# Revisiting Counter Mode to Repair Galois/Counter Mode

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

Aug 12, 2013

# Revisiting Counter Mode to Repair Galois/Counter Mode and Simeck: An Authenticated Cipher Design

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

Aug 12, 2013

# Motivations

- To study existing modes of operations
    - Before designing authenticated ciphers

# Motivations

- To study existing modes of operations
  - Before designing authenticated ciphers
  - Recent attacks on GCM
    - A flaw found in GCM's security proofs in Crypto'12
    - Forgery attacks in FSE'12 and FSE'13

# Motivations

- To study existing modes of operations
  - Before designing authenticated ciphers
  - Recent attacks on GCM
    - A flaw found in GCM's security proofs in Crypto'12
    - Forgery attacks in FSE'12 and FSE'13
- To study lightweight cipher designs
  - To use with mode of operation

# Motivations

- To study existing modes of operations
  - Before designing authenticated ciphers
  - Recent attacks on GCM
    - A flaw found in GCM's security proofs in Crypto'12
    - Forgery attacks in FSE'12 and FSE'13
- To study lightweight cipher designs
  - To use with mode of operation
  - Two block ciphers designed by people from NSA

# Outline

# Outline

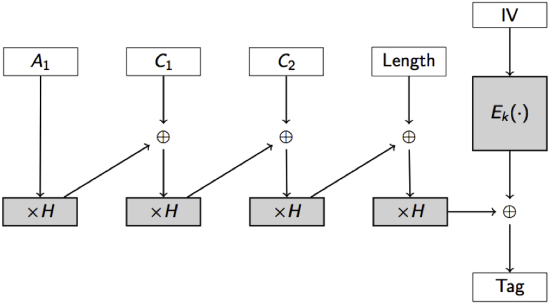
# Galois/Counter Mode (GCM)

- One design of AEAD by McGrew and Viega in 2005
  - Counter Mode (CM) for encryption
  - Galois MAC (GMAC) for authentication
- GCM comparing to CCM (CM + CBC-MAC)
  - Less popular than CCM for historical reasons
    - Supported by OpenSSH from v6.2 (March 2013)
  - Incluced in NSA Suite B (CCM isn't in)
    - Suite A is classified
  - Parallelizable computation

# Authentication by Galois MAC (GMAC)

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

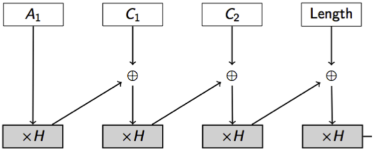
- Authentication key: $H = E_K(0)$




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

# Polynomial Based GHASH

- $GMAC = GHASH(H, A, C) + E_K(IV)$

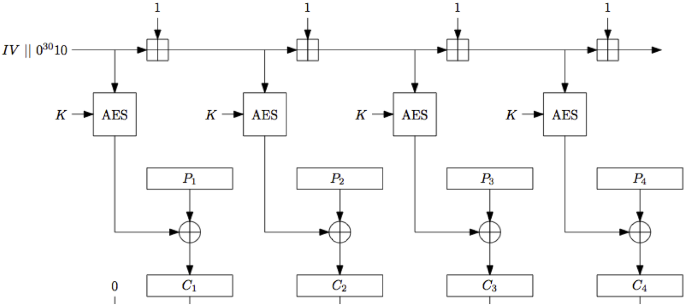



- GHASH

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

- Note: constant term is zero

# Encryption in Counter Mode (CM)




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

# Counter Generation

- Initial counter
  - $N_0 = IV||0^{32}$, if $len(IV) = 96$
  - $N_0 = GHASH_H(IV)$, if $len(IV) \neq 96$

# Counter Generation

- Initial counter
  - $N_0 = IV||0^{32}$, if $len(IV) = 96$
  - $N_0 = GHASH_H(IV)$, if $len(IV) \neq 96$
- Generating counters

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

# Counter Generation

- Initial counter
  - $N_0 = IV||0^{32}$, if $len(IV) = 96$
  - $N_0 = GHASH_H(IV)$, if $len(IV) \neq 96$
- Generating counters

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

- Security of GCM highly depends the prob of counter collisions
  - $N'_0 = N''_0$,
    $N'_{r_1} = N''_{r_2}$

# Counter Generation

- Initial counter
  - $N_0 = IV||0^{32}$, if $len(IV) = 96$
  - $N_0 = GHASH_H(IV)$, if $len(IV) \neq 96$
- Generating counters

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

- Security of GCM highly depends the prob of counter collisions
  - $N'_0 = N''_0$,
    $N'_{r_1} = N''_{r_2}$
  - if $len(IV) \neq 96$,
    $GHASH(IV_1) = GHASH(IV_2)$,
    $GHASH(IV_1) \boxplus r_1 = GHASH(IV_2) \boxplus r_2$

# Counter Generation

- Initial counter
  - $N_0 = IV||0^{32}$, if $len(IV) = 96$
  - $N_0 = GHASH_H(IV)$, if $len(IV) \neq 96$
- Generating counters

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

- Security of GCM highly depends the prob of counter collisions
  - $N_0' = N_0''$,
    $N_{r_1}' = N_{r_2}''$
  - if $len(IV) \neq 96$,
    $GHASH(IV_1) = GHASH(IV_2)$,
    $GHASH(IV_1) \boxplus r_1 = GHASH(IV_2) \boxplus r_2$
  - $GHASH(IV_1) \boxplus (r_1 - r_2) = GHASH(IV_2)$

# Counter Generation (Cont.)

$$\begin{aligned} GHASH(IV_1) \boxplus r &= GHASH(IV_2) \\ h_H(IV_1) \boxplus r &= h_H(IV_2) \end{aligned}$$

# Counter Generation (Cont.)

$$\begin{aligned} GHASH(IV_1) \boxplus r &= GHASH(IV_2) \\ h_H(IV_1) \boxplus r &= h_H(IV_2) \\ g_{IV_1}(H) \boxplus r &= g_{IV_2}(H) \end{aligned}$$

# Counter Generation (Cont.)

$$
\begin{aligned}
GHASH(IV_1) \boxplus r &= GHASH(IV_2) \\
h_H(IV_1) \boxplus r &= h_H(IV_2) \\
g_{IV_1}(H) \boxplus r &= g_{IV_2}(H)
\end{aligned}
$$

- For a randomly chosen $H$, the collision prob is

$$
\frac{\#\{x : x \in GF(2^{128}) | g_{IV1}(x) \boxplus r = g_{IV2}(x)\}}{2^{128}}
$$

# Counter Generation (Cont.)

$$\begin{aligned} GHASH(IV_1) \boxplus r &= GHASH(IV_2) \\ h_H(IV_1) \boxplus r &= h_H(IV_2) \\ g_{IV_1}(H) \boxplus r &= g_{IV_2}(H) \end{aligned}$$

- For a randomly chosen $H$, the collision prob is

$$\frac{\#\{x : x \in GF(2^{128}) | g_{IV1}(x) \boxplus r = g_{IV2}(x)\}}{2^{128}}$$

- In the original security proofs of GCM, it was believed

$$g_{IV_1}(x) \boxplus r = g_{IV_2}(x)$$

has the same number of solutions as

$$g_{IV_1}(x) \oplus r = g_{IV_2}(x)$$

# Counter Generation (Cont.)

$$\begin{aligned} GHASH(IV_1) \boxplus r &= GHASH(IV_2) \\ h_H(IV_1) \boxplus r &= h_H(IV_2) \\ g_{IV_1}(H) \boxplus r &= g_{IV_2}(H) \end{aligned}$$

- For a randomly chosen $H$, the collision prob is

$$\frac{\#\{x : x \in GF(2^{128}) | g_{IV1}(x) \boxplus r = g_{IV2}(x)\}}{2^{128}}$$

- In the original security proofs of GCM, it was believed

$$g_{IV_1}(x) \boxplus r = g_{IV_2}(x)$$

has the same number of solutions as

$$g_{IV_1}(x) \oplus r = g_{IV_2}(x)$$

which is upper-bounded by

$$\max\{deg(g_{IV_1}(x)), deg(g_{IV_2}(x))\} = \max\{len(IV_1), len(IV_2)\} + 1$$

# Outline

Intro to Galois/Counter Mode

Repairing Galois/Counter Mode
- The flaw in GCM's proofs discovered by Iwata et al.
- A fix to GCM's security proofs and bounds

Simeck: A Simple Authenticated Cipher Design
- Design Rationales
- Specifications

Summery and Future Work

# Problem in $N_r \boxplus 1$

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

# Problem in $N_r \boxplus 1$

- Pointed out by Iwata *et al.* in Crypto'12
- $N_r \boxplus 1$ 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 expected
  $$\max\{len(IV_1), len(IV_2)\} + 1$$
- $\alpha_r$ times more solutions
  - for $r < 2^{32}$, $\alpha_r$ **is up to** $2^{22}$
    $$\alpha_r \cdot (\max\{len(IV_1), len(IV_2)\} + 1) \leq 2^{22} \cdot (\max\{len(IV_1), len(IV_2)\} + 1)$$

# 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 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 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

# Revisiting Counter Mode

- In CM, counter is incremented by 1, i.e.

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

# Revisiting Counter Mode

- In CM, counter is incremented by 1, i.e.

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

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

# Revisiting Counter Mode

- In CM, counter is incremented by 1, i.e.

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

- 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.

# Revisiting Counter Mode

- In CM, counter is incremented by 1, i.e.

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

- 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 one circle
    - non-repeating

# Requirements of $next()$

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

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

   should be as small as possible compared to

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

   - To reduce counter collision probability

# Requirements of *next*()

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

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

   should be as small as possible compared to

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

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

# Designing *next*()

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

- addition, i.e. XOR

# Designing *next*()

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

- addition, i.e. XOR
  - not a permutation

# Designing *next*()

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

- addition, i.e. XOR
  - not a permutation
  - unless defined as $next(N, r) = N \oplus r$, with $r$ as another input

# Designing *next*()

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

- addition, i.e. XOR
  - not a permutation
  - unless defined as $next(N, r) = N \oplus r$, with $r$ as another input
  - but $f \oplus r_1 = g \oplus r_2 \not\Rightarrow f \oplus (r_1 \boxminus r_2) = g$
    - e.g., $f \oplus 2 = g \oplus 1 \not\Rightarrow f \oplus (2 \boxminus 1) = f \oplus 1 = g$

# Designing *next*()

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

- addition, i.e. XOR
  - not a permutation
  - unless defined as $next(N, r) = N \oplus r$, with $r$ as another input
  - but $f \oplus r_1 = g \oplus r_2 \not\Rightarrow f \oplus (r_1 \boxminus r_2) = g$
    - e.g., $f \oplus 2 = g \oplus 1 \not\Rightarrow f \oplus (2 \boxminus 1) = f \oplus 1 = g$
- multiplication, by a constant
  - multiplying with a primitive element $w$

# Designing *next*()

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

- addition, i.e. XOR
  - not a permutation
  - unless defined as $next(N, r) = N \oplus r$, with $r$ as another input
  - but $f \oplus r_1 = g \oplus r_2 \not\Rightarrow f \oplus (r_1 \boxminus r_2) = g$
    - e.g., $f \oplus 2 = g \oplus 1 \not\Rightarrow f \oplus (2 \boxminus 1) = f \oplus 1 = g$
- multiplication, by a constant
  - multiplying with a primitive element $w$
  - $w^{r_1} f = w^{r_2} g \implies w^{r_1 \boxminus r_2} f = g$

# Designing *next*()

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

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

# Merging Two Circles into One

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

# Merging Two Circles into One

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

- $L_w(x)$ is full-cycle permutation

# Merging Two Circles into One

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

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

# Merging Two Circles into One

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

- $L_w(x)$ is full-cycle permutation
- $L_w^{r_1}(f(x)) = L_w^{r_2}(g(x)) \Leftrightarrow L_w^{r_1 \boxminus r_2}(f(x)) = g(x)$
- 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 $g(x) = 0$.

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

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 $g(x) = w^{r-1}$.

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

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 $g(x) = 0$.

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

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

# LGCM – Revised GCM

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

$$\begin{aligned} N_0 &= GHASH_H(IV) \\ N_i &= L_w^i(N_0) \end{aligned}$$

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

# For Timing-based Side-channel

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

can change to

$$\begin{aligned} y &= w \cdot x, \\ L_w(x) &= \begin{cases} 1 & \text{if } y = 0, \\ 0 & \text{if } y = 1, \\ y & \text{otherwise.} \end{cases} \end{aligned}$$

# Outline

Intro to Galois/Counter Mode

Repairing Galois/Counter Mode
- The flaw in GCM's proofs discovered by Iwata et al.
- A fix to GCM's security proofs and bounds

Simeck: A Simple Authenticated Cipher Design
- Design Rationales
- Specifications

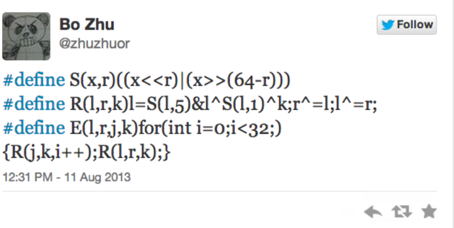
Summery and Future Work

# Simeck: An Authenticated Cipher Design

- LGCM + a lightweight block cipher

# Simeck: An Authenticated Cipher Design

- LGCM + a lightweight block cipher
- Specs of the block cipher in one tweet (140 chars)




- *tweetcipher* designed by Aumasson needs 6 tweets

**JP Aumasson** @veorq 8 Jun
x[0]^=1; ROUNDS LOOP(8) putchar(255&((x[4]^x[5])>>8*i)); LOOP(8) putchar(255&((x[6]^x[7])>>8*i)); return 0;}
Expand

**JP Aumasson** @veorq 8 Jun
x[i+4]=W(v[3],i); ROUNDS while((c=getchar())!=EOF){if(!f&&10==(x[0]^c)%256)return 0;putchar(x[0]^c);x[0]=c^(f?x[0]:x[0]&~255ULL);ROUNDS}
Expand

**JP Aumasson** @veorq 8 Jun
int main(int _,char**v){ uint64_t x[16],i,c,r,f='e'==*v[1]; LOOP(16) x[i]=i*0x7477697468617369ULL; LOOP(4) x[i]=W(v[2],i); LOOP(2)
Expand

**JP Aumasson** @veorq 8 Jun
AXR(a,b,d,16) AXR(c,d,b,11)} #define ROUNDS {for(r=6;r--;){LOOP(4) G(i,i+4,i+8,i+12) LOOP(4) G(i,(i+1)%4+4,(i+2)%4+8,(i+3)%4+12)}}
Expand

**JP Aumasson** @veorq 8 Jun
#define R(v,n)(((v)<<(64-n))|((v)>>n)) #define AXR(a,b,c,r) x[a]+=x[b];x[c]=R(x[c]^x[a],r); #define G(a,b,c,d) {AXR(a,b,d,32) AXR(c,d,b,25)
Expand

**JP Aumasson** @veorq 8 Jun
#include <stdint.h> #include <stdio.h> #define LOOP(n) for(i=0;i<n;++i) #define W(v,n) ((uint64_t*)v)[n]
Expand

# Block Cipher Design

- Consider the two block ciphers designed by Beaulieu *et al.* from NSA
  - hardware-optimized cipher Simon
  - software-optimized cipher Speck

# Block Cipher Design

- Consider the two block ciphers designed by Beaulieu *et al.* from NSA
  - hardware-optimized cipher Simon
  - software-optimized cipher Speck
- Design comparisons
  - Round function, both Feistel-like network

Simon Use AND for efficiency of hardware

Speck ARX construction; decryption cannot reuse encryption functions

# Block Cipher Design

- Consider the two block ciphers designed by Beaulieu *et al.* from NSA
  - hardware-optimized cipher Simon
  - software-optimized cipher Speck
- Design comparisons
  - Round function, both Feistel-like network

    Simon: Use AND for efficiency of hardware

    Speck: ARX construction; decryption cannot reuse encryption functions
  - Key schedule

    Simon: Linear operations with constant sequences

    Speck: Cleverly reuse round function

# Block Cipher Design

- Consider the two block ciphers designed by Beaulieu *et al.* from NSA
  - hardware-optimized cipher Simon
  - software-optimized cipher Speck
- Design comparisons
  - Round function, both Feistel-like network

    Simon Use AND for efficiency of hardware

    Speck ARX construction; decryption cannot reuse encryption functions
  - Key schedule

    Simon Linear operations with constant sequences

    Speck Cleverly reuse round function
- How about we combine them two?

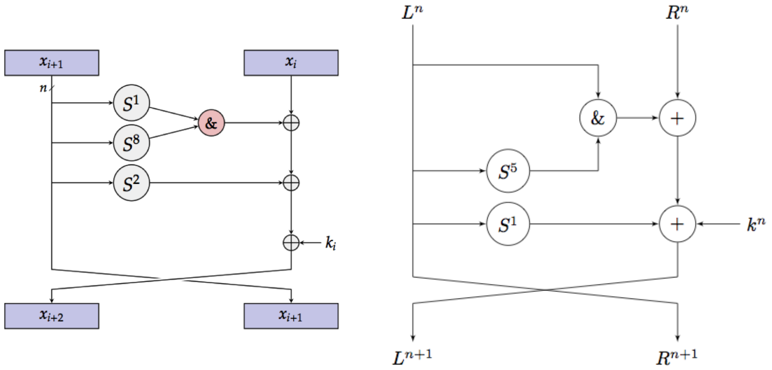
# Simeck = Simon + Speck

- Combine the efficient designs
  - Round function of Simon
  - Key schedule of Speck
- Minimal design
  - Keep the design as simple as possible
  - If we could find attacks on the mini design
    - Get attacks on Simon and/or Speck
    - or understand more about Simon and Speck
  - Get a fairly good authenticated cipher design if no serious attack is found

# Simeck Round function

Simplified from Simon

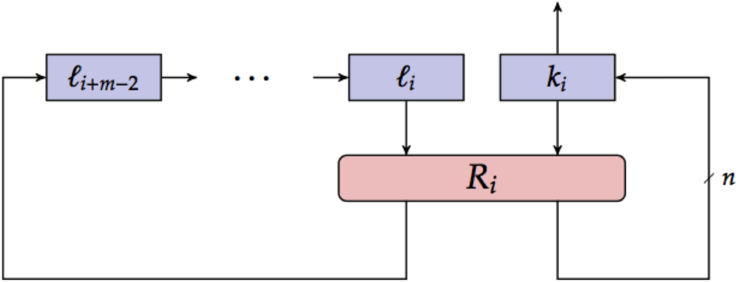
- Remove $S^1$
- Change $S^8$ to $S^5$, $S^2$ to $S^1$




The left image is from the Simon and Speck design paper.

# Simeck Key Schedule

Learn from Speck




The image is from the Simon and Speck design paper.

# Parameters and Performance

- 128-bit block cipher, compatible with LGCM
- 128/196/254 bits for master keys
- 32/48/64 rounds for security-levels

# Parameters and Performance

- 128-bit block cipher, compatible with LGCM
- 128/196/254 bits for master keys
- 32/48/64 rounds for security-levels
- Hardware implementation
  - Reuse the round function in key schedule
  - Less bits of rotations
  - Smaller footprint than hardware-optimized Simon

# Parameters and Performance

- 128-bit block cipher, compatible with LGCM
- 128/196/254 bits for master keys
- 32/48/64 rounds for security-levels
- Hardware implementation
  - Reuse the round function in key schedule
  - Less bits of rotations
  - Smaller footprint than hardware-optimized Simon
- Software implementation
  - Comparable software performance with software-oriented Speck
  - Decryption can reuse encryption round function
  - Small code size (ROM) for software

# Parameters and Performance

- 128-bit block cipher, compatible with LGCM
- 128/196/254 bits for master keys
- 32/48/64 rounds for security-levels
- Hardware implementation
  - Reuse the round function in key schedule
  - Less bits of rotations
  - Smaller footprint than hardware-optimized Simon
- Software implementation
  - Comparable software performance with software-oriented Speck
  - Decryption can reuse encryption round function
  - Small code size (ROM) for software
- Compact and clean specification (in one tweet!)
  - Ideal for "lazy" programmers
  - Neither Simon, nor Speck can fit into 140 chars

# Outline

# Summery and Future Work

- Repairing GCM
  - Merging two cycles by $L_w$
  - Consider cyclic permutation polynomials?
  - Redo proofs and recompute bounds with other fixes?
- Designing Simeck
  - Ideas/designs from Simon and Speck
  - To attack Simeck?
  - More efficient mode of operation than GCM?

Thanks for your attention!