function $\Rightarrow$ chance of success is $D[(1 - C/\#G)^{-(C+1)/2}]\epsilon$ (bracketed part equals $\delta$).

# Poly1305-AES security proof

First, the authors prove the following.

- $h$ random (not necessarily uniform) with maximum differential probability $\epsilon$, $f = AES \Rightarrow$ chance of success (distinguish AES or forgery) is $\beta + D[(1 - C/2^{128})^{-(C+1)/2}]\epsilon$, where $\beta$ is the probability of distinguishing AES.

Note that the criterion for success is now *either* that we distinguish AES or that we get a successful forgery. AES is modelled (ideally) as a uniform random injective function.

(AES is not special; Poly1305-XYZ works with suitable XYZ.)

# Poly1305-AES security proof ...

Finally, we consider the concrete functions involved in Poly1305-AES:

- $h(m) = h_r(m)$ as defined in Poly1305-AES paper (polynomial defined by message, evaluated at additional key $r$), $f = AES$, simulates uniform random injective function $\Rightarrow$ $h$ has small differential probabilities, $\epsilon \leq 8\lceil L/16 \rceil / 2^{106}$ where $L$ is (maximum) length of message (separate proof). Chance of success is at most $\beta + D[(1 - C/2^{128})^{-(C+1)/2}][8\lceil L/16 \rceil / 2^{106}]$. In particular, if $C \leq 2^{64}$, then chance of success is at most $\beta + 14D\lceil L/16 \rceil / 2^{106}$.

The first bracketed part is (a bound for) $\delta$ and the second is (a bound for) $\epsilon$. The Poly1305-AES paper contains a bound on the differential probabilities of $h(m) = h_r(m)$, which is one key ingredient in the proof. (Due to time constraints we have to skip the proof.)

# Review of security proof

- $(n, m) \to h(m) + f(n)$ secure if $h$ has small differential probabilities and $f$ has small interpolation probabilities. Assume $C$ oracle queries and $D$ forgery attempts in what follows.

- $h$ random (not necessarily uniform) with maximum differential probability $\epsilon$, $f$ random (not necessarily uniform) with maximum $C$-interpolation probability $\delta/\#G^C$, $C+1$-interpolation probability $\delta\epsilon/\#G^C$, $h$ and $f$ independent $\Rightarrow$ chance of success is $D\delta\epsilon$.

- $h$ random (not necessarily uniform) with maximum differential probability $\epsilon$, $f$ **uniform random injective** function $\Rightarrow$ chance of success is $D(1 - C/\#G)^{-(C+1)/2}\epsilon$.

# Review of security proof ...

- $h$ random (not necessarily uniform) with maximum differential probability $\epsilon$, $f = AES$, simulates uniform random injective function $\Rightarrow$ chance of success (distinguish AES or forgery) is $\beta + D(1 - C/2^{128})^{-(C+1)/2}\epsilon$, where $\beta$ is the probability of distinguishing AES.

# Review of security proof ...

- $h(m) = h_r(m)$ as defined in Poly1305-AES paper (polynomial defined by message, evaluated at additional key $r$), $f = AES$, simulates uniform random injective function $\Rightarrow$ proved that $h$ has small differential probabilities, $\epsilon \leq 8\lceil L/16\rceil/2^{106}$ where $L$ is (maximum) length of message. Chance of success is at most $\beta + D(1 - C/2^{128})^{-(C+1)/2} 8\lceil L/16\rceil/2^{106}$. In particular, if $C \leq 2^{64}$, then chance of success is at most $\beta + 14D\lceil L/16\rceil/2^{106}$.

- Note that the special form of $r$ is not required for the proof; it's to make the implementation easier.

- Example (IPsec): $L \leq 65536 \Rightarrow \beta + 14D2^{12}/2^{106} < \beta + D/2^{90}$. Assume $2^{32}$ forgery attempts, then total probability of success less than $\beta + 1/2^{58}$.

# Poly1305-AES implementation

# Poly1305-AES implementation

The author describe an implementation based on x86 floating point (!) arithmetic. A few key facts about the implementation:

- Precomputation (key schedule or similar) not necessary
- The special form of $r$ helps in doing floating point carries of a "multipart" representation $x = x_0 + x_1 + x_2 + x_3$
- 1024-byte message and code in cache $\Rightarrow$ about 4-5 cycles / byte
- 1600MHz AMD Duron can handle 3 gbps (384 MB/s) of 1500-byte messages
- Comparison: 1600MHz Athlon XP, OpenSSL HMAC-MD5 $\Rightarrow$ 1.7 gpbs (216 MB/s) of 1024-byte messages

# Summary

$$\boxed{\text{Poly1305-AES}_{(k,r)}(n,m) = h_r(m) + \text{AES}_k(n) \pmod{2^{128}}}$$

- Poly1305-AES is a fast MAC with a security proof.
- AES can be replaced with another cipher should AES break.
- Security proof is based on modelling *interpolation probabilities* of $f$ and *differential probabilities* of $h$.

Thank you!