Investigating the security properties of MACs based on stream ciphers

Leonie Simpson, Mufeed Al Mashrafi, Harry Bartlett, Ed Dawson and Kenneth Wong

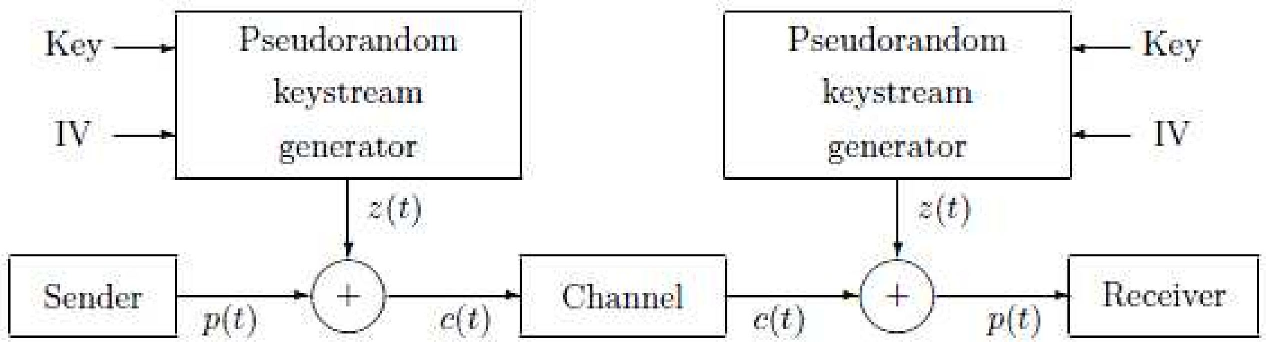
Institute for Future Environments
Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia

Outline

- Introduction
- Indirect injection
  - Matrix Representation
  - Security Analysis
  - Examples
- Direct injection
  - Matrix representation
  - Security analysis
  - Examples
- Summary

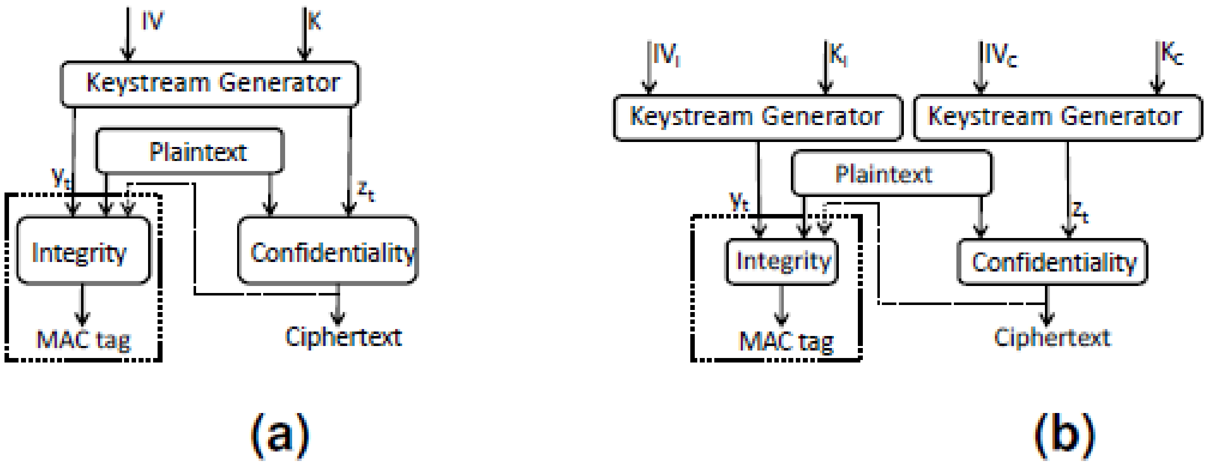
Introduction: Stream ciphers

- Keystream generator for a stream cipher
  - Inputs: secret key *K* and public *IV*
  - Outputs: Pseudorandom binary sequence
- Sequence commonly used as keystream for binary additive stream cipher to provide ***confidentiality***



Introduction: Stream ciphers

- Keystreams also used for ***integrity*** applications
- Stream ciphers providing authenticated encryption (AE) use binary sequences for both confidentiality and integrity
- These sequences can be produced by:
  a) the *same* keystream generator
  b) *different* keystream generators



Introduction:
Stream ciphers and MAC generation

- **Phases of MAC generation**:
  1. Preparation:
     - Initialise the internal state of the integrity components of the device
     - Prepare the input message: may involve appending padding bits to either end of message
     - NOTE: for AE, message may be plaintext or ciphertext
  2. Accumulation:
     - Iterative process where input message used to accumulate values in the internal state of the integrity component
  3. Finalisation:
     - Complete the processing of MAC tag (possible masking)

Introduction:
Stream ciphers and MAC generation

- Q: <u>How do stream ciphers use the message in the accumulation phase</u>?
  - Message dependent updating of internal state of integrity component
  - Two approaches to this:
    1. **<u>Directly</u>**: using message content as an input into the internal state component
    2. **<u>Indirectly</u>**: using the message content to control accumulation of some unknown keystream into an internal state component

Introduction:
AE Stream ciphers and MAC security

- Consider security against forgery attacks:
  - Assume keystream sequences are pseudorandom
  - Consider a Man-In-The-Middle attacker who can:
    - Intercept transmission of M and $MAC_{K,IV}(M)$, and
    - Modify M and possibly also $MAC_{K,IV}(M)$:
      - Flip, delete or insert bits in M,
      - Alter bits in $MAC_{K,IV}(M)$

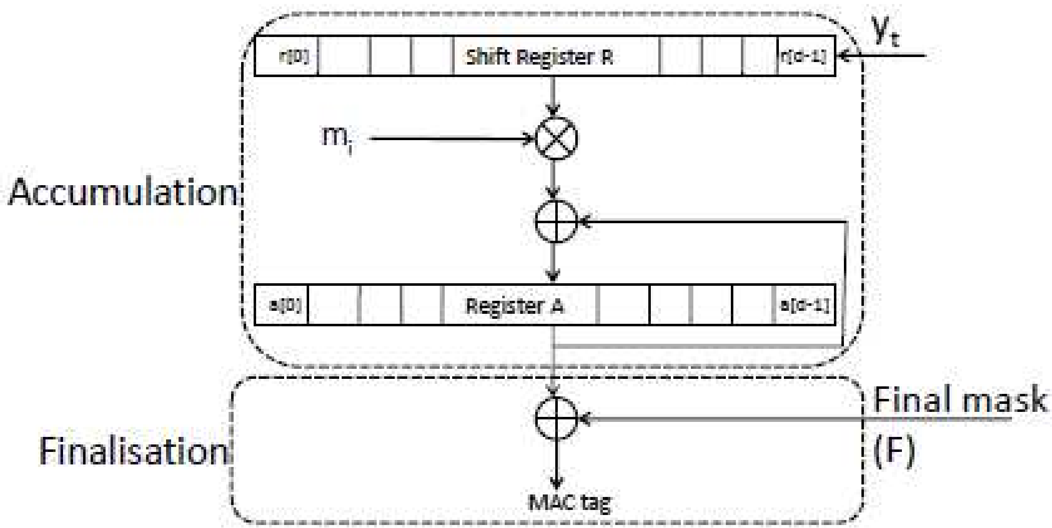
  - Forgery succeeds if attacker can produce valid pair: *M' and $MAC_{K,IV}(M')$*

Outline

- Introduction
- Indirect injection
  - Matrix Representation
  - Security Analysis
  - Examples
- Direct injection
  - Matrix representation
  - Security analysis
  - Examples
- Summary

Indirect injection

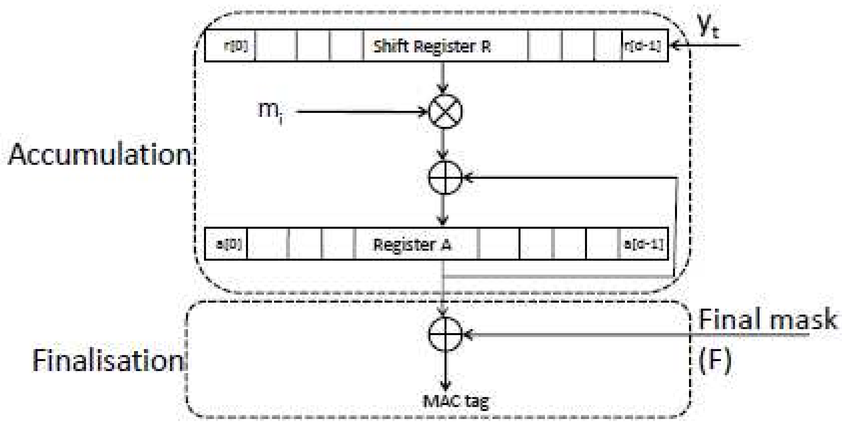
- Modelling the ***integrity*** component:
  - Two registers, *R* and *A*, same length as MAC: *d* bits
  - Two inputs: message *M* and keystream sequence *y*
  - *M* used to control values from *R* accumulated in *A*



Indirect injection

- During accumulation:
  - Register R update:
    - Sliding window on keystream

$$r_t[i] = \begin{cases} r_{t-1}[i+1], & \text{for } i = 0, \ldots, d-2 \\ y_{t-1}, & \text{for } i = d-1 \end{cases}$$



  - Register A update:
    - Message dependent

$$A_t = \begin{cases} A_{t-1} \oplus R_{t-1}, & \text{if } m_t = 1 \\ A_{t-1}, & \text{otherwise} \end{cases}$$

Indirect injection: examples

- Stream cipher based MACs using indirect injection:

Cipher	Date	MAC size	message	initialisation	finalisation
Sfinks	2005	64	plaintext	R_0 and A_0 with y^t	z^t
Grain-128a	2011	32	ciphertext	R_0 and A_0 with y^t	$F = 0$
ZUC	2011	32	ciphertext	R_0 with y^t and $A_0 = 0$	y^t

Indirect injection: matrix representation

- Consider contents of register A at time *i*:
  - Each stage of A contains a message dependent linear combination of values previously in register R, combined with the initial values in A:

$$A_i = A_0 \oplus T_i M_i$$

$$= \begin{pmatrix} a_0[0] \\ a_0[1] \\ \vdots \\ a_0[d-1] \end{pmatrix} \oplus \begin{pmatrix} r_0[0] & r_0[1] & \dots & r_0[d-1] & y_0 & \dots & y_{i-d-1} \\ r_0[1] & r_0[2] & \dots & y_0 & y_1 & \dots & y_{i-d} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots \\ r_0[d-1] & y_0 & \dots & y_{d-2} & y_{d-1} & \dots & y_{i-2} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{i-1} \end{pmatrix}$$

Indirect injection: matrix representation

- Computing the MAC for an input message of length *l*:
  - Compute the value in the accumulation register A
  - Combine with (optional) final mask

$$MAC(M_l) = A_l \oplus F = A_0 \oplus T_l M_l \oplus F$$

- NOTE: really only need to consider two aspects:
  - the accumulation phase, and
  - the linear combination of A_0 and F

Indirect injection: security analysis

- Analysis of the accumulation phase only:

$$T_l\, M_l = \begin{pmatrix} r_0[0] & r_0[1] \ldots r_0[d-1] & y_0 & \ldots y_{l-d-1} \\ r_0[1] & r_0[2] \ldots & y_0 & y_1 & \ldots & y_{l-d} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots \\ r_0[d-1] & y_0 & \ldots & y_{d-2} & y_{d-1} \ldots & y_{l-2} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \end{pmatrix}$$

- Bit flipping forgeries:
  - Forge MAC(*M'*) by flipping appropriate bit/s in MAC(*M*)
  - For known R_0 attacker can flip:
    - first bit of *M* and forge valid MAC with probability 1
    - first 2 bits of *M* and forge valid MAC with probability ½
    - first *i* bits of *M* and forge valid MAC with probability 2^{-i}

Indirect injection: security analysis

- Analysis of the accumulation phase only:

$$T_l\, M_l = \begin{pmatrix} r_0[0] & r_0[1] \ldots r_0[d-1] & y_0 & \ldots y_{l-d-1} \\ r_0[1] & r_0[2] \ldots & y_0 & y_1 & \ldots & y_{l-d} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots \\ r_0[d-1] & y_0 & \ldots & y_{d-2} & y_{d-1} \ldots & y_{l-2} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \end{pmatrix}$$

- Bit deletion forgeries:
  - Forge MAC(*M'*) by shifting MAC(*M*) and guessing appropriate bit/s
  - For known R_0 attacker can delete:
    - first bit of *M* and forge valid MAC with probability ½
    - first 2 bits of *M* and forge valid MAC with probability ¼
    - first *i* bits of *M* and forge valid MAC with probability 2^{-i}
  - Similarly, can forge MACs for unknown R_0 but known *M* by deleting leading/trailing zeroes

Indirect injection: security analysis

- Analysis of the accumulation phase only:

$$T_l\, M_l = \begin{pmatrix} r_0[0] & r_0[1] & \dots & r_0[d-1] & y_0 & \dots & y_{l-d-1} \\ r_0[1] & r_0[2] & \dots & y_0 & y_1 & \dots & y_{l-d} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots \\ r_0[d-1] & y_0 & \dots & y_{d-2} & y_{d-1} & \dots & y_{l-2} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \end{pmatrix}$$

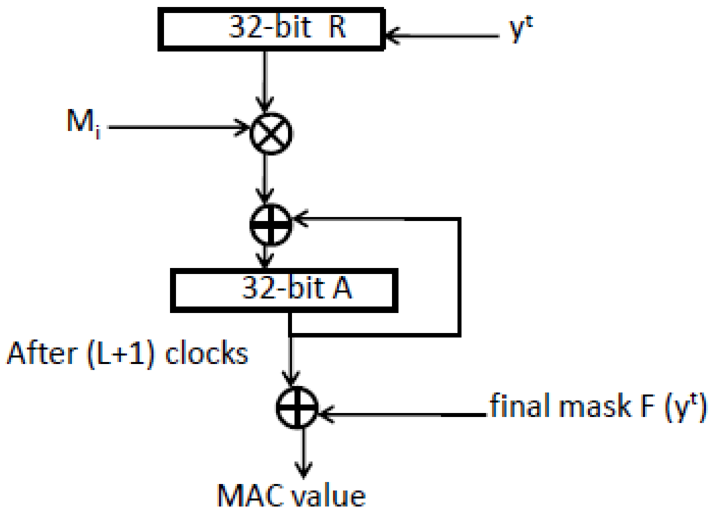
- Bit insertion forgeries:
  - For any R_0,
    - Can insert zeroes at the end of M:
      - Does not change accumulated value, so MAC(*M'*) = MAC(*M*)
      - Forge valid MAC with probability 1
    - Can insert zeroes at the start of M
      - Forge MAC(*M'*) by shifting MAC(*M*) and guessing appropriate bit/s
      - Insert one zero - forge valid MAC with probability ½
      - Insert *i* zeroes - forge valid MAC with probability 2^{-i}
  - For known R_0 can insert 1's at start (Forge MAC(*M'*) by shift & guessing)

Indirect injection: security analysis

- Analysis of the masking phase: $\boldsymbol{A_0 \oplus F}$
  - Forgeries involving *insertions or deletions at the start of the message* rely on the sliding property of $T_l M_l$
    - Prevent the MAC tag sliding by by initialising A with bits from a fixed position, such as the start of the keystream sequence y
  - Forgeries involving *zeroes inserted or deleted at the end of the message* rely on the these zeroes having no effect on the accumulated value
    - Choice of A_0 does not prevent this
    - Prevent by using unknown mask that depends on message length
  - Choices for A_0 and F provide effective means to prevent bit insertion and deletion attacks

Indirect injection: ZUC

- 128-EIA3 based on ZUC
  - Prep phase: input message padded with a 1 at end
  - Finalisation phase: final mask from same sequence, as accumulation, but segment not previously used



Indirect injection: ZUC

- Matrix representation: MAC tag for 128-EIA3 Version 1.4

$$MAC(M_p) = \begin{pmatrix} y_{-32} & y_{-31} & \cdots & y_{-1} & y_0 & \cdots & y_{l-32} \\ y_{-31} & y_{-30} & \cdots & y_0 & y_1 & \cdots & y_{l-31} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots \\ y_{-2} & y_{-1} & \cdots & y_{29} & y_{30} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_{30} & y_{31} & \cdots & y_{l-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \\ 1 \end{pmatrix} \oplus \begin{pmatrix} y_{l+32} \\ y_{l+33} \\ \vdots \\ y_{l+62} \\ y_{l+63} \end{pmatrix}$$

- Fuhr et al, 2012
  - Possible forgery if zero inserted at start of message
  - Forge MAC from existing by shifting and guessing bit
- Our work, 2012
  - For messages with leading zeroes, possible to delete zeroes and forge MACs by shifting and guessing

Outline

- Introduction
- Indirect injection
  - Matrix Representation
  - Security Analysis
  - Examples
- Direct injection
  - Matrix representation
  - Security analysis
  - Examples
- Summary

Direct injection

- Model for the ***integrity*** component:
  - Consider simple case: accumulation component is single register
  - Aspects to consider:
    - component state update function
    - how and where message inputs are injected
  - We extend the Nakano et al. 2011 model for stream cipher-based hash functions:
    - Hash function based on nonlinear filter generator
    - Uses structure of generator, but hash function is unkeyed
    - State update function includes both:
      - LFSR update, and
      - nonlinear filter feedback

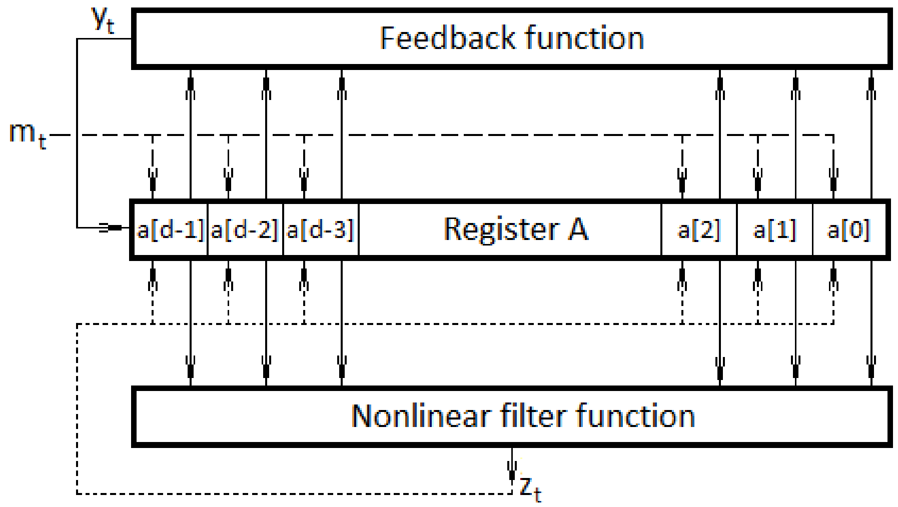
Direct injection: examples

- SOBER family of stream cipher based MACs or MAC components use direct injection:

Cipher	Date	MAC size	Message	Initialisation	Finalisation
SOBER -128	2003	32 bits	plaintext if transmission is ciphertext	keystream	Nonlinear
SSS	2005	≤ 128	plaintext	keystream	Encrypts MAC
NLSv2	2006	variable	plaintext	keystream	2 components combined

Direct injection

- Accumulation using nonlinear filter generator
  - Inject message and filter output into LFSR
    - Consider *where* input will be injected (which stages)
    - Consider *how* input will be injected (combine or replace)



Direct injection: matrix representation

- For autonomous LFSR: $A_{t+1} = C\,A_t$ where

$$A_t = \begin{bmatrix} a_t[0] \\ a_t[1] \\ \vdots \\ \vdots \\ a_t[d-1] \end{bmatrix} \text{ and } C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ c_0 & c_1 & c_2 & \cdots & c_{d-1} \end{bmatrix}$$

- Extend to include injection of message and/or nonlinear filter output bit by combining:

$$A_{t+1} = C\,A_t \oplus m_t\sigma_m \oplus z_t\sigma_z$$

Direct injection: matrix representation

- In the accumulation phase, as the message is processed the contents of register A are updated:

$$A_{t+1} = C\,A_t \oplus m_t\sigma_m \oplus z_t\sigma_z$$

- Matrix representation for this:

$$A_L = C^L A_0 \oplus K_m M_{L-1} \oplus K_z Z_{L-1}$$

- where

$$K_m = [C^{L-1}\sigma_m \; C^{L-2}\sigma_m \; \ldots \; C\sigma_m \; \sigma_m]$$

$$M_{L-1} = [m_0 \; m_1 \; \ldots \; m_{L-2} \; m_{L-1}]^T,$$

Direct injection: matrix representation

- At the end of accumulation phase:

  $A_L = C^L A_0 \oplus K_m M_{L-1} \oplus K_z Z_{L-1}$
- For injection performed by **replacing** stage contents with feedback, rather than combining, can construct a similar matrix model:
  - Modify matrix C by changing relevant 1 to 0.
  - Also affects definitions of K_m and K_z
- Matrix model also permits mixtures of **combining / replacing**
  - Through choices for entries in state update matrix C

Direct injection: security analysis

- Analyse matrix model for possible collisions obtained through manipulating contents of M
  - If M and M' produce same A_L then forgery possible
  - Assume A_0 is unknown
    - NOTE: MAC(M) is reproducible if M and A_0 are both known, consider this for completeness
- Consider two cases:

1. Message injection by combining
2. Message injection with replacement

Direct injection: security analysis

- 1. Message injection by combining
  - 2 subcases: is nonlinear filter output z injected into state?

  - Case 1: *z* is not injected: then $A_L = C^L A_0 \oplus K_m M_{L-1}$
  - Theorem: the final *d* columns of K_m form a basis for

$U = \{C^i \sigma_m \mid i \geq 0\}$ = column space of K_m

$\Rightarrow$ if $L > d$, can always force collisions:

- the results of any changes to the first $L-d$ words of the message can be reversed by a suitable set of changes to the final d words

  - Applies whether A_0 is known or not (due to linearity)

Direct injection: security analysis

- 1. Message injection by combining (cont'd)
  - Case 2: z injected: then $A_L = C^L A_0 \oplus K_m M_{L-1} \oplus K_z Z_{L-1}$

a) If M_{L-1}, A_0 known, $\sigma_m = \sigma_z \rightarrow K_m = K_z$
   - z_t known at each step, so adjust m_t by $-z_t$ to obtain forgery as before

b) If M_{L-1}, A_0 known, $\sigma_m \neq \sigma_z \rightarrow K_m \neq K_z$
   - now z_t, m_t affect different stages: can't adjust for z_t

c) If M_{L-1} and/or A_0 unknown
   - now z_t unknown, so can't adjust for it

Direct injection: security analysis

- Now consider message injection with some replacing:
  - Arguments for
    - Case 1: Z injected, and
    - Case 2: Z not injected

  apply as before, except that the dimension of the column space is reduced
  - This means that **only a reduced basis is required** to guarantee forgeries in Cases 1 and 2a
    - see SOBER-128 example later

Direct injection: security analysis

- Summary of analysis

Case	Nonlinear filter	M / A_0	Other condition	Forced collisions ?	Overall outcome
1	not used	any	—	Yes	not secure (collisions)
2a	used	both known	$\sigma_m = \sigma_z$	Yes	not secure (collisions)
2b	used	both known	$\sigma_m \neq \sigma_z$	Unlikely	not secure – other
2c	used	either unknown	—	No	secure

Direct injection: security analysis

- Nakano et al. model for hash functions:
  - bit based LFSR with known (zero) initial state
  - message (plaintext) known
- Hash function model considered two configurations with $\sigma_m = \sigma_z$ and **combining** into register:
  1. into final stage a[d–1] only
  2. into r regularly spaced stages
- Both configurations are Case 2a,
  - Therefore **collisions can be forced in both cases** – contrary to their claim for (2)

Direct injection: security analysis

- Several members of the **Sober** stream cipher family include a MAC component that fits our model:
  - SOBER-128:
    - **replacing** Case 2c: accumulation should be secure but nonlinear filter is weak
  - SSS:
    - **combining** Case 1 $\Rightarrow$ accumulation insecure
    - but MAC secure as cipher self-synchronous
  - NLSv2:
    - **combining** Case 1 $\Rightarrow$ accumulation insecure
    - but has second (n.l.) accumulation

Summary

- Can generate MAC tags using stream ciphers by injecting the input message (plaintext or ciphertext)
  - Indirectly
  - Directly
- Matrix model for the accumulation phase facilitates analysis of potential forgeries
  - that do not require knowledge of the keystream
- Different options available for preparation and finalization phases of MAC generation
  - Security implications associated with these options with respect to forgery attacks

References

- Mufeed Almashrafi, Harry Bartlett, Leonie Simpson, Ed Dawson and Kenneth Wong. Analysis of indirect message injection for MAC generation using stream ciphers. In *17th Australasian Conference on Information Security and Privacy (ACISP 2012)*, vol 7372 of Lecture Notes in Computer Science, pages 138-151, Springer, Heidelberg (2012).

- Harry Bartlett, Mufeed Almashrafi, Leonie Simpson, Ed Dawson and Kenneth Wong. A general model for MAC generation using direct injection. In *8th China International Conference on Information Security and Cryptology (INSCRYPT 2012)*, vol 7763 of Lecture Notes in Computer Science, pages 198-215, Springer, Heidelberg (2012).

- Mufeed Almashrafi, Harry Bartlett, Ed Dawson, Leonie Simpson and Kenneth Wong. Indirect message injection for MAC generation. to appear in *Journal of Mathematical Cryptology*