Stream ciphers: native stream ciphers and security notion

Cryptography, Spring 2020

L. Batina, J. Daemen

February 12, 2020

Institute for Computing and Information Sciences
Radboud University

Outline

How not to build modern stream ciphers

Some real-world stream ciphers
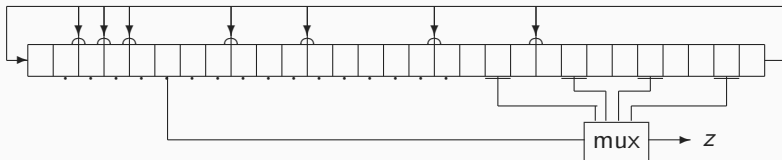
Modelling attacks on stream ciphers

The Random Oracle

Indistinguishability security notion

# How not to build modern stream ciphers

- ▶ Real-world stream cipher (early '90s), approach:
  - • choose LFSR and mux dimension d to resist best known attacks
  - • initialize state with $K+D$
- ▶ Adversary may get keystream for multiple, say n, diversifier values:
  - • $D^{(0)}$ gives $Z^{(0)}$
  - • $D^{(1)}$ gives $Z^{(1)}$
  - • $\cdots$
  - • $D^{(i)}$ gives $Z^{(i)}$
  - • $\cdots$
- ▶ We zoom in on the state $s_t^{(i)}$ and keystream bit $z_t^{(i)}$ at cycle t
- ▶ For compactness, we omit the t subscript and write $s^{(i)}$ and $z^{(i)}$
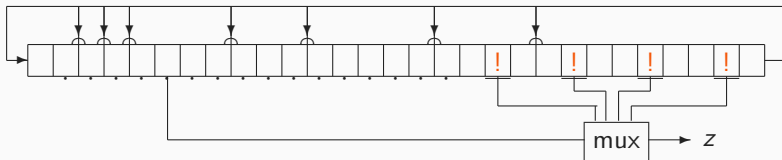
- ▶ Some notation:
  - we have $s^{(i)} = M^t(K + D^{(i)}) = M^tK + M^tD^{(i)}$
  - we write K' for M^tK and $E^{(i)}$ for $M^tD^{(i)}$
  - we now have $s^{(i)} = K' + E^{(i)}$:
- ▶ Guess-and-determine attack starting at cycle t: $z_t^{(i)}$
  - make hypothesis for 4 bits of K', in mux address positions (!)
  - ... this allows computing corresponding bits of $s^{(i)}$
  - ... and for each, allows appointing $z^{(i)}$ to a bit of $s^{(i)}$
  - each of these n statebits can be converted to K'
    - ▶ if wrong hypothesis, huge probability for inconsistency
    - ▶ if right hypothesis, known part of the state fills up fast
- ▶ Leads to immediate break even if n is quite modest, for any t

- ▶ State update function is linear
  - lightweight and convenient to implement in *constant time*
  - analyzable: length of cycle known in advance
  - but difference between states known after init, remains known forever
- ▶ Mapping from D to initial state is linear
  - simple and cheap
  - but difference of D's known $\rightarrow$ state difference known
- ▶ Output function (multiplexer) is simple
  - compact and cheap
  - but allows for partial reconstruction of state

(1) Introduce non-linearity in state updating function
- irregular clocking: let # LFSR cycles depend on state bit values
- make recursion formula non-linear, e.g., NLFSR

(2) After writing D and K in state, do *blank cycles* (no output)
- non-linearity from D and K to s_t is weak for small t
- but increases fast with growing t
- Note: requires state updating function to be non-linear

(3) Make output function stronger
- *research* has led to many published criteria
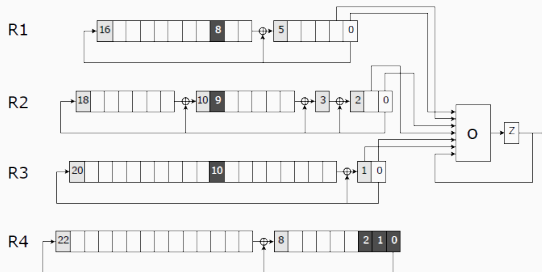- choose an output function guided by these

Alternative approach: build stream cipher from a strong cryptographic primitive, e.g., a block cipher or a cryptographic permutation
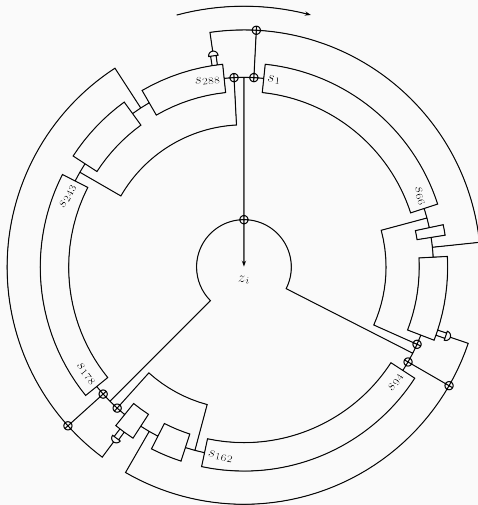
Some real-world stream ciphers

- ▶ In use in hundreds of millions of wireless phones today
- ▶ 4 maximum-length LFSRs with coprime lengths
- ▶ Top 3 clocked 2 or 3 times in between time steps t
- ▶ Bottom LFSR determines clocking of top 3 ones
- ▶ Output function O with 1 bit of *memory*
- ▶ Practically broken with statistical key recovery attack

- claims 80 bits of security
- 80-bit K and 80-bit D
- 288-bit state
- linear output function
- regularly clocked
- non-linearity in update: only 3 AND gates
- output period not known in advance *but likely OK*
- init. takes 1152 cycles
- as yet unbroken

- 5 to 256-byte K, no dedicated D
- State: 256-byte array, 2 pointers
- Software-oriented

- Used in TLS and WEP
- Biases in keystream
- Broken in practice
  [wikipedia]

Key Schedule Algorithm (KSA), initialization:

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Pseudo-random generation algorithm (PRGA), update/output:

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

Native stream ciphers: summary

- Filter and combiner LFSR: mostly of historical significance
  - from days before *resync attacks*
- Irregularly clocked still in use today:
  - DECT, Mifare Classic, GSM A5/1, . . .
  - each and every one of them is broken
- So-called *state-of-the-art*: eSTREAM portfolio
  http://www.ecrypt.eu.org/stream/
  - NLFSR: Trivium, Grain, Snow and Sosemanuk
  - RC4-inspired design: HC-128

  by academics, for academics
- Reality check:
  - stream encryption in practice today: modes of block ciphers
  - future of stream encryption: modes of permutations

  (both treated later in this course)
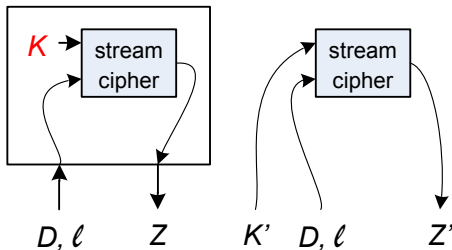
Modelling attacks on stream ciphers
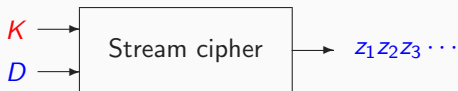
- ▶ Generates keystream bits z_t from
  - K: secret, typically 128 or 256 bits
  - D: diversifier, for generating multiple keystreams per key
- ▶ z_t can be a bit or a sequence of bits, e.g., a byte or a 32-bit word

We can formally write (asking for ℓ output bits):

$$Z = \mathsf{SC}_K(D, \ell)$$

- ▶ Adversary has query access to:
  - SC_K: stream cipher instance with unknown key K
  - $SC_{K'}$: stream cipher instance with chosen key K'
- ▶ Can make queries Q
  - Q_d: queries to SC_K with cost (e.g., total length) M
  - Q_c: queries to $SC_{K'}$ with cost N
- ▶ Express probability of success as function of M and N
- ▶ Example: generic exhaustive key search: $\Pr(\text{success}) = N2^{-|K|}$ with N number of key-trial queries to $SC_{K'}$
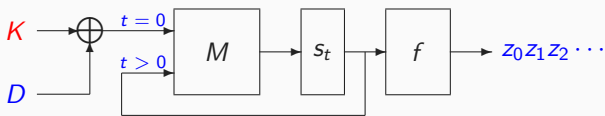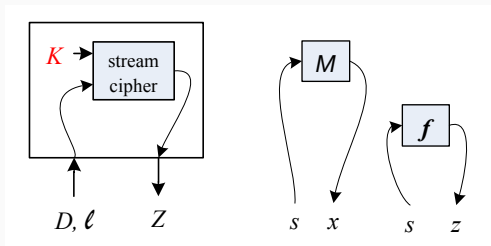
- ▶ Operates on an evolving state s_t
- ▶ In our multiplexer LFSR example:
  - State update function $s_t = M s_{t-1}$: LFSR update
  - Output function $z_t = f(s_t)$: multiplexer
- ▶ More in general
  - State update function $s_t = F(s_{t-1})$
  - Output function: $z_t = f(s_t)$

- ▶ Limitation of previous model: can only describe *generic* attacks
  - generic means: making abstraction of the inner working
- ▶ We give adversary query access to inner functions F (here M) and f
  - models the fact that these are *public* (see Kerckhoffs)
- ▶ Breakdown can be even more fine-grained
  - queries to inner functions finally become *computation*
  - . . . with some measure of computational effort
- ▶ When do we consider a stream cipher *secure*?
  - if no attacks with success probability above the one in claim!
  - but what would be reasonable to claim in the first place? 14

The Random Oracle

- What would the ideal cryptographic function look like?
- It is called a Random Oracle (RO)
- Random Oracle can be built, but is not practical
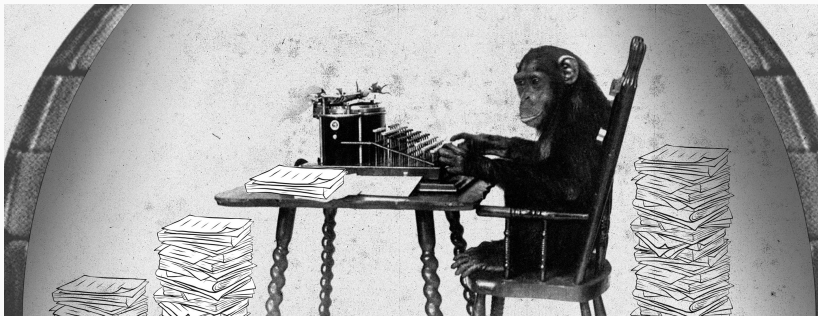
Random Oracle Inc.: letter answering service!

1) (m, ℓ) arrives at Random Oracle Inc., with

- m: the letter
- ℓ: the length of the requested answer

2) Frank checks archive for presence of a file (m, Z)

3) Employee Cheetah is put at work
   a) if no (m, Z) in archive, Cheetah types random Z with $|Z| = \ell$
   b) else if $|Z| < \ell$, Cheetah extends Z to length ℓ with random string
   c) else Cheetah takes a break and eats a banana

4) Frank copies Z

5) Frank puts file with (m, Z) (back) in archive

5) Frank sends response Z truncated to length ℓ to sender

Random Oracle returns unrelated responses for different inputs m

- ▶ Say:
  - Alice wants so send messages P_i confidentially to Bob
  - Both Alice and Bob can query $\mathcal{RO}$, but nobody else can
- ▶ For message P_i:
  - Alice queries $\mathcal{RO}$ with $(i, |P_i|)$ and $\mathcal{RO}$ returns Z_i
  - Alice enciphers P_i to $C_i \leftarrow P_i + Z_i$
  - Alice sends (i, C_i) to Bob
  - Bob queries $\mathcal{RO}$ with $(i, |C_i|)$ and $\mathcal{RO}$ returns Z_i
  - Bob deciphers C_i to $P_i \leftarrow C_i + Z_i$
- ▶ The $\mathcal{RO}$ returns a one-time pad, so provides perfect secrecy
- ▶ If we have stream cipher that, if keyed, is indistinguishable from $\mathcal{RO}$
  - Alice and Bob use that to get Z_i instead of $\mathcal{RO}$ and it's OK!
  - that suggests a useful security goal for a stream cipher

Indistinguishability security notion
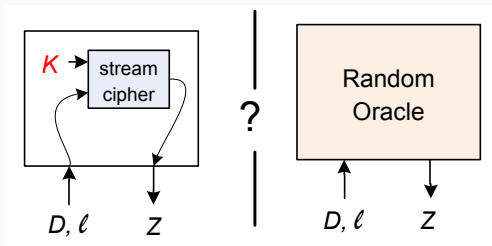
Distinguishing game (black box version):

▶ Adversary $\mathcal{A}$ has query access to a system that is either:
  • SC_K: stream cipher with unknown key K
  • $\mathcal{RO}$: ideal stream cipher in the form of a random oracle
▶ She does not know which one and has to guess that
▶ Adversary $\mathcal{A}$ is actually an *attack algorithm* that returns either:
  • 1 if it estimates the system is SC_K
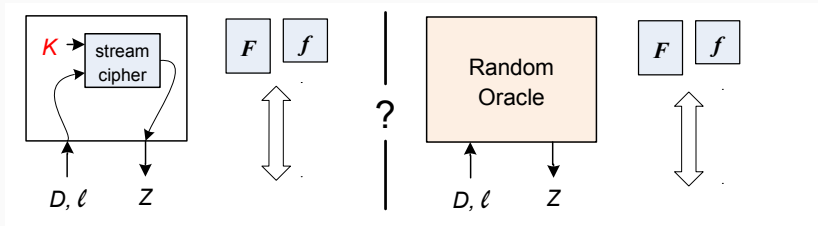 • 0 if it estimates the system is $\mathcal{RO}$
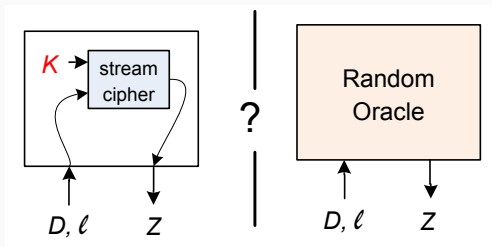
- ▶ Let:
  - $\Pr(\mathcal{A} = 1 \mid \mathsf{SC}_K)$: probability that $\mathcal{A}$ returns 1 in case of SC_K
  - $\Pr(\mathcal{A} = 1 \mid \mathcal{RO})$: probability that $\mathcal{A}$ returns 1 in case of $\mathcal{RO}$

Advantage of an adversary $\mathcal{A}$

$\mathrm{Adv}_{\mathcal{A}} = |\Pr(\mathcal{A} = 1 \mid \mathsf{SC}_K) - \Pr(\mathcal{A} = 1 \mid \mathcal{RO})|$
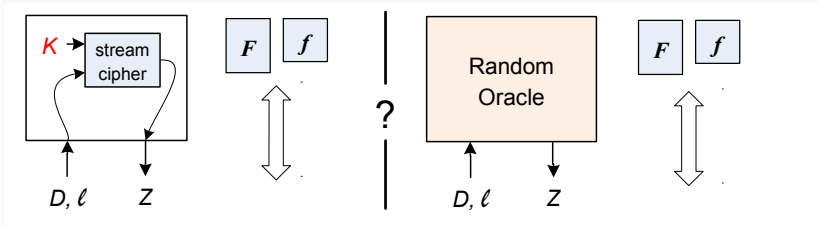
Note: $\mathrm{Adv}_{\mathcal{A}}$ is in interval $[0 \dots 1]$

- ▶ Black box fails to model that F and f are public
- ▶ We give additional query access to F and f
- ▶ We model query complexity in two parts again:
  - • M: called online or data complexity
  - • N: called offline or computational complexity
- ▶ We express $\mathrm{Adv}_{\mathcal{A}}$ as $\epsilon(M, N)$

$\epsilon(M, N)$ **indistinguishability claim for a stream cipher** SC

There exists no attack algorithm $\mathcal{A}$ that distinguishes SC_K, with K a uniformly chosen unknown key, from a random oracle with

$\text{Adv}_{\mathcal{A}} > \epsilon(M, N)$

Note: this is a very powerful type of claim

Implications of $\mathcal{RO}$ indistinguishability

A $\epsilon(M, N)$ indistinguishability claim implies:

- There are no key recovery attacks with success above $\epsilon(M, N)$
- Probability that keystream Z is periodic is below $\epsilon(M, N)$
- Success in exploiting biases in Z limited to $\epsilon(M, N)$,
- ...

Implications of a $\epsilon(M, N)$ indistinguishability claim

It claims for any imaginable attack:

$\Pr(\text{success of attack on } SC_K) \leq \epsilon(M, N) + \Pr(\text{success of attack on } \mathcal{RO})$

- Recipe for distinguishing adversary $\mathcal{A}$ based on the attack:
  - (1) Spend resources N and M on the attack
  - (2) If it works, return 1, else return 0
- $\Pr(\mathcal{A} = 1 \mid \mathcal{RO}) = \Pr(\text{success of attack on } \mathcal{RO})$
- $\Pr(\mathcal{A} = 1 \mid SC_K) = \Pr(\text{success of attack on } SC_K)$
- Due to the claim their difference is at most $\epsilon(M, N)$

Conclusion: what is a secure stream cipher?

- ▶ A stream cipher that, when keyed with a fixed and unknown key K, is hard to distinguish from a random oracle
- ▶ How hard it actually is, is expressed by a bound on the advantage
- ▶ We cannot prove such bounds for concrete stream ciphers
- ▶ But we can make claims and assumptions
- ▶ For stream ciphers built on an underlying primitive we can prove conditional bounds . . .