CS 5830
Cryptography

Instructor: Tom Ristenpart

TAs: Yan Ji, Sanketh Menda

CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Last time: passwords

- Passwords widely used in practice
- Never store passwords in clear, should at least be salted and slow hashed (PBKDF)
  - Memory-hard hashes are state-of-the art (scrypt)
- Password breach alerting increasingly used to warn users about exposed passwords
  - Cloudflare product that I helped with: https://blog.cloudflare.com/privacy-preserving-compromised-credential-checking/
- Password-based encryption should derive key for AEAD scheme using PBKDF
  - Beware of what you see on internet about applied crypto

Today: random number generation

- Cryptography relies on random numbers. What have we been using RNGs for?
  - Secret key generation
  - IVs for AEAD modes (CTR mode, CBC mode, AES-GCM nonces)
  - Salts for password hashing

TLS 1.2 handshake for RSA transport

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods

ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Pick random Ns

CERT = (pk of bank, signature over it)

Check CERT
using CA public
verification key

C

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

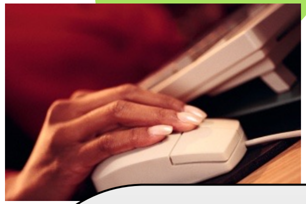
PMS <- D(sk,C)

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

Random

[Wagner, G

[Guttermar

[Guttermar

[Dorrendor

[Woolley et

[Bello 2008

[Mueller 20

[Abeni et al.

[Yilek et al. 2009]

[Heninger et al. 2012]

[Everspaugh et al. 2015]

```
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!

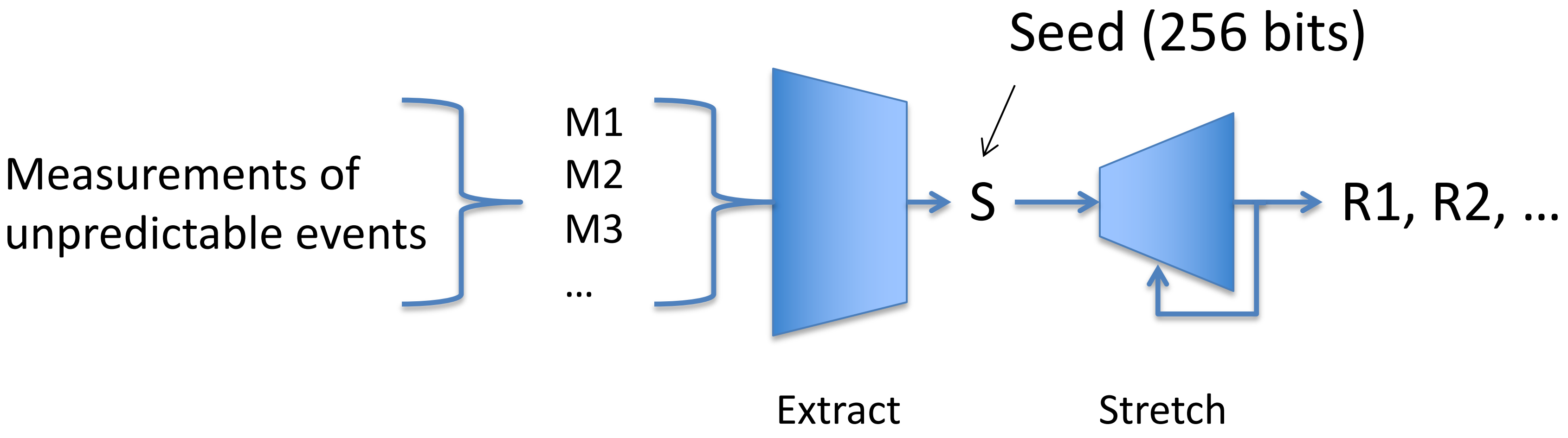
Debian OpenSSL bug lead to small set of possible R

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



Extract-and-Expand approaches: HKDF

Hash-based key derivation function (HKDF)

- Derive more key material from one key
- Extract key from non-uniform key material

Uses HMAC with underlying hash function (SHA256)

<u>HKDF(L,K,salt,info):</u>

prk <- HMAC(salt,K)

K_0 <- empty byte string

m <- ceil(L/hashLen)

For i = 1 to m

  K_i <- HMAC(prk,K_{i-1} || info || <i +1>)

Return $\text{truncate}_L(K_1 \,||\, \ldots \,||\, K_m)$

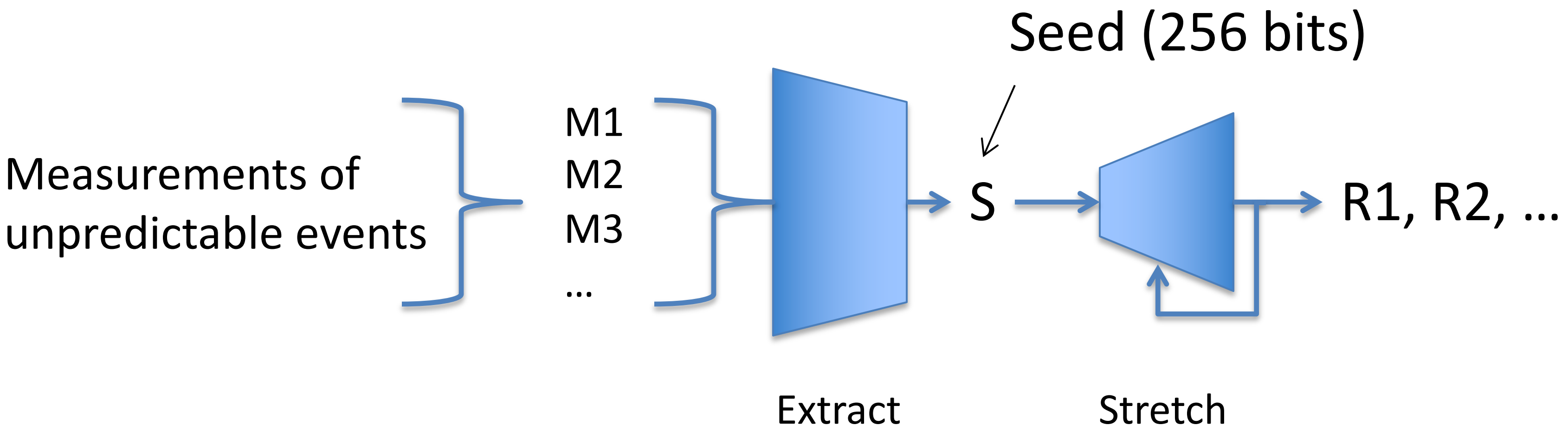
https://datatracker.ietf.org/doc/html/rfc5869

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
  - RDRAND instruction
- Health tests

How can we test security of an RNG?

- Statistical tests
  - Diehard tests, NIST 800-22 tests
- Apply to inputs or to outputs?
  - Careful of applying to outputs: H(0), H(1), H(2), … may pass tests

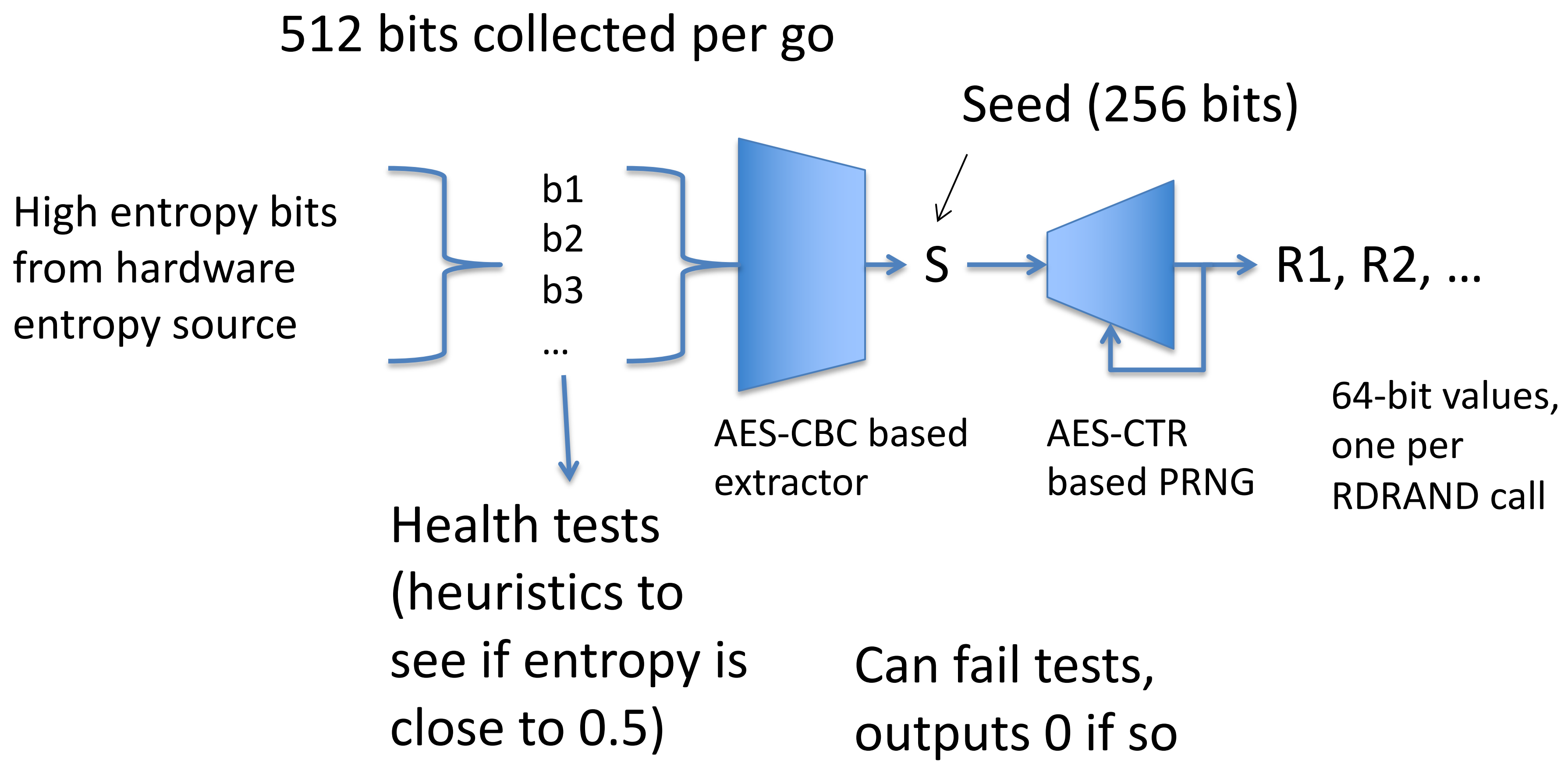


Testing randomness

- "However, no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application, i.e., statistical testing cannot serve as a substitute for cryptanalysis."   from NIST 800-22 (https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf)

- Tests can help catch problems, but best bet is to try to build attacks against specific RNGs

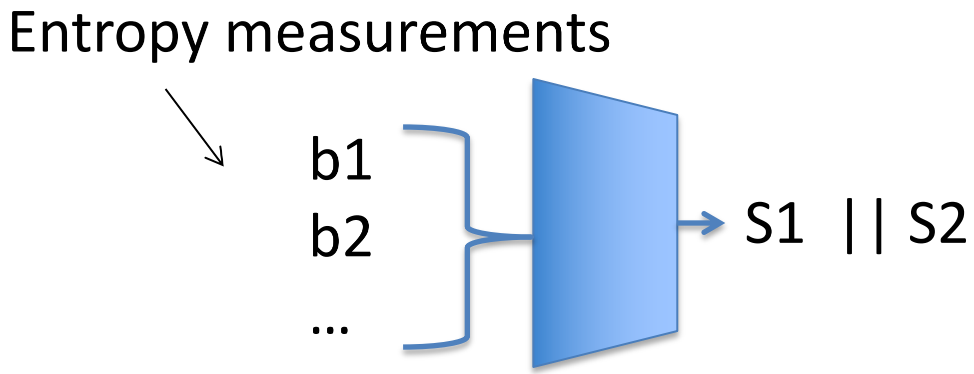
Intel RNG system

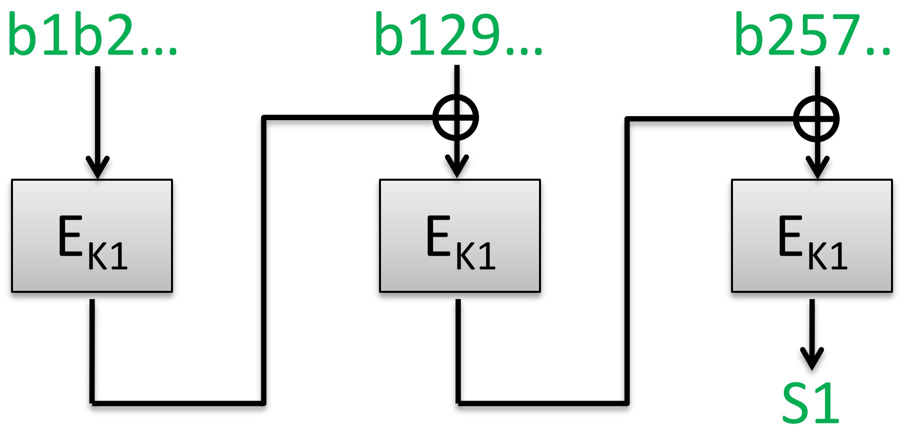


Good writeup: http://eprint.iacr.org/2014/504.pdf

AES CBC MAC as an extractor

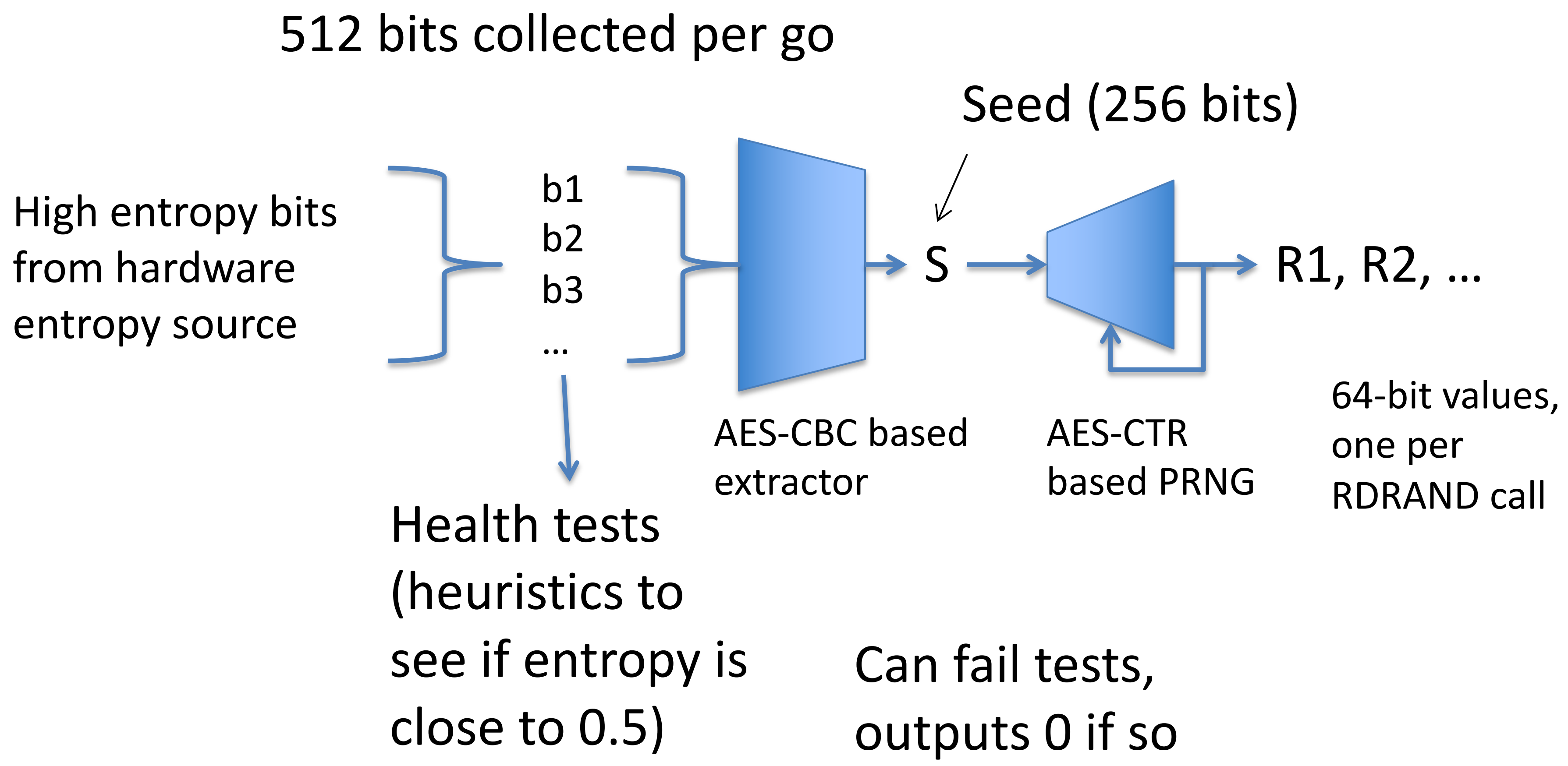
$K1 = AES(0^n, 1^n)$





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

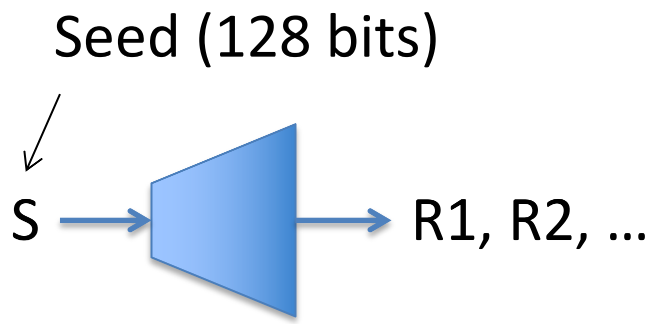
Intel RNG system

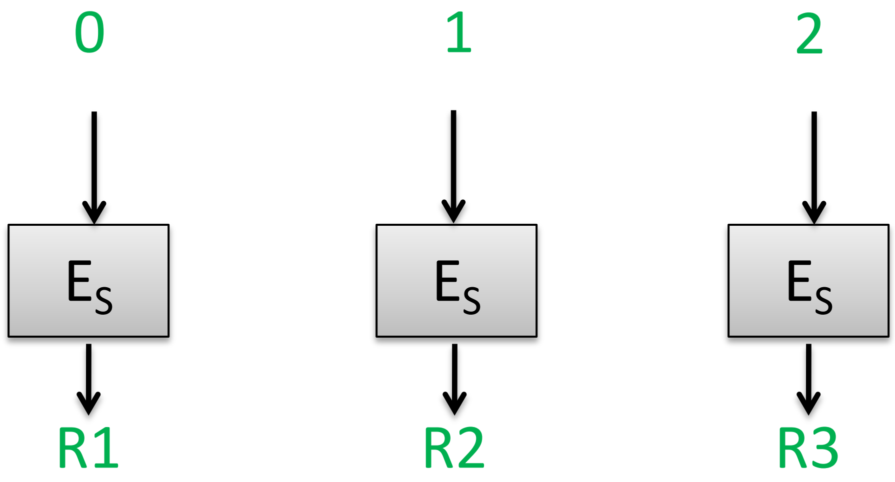


Good writeup: http://eprint.iacr.org/2014/504.pdf

AES CTR mode as PRG

AES-CTR(S) -> R1, R2, R3…



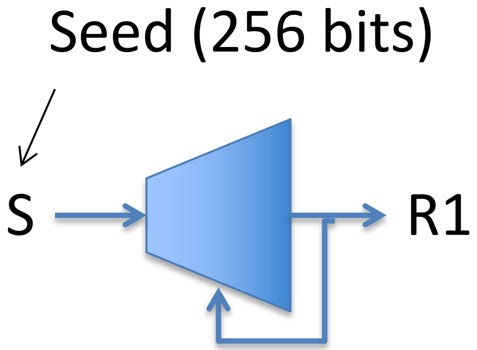


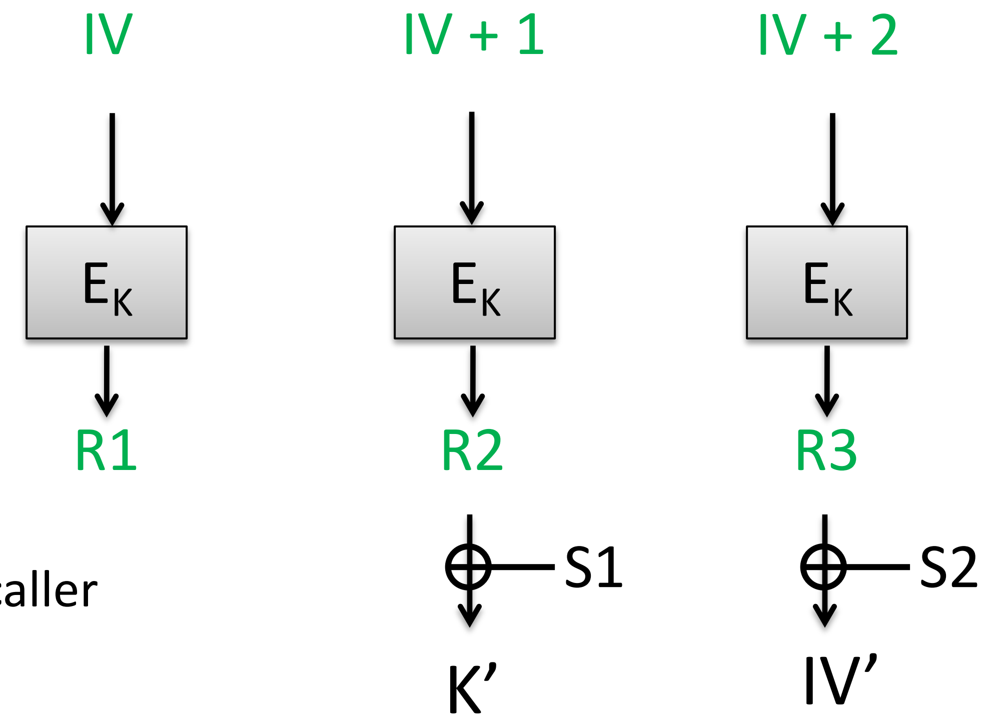
AES CTR mode in Intel RNG

AES-CTR(K,IV,S) -> R1, K’, IV’

K, IV initially all zeros

S = S1 || S2   (128 bits each)



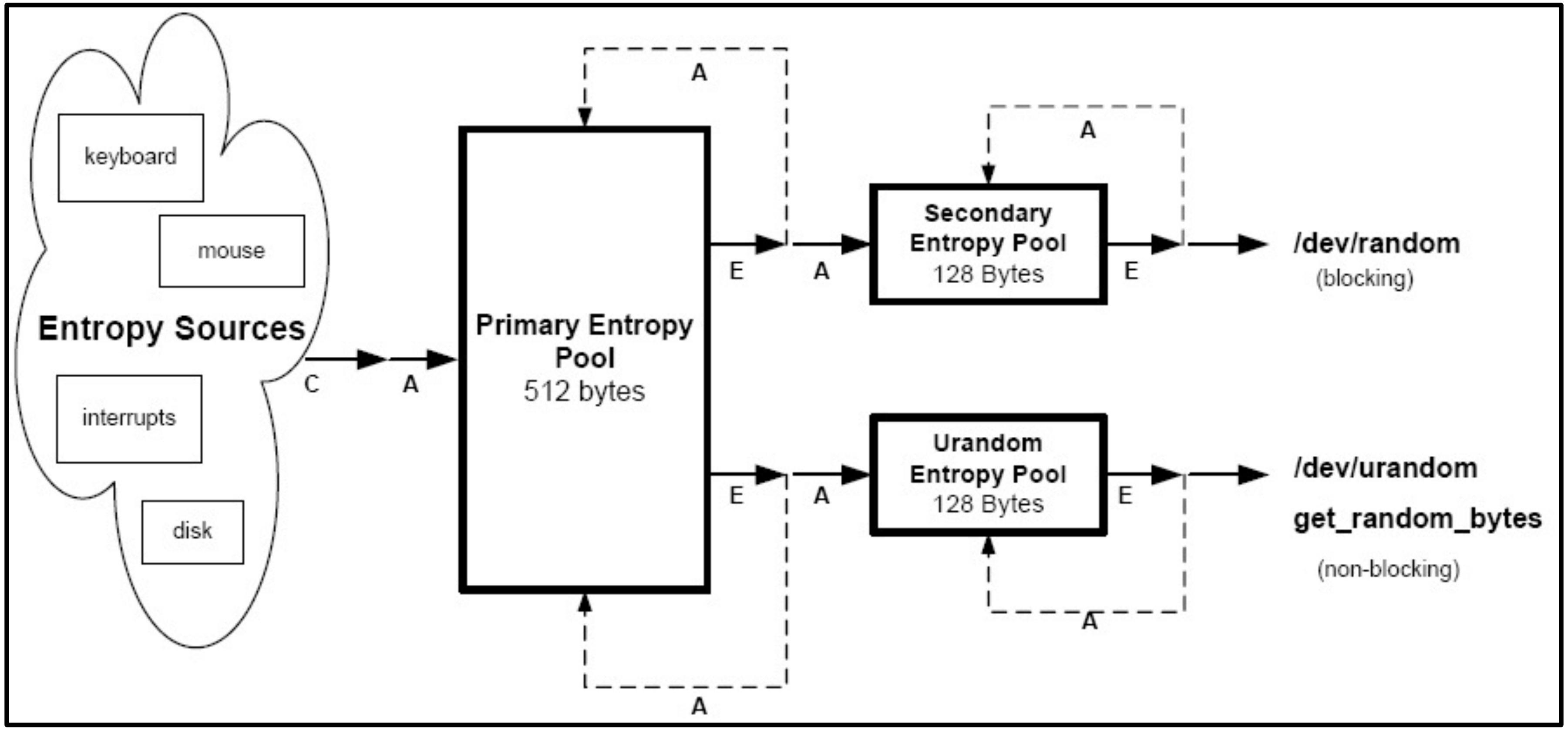


R1 output to caller
of instruction

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

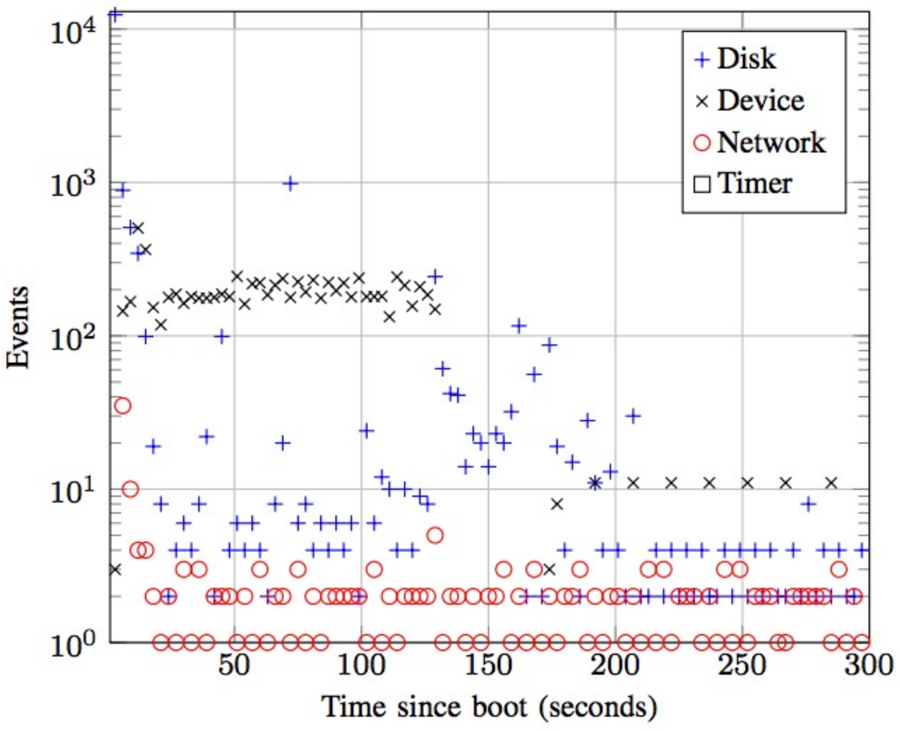
Questions

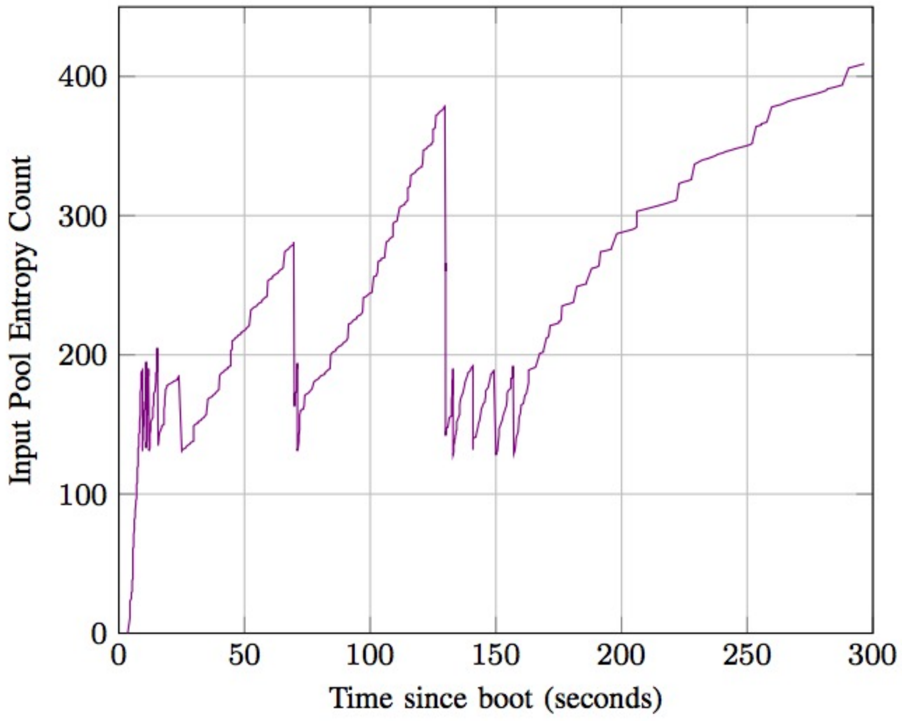
- Does /dev/(u)random collect sufficient entropy during boot?
  - Stamos, Becherer, and Wilcox conjecture not in virtualized environments (BlackHat 2009)
- What happens when a full-state snapshot is resumed?

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





Our 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

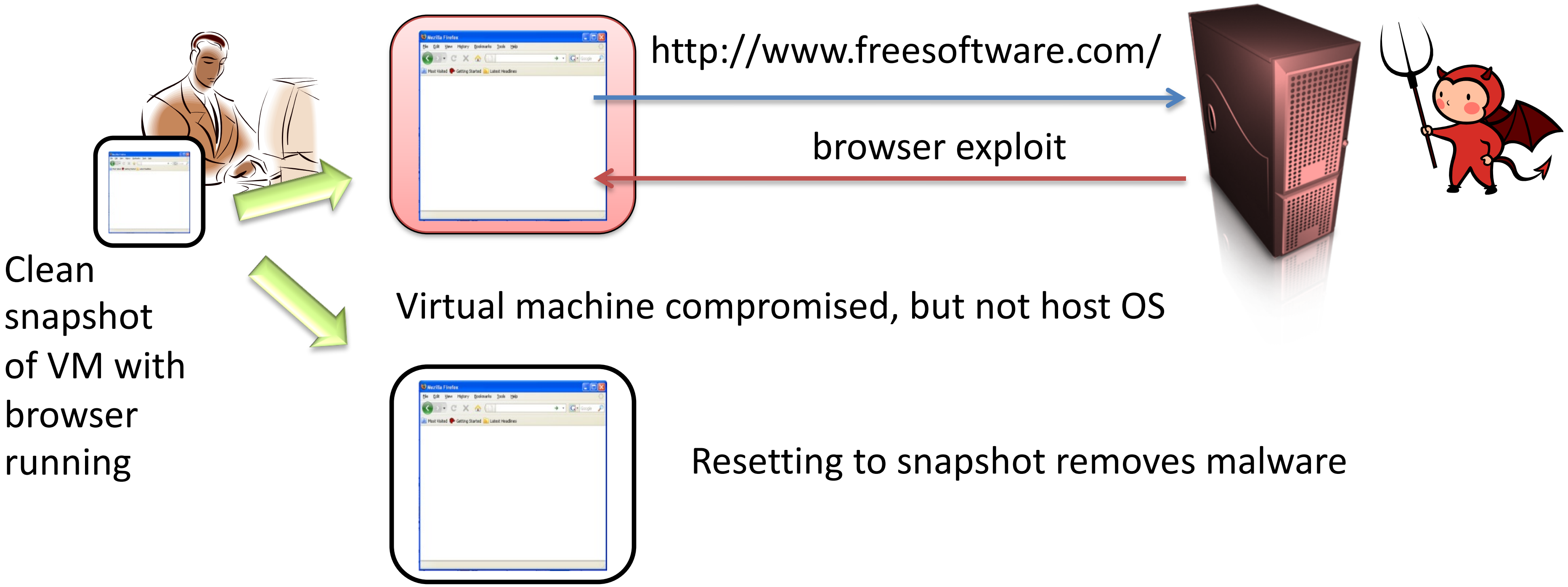
- Does /dev/(u)random collect sufficient entropy during boot?
  - Stamos, Becherer, and Wilcox conjecture not in virtualized environments (BlackHat 2009)
- What happens when a full-state snapshot is resumed?

We carefully instrumented Linux kernel to track entropy accumulation

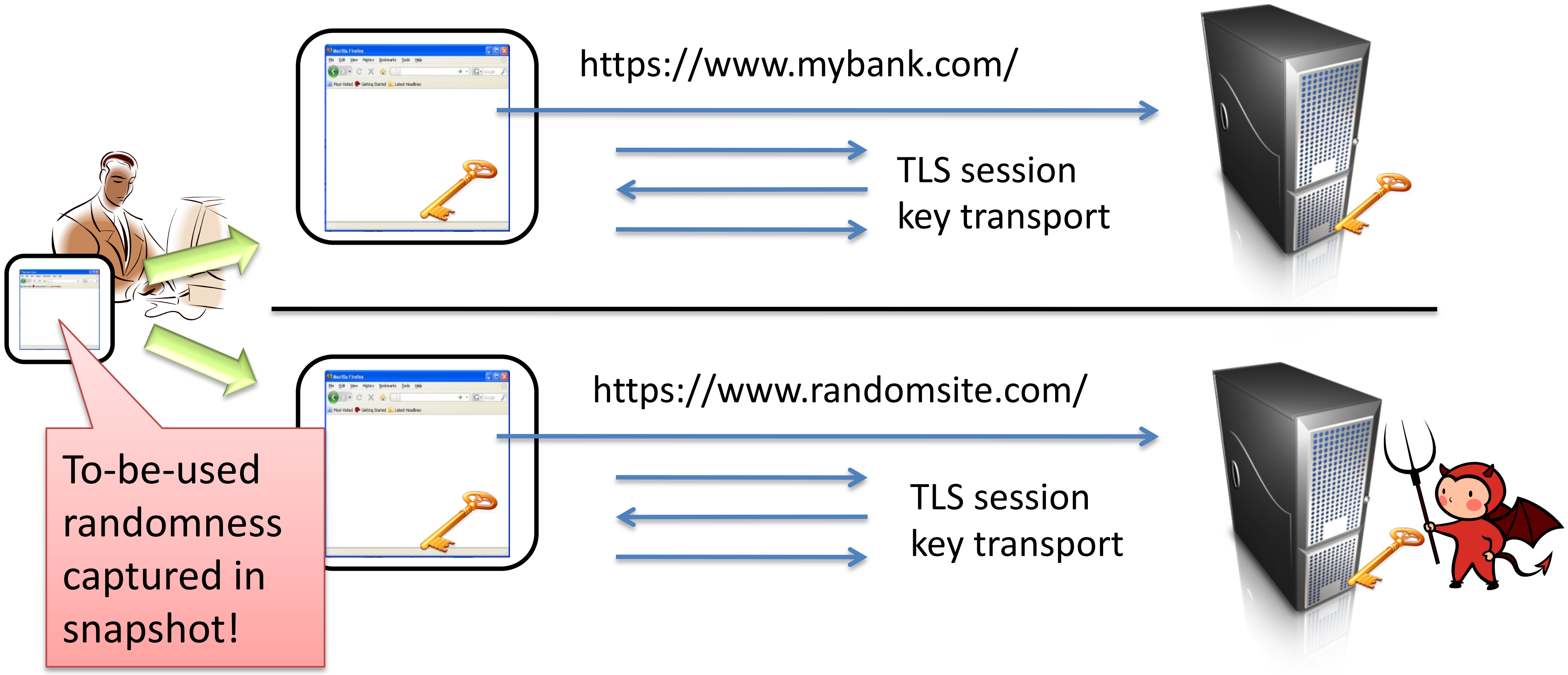
http://pages.cs.wisc.edu/~ace/papers/not-so-random.pdf

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]

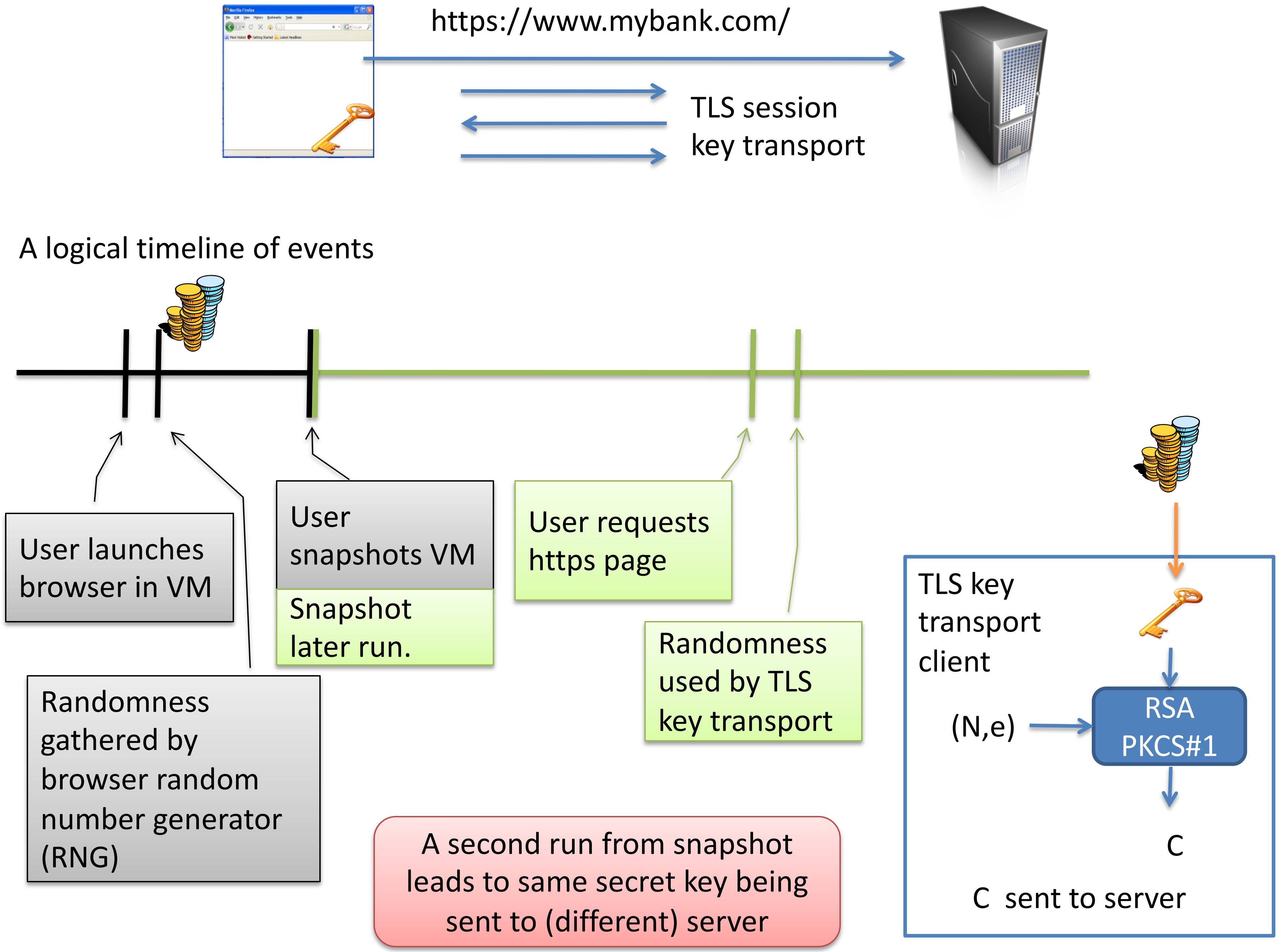


Virtual machine resets lead to RNG failures for applications



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
Randomness
gathered by
browser random
number generator
(RNG)
User
snapshots VM
Snapshot
later run.
User requests
https page
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
  - Can carefully mix in local sources of entropy if available
  - 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!)