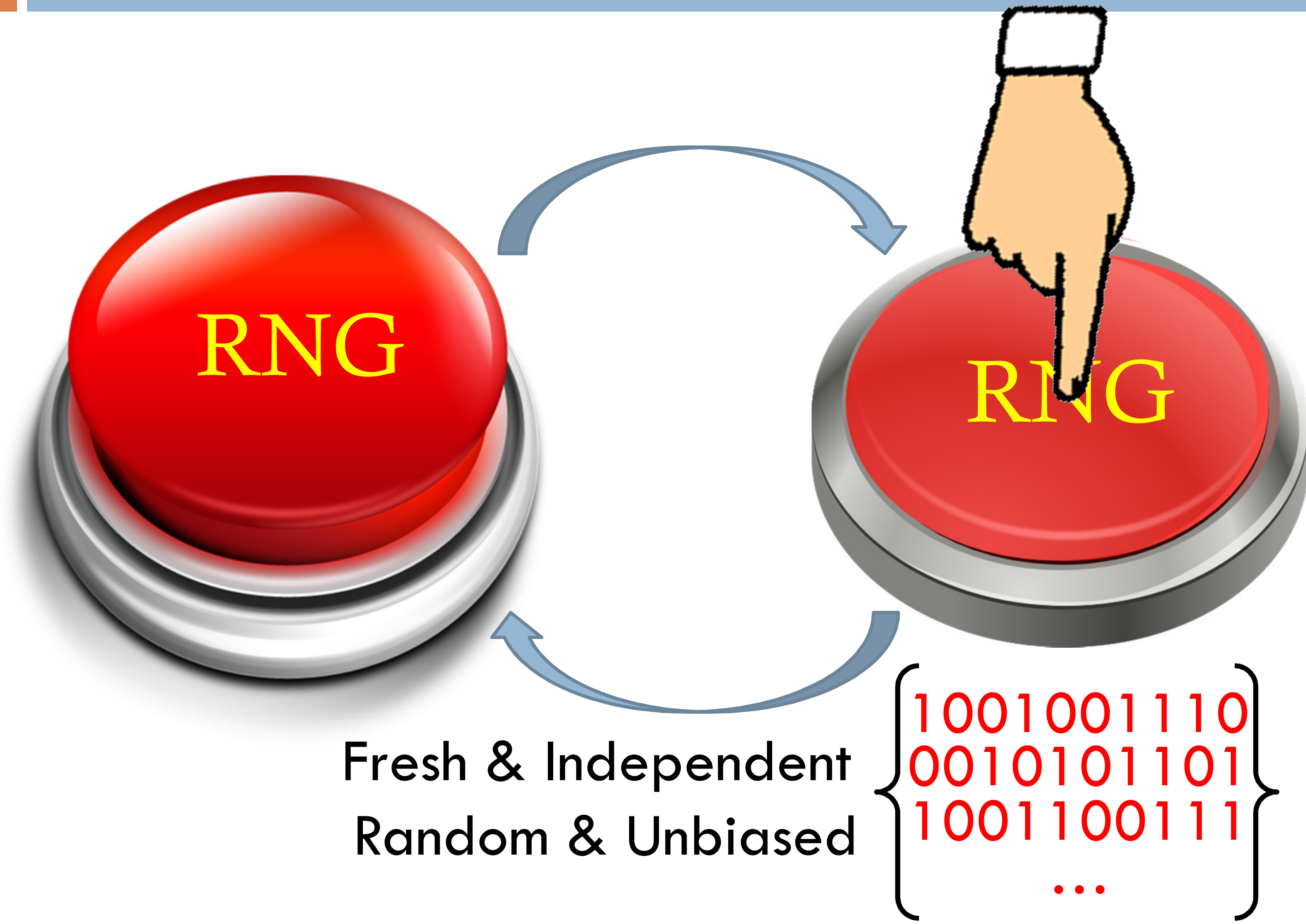
RANDOM NUMBER GENERATION, REVISITED

Joint work with David Pointcheval, Sylvain Ruhault, Damien Vergnaud and Daniel Wichs

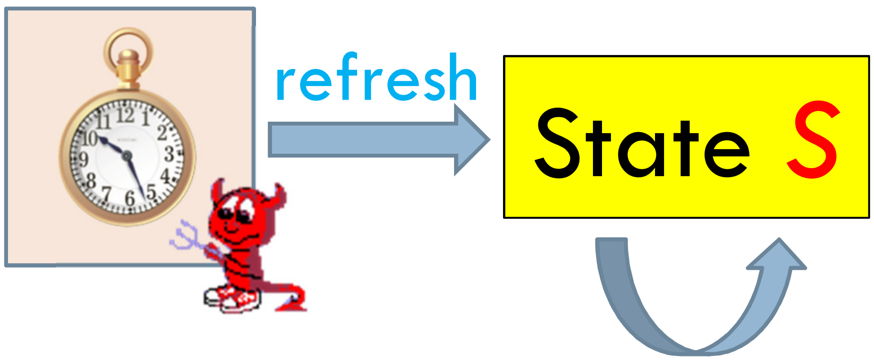
Yevgeniy Dodis (New York University)

Random Number Generators (RNGs)



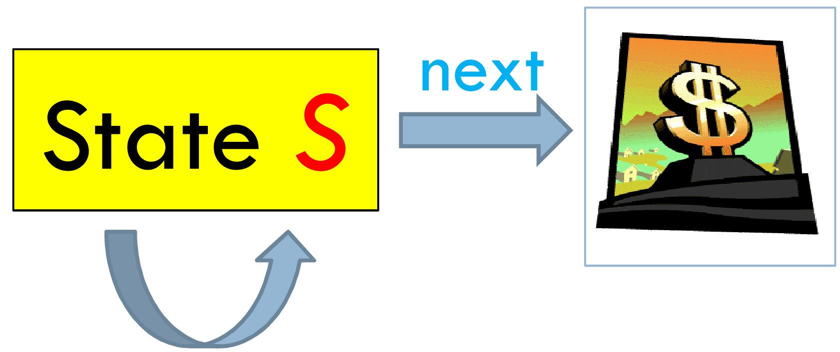
Random Number Generators (RNGs)

Input I



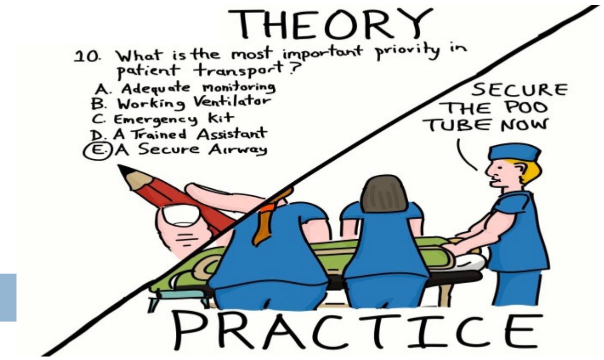
- refresh(current S, I) = new S
  - runs in background
  - input I possibly adversarial (but must "have entropy")
  - Goal: "entropy accumulation"

Output R



- next(current S) = (new S, R)
  - runs when called by user
  - output R "looks random" (if "not compromised")

Theory vs. Practice



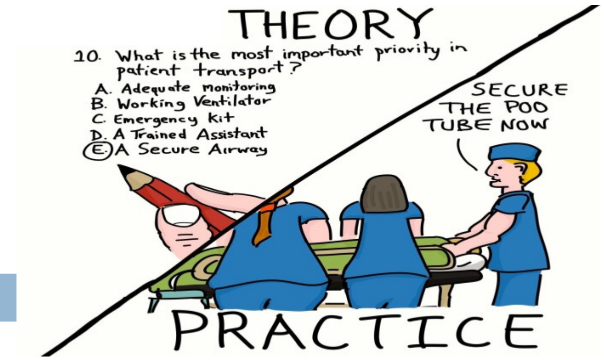
Case study: Linux /dev/random

- complex: over 800 lines of code
- "security-by-obscurity" (appears)
  - everything ad hoc and heuristic
  - uses "cryptographic hashing" (SHA1), but in ad hoc manner
- keeps multiple "entropy pools"
- (most complex) key components:
  - heuristic "mixing function" *M*
  - ad-hoc "entropy estimation" *E*
- completely unintuitive
- no security proof

Case study: [BH05] RNG

- formal, intuitive model
- simple, natural construction
  - much simpler than "practice"
  - elementary security proof
- "trivialize" the heart of real-world RNGs:
  - no entropy estimation, entropy pools or mixing function
  - strong advice against entropy estimation
- no "entropy accumulation" (model or construction)

Theory vs. Practice



Case study: Linux /dev/random

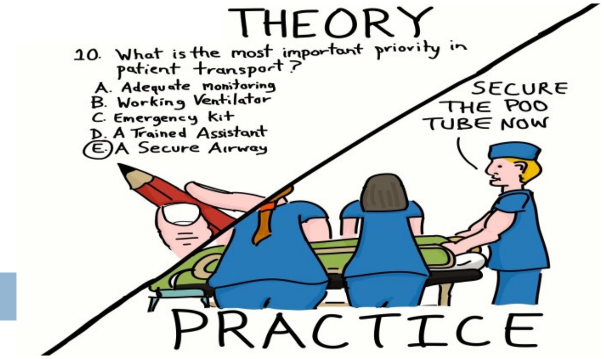
- complex
- "security
  - every
  - uses " but in
- keeps m
- (most co
  - heuristic "mixing function" M
  - ad-hoc "entropy estimation" E
- completely unintuitive
- no security proof

Case study: [BH05] RNG

- el
- uction
- actice"
- roof
- of real-
- , entropy
- xing function
  - advice against entropy es ation
- no "entropy accumulation" (model or construction)

Recover from compromise as long as the *total* amount of fresh entropy accumulated over some *potentially long* period time crosses a threshold e^*

Theory vs. Practice



Case study: Linux /dev/random

- Good security intuition, but too complex, and too much reliance on heuristics (security-by-obscurity)

Case study: [BH05] RNG

- Nice and clean, but "over-simplified" reality, failing to account for a key security concern

Our Results

- New rigorous model for RNG security
  - Captures "entropy accumulation" (and more)
  - Explicit (adversarial) "distribution sampler" 
- Explicit attacks on both theory (Barak-Halevi) and practice (Linux /dev/random)
- Provably Secure Construction
  - As simple/efficient as Barak-Halevi (+ secure)
  - Cleaner and more efficient than /dev/random

Our RNG Model

- Two adversaries:  and 
-  : Distribution sampler *D* ("Devil")
  - outputs "entropic" inputs I_1, I_2, … (and more)
  - explicitly models (adversarial) "nature"
-  : (traditional) Attacker *A* ("Alice")
  - tries to distinguish outputs of RNG from truly random strings (when RNG is "uncompromised")
  - has power to "compromise" RNG or call 

Provably Secure Construction (simplified)

- Let k – security parameter, n = e* = $3k$
- chop$_k$(x) – truncation of n-bit string x to k bits
- $\mathbf{G}:\{0,1\}^k \rightarrow \{0,1\}^{4k}$ pseudorandom generator
- Define RNG= (*setup*, *refresh*, *next*) as follows (here *length*(*S*) = *length*(*I*) = n, *length*(*R*)=k):
  - *setup*(): output random n-bit string *x*,*y*
  - *refresh*$_{x,y}$(*S*,*I*): set $S \leftarrow S \cdot x + I$ (multiply in $\mathrm{GF}[2^n]$)
  - *next*$_{x,y}$(*S*): set $(S,R) \leftarrow \mathbf{G}(\mathrm{chop}_k(S \cdot y))$

Lessons Learned

- Security-by-obscurity is so 20-th century!
- We can do better now!

Paper to appear at CCS'2013

Full version available at

http://eprint.iacr.org/2013/338