Today in Cryptography (5830)

Random number generators
Pseudorandom number generators

TLS handshake for RSA transport

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Pick random Ns

Check CERT using CA public verification key

← CERT = (pk of bank, signature over it)

Pick random PMS
C <- E(pk,PMS)

C →

PMS <- D(sk,C)

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

Random number generator (RNG) → R

```
MD_Update(&m,buf,j);
....
MD_Update(&m,buf,j);   /* purify complains */
```

These lines of code commented out from OpenSSL random number generator code (md_rand.c) to **address complaints by security tools Purify and Valgrind**

Only the process ID (PID) was used as input to RNG.

It took a ~2 years for the bug to be (publicly) discovered!

Ra

[W
[G
[G
[D
[W
[B
[M
[Abe
[Yilek et al. 2009]
[Heninger et al. 2012]
[Everspaugh et al. 2015]

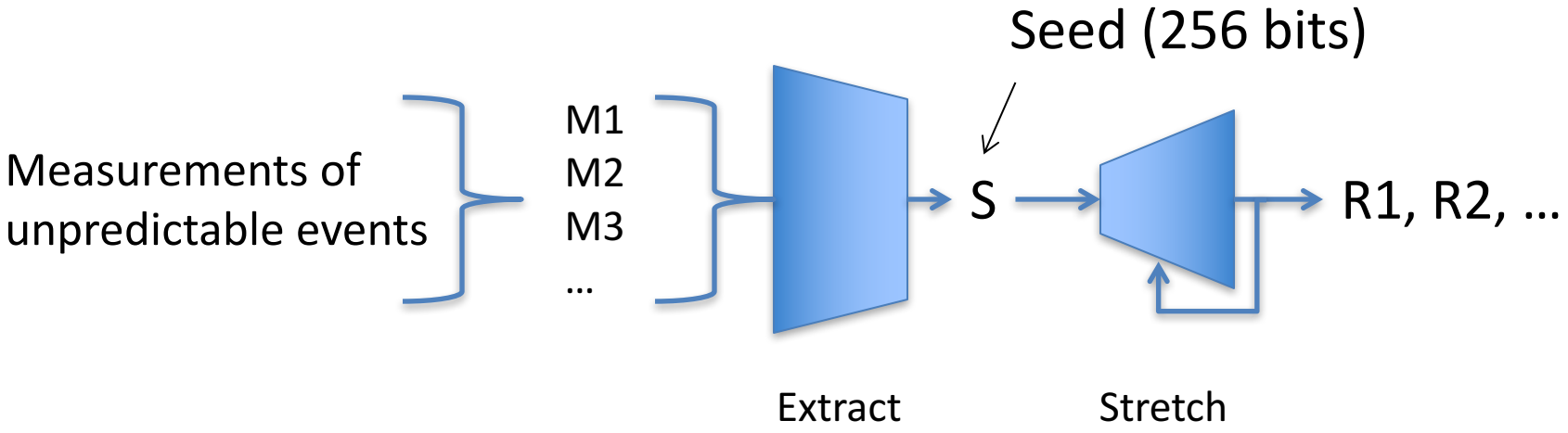
Debian OpenSSL bug lead to small set of possible R

Cryptographically strong randomness

- Must be maximally unpredictable from adversary's perspective
- This means (computationally) indistinguishable from uniform bit string of same length
- "True" randomness vs. cryptographic randomness
  - Typically false dichotomy in practice

RNG pipelines

1. Entropy gathering
2. Extracting from measurements a cryptographically strong value called seed
3. Using seed to deterministically produce pseudorandom values

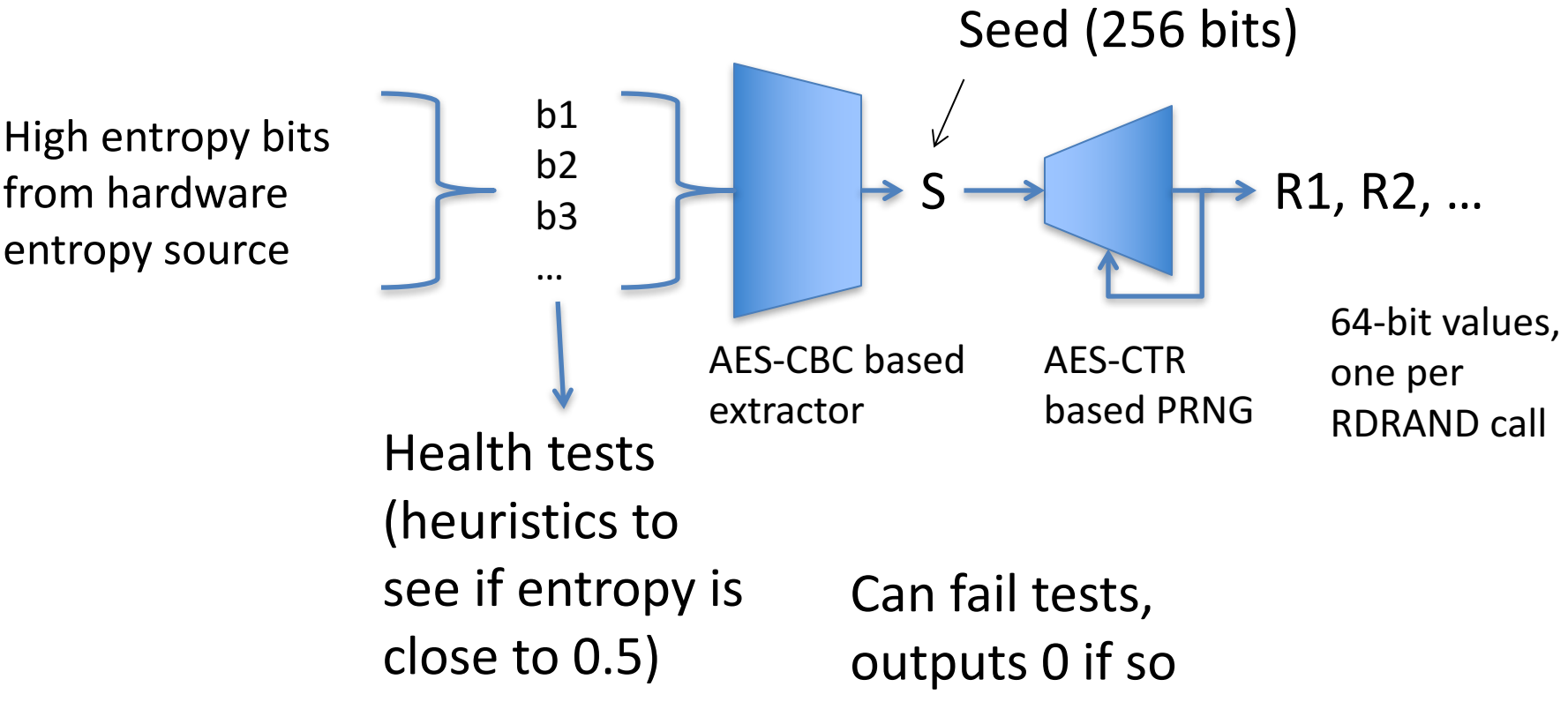


Entropy sources

- Timing and description of various events
  - keyboard presses and timing
  - file/network interrupts
  - mouse movements
- Hardware RNGs
  - Intel RNG has custom hardware for generating unpredictable bits using thermal noise
- Health tests

Intel RNG system

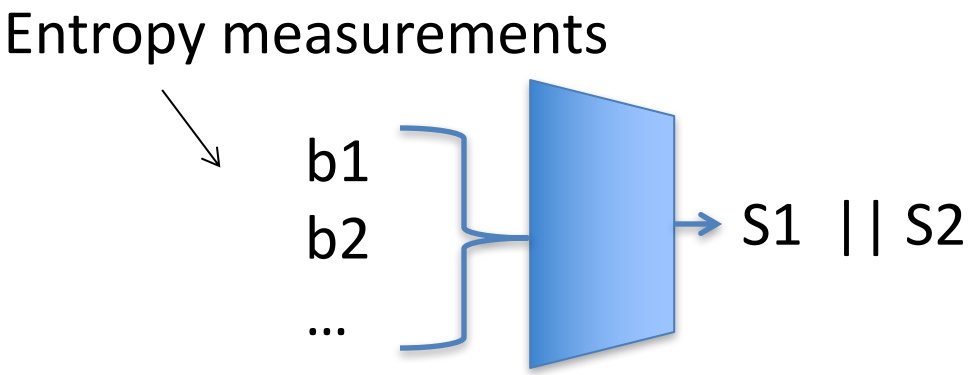
512 bits collected per go

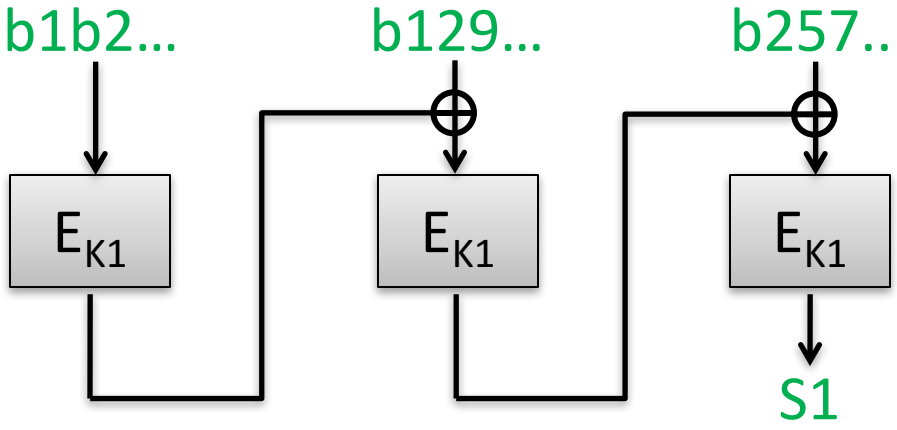


Good writeup: http://eprint.iacr.org/2014/504.pdf

AES CBC MAC as an extractor

K1 = AES(0,1)

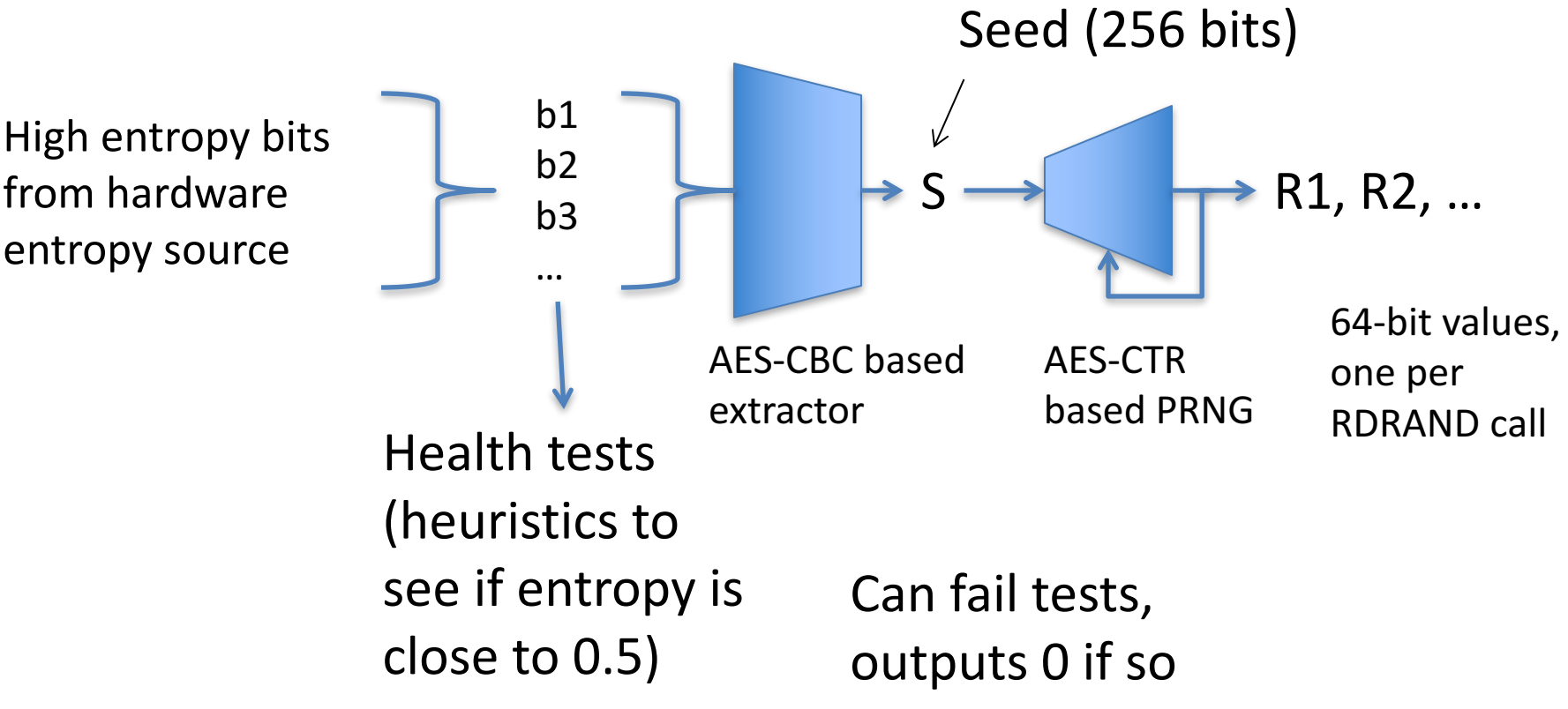




Repeat process of collecting entropy values and CBC-MACing to get S2

Intel RNG system

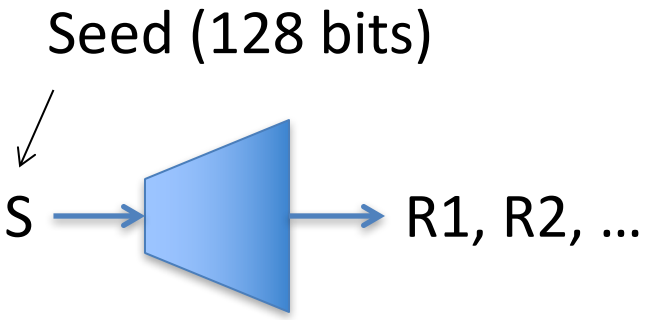
512 bits collected per go

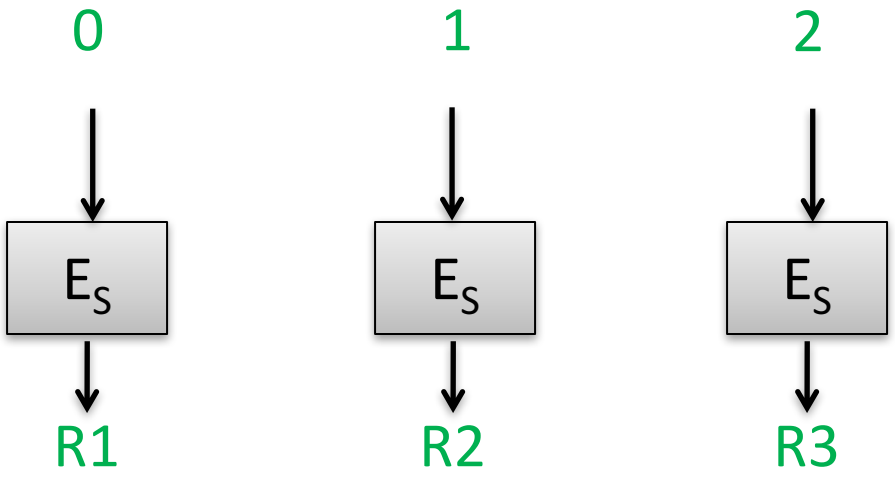


Good writeup: http://eprint.iacr.org/2014/504.pdf

AES CTR mode as PRG

AES-CTR(S) -> R1, R2, R3…

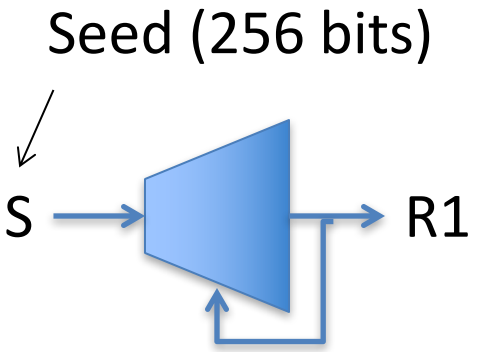


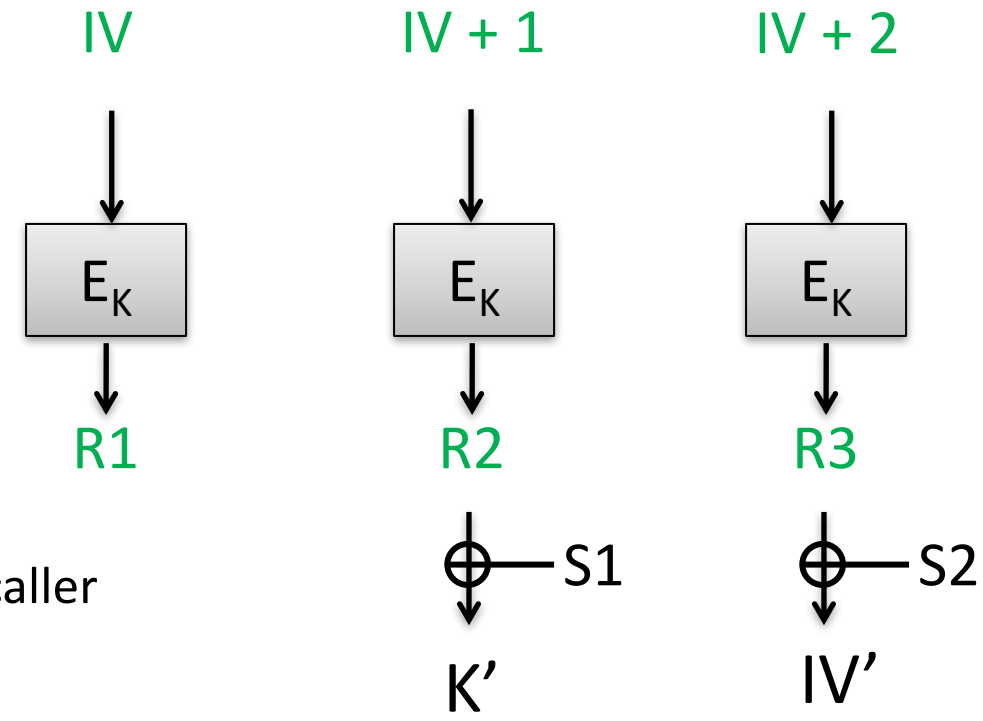


AES CTR mode in Intel RNG

AES-CTR(K,IV,S) -> R1, K', IV'

S = S1 || S2 (128 bits each)



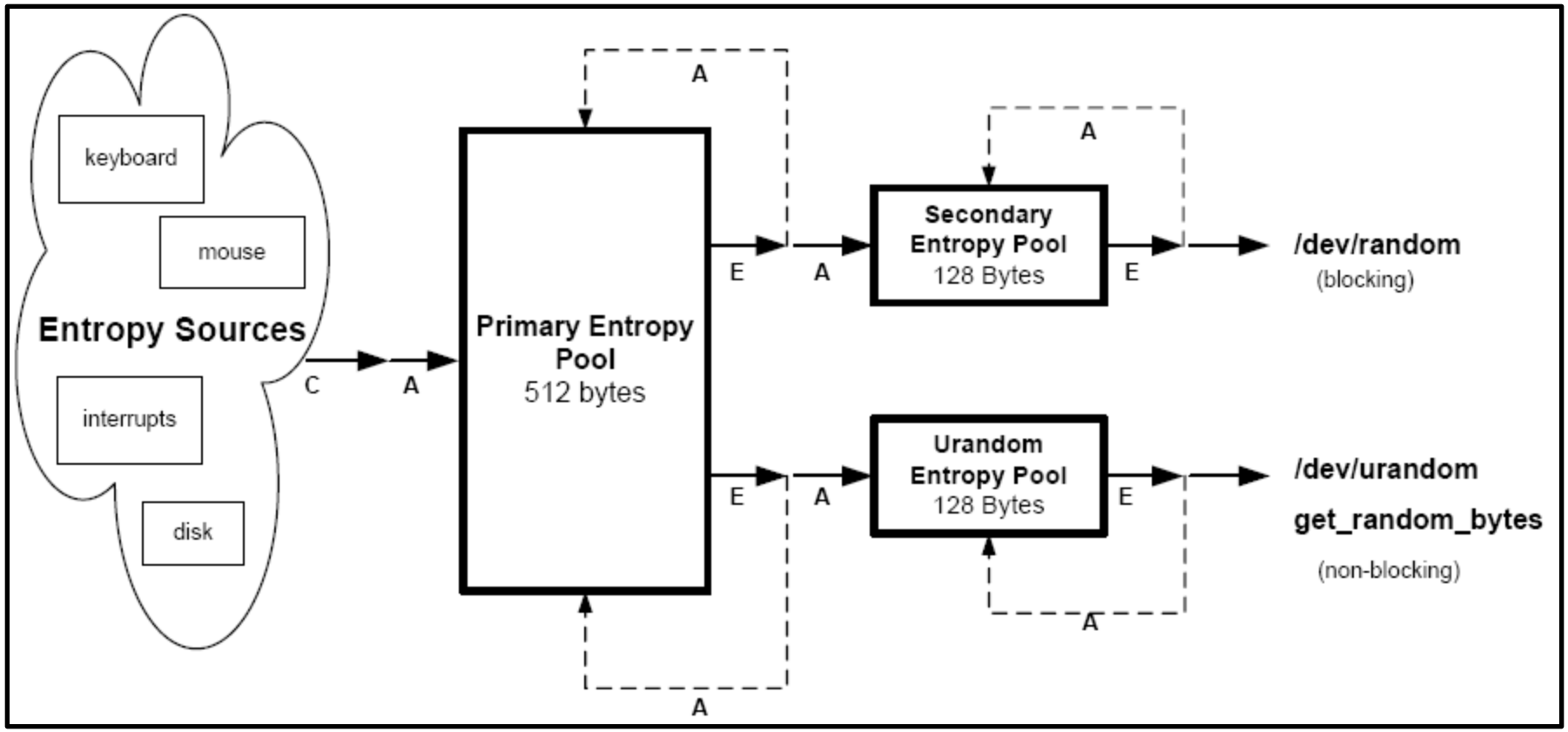


R1 output to caller of instruction

Linux /dev/(u)random

Linux random number generator (2500 lines of undocumented code)

Diagram from [Gutterman, Pinkas, Reinman 2006]



Primary entropy pool feeds into other entropy pools only when 192 bits of entropy are estimated. Favors /dev/random

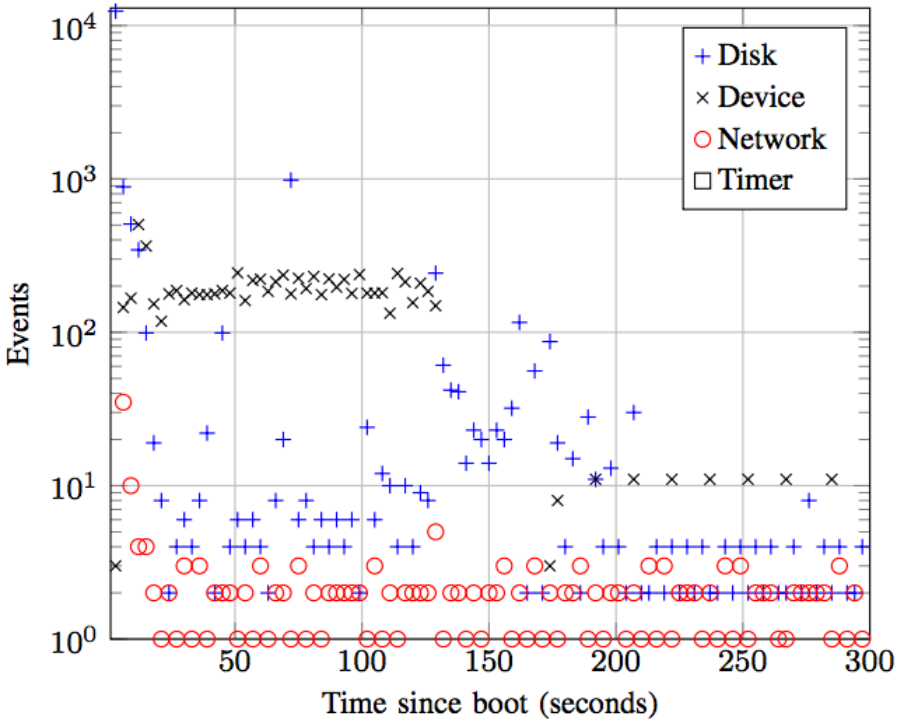
Questions

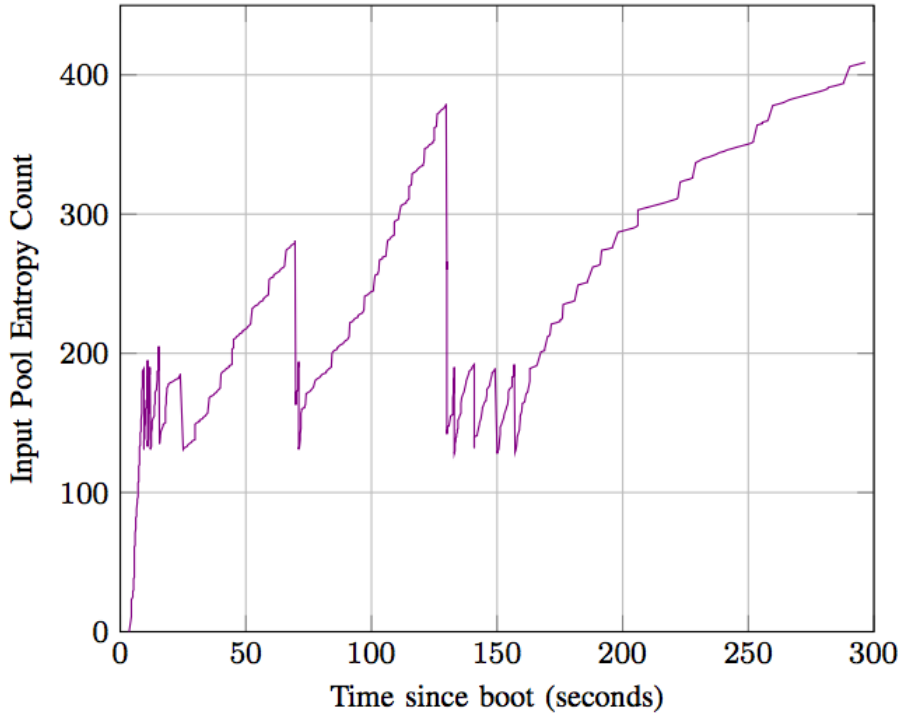
- Does /dev/(u)random collect sufficient entropy during boot?
  - Stamos, Becherer, and Wilcox conjecture not in virtualized environments (BlackHat 2009)
- What happens when a full-state snapshot is resumed?

We carefully instrumented Linux kernel to track entropy accumulation

http://pages.cs.wisc.edu/~ace/papers/not-so-random.pdf

Entropy accumulation during boot of Linux VM within VMWare





We analysis suggests that, after first use of /dev/urandom during boot, entropy is sufficient to prevent attacks

Boot-time entropy holes

First read from /dev/urandom before any entropy inputs.
Output is always: 0x22DAE2A8 862AAA4E

Combined with cycle counter to seed stack canary on init process
Not clear how to exploit directly

Embedded systems also exhibit boot-time entropy holes:

- urandom entropy pool not updated long into boot
- ssh keys generated on first boot --- broken!

https://factorable.net/weakkeys12.extended.pdf

Questions

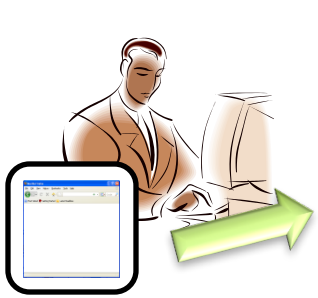
- Does /dev/(u)random collect sufficient entropy during boot?
  - Stamos, Becherer, and Wilcox conjecture not in virtualized environments (BlackHat 2009)
- What happens when a full-state snapshot is resumed?

We carefully instrumented Linux kernel to track entropy accumulation

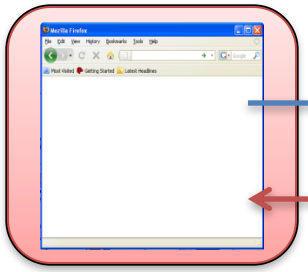
http://pages.cs.wisc.edu/~ace/papers/not-so-random.pdf

Virtual machines and secure browsing

"**Protect Against Adware and Spyware:** Users protect their PCs against adware, spyware and other malware while browsing the Internet with Firefox in a virtual machine."
[http://www.vmware.com/company/news/releases/player.html]

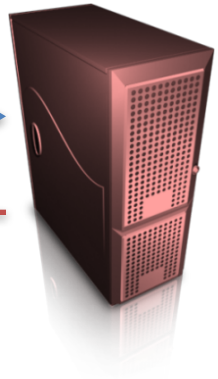


Clean snapshot of VM with browser running

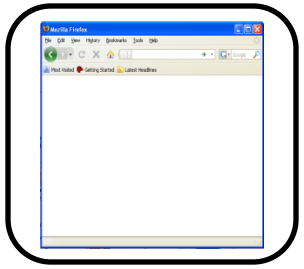


http://www.freesoftware.com/

browser exploit

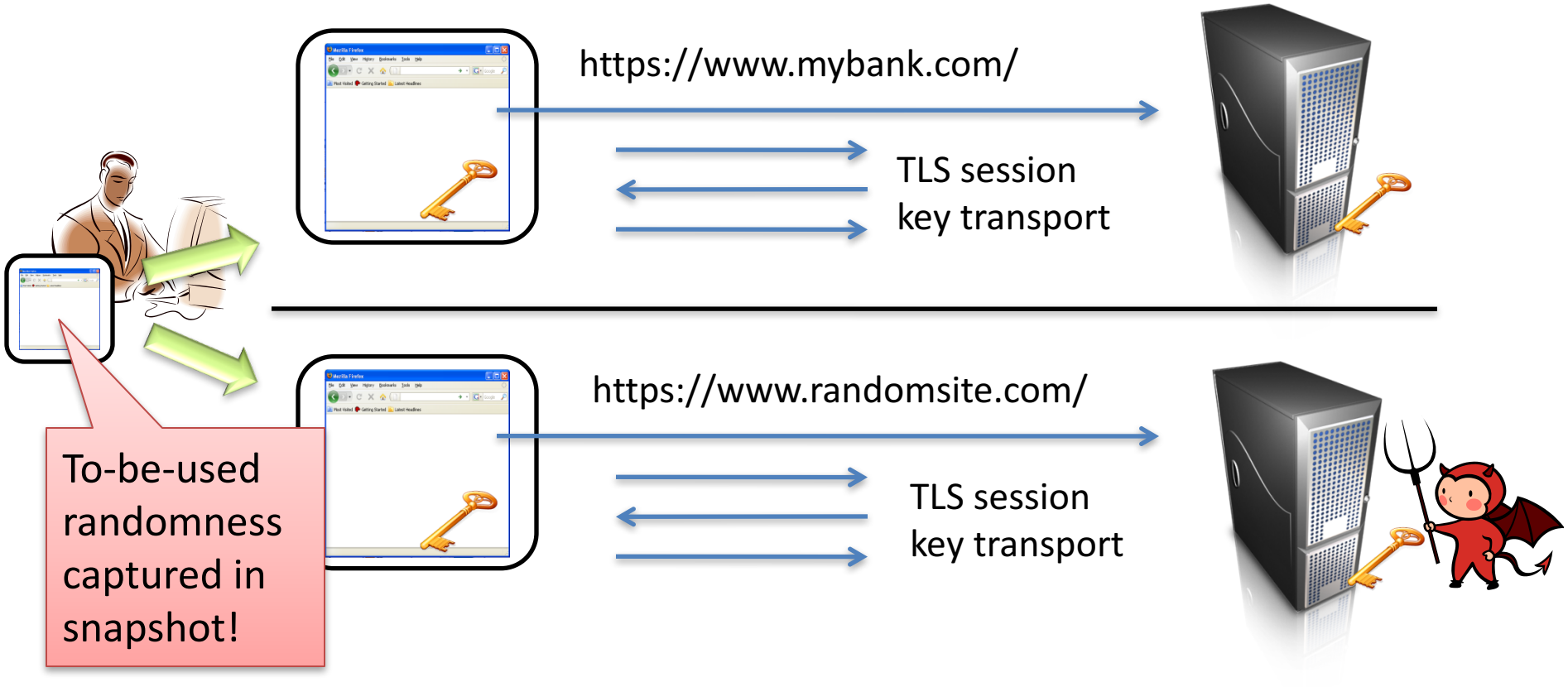


Virtual machine compromised, but not host OS



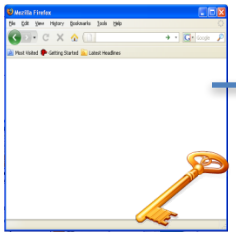
Resetting to snapshot removes malware

Virtual machine resets lead to RNG failures for applications



Older versions of Firefox, Chrome allow session compromise attacks

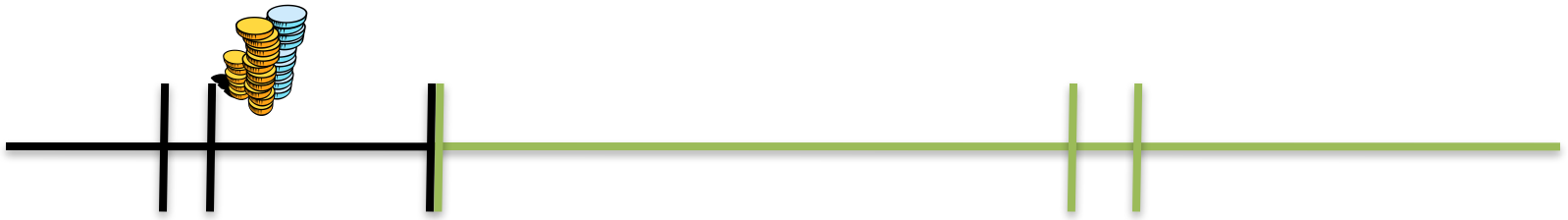
Apache mod_ssl TLS server: server's secret DSA key can be stolen!



https://www.mybank.com/

TLS session
key transport

A logical timeline of events



User launches browser in VM

User snapshots VM

Snapshot later run.

User requests https page

Randomness gathered by browser random number generator (RNG)

Randomness used by TLS key transport

TLS key transport client

(N,e)

RSA PKCS#1

C

C sent to server

A second run from snapshot leads to same secret key being sent to (different) server

Reset vulnerabilities when using /dev/urandom after resumption?

- We showed that Linux /dev/(u)random and Windows system RNG are also vulnerable to resets
  - openssl genrsa will sometimes use repeat randomness (if ALSR is turned off, always)
- Primary problem is pooling structure of /dev/(u)random
- Changes have been made to Windows to fix

Using RNGs

- Rule of thumb: more entropy is better
- In consuming applications:
  - Call cryptographically strong RNG such as /dev/urandom or Intel RDRAND
  - Mix in local entropy if you have any
  - Hash it all together with cryptographic hash function to derive randomness to use
  - Minimize time between collection and use
- If efficiency is problem, use your own PRG seeded with above (be careful of reset vulnerabilities!)