CS171: Cryptography

Lecture 7

Sanjam Garg

Approach – Stream Ciphers/Block Ciphers

- Heuristic
  - no lower level assumptions
- Formal Definitions Help
- Clear Design Principles

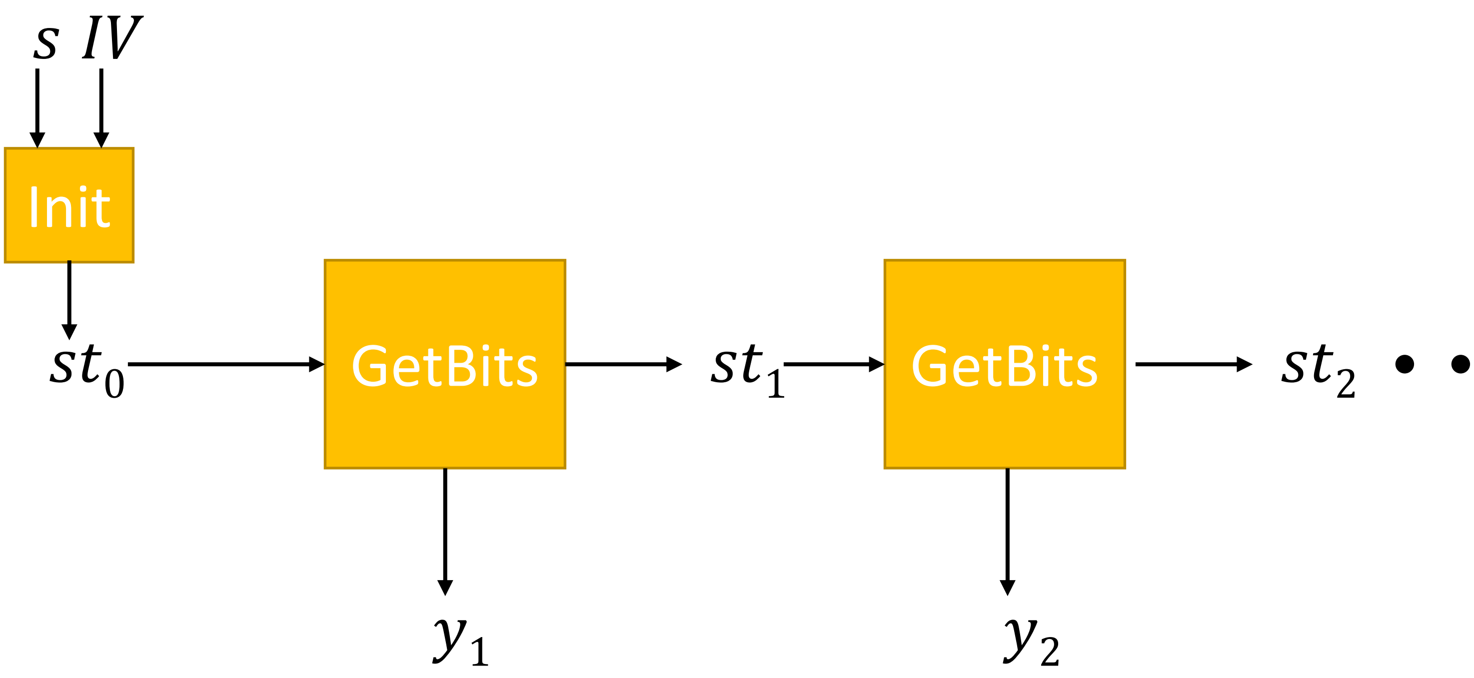
Stream Ciphers

- Init algorithm
  - Input: a key and an *optional* initialization vector (IV)
  - Output: initial state

- GetBits algorithm
  - Input: the current state
  - Output: next bit and updated state
  - Multiple executions allow for generation of desired number of bits

Stream Ciphers

- Use (Init, GetBits) to generate the desired number of output bits from the seed



Security

- Without IV: For a uniform key, output of GetBits should a pseudorandom stream of bits

- With IV: : For a uniform key, and uniform IVs (a*vailable to the attacker*), output of GetBits should be pseudorandom streams of bits (weak PRF)

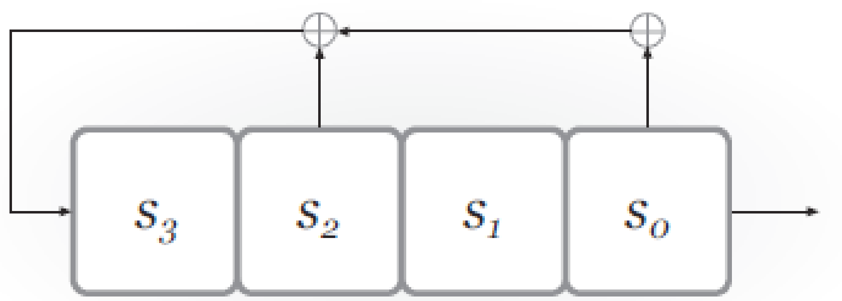
Security

- We care about concrete security and not just asymptotic security

- Efficiency: Keys of length n should give security against adversaries running in time $\approx 2^n$.

LFSRs (Linear Feedback Shift Register)

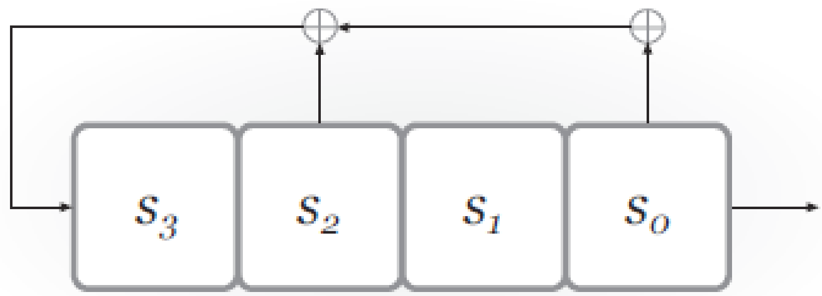
- Degree-n LFSR has n registers
- $s_{n-1} \dots s_0$ are the contents of the registers
- $c_{n-1} \dots c_0$ are the feedback coefficients



- Registers updated in each clock cycle

$$s'_{n-1} = \sum c_j s_j \; mod\; 2$$
$$s'_i = s_{i+1} \text{ for } i < n-2$$

LFSR



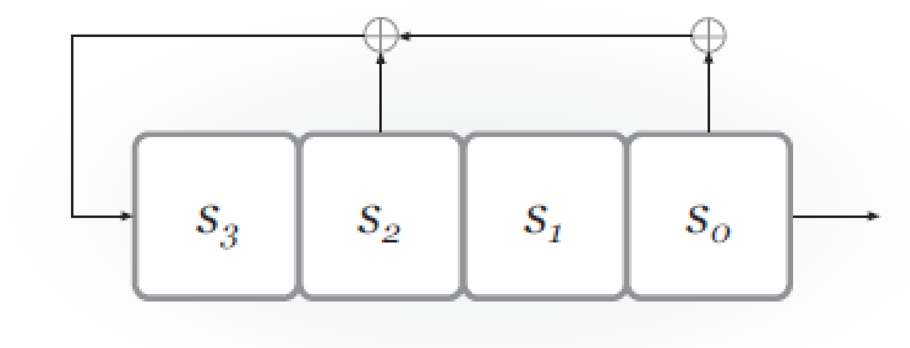
- 0100
- 1010 -> 0
- 0101 -> 0
- 0010 -> 1

Quest for a good LFSR

Output bits will start to repeat for short cycles.

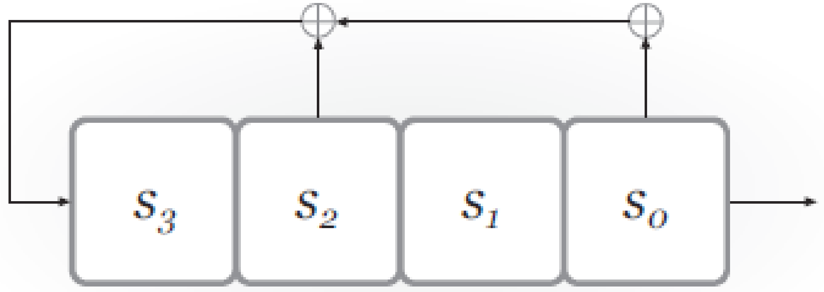
- Intuitively: Should cycle all $2^n - 1$ non-zero states.
- It is known how to set the feedback coefficients to get such an LFSR (also called maximum length LFSR)

- Max length LFSR has good statistical properties but is not cryptographically secure



Can you see an attack?

Attacks on LFSR



- If the feedback coefficients are fixed (and known to the attacker),

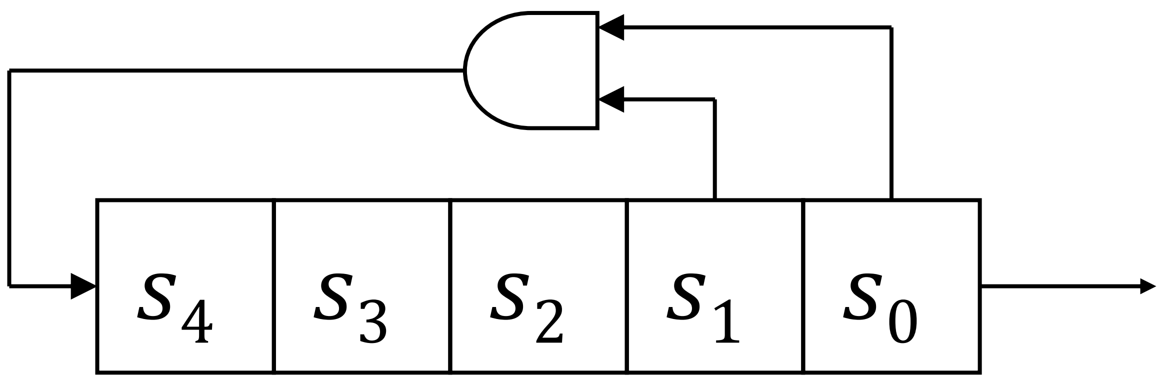
  then the first n output bits fix the key entirely.
- If the feedback coefficients are unknown (and derived from the key),

  then the first $2n$ output bits fix the key and the coefficients. (linear algebra is very powerful)
- Lesson: linearity is *bad* for pseudorandomness

Non-linear FSR

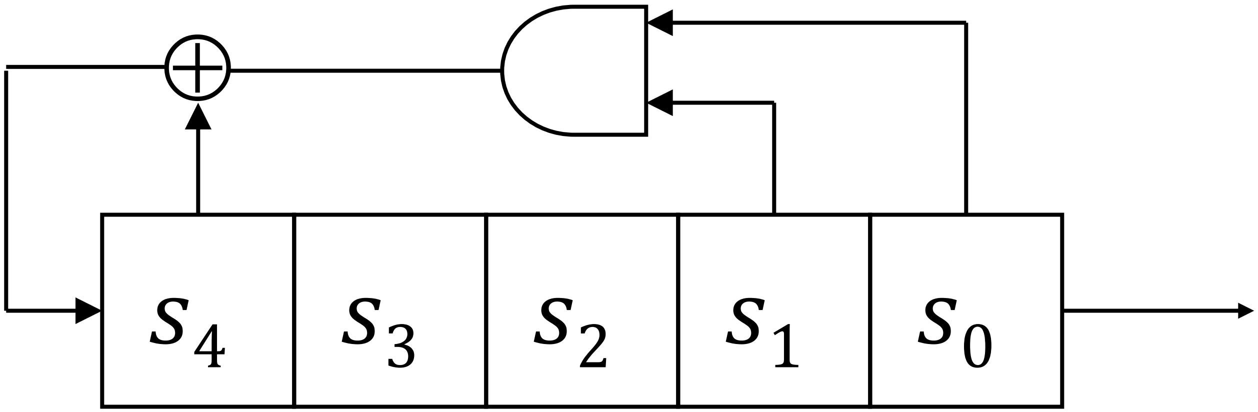
- Adding non-linearity
  - Make the feedback non-linear
  - Make the output non-linear
  - Use multiple LFSRs
  - Mix the above methods.
- Allow for long-cycle and preserve the statistical properties.

Non-linear Feedback



- Is it secure?
- Linear-algebra is not useful!
- However, AND biases the bits!
- How can we fix this?

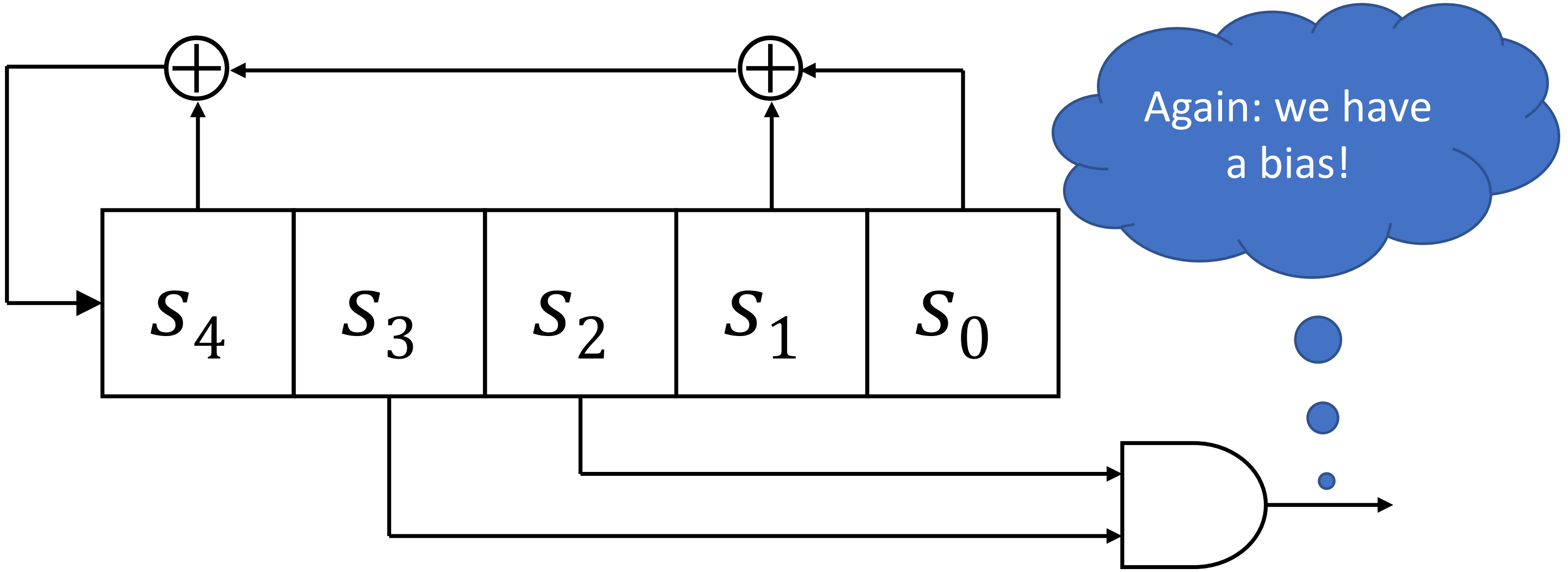
Non-linear Feedback (avoiding bias)



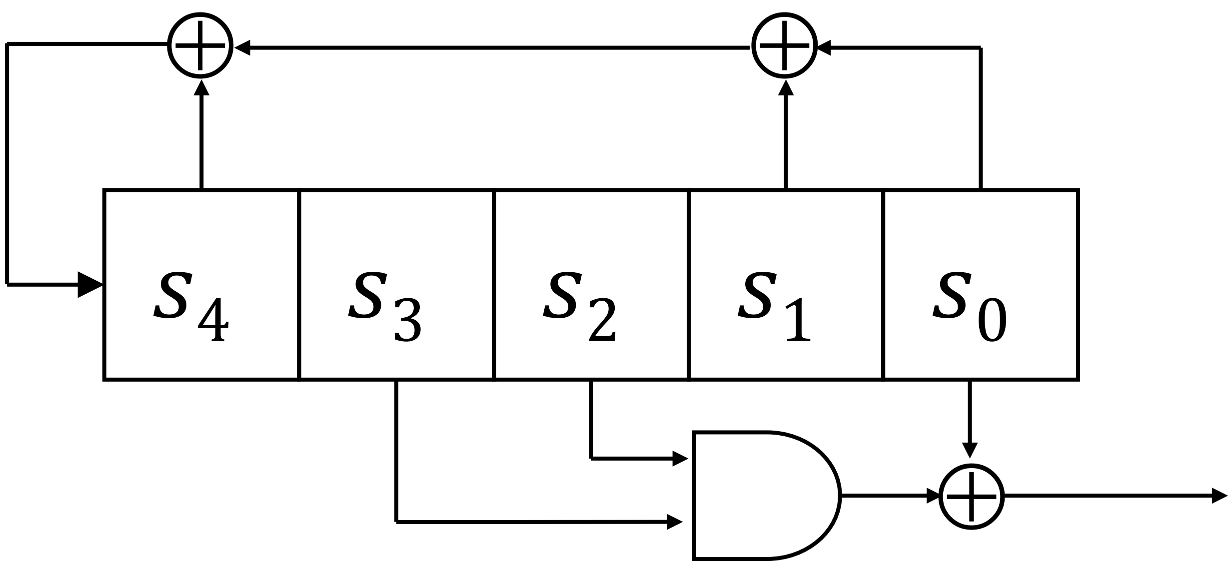
- Use of xor helps remove bias!

Non-linear output

- Update of the LFSR state is linear but the output is obtained as a non-linear function of the state



Non-linear Output (avoiding bias)

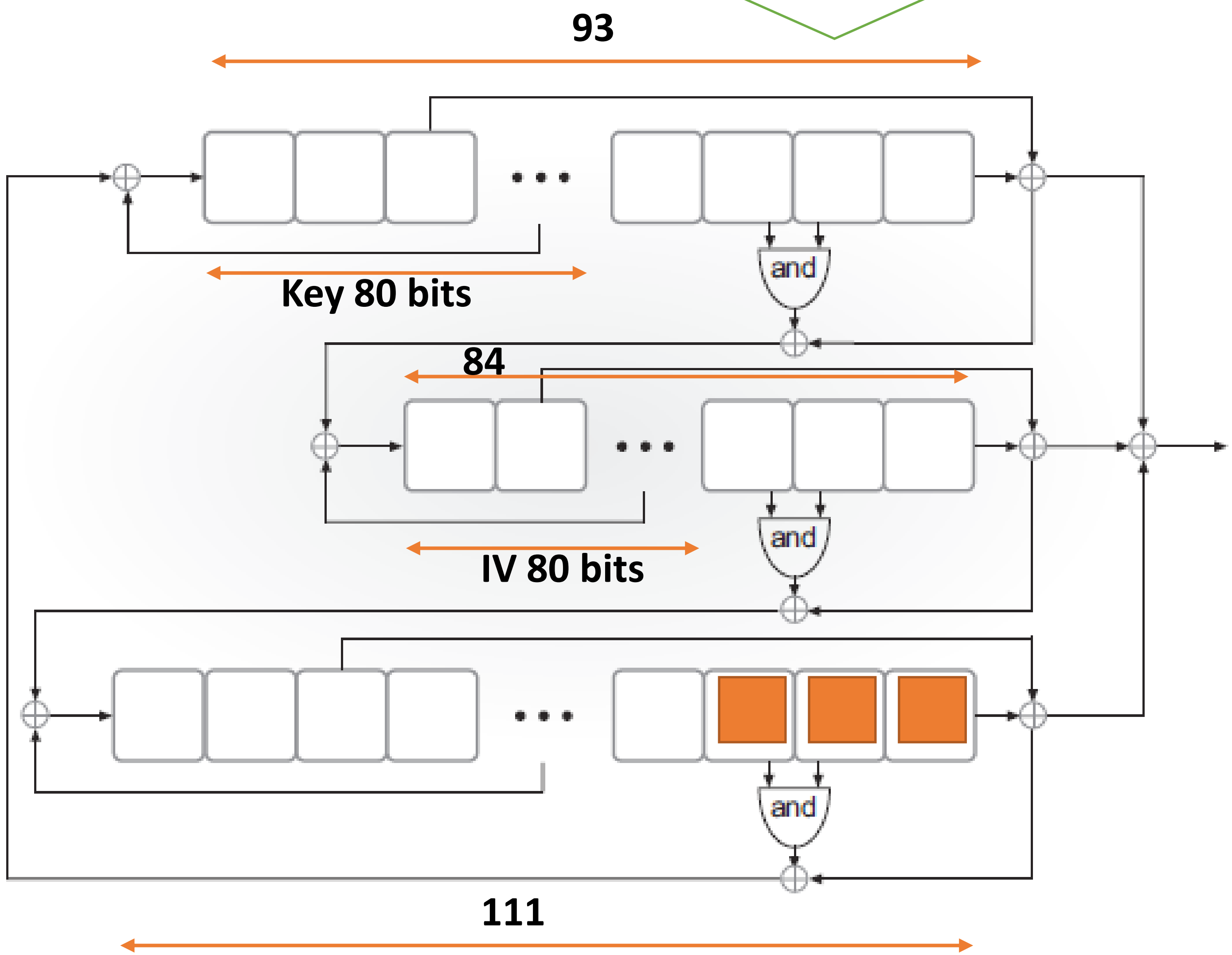


Trivium

- Designed by De Cannière and Preneel in 2006 as part of eSTREAM competition
- Designed for efficiency in hardware
- No attacks better than brute-force search are known!

Trivium

Set everything else to 0, except the last three registers (of the last FSR) which are set to 1. Then, initialize by executing for $4 \cdot 288$ times and discarding the output bits.



RC4

- Designed in 1987
- Designed for efficiency in software, rather than hardware

- *No longer considered secure*, but still interesting to study
  - Simple description; not LFSR-based
  - Still encountered in practice (WEP 802.11)
  - Interesting attacks

RC4

Set $S[i]$ to be the identity permutation of $\{0 \ldots 255\}$.

One pseudorandom swap and obtain information for a pseudorandom location.

ALGORITHM 6.1
Init algorithm for RC4

Input: 16-byte key k
Output: Initial state (S, i, j)
(Note: All addition is done modulo 256)

for $i = 0$ to 255:
  $S[i] := i$
  $k[i] := k[i \bmod 16]$
$j := 0$
for $i = 0$ to 255:
  $j := j + S[i] + k[i]$
  Swap $S[i]$ and $S[j]$
$i := 0, \quad j := 0$
return (S, i, j)

ALGORITHM 6.2
GetBits algorithm for RC4

Input: Current state (S, i, j)
Output: Output byte y; updated state (S, i, j)
(Note: All addition is done modulo 256)

$i := i + 1$
$j := j + S[i]$
Swap $S[i]$ and $S[j]$
$t := S[i] + S[j]$
$y := S[t]$
return $(S, i, j), y$

Repeat the key to make it 256 byte long.

Each entry of S is swapped with another pseudorandom entry of S.

RC4 used with an initialization vector

- Was not designed for that.
- Set key to be $k \ = \ IV||k'$

Attack: Biased 2^{nd} output byte

- Let S_t denote the state of array S after t executions.
- Say S_0 is uniform for simplicity
- Thus, $S_0[2] = 0$ and $S_0[1] = X \neq 2$ happens with probability $\frac{1}{256} \cdot \left(1 - \frac{1}{256}\right) \approx \frac{1}{256}$.

Probability 2^{nd} output byte is 0 is $\approx 1/256 + 1/256$

ALGORITHM 6.2
GetBits algorithm for RC4

Input: Current state (S, i, j)
Output: Output byte y; updated state (S, i, j)
(Note: All addition is done modulo 256)

$i := i + 1$
$j := j + S[i]$
Swap $S[i]$ and $S[j]$
$t := S[i] + S[j]$
$y := S[t]$
return $(S, i, j), y$

After 1 step, $i = 1, j = X$ and $S_1[X] = X$.

After 2 step, $i = 2, j = X$ and $S_2[X] = 0$. $t = X$

$S_1[t] = 0$

More attacks

- Already enough to break EAV-security
- More serious attacks when IV is used
- Attacks can recover keys in WEP

Thank You!