Revisiting MAC Forgeries, Weak Keys and Provable Security of GCM

Bo Zhu, Yin Tan and Guang Gong
University of Waterloo, Canada

CANS 2013
Nov 20, 2013

Galois/Counter Mode (GCM)

- One design of AEAD by McGrew and Viega in 2005
  - Counter Mode (CM) for encryption
  - Galois MAC (GMAC) for authentication
    - Polynomial-based MAC
- Features
  - Parallelizable computation
  - Intel CPU hardware instructions (around 1 cycle/byte)
  - IEEE 802.1AE, IPsec, and TLS v1.2
  - To replace RC4 and AES-CBC in TLS

Galois/Counter Mode (GCM)

- One design of AEAD by McGrew and Viega in 2005
  - Counter Mode (CM) for encryption
  - Galois MAC (GMAC) for authentication
    - Polynomial-based MAC
- Features
  - Parallelizable computation
  - Intel CPU hardware instructions (around 1 cycle/byte)
  - IEEE 802.1AE, IPsec, and TLS v1.2
  - To replace RC4 and AES-CBC in TLS
- Recent attacks
  - A flaw found in GCM's security proofs in Crypto'12
  - Forgery attacks in FSE'12 and FSE'13

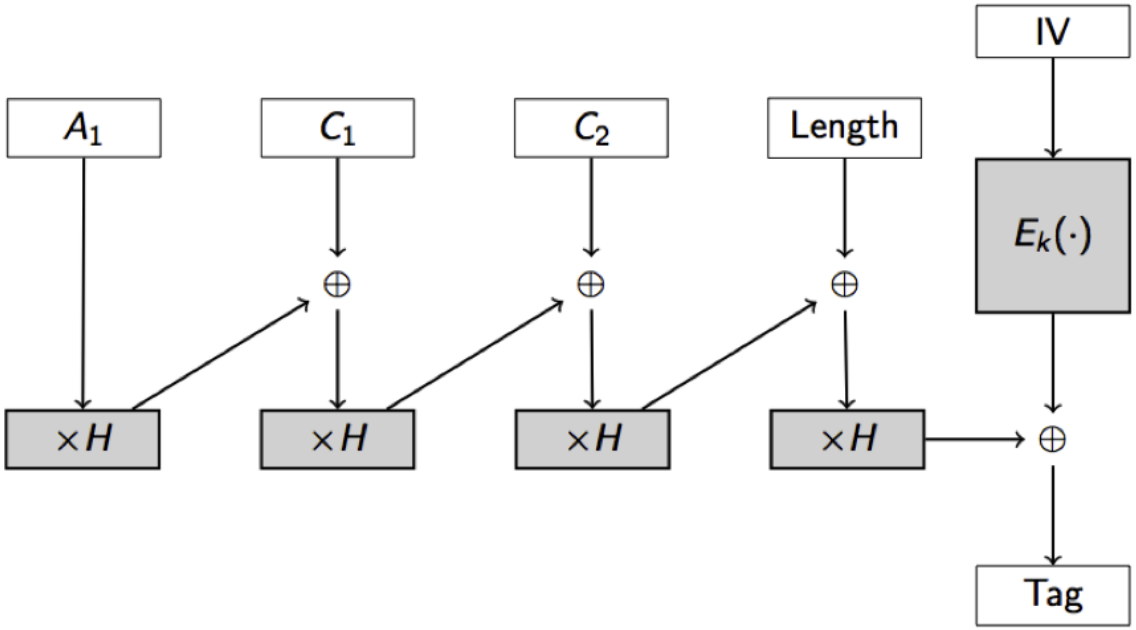
Outline

Outline

Authentication by Galois MAC (GMAC)

Additions and multiplications in $GF(2^{128})$

- Authentication key: $H = E_K(0)$

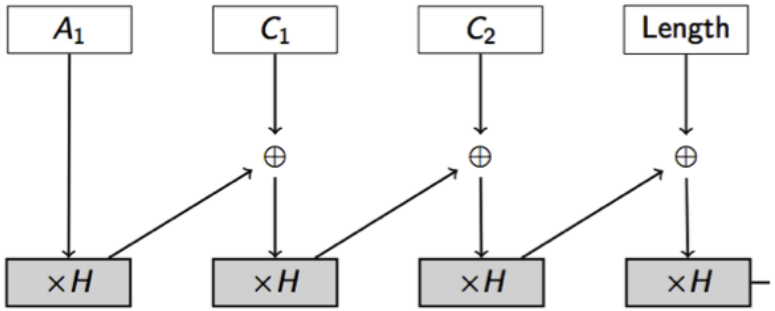


The image is from Procter and Cid's slides in FSE'13.

Polynomial Based GHASH

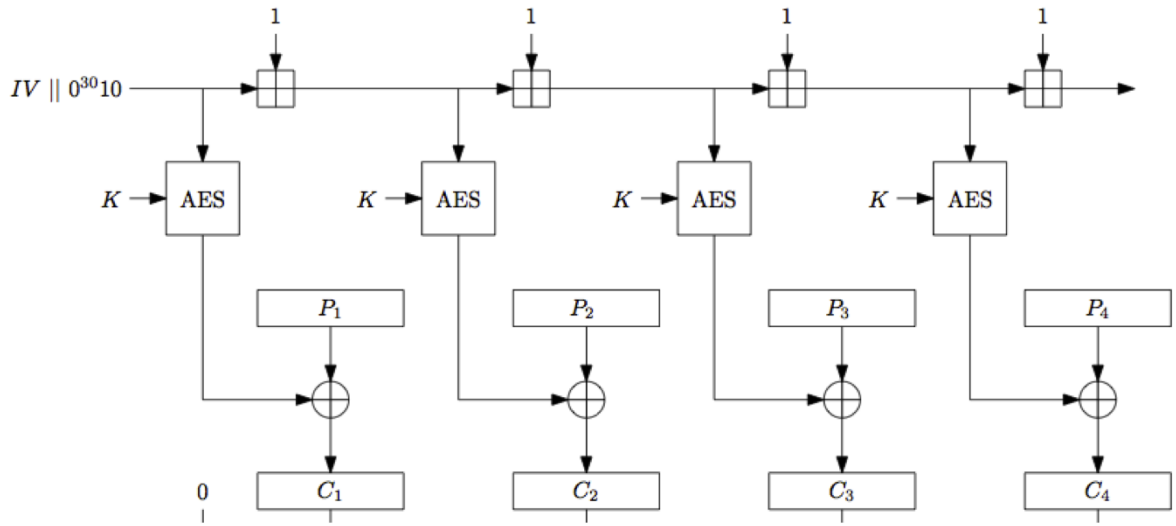
- $GMAC = GHASH_H(A, C) + E_K(N)$
  - N: non-repeating nonce
- GHASH-like, polynomial based (keyed) hash

$$h_H(M) = \sum_{i=1}^{m} M_i \times H^i = g_M(H)$$



- Note: constant term is zero

Encryption in Counter Mode (CM)



The image is from Saarinen's paper in FSE'12.

- Initial counter
  - If $len(N) = 96$, $Y_0 = N||0^{32}$
  - If $len(N) \neq 96$, $Y_0 = GHASH_H(N)$
- Consecutive counters

$$Y_{r+1} = msb_{96}(Y_r)||lsb_{32}(Y_r) \boxplus 1$$

Outline

Forgery Attacks on Polynomial-based MACs

- General forgeries by Procter and Cid in FSE'13
  - Based on the work by Saarinen in FSE'12
- Attacking the polynomial-based hash functions

$$h_H(M) = \sum_{i=1}^{m} M_i \times H^i = g_M(H)$$

Forgery Attacks on Polynomial-based MACs

- General forgeries by Procter and Cid in FSE'13
  - Based on the work by Saarinen in FSE'12
- Attacking the polynomial-based hash functions

$$h_H(M) = \sum_{i=1}^{m} M_i \times H^i = g_M(H)$$

- If we can find a polynomial $f(x) \in \mathbb{F}[x]$
  - Constant term is zero
  - $f(H) = 0$

Forgery Attacks on Polynomial-based MACs

- General forgeries by Procter and Cid in FSE'13
  - Based on the work by Saarinen in FSE'12
- Attacking the polynomial-based hash functions

$$h_H(M) = \sum_{i=1}^{m} M_i \times H^i = g_M(H)$$

- If we can find a polynomial $f(x) \in \mathbb{F}[x]$
  - Constant term is zero
  - $f(H) = 0$

  then

$$h_H(M \oplus F) = g_{M \oplus F}(H) = g_M(H) \oplus g_F(H) = g_M(H) = h_H(M)$$

Our Generalized Forgery Attack on GCM-like Modes

For GMAC-like MACs, the MAC tag is computed as

$$T = h_H(M) \oplus E_K(N)$$

Our Generalized Forgery Attack on GCM-like Modes

For GMAC-like MACs, the MAC tag is computed as

$$T = h_H(M) \oplus E_K(N)$$

If we find a polynomial $q(x) = q^*(x) \oplus Q_0 \in \mathbb{F}[x]$ such that

- $q(H) = 0$
- Note: constant term Q_0 does NOT need to be zero

Our Generalized Forgery Attack on GCM-like Modes

For GMAC-like MACs, the MAC tag is computed as

$$T = h_H(M) \oplus E_K(N)$$

If we find a polynomial $q(x) = q^*(x) \oplus Q_0 \in \mathbb{F}[x]$ such that

- $q(H) = 0$
- Note: constant term Q_0 does NOT need to be zero

then

$$T = h_H(M) \oplus E_k(N) \oplus q(H),$$

Our Generalized Forgery Attack on GCM-like Modes

For GMAC-like MACs, the MAC tag is computed as

$$T = h_H(M) \oplus E_K(N)$$

If we find a polynomial $q(x) = q^*(x) \oplus Q_0 \in \mathbb{F}[x]$ such that

- $q(H) = 0$
- Note: constant term Q_0 does NOT need to be zero

then

$$T = h_H(M) \oplus E_k(N) \oplus q(H),$$

which implies

$$\begin{aligned} T \oplus Q_0 = & \quad E_k(N) \oplus h_H(M) \oplus q^*(H) \\ = & \quad E_k(N) \oplus g_M(H) \oplus q^*(H) \\ = & \quad E_k(N) \oplus g_{M \oplus Q^*}(H). \end{aligned}$$

So $(N, M \oplus Q^*, T \oplus Q_0)$ is a successful forgery.

All Subsets with ≥ 2 Auth Keys are Weak

- Definition of weak key classes by Handschuh and Preneel
  - Members of the key class make the algorithm behaves in an unexpected way
    - e.g., high probability for MAC forgeries
  - Easy to detect whether a key belongs to the class
    - e.g., less #queries than #elements of the class

All Subsets with ≥ 2 Auth Keys are Weak

- Definition of weak key classes by Handschuh and Preneel
  - Members of the key class make the algorithm behaves in an unexpected way
    - e.g., high probability for MAC forgeries
  - Easy to detect whether a key belongs to the class
    - e.g., less #queries than #elements of the class
- For any subset of authentication keys, we can determine if the used key is in the subset
  - Try to make a forgery by

$$q(x) = \prod_{i=1}^{n} (x \oplus H_i)$$

  and query the verification oracle once

All Subsets with ≥ 2 Auth Keys are Weak

- Definition of weak key classes by Handschuh and Preneel
  - Members of the key class make the algorithm behaves in an unexpected way
    - e.g., high probability for MAC forgeries
  - Easy to detect whether a key belongs to the class
    - e.g., less #queries than #elements of the class
- For any subset of authentication keys, we can determine if the used key is in the subset
  - Try to make a forgery by

$$q(x) = \prod_{i=1}^{n} (x \oplus H_i)$$

    and query the verification oracle once
- For comparison, the original forgery attack by Procter and Cid
  - Cannot get rid of 0 by only one query
  - For $|S| \geq 3$, use two queries
  - For $|S| \geq 2$ and $0 \in S$, use one query

Outline

Birthday-bound based Forgery Attacks

- Previously mentioned forgery attacks are all trial-and-error
  - (Perhaps randomly) choose a $q(x)$
  - Forge a tuple (N, M, T) and send it to verification oracle
  - If fails, try another $q(x)$

Birthday-bound based Forgery Attacks

- Previously mentioned forgery attacks are all trial-and-error
  - (Perhaps randomly) choose a $q(x)$
  - Forge a tuple (N, M, T) and send it to verification oracle
  - If fails, try another $q(x)$
- GCM's special structure can amplify this probability
  - GHASH is reused to compute the initial counter number if $len(N) \neq 96$.
  - Previous forgeries also work for this GHASH

Birthday-bound based Forgery Attacks

- Previously mentioned forgery attacks are all trial-and-error
  - (Perhaps randomly) choose a $q(x)$
  - Forge a tuple (N, M, T) and send it to verification oracle
  - If fails, try another $q(x)$
- GCM's special structure can amplify this probability
  - GHASH is reused to compute the initial counter number if $len(N) \neq 96$.
  - Previous forgeries also work for this GHASH
- New forgery attack
  1. Obtain a valid tuple (N, P, C)
  2. Apply $q(x)$ to N, and feed (N', P) to the encryption oracle
  3. Collect $P \oplus C$ to a set for collisions

Birthday-bound based Forgery Attacks

- Previously mentioned forgery attacks are all trial-and-error
  - (Perhaps randomly) choose a $q(x)$
  - Forge a tuple (N, M, T) and send it to verification oracle
  - If fails, try another $q(x)$
- GCM's special structure can amplify this probability
  - GHASH is reused to compute the initial counter number if $len(N) \neq 96$.
  - Previous forgeries also work for this GHASH
- New forgery attack
  1. Obtain a valid tuple (N, P, C)
  2. Apply $q(x)$ to N, and feed (N', P) to the encryption oracle
  3. Collect $P \oplus C$ to a set for collisions
- Once a collision is found
  - Obtain a polynomial for more forgeries
  - Solve the equation to get the auth key

Outline

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$
  - if $len(N) \neq 96$, $GHASH(N_1) \boxplus r_1 = GHASH(N_2) \boxplus r_2$

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$
  - if $len(N) \neq 96$, $GHASH(N_1) \boxplus r_1 = GHASH(N_2) \boxplus r_2$

$$GHASH(N_1) \boxplus r \quad = \quad GHASH(N_2)$$

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$
  - if $len(N) \neq 96$, $GHASH(N_1) \boxplus r_1 = GHASH(N_2) \boxplus r_2$

$$\begin{aligned} GHASH(N_1) \boxplus r &= GHASH(N_2) \\ h_H(N_1) \boxplus r &= h_H(N_2) \\ g_{N_1}(H) \boxplus r &= g_{N_2}(H) \end{aligned}$$

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$
  - if $len(N) \neq 96$, $GHASH(N_1) \boxplus r_1 = GHASH(N_2) \boxplus r_2$

$$\begin{aligned} GHASH(N_1) \boxplus r &= GHASH(N_2) \\ h_H(N_1) \boxplus r &= h_H(N_2) \\ g_{N_1}(H) \boxplus r &= g_{N_2}(H) \end{aligned}$$

- For a randomly chosen H, the collision probability is

$$\frac{\#\{x : x \in GF(2^{128}) | g_{N_1}(x) \boxplus r = g_{N_2}(x)\}}{2^{128}}$$

Counter Generation in GCM

- Security of GCM highly depends the prob of counter collisions
  - $Y'_{r_1} = Y''_{r_2}$
  - if $len(N) \neq 96$, $GHASH(N_1) \boxplus r_1 = GHASH(N_2) \boxplus r_2$

$$\begin{aligned} GHASH(N_1) \boxplus r &= GHASH(N_2) \\ h_H(N_1) \boxplus r &= h_H(N_2) \\ g_{N_1}(H) \boxplus r &= g_{N_2}(H) \end{aligned}$$

- For a randomly chosen H, the collision probability is

$$\frac{\#\{x : x \in GF(2^{128}) | g_{N_1}(x) \boxplus r = g_{N_2}(x)\}}{2^{128}}$$

- In the original proofs of GCM, it was believed it has the same number of solutions as $g_{N_1}(x) \oplus r = g_{N_2}(x)$:

$$\max\{deg(g_{N_1}(x)), deg(g_{N_2}(x))\} = \max\{len(N_1), len(N_2)\} + 1$$

Problem of $Y_r \boxplus 1$

- Pointed out by Iwata *et al.* in Crypto'12
- $Y_r \boxplus 1$ is non-linear in Galois field
- $$f(x) \boxplus r = g(x)$$

  can be converted to multiple forms of equations in GF

Problem of $Y_r \boxplus 1$

- Pointed out by Iwata *et al.* in Crypto'12
- $Y_r \boxplus 1$ is non-linear in Galois field
- $$f(x) \boxplus r = g(x)$$
  can be converted to multiple forms of equations in GF
- Much more solutions than the expected

$$\max\{len(N_1), len(N_2)\} + 1$$

- α_r times more solutions
  - for $r < 2^{32}$, α_r **is up to** 2^{22}

$$\begin{aligned} &\alpha_r \cdot (\max\{len(N_1), len(N_2)\} + 1) \\ \leq\ &2^{22} \cdot (\max\{len(N_1), len(N_2)\} + 1) \end{aligned}$$

Actual Security Bounds of GCM

- New security bounds of GCM were also given by Iwata *et al.*
  - for both of privacy (encryption) and authenticity (MAC)
  - almost 2^{22} looser than the originally claimed

Actual Security Bounds of GCM

- New security bounds of GCM were also given by Iwata *et al.*
  - for both of privacy (encryption) and authenticity (MAC)
  - almost 2^{22} looser than the originally claimed
- It would be better to repair GCM s.t.
  - retain the original bounds, and
  - leave original proofs largely unchanged
  - with a small fix to the original design

Revisit Counter Mode

- Generally in CM, the counter is incremented by 1, i.e.

$$next(Y) = Y \boxplus 1$$

Revisit Counter Mode

- Generally in CM, the counter is incremented by 1, i.e.

$$next(Y) = Y \boxplus 1$$

- CM is secure if $next()$ outputs uniquely
  - $next()$ is indistinguishable if the underlying block cipher is secure

Revisit Counter Mode

- Generally in CM, the counter is incremented by 1, i.e.

$$next(Y) = Y \boxplus 1$$

- CM is secure if *next*() outputs uniquely
  - *next*() is indistinguishable if the underlying block cipher is secure
- McGrew, Counter Mode Security: Analysis and Recommendations, 2002
  - The details of the next-counter function are unimportant;
  - That function does not provide any security properties other than the uniqueness of the inputs to the block cipher.

Revisit Counter Mode

- Generally in CM, the counter is incremented by 1, i.e.

$$next(Y) = Y \boxplus 1$$

- CM is secure if *next*() outputs uniquely
  - *next*() is indistinguishable if the underlying block cipher is secure
- McGrew, Counter Mode Security: Analysis and Recommendations, 2002
  - The details of the next-counter function are unimportant;
  - That function does not provide any security properties other than the uniqueness of the inputs to the block cipher.
- Design a different *next*() to "fix" GCM?

Requirements of *next*()

1. Cyclic permutation with only one cycle
   - non-repeating

Requirements of *next*()

1. Cyclic permutation with only one cycle
   - non-repeating
2. Number of solutions for

$$next^r(f(x)) = g(x)$$

should be as small as possible compared to

$$\max\{deg(f), deg(g)\}$$

in order to reduce counter collision probability

Requirements of *next*()

1. Cyclic permutation with only one cycle
   - non-repeating
2. Number of solutions for

$$next^r(f(x)) = g(x)$$

   should be as small as possible compared to

$$\max\{deg(f), deg(g)\}$$

   in order to reduce counter collision probability
3. Assume $r_1 \geq r_2$,
   $next^{r_1}(f(x)) = next^{r_2}(g(x)) \Leftrightarrow next^{r_1 - r_2}(f(x)) = g(x)$
   - e.g., $f(x) \boxplus r_1 = g(x) \boxplus r_2 \Leftrightarrow f(x) \boxplus (r_1 - r_2) = g(x)$
   - to keep the original proofs largely unchanged

Design *next*()

Two basic operations that won't increase degrees of $f(x)$ and $g(x)$

- addition, i.e. XOR
  - not a permutation
  - unless defined as $next(Y, r) = Y \oplus r$, with r as an index
  - but $f \oplus r_1 = g \oplus r_2 \implies f \oplus (r_1 - r_2) = g$
    - e.g., $f \oplus 2 = g \oplus 1$ implies
    - $f \oplus 2 \oplus 1 = f \oplus 3 = g$
    - but not $f \oplus (2 - 1) = f \oplus 1 = g$
- multiplication, by a constant
  - multiplying with a primitive element w
  - $w^{r_1} f = w^{r_2} g \implies w^{r_1 - r_2} f = g$
  - cyclic permutation, but with two cycles
    - $\{1, w, w^2, \cdots, w^{2^n - 2}\}$, and $\{0\}$

Merge Two Cycles

To merge $\{1, w, w^2, \cdots, w^{2^n-2}\}$, and $\{0\}$, we define

$$L_w(x) = \begin{cases} 0 & \text{if } x = w^{2^n-2}, \\ 1 & \text{if } x = 0, \\ w \cdot x & \text{otherwise.} \end{cases}$$

Merge Two Cycles

To merge $\{1, w, w^2, \cdots, w^{2^n-2}\}$, and $\{0\}$, we define

$$L_w(x) = \begin{cases} 0 & \text{if } x = w^{2^n-2}, \\ 1 & \text{if } x = 0, \\ w \cdot x & \text{otherwise.} \end{cases}$$

- $L_w(x)$ is a full-cycle permutation
- $L_w^{r_1}(f(x)) = L_w^{r_2}(g(x)) \Leftrightarrow L_w^{r_1-r_2}(f(x)) = g(x)$

Merge Two Cycles

To merge $\{1, w, w^2, \cdots, w^{2^n-2}\}$, and $\{0\}$, we define

$$L_w(x) = \begin{cases} 0 & \text{if } x = w^{2^n-2}, \\ 1 & \text{if } x = 0, \\ w \cdot x & \text{otherwise.} \end{cases}$$

- $L_w(x)$ is a full-cycle permutation
- $L_w^{r_1}(f(x)) = L_w^{r_2}(g(x)) \Leftrightarrow L_w^{r_1-r_2}(f(x)) = g(x)$
- Next, to investigate the number of solutions for

$$L_w^r(f(x)) = g(x)$$

$$L_w^r(f(x)) = g(x)$$

1. If $f(x) = 0$,
   - 1.1 If $L_w^r(f(x)) = 0$, then $g(x) = 0$.
   - 1.2 If $L_w^r(f(x)) \neq 0$, then $g(x) = w^{r-1}$.
2. If $f(x) \neq 0$,
   - 2.1 If $L_w^r(f(x)) = 0$, then $g(x) = 0$.
   - 2.2 If $L_w^r(f(x)) \neq 0$, let $f(x) = w^{r_1}$ and $L_w^r(f(x)) = w^{r_2}$, where $0 \leq r_1, r_2 < 2^n - 1$. Then we have
     - 2.2.1 If $r_1 \leq r_2$, then $w^r f(x) = g(x)$.
     - 2.2.2 If $r_1 > r_2$, then $w^{r-1} f(x) = g(x)$.

$$L_w^r(f(x)) = g(x)$$

1. If $f(x) = 0$,
   - 1.1 If $L_w^r(f(x)) = 0$, then $g(x) = 0$.
   - 1.2 If $L_w^r(f(x)) \neq 0$, then $g(x) = w^{r-1}$.
2. If $f(x) \neq 0$,
   - 2.1 If $L_w^r(f(x)) = 0$, then $g(x) = 0$.
   - 2.2 If $L_w^r(f(x)) \neq 0$, let $f(x) = w^{r_1}$ and $L_w^r(f(x)) = w^{r_2}$, where $0 \leq r_1, r_2 < 2^n - 1$. Then we have
     - 2.2.1 If $r_1 \leq r_2$, then $w^r f(x) = g(x)$.
     - 2.2.2 If $r_1 > r_2$, then $w^{r-1} f(x) = g(x)$.

x must be a root of one of

$$\begin{cases} g(x) = 0, \\ g(x) = w^{r-1}, \\ w^r f(x) = g(x), \\ w^{r-1} f(x) = g(x). \end{cases}$$

So #solutions $\leq 4 \cdot (\max\{deg(f), deg(g)\})$.

LGCM – Revised GCM

- Replacing *counter* $\boxplus 1$ by L_w

$$\begin{aligned} Y_0 &= GHASH_H(N) \\ Y_i &= L_w^i(Y_0) \end{aligned}$$

- The upper bound of counter collision will decrease
  - from $2^{22}d$ to $2^2 d$
- Tighten the bounds of GCM by around 2^{20} (1 million) times
  - both privacy and authenticity

Outline

Attacking GCM in MAC-then-Enc

- The attacks focus on the MAC scheme
- Successful forgeries will render the encryption not trustworthy

Attacking GCM in MAC-then-Enc

- The attacks focus on the MAC scheme
- Successful forgeries will render the encryption not trustworthy
- Previous papers mentioned one straightforward way to fix this
  - Change GCM to MAC-then-Enc
    - Computer T from P
    - Encrypt T by CM encryption

Attacking GCM in MAC-then-Enc

- The attacks focus on the MAC scheme
- Successful forgeries will render the encryption not trustworthy
- Previous papers mentioned one straightforward way to fix this
  - Change GCM to MAC-then-Enc
    - Computer T from P
    - Encrypt T by CM encryption
- However, since the forgery attacks are based the linear structures of GMAC
  - Also penetrate Counter Mode
  - Applying $q(x)$ to C is equivalent to applying to P
  - Please see our paper for details and computational examples

Outline

Summery

We showed

- All polynomials can be used in the forgery attacks
  - With or without constant terms
- All subsets with at least two authentication keys are weak
  - Only one query to verification oracle
- How to turn forgery attacks to birthday attacks
  - Attack encryption oracle instead of verification oracle
  - To increase forgery success probabilities
- A simple fix to GCM
  - Improve security bounds of GCM by a factor of around 2^{20}
  - Keep most parts of security proofs unchanged
- MAC-then-Enc won't make GCM more secure

Thanks for your attention!