Universal Hash Functions are Not so Universal

or why efficient provable security without any assumption may sometimes be too good to be true

Bart Preneel

Katholieke Universiteit Leuven – COSIC

`bartDOTpreneel(AT)esatDOTkuleuvenDOTbe`

`http://homes.esat.kuleuven.be/∼preneel`

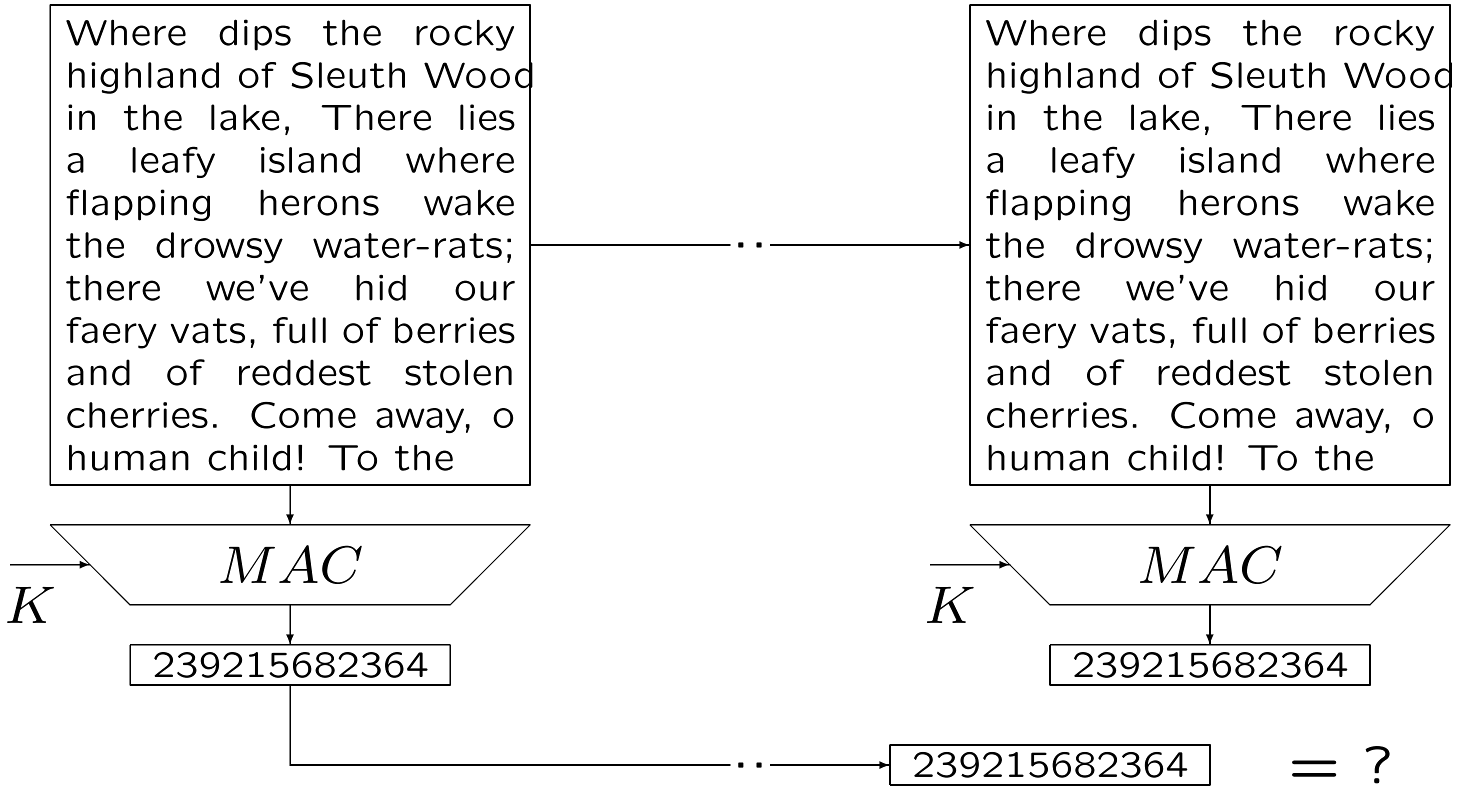
joint work with Helena Handschuh, Spansion

January 2008

Outline

1. definitions
2. attack principles
3. application to some examples
4. concluding remarks

MAC Algorithm


Where dips the rocky highland of Sleuth Wood in the lake, There lies a leafy island where flapping herons wake the drowsy water-rats; there we've hid our faery vats, full of berries and of reddest stolen cherries. Come away, o human child! To the
Where dips the rocky highland of Sleuth Wood in the lake, There lies a leafy island where flapping herons wake the drowsy water-rats; there we've hid our faery vats, full of berries and of reddest stolen cherries. Come away, o human child! To the
MAC
K
MAC
K
239215682364
239215682364
239215682364
= ?

MAC: definition

- Key generation algorithm
- MAC generation algorithm: stateful, can be randomized
- MAC verification algorithm: memoryless, deterministic

A MAC is $(\epsilon, t, q, q', q'', L)$ secure if, an adversary who does not know K, and

- can spend time t (operations),
- can obtain the MAC for q texts of his choice,
- can observe the MAC for q' texts (not of his choice),
- and can obtain the result of q'' verification queries on text-MAC pairs of his choice.

(each text of length L),
cannot produce an existential forgery with probability of success larger than ϵ.

Cryptanalysis

key length: k bits, MAC length m bits; internal memory n bits

notation: work – known texts – chosen texts – on-line verifications

key search: $[2^k, k/m, 0, 0]$ or $[2^k, 0, 0, 2^k]$ or
$[2^{2k/3}, 0, k/m, 0]$ with 2^k offline work

guess MAC: $[0, 0, 0, 2^m]$ or $[2^k, 0, 0, 2^k]$ (but not verifiable)

birthday forgery for iterated MAC algorithms with n-bit int. memory:
$[0, 2^{n/2}, \min(2^{n/2}, 2^{n-m}), 0]$

shortcut attacks: e.g., RFC 1828 (envelope MAC), ANSI Retail MAC

Information-theoretic authentication

authentication codes (AC)/universal hash functions
[1970s (Zobrist/Simmons/Carter-Wegman)]

- advantages
  - provably secure: only combinatorial
  - extremely fast (10-15 times faster than AES/HMAC)
  - parallelizable and incremental
- disadvantages
  - use key only once
  - sometimes very large keys
  - security level in bits against forgery is at most half the key size

Information-theoretic authentication

[Black-Halevi-Krawczyk-Krovetz-Rogaway99]

[...] "since the combinatorial property of the universal hash-function family is mathematically proven (making no cryptographic hardness assumptions), it needs no "over-design" or "safety margin" the way a cryptographic primitive would. Quite the opposite: the [UMAC] hash-function family might as well be the fastest, simplest thing that one can prove universal."

Example: polynomial authentication code

(Change of notation: k is key rather than key size in bits)

- key k', $k \in GF(2^n)$
- split x into x_1, x_2, . . . , x_t, with $x_i \in GF(2^n)$
- note $\ell = t \cdot n$

$$g(x) = k' + \sum_{i=1}^{t} x_i \cdot k^i$$

Pr(success of forgery after seeing 1 text/MAC pair) $= (\ell/n)/2^n = t/2^n$

In practice: value k can be reused

Step 1: Compress

family of functions $g_k : A \longrightarrow B$ with $a = |A|$ and $b = |B|$.
$B, \star$ an Abelian group
Let ϵ be any positive real number.

g_k is an **ϵ-almost universal** class (or $\epsilon - AU$ class) $\mathcal{G}$ of hash functions if $\forall x, x' \neq \in A$

$$\Pr_k \left\{ g_k(x) = g_k(x') \right\} \leq \epsilon \, .$$

g_k is an **ϵ-almost $\star$ universal** class (or ϵ-A$\star$U class) G of hash functions if $\forall x, x' \neq x \in A$ and $\forall \Delta \in B$

$$\Pr_k \left\{ g_k(x) = g_k(x') \star \Delta \right\} \leq \epsilon \, .$$

Step 1: Compress (2)

functions that are ϵ-AU

- $g_k(x) = \sum_{i=0}^{t} x_i \cdot k^i$ with k, $x_i \in$ GF(2^r) or GF(p)

functions that are ϵ-A$\star$U

- $g_k(x) = \sum_{i=1}^{t} x_i \cdot k^i$ with k, $x_i \in$ GF(2^r) or GF(p)
- MMH: $g_k(x) = \left(\sum_{i=1}^{t} x_i \cdot k_i\right) \bmod p$
  x_i, k_i, $\in \mathbf{Z}_{2^{32}}$ and $p = 2^{32} + 15$ (inner sum mod 2^{64}) [Halevi-Krawczyk97]
- NMH: $g_k(x) = \left(\sum_{i=1}^{t/2} (x_{2i-1} + k_{2i-1}) \cdot (x_{2i} + k_{2i})\right) \bmod p$
  x_i, $k_i \in \mathbf{Z}_{2^{32}}$ and $p = 2^{32} + 15$ [Wegman-Carter81 and Halevi-Krawczyk97]
- NH: $g_k(x) = \left(\sum_{i=1}^{t/2} ((x_{2i-1} + k_{2i-1}) \bmod 2^w) \cdot ((x_{2i} + k_{2i}) \bmod 2^w)\right) \bmod 2^{2w}$
  x_i, $k_i \in \mathbf{Z}_{2^w}$ [BHKKR99]
- WH: $g_k(x) = \left(\sum_{i=1}^{t/2} (x_{2i-1} + k_{2i-1}) \cdot (x_{2i} + k_{2i}) x^{(t/2-i)w}\right) \bmod p(x)$
  x_i, $k_i \in$ GF(2^w) (polynomials) [Kaps-Yüksel-Sunar04]

Step 2: Replace addition $k'+$

pseudorandom function family $f_{k'}$ $\quad \rightsquigarrow$ computational security

Option 1: $\text{MAC}_{k||k'}(x) = f_{k'}(g_k(x))$ with g ϵ-AU

Option 2: $\text{MAC}_{k||k'}(x) = f_{k'}(n) \star g_k(x)$ with g ϵ-A$\star$U
need nonce but better security

Option 3: $\text{MAC}_{k||k'}(x) = f_{k'}(n||g_k(x))$ with g ϵ-AU
need nonce and larger input of f

Observations on universal hash functions for MAC

1. reuse of k is common in practice, in particular if k is large
2. nonce n is supposed to be unique; what if it isn't?
   - nonce is always used twice (generation/verification)
   - nonce reuse for verification (e.g., if random numbers)
   - nonce reuse for MAC generation more problematic
3. weak keys: $\forall x, x', x' \neq x \ \Pr_k \{g_k(x) = g_k(x') \star \Delta\} \leq \epsilon$ does not imply that $\forall k \ \Pr_{x,x'} \{g_k(x) = g_k(x') \star \Delta\} \approx 1/|B|$. For some keys $\overline{k}$
$$\Pr_{x,x'} \left\{ g_{\overline{k}}(x) = g_{\overline{k}}(x') \star \Delta \right\} \gg 1/|B|$$
4. simple combinatorial scheme: partial knowledge of k may be devastating (e.g., verification by leaking first two key bytes)
5. messages of special form reduce hash value to subspace

Related Work

[Coppersmith96] Finding a small root of a bivariate integer equation; factoring with high bits known.

[Bellare-Goldwasser97] Verifiable partial key escrow

[McGrew-Fluhrer05] Multiple forgery attacks

[Preneel-vanOorschot-Knudsen96] Key recovery attacks on ANSI retail MAC

[Blackburn-Paterson04] Cryptanalysis of a MAC due to Cary and Venkatesan

[Black-Cochran 06] Focus on reforgeability

UMAC RFC 4418 contains warnings:

- be careful if too many wrong MAC values
- “once an attempted forgery is successful, it is possible, in principle, that subsequent messages under this key may be easily forged. This is important to understand in gauging the severity of a successful forgery, **even though no such attack on UMAC is known to date.**”

Even two appendices full of warnings for GCM NIST SP 800-38D

Polynomial hash

$g_k(x) = \sum_{i=1}^{t} x_i \cdot k^i$ with k, $x_i \in \mathsf{GF}(2^n)$

GCM NIST SP-800 38-D:

- Option 2: $\mathsf{MAC}_{k||k'}(x) = \mathsf{trunc}_\tau\left(\mathsf{AES}_{k'}(n) \oplus g_k(x)\right)$
- $k = \mathsf{AES}_{k'}(000\ldots00)$

trivial weak key: $k = 0$ (extremely unlikely)

trivial to verify a guess for k, even if we do not know k' (ok for GCM)

if order of k divides $l < t$: swap of two blocks leaves $g_k(x)$ unchanged

[Joux06] attack on GCM (nonce reuse by sender):
compute $\mathsf{MAC}_{k||k'}(x) \oplus \mathsf{MAC}_{k||k'}(x') = g_k(x) \oplus g_k(x') = g_k^*(x, x')$
k is one of the t roots of the polynomial g_k^*

Polynomial hash (2)

Joux: *"replacing the counter encryption for MACs by the classical encryption with the block cipher usually used with Wegman-Carter MACs seems a safe option."*

Variant on Joux attack that works even for Option 1 or 3
(more expensive, same cost as forgery, but no nonce reuse):

- obtain 1 MAC value for a text x of your choice
- choose x' such that the polynomial with coefficients from $x - x'$ has t distinct roots (each time)
- perform a MAC verification query for x'
- after $2^n/t$ trials you know that k is one of t values
- perform another t MAC verification queries to find out which one of the t

easy to take into account any information you may have on k.

Polynomial hash (3)

special message attack [Ferguson05]

- $x_i = 0$ except if $i = 2^j$ for some j: $g_k(x) = \sum_{j=0}^{l} x_{2^j} \cdot k^{2^j}$
- squaring is a linear operation in GF(2^n)
  hence we can write the bits of the hash as follows:

$$g_k(x)[.] = \sum_{i*,j*,u*} x_{i*}[j*] \cdot k[u*]$$

- choose x, x' such that trunc$_s(g_k(x))$ = trunc$_s(g_k(x'))$ ($1 \leq s < \tau$), independent of the value of k
- submit x' for verification (same nonce); success prob. $1/2^{\tau-s}$
- collect $r \cdot 2^{\tau-s}$ messages (same nonce!) resulting in r forgeries; each forgery yields $\tau - s$ linear equations in the key bits.

response by NIST: be careful with wrong MAC values; no error messages

Polynomial hash (4)

variant of Ferguson's attack
extends to Option 1 and 3 (but nonce reuse for Option 3)

- same special messages
- guess the linear combination of s ($1 \leq s < n$) bits of the key k, we can generate a set of λ message for which the hash result is restricted to a subspace of size 2^{n-s}.
- collect $\lambda = 2^{(n-s)/2+1}$ messages (same nonce!) resulting in 2 collisions; each collision yields $n-s$ linear equations for the remaining $n-s$ key bits.

example: $n = 64$, $s = 24$, $n - s = 40$
2^{21} messages yield 2 collisons, which gives 80 linear equations in the remaining 40 key bits.

Bucket hashing with short key [Johansson97]

$g_k(x) = \sum_{j=1}^{r} \sum_{i=1}^{t} x_{ij} \cdot k_j^i$ with $x_{ij} \in \mathrm{GF}(2)$, $k_j \in \mathrm{GF}(2^n)$

input size: $m = r \cdot t$ bits; key size: $r \cdot n$ bits, $\epsilon = t/2^n$

weak keys: if $k_j = 0$, $g_k(x)$ independent of t message bits (prob. 2^{-n}).
fraction of weak keys $\approx r/2^n$

simple key recovery more complex as coefficients are bits.

still easy to test an n-bit key value k_{j*}: make a modification to the values x_{ij*} only (but k_{j*} is only a very small part of a large key).

example: $n = 40$, $t = 512$, $r = 16$ (4 rows of 1024 buckets)
input length = $m = 2^{13}$ bits, key size = 640 bits, output = 40 bits,
$\epsilon = 2^{-31}$.

Bucket hashing with short key with nonce reuse (sender)

$$g_k(x) = \sum_{j=1}^{r} \sum_{i=1}^{t} x_{ij} \cdot k_j^i \text{ with } x_{ij} \in \mathsf{GF}(2),\ k_j \in \mathsf{GF}(2^n)$$

special message attack

- set $x_{ij} = 0$ except for i a power of 2 less than t.
- then $g_k(x)[.] = \sum_{i*,j*,u*,v*} x_{i*}[j*] \cdot k_{u*}[v*]$
- collect $2^{(n+1)/2}\sqrt{r}$ messages (same nonce!) resulting in r collisions; each collision yields n linear equations for the key bits.

example: $n = 40$, $t = 512$, $r = 16$
2^{23} messages yield 32 collisions, which gives 1280 linear equations in the 640 key bits.

MMH [Halevi-Krawczyk97]

$g_k(x) = \left(\sum_{i=1}^{t}\ x_i \cdot k_i\right)$ mod p
$x_i,\ k_i,\ \in \mathbf{Z}_{2^{32}}$ and $p = 2^{32} + 15$ (inner sum mod 2^{64})

attacks (consider $t = 2$):

- $k_1 = k_2 = 0$ (trivial): all messages map to 0.
- assume $k_1 = \alpha \cdot k_2$ mod 2^{64} for $\alpha \in \mathbf{Z}_{2^{32}}$; then $g_k(x)$ unchanged if one replaces (x_1, x_2) by $(x'_1, (x_2 + (x_1 - x'_1) \cdot \alpha)$ mod $2^{64})$
- recover 2 key words with $2 \cdot 2^{32}$ verification queries
- easy to exploit partial key information $x_1 k_1 + x_2 k_2 = x'_1 k_1 + x'_2 k_2$

Square Hash [Etzel-Patel-Ramzan99]

$g_k(x) = \sum_{i=1}^{t} (x_i + k_i)^2 \bmod p$ with $x_i,\ k_i, \in \mathbf{Z}_{2^w}$

weak keys (consider $t = 2$):

- $k_1 = k_2 \bmod p$: message with x_1 and x_2 swapped collide
- confirm guess for any key word (w bits) with one verification query:
  change x_i into $x_i' = (-2k_i - x_i) \bmod p$
  easy to take into account any information you may have on k_i

preclude 2nd attack by $x_i < p/2$

NH – UMAC [BHKKR99] – VMAC – WMAC

$$g_k(x) = \left(\sum_{i=1}^{t/2} ((x_{2i-1} + k_{2i-1}) \bmod 2^w) \cdot ((x_{2i} + k_{2i}) \bmod 2^w) \right) \bmod 2^{2w},$$

$x_i,\ k_i \in \mathbf{Z}_{2^w}$

- if $((x_{2i-1} + k_{2i-1}) \bmod 2^w) = 0$, $g_k(x)$ is independent of x_{2i}.
- assume that $k_{2i} = k_{2i-1} + \Delta \bmod 2^w$ for $\Delta \in \mathbf{Z}_{2^w}$,
  then $g_k(x)$ unchanged if one replaces
  (x_{2i-1}, x_{2i}) by $(x_{2i} + \Delta \bmod 2^w, x_{2i-1} - \Delta \bmod 2^w)$
- recover t key words with $t \cdot 2^w$ verification queries
- with an oracle for s bits of one key word, expected complexity to find one word reduces to 2^{w-s} verification queries.

NH – UMAC [BHKKR99] – VMAC – WMAC (2)

UMAC: $w = 32$ marginal; ok if output length of 64, 96 or 128 bits.
(earlier version had $w = 16$)

VMAC: $w = 64$ (probably ok)

WMAC: variant with $w = 16$, 32 and 64

UMAC RFC 4418 [March 2006]
"It should be pointed out that once an attempted forgery is successful, it is possible, in principle, that subsequent messages under this key may be easily forged. This is important to understand in gauging the severity of a successful forgery, even though no such attack on UMAC is known to date."

Conclusions

While universal hash functions have very attractive performance and provable security, they can be very brittle in practice

- avoid reusing keys (Snow3G is good example)
- sender/verifier needs to guarantee/check uniqueness of nonces
- vulnerable to oracle that reveals part of the key (and thus to side-channel attacks)

Some schemes are more secure than others. . .

EMAC based on AES is slower but more "robust"

- internal collisions (2^{64} texts) lead to forgeries, but not to key recovery
- no known way to use an oracle that gives access to 32 or 64 key bits
- faster key setup

Ongoing: other universal hash functions, including improved attack on MAC of Cary and Venkatesan

Summary

	number of weak keys	verify key guess	partial key information
Polynomial hash GF(2^n)	> 1	k only	yes
Polynomial hash GF(p)	> 1	k only	?
Bucket hashing w/ small key	$\approx r \cdot 2^{(r-1)n}$	subkey k_j	yes
MMH	$\approx t \cdot 2^{(t-1)w}$	w-bit subkey k_i	yes
Square Hash	$p^t - p!/(p-t)!$	w-bit subkey k_i	yes
NMH/NH/WH	$-$	w-bit subkey k_i	yes