Entropy as a Service

Unlocking the full potential of cryptography

Apostol Vassilev
Robert Staples

ICMC, May, 2016, Ottawa

A perspective

Cryptography enjoys a Renaissance period of increasingly fast evolution

- IoT and PQC are the next big frontiers

Emerging crypto technologies abound

- lightweight crypto
- lighter versions of legacy protocols
  - tinyDTLS, lightweight DTLS
- Post-Quantum Cryptography (PQC)

New crypto is cool but have we solved all problems with conventional cryptography?

Observation

In modern cryptography the algorithms are known

Key generation and ma
strengt and security o

Key generation is stron
dependent on entropy

Image Courtesy: XKCD, https://en.wikipedia.org/wiki/Xkcd

The elephant in the room

Where are the keys coming from?

Image Courtesy: Web

Real World Examples 2013

"Factoring RSA keys from certified smart cards (Coppersmith in the wild)",

Bernstein, Chang, Cheng, Chou, Heninger, Lange, van Someren

Problem: Low-quality hardware RNG, stuck in a short cycle:

0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
etc.

Likely reasons for using this weak design: <u>cost</u> of high-quality hardware, cost of licensing patents

Real World Examples 2012

"*Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices*"

Heninger, Durumeric, Wustrow, Halderman

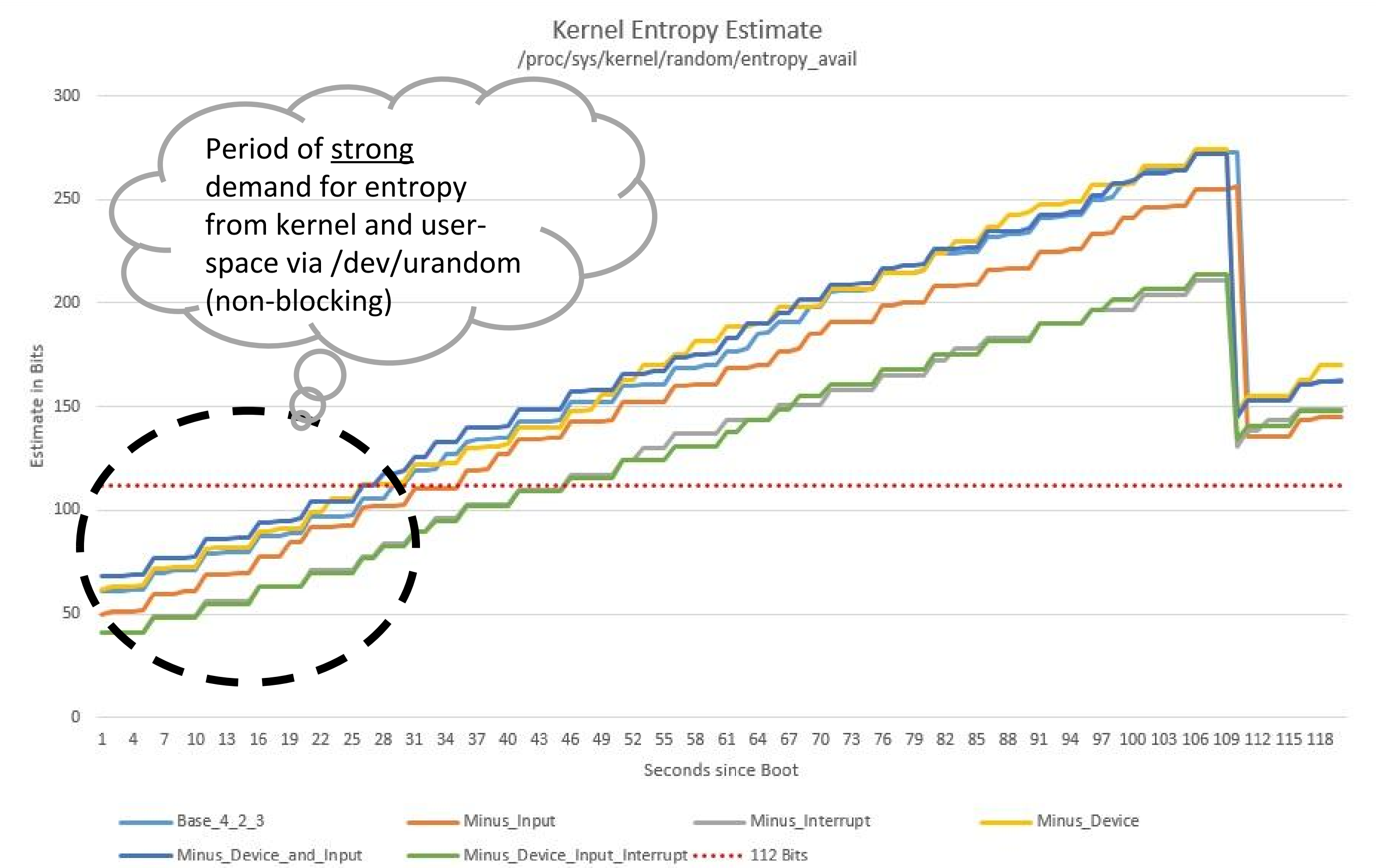
Scanned 28 Mil TLS and 23 Mil SSH hosts on the Internet

- 0.75% of TLS certificates share keys
  - due to insufficient entropy during key generation
  - another 1.70% come from same faulty implementations
- 0.50% of TLS hosts and 0.03% of SSH hosts revealed RSA private keys
  - public keys shared nontrivial common factors due to entropy problems

Real World Examples 2016

The Linux kernel dissected – four sources of entropy

- Device
- Input
- Interrupt
- Disk



"minimal" (no GUI) Ubuntu Server v14.04.3 64-bit w/ Kernel v4.2.3

Testing randomness is hard

Using a finite set of statistical tests on data samples can lead to misleading results

Example 1: expand a well-known irrational number, e.g. π, and test the output bit sequence for randomness - it will be reported as random.

Example 2: challenges in hardware-based sources of randomness – see "Sources of Randomness in Digital Devices and Their Testability"

Viktor Fischer, Univ Lyon, UJM-Saint-Etienne, Laboratoire Hubert Curien; NIST DRBG Workshop 2016

http://csrc.nist.gov/groups/ST/rbg-workshop-2016/presentations/SessionVI-2-viktor-fischer-presentation.pdf

Using the statistical test approach of **SP 800-90B** makes it hard to automate the estimation of entropy

automation is critically important for the new CMVP @ NIST

Our approach

How about delivering high-entropy random data from a provably good source to needy clients?

Public service providing high-entropy random data for use in cryptography

Entropy as a Service (EaaS)

- delivers entropy securely (no one can see) upon request from clients

Our solution is

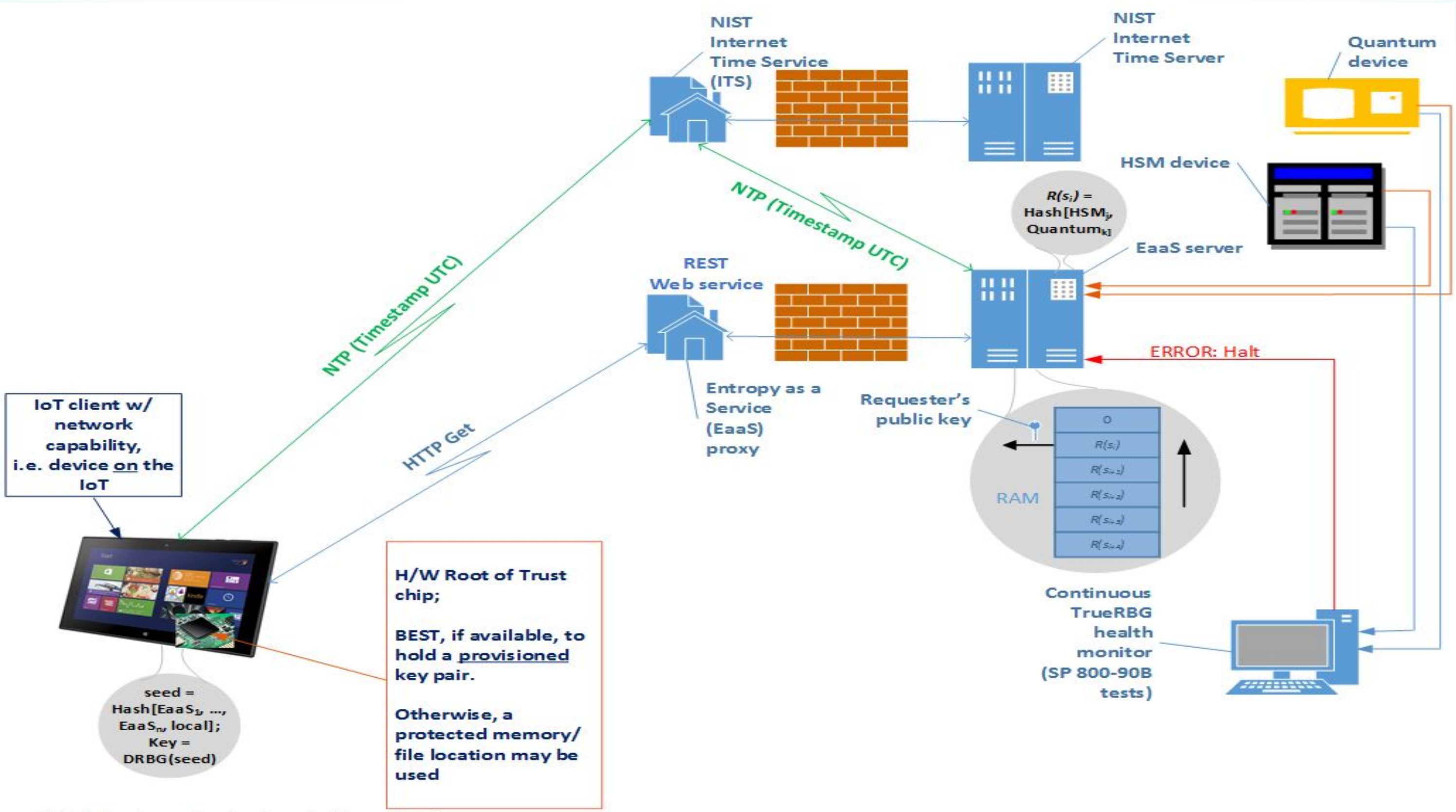
Not a key generation service

- cryptographic keys are generated locally on the client using DRBG's

Not similar to the NIST beacon

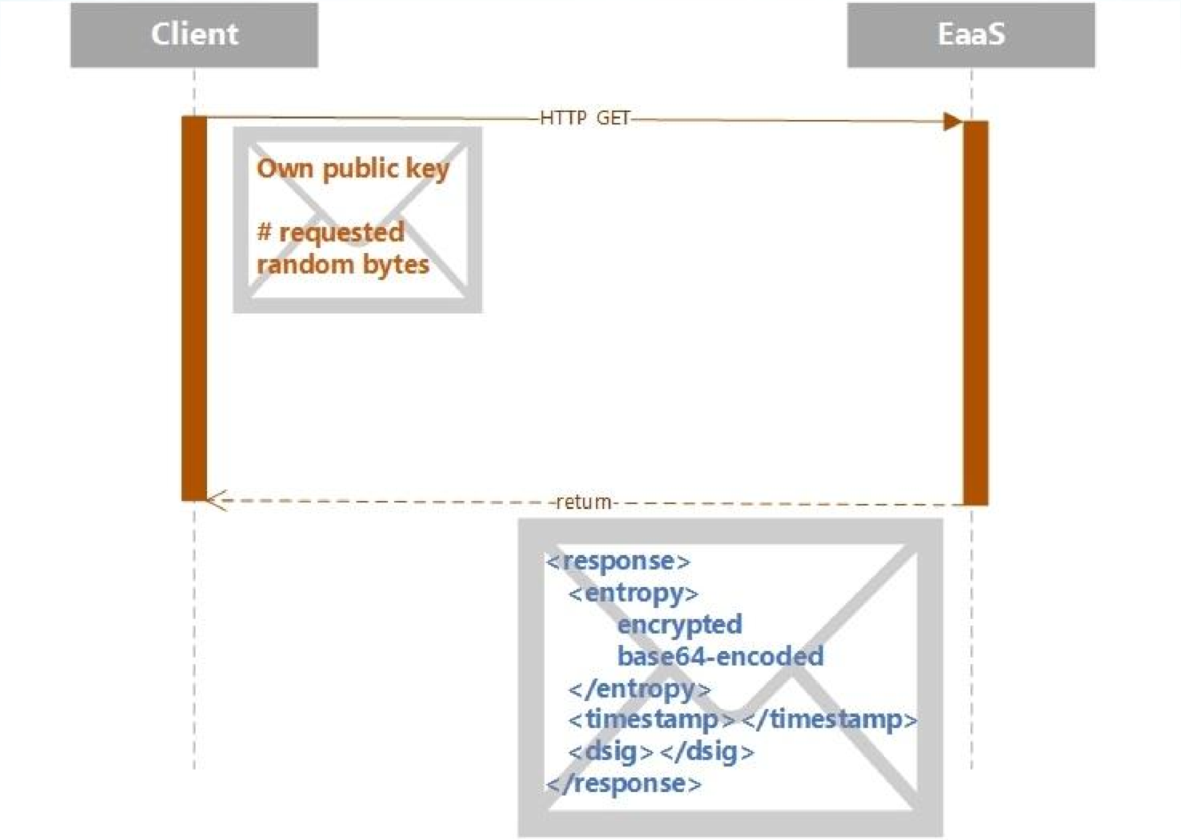
- EaaS does not record incoming or outgoing requests
- EaaS does not record generated entropy

EaaS architecture

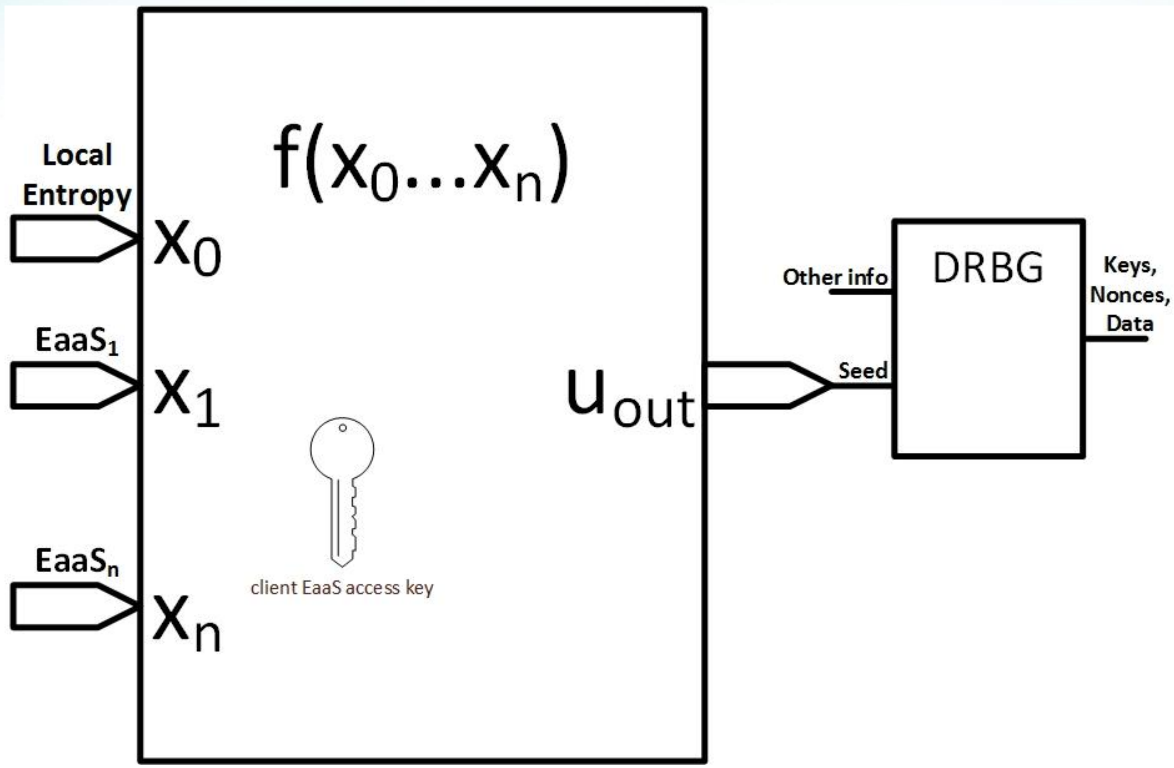


NOTE: **EaaS$_1$, ..., EaaS$_n$** above indicate data from ***n*** different **EaaS** server instances;
local indicates locally available random data, if any

A protocol sketch

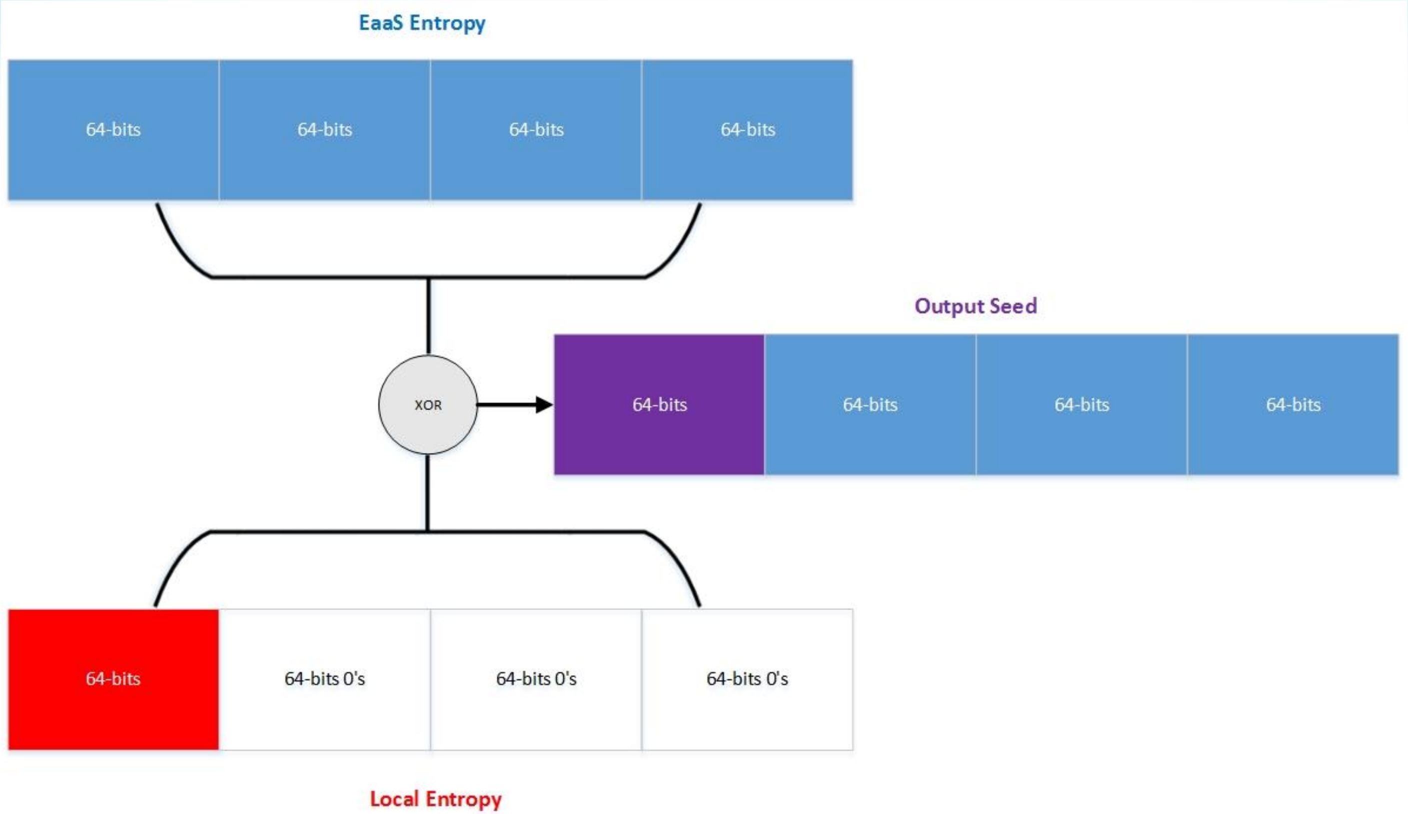


EaaS client usage model



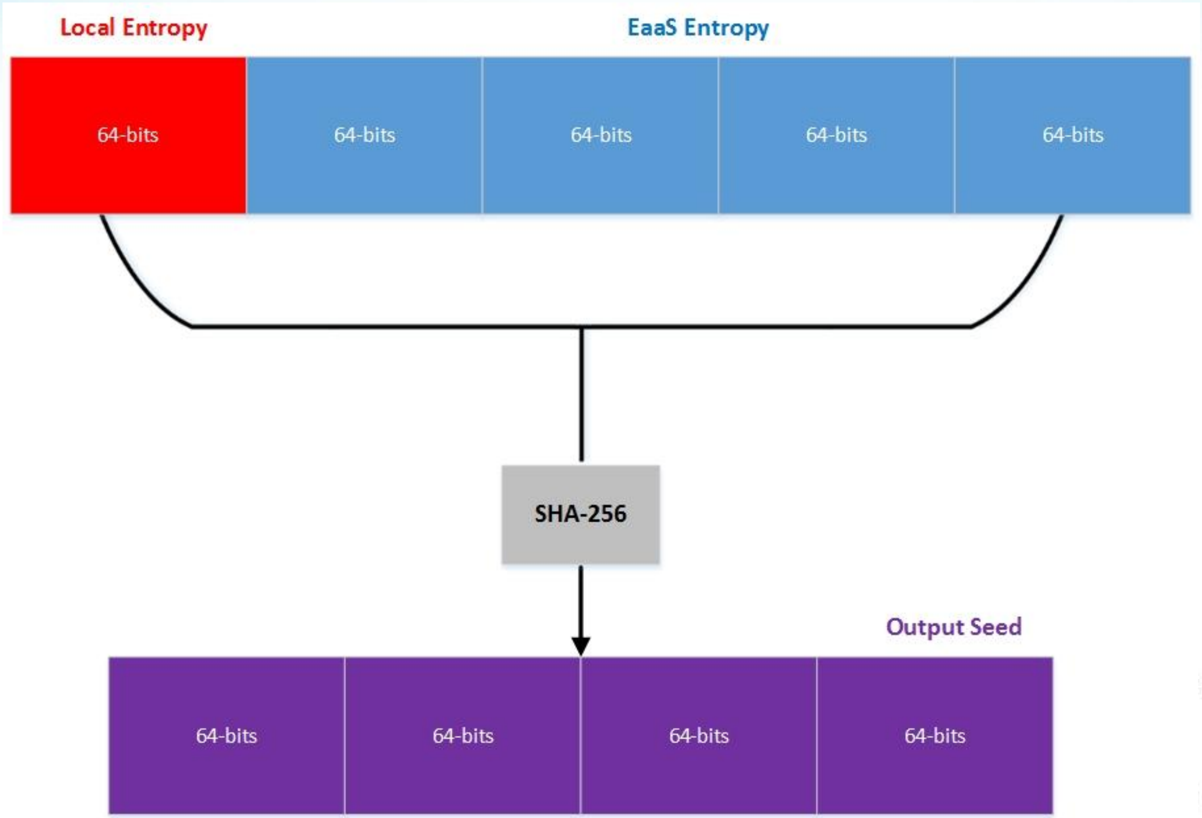
f: a hash function; **EaaS$_1$** … **EaaS$_n$**: independent EaaS instances providing data for computing **u_{out}** = **$f(x_0 \ldots x_n)$**, where **x_i**,*$1 \le i \le n$*, is data obtained from the **EaaS$_i$** instance using the client EaaS access key; Note, there is <u>one</u> client access key for accessing all **EaaS$_i$** instances.

A note on XoR mixing



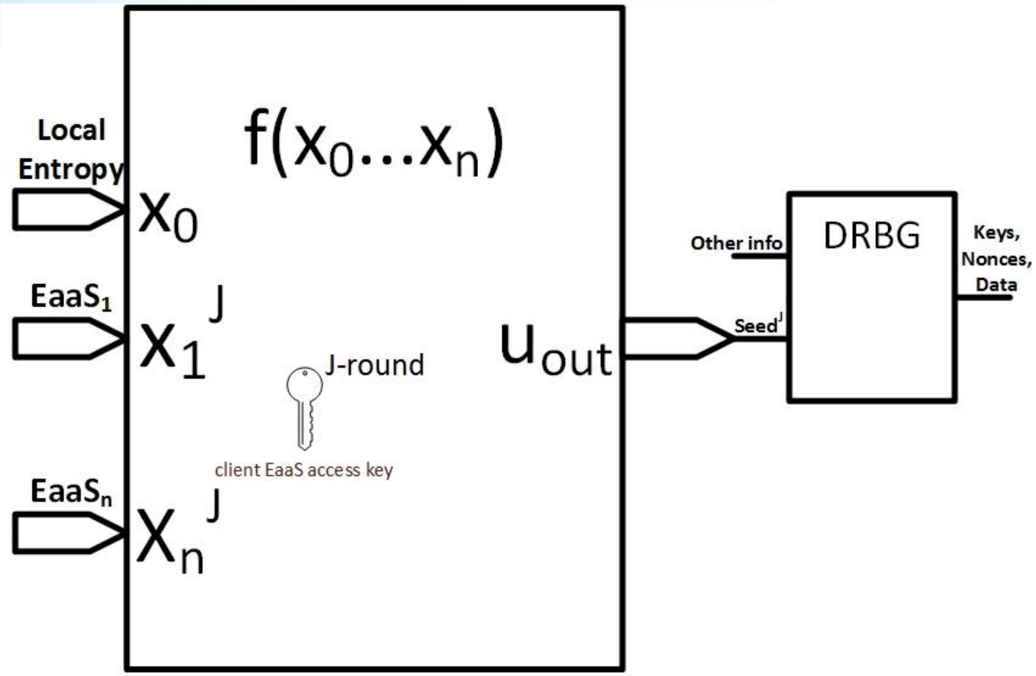
A dishonest EaaS instance may gain insight into the output seed if different size buffers are padded and XoR-ed.

Hash-based mixing



Safe and simple

Client EaaS access key management protocol



0-round

Initial key for accessing EaaS, provisioned out-of-band (factory, enterprise, etc.)

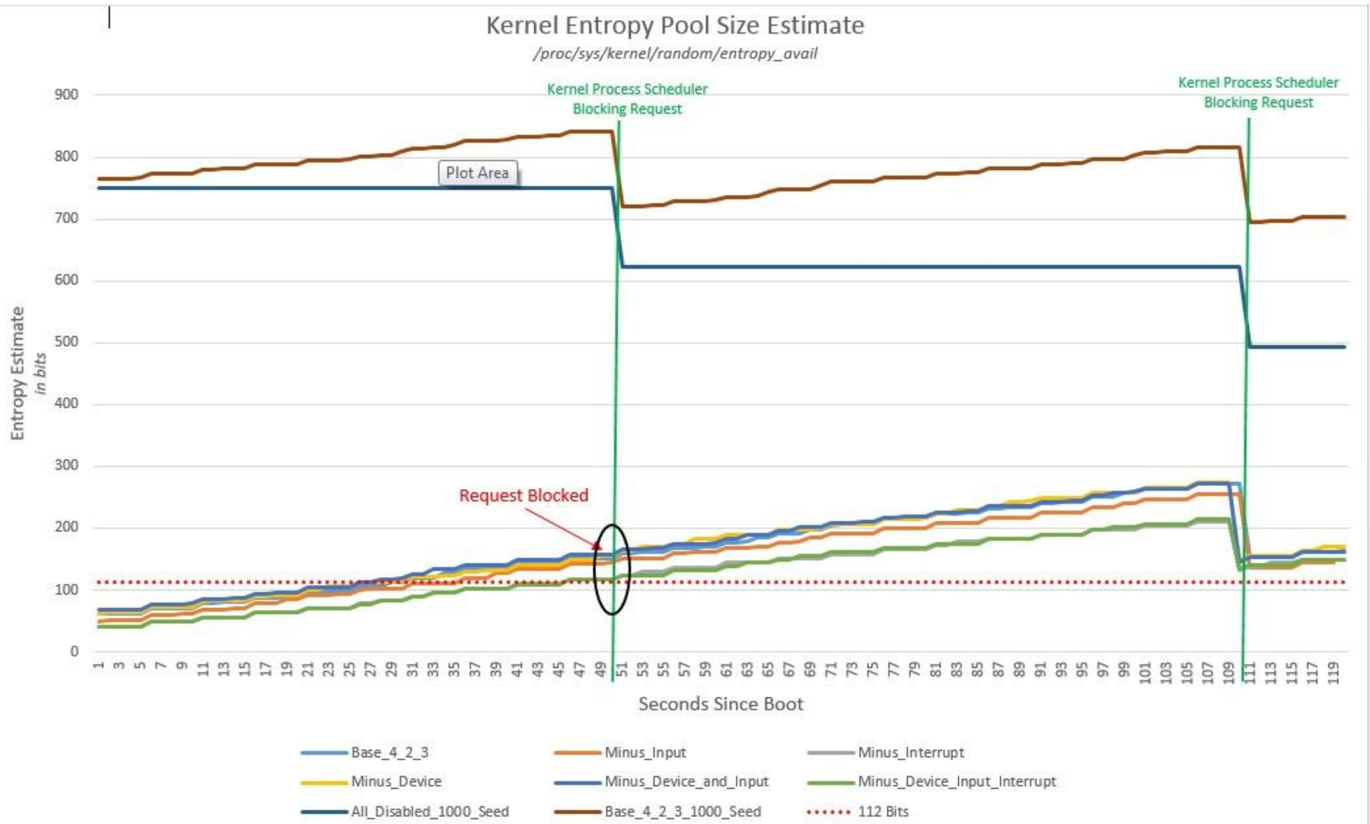
J-round

= **KDF** (DRBG(**Seed**$^{J-1}$)); **Seed**$^{J-1}$ = $\mathbf{u}_{out}$ = $\mathbf{f}(x_1\, x_2^{J-1} \ldots x_n^{J-1})$; x_i^{J-1}, $1 \le i \le n$, denotes data obtained from the **EaaS**$_i$ instance using the (J-1)-round client EaaS access key; **KDF** denotes an appropriate procedure for asymmetric key generation, cf. **SP 800-133.**

Protocol sketch for managing EaaS access keys with Perfect Forward Secrecy (PFS)

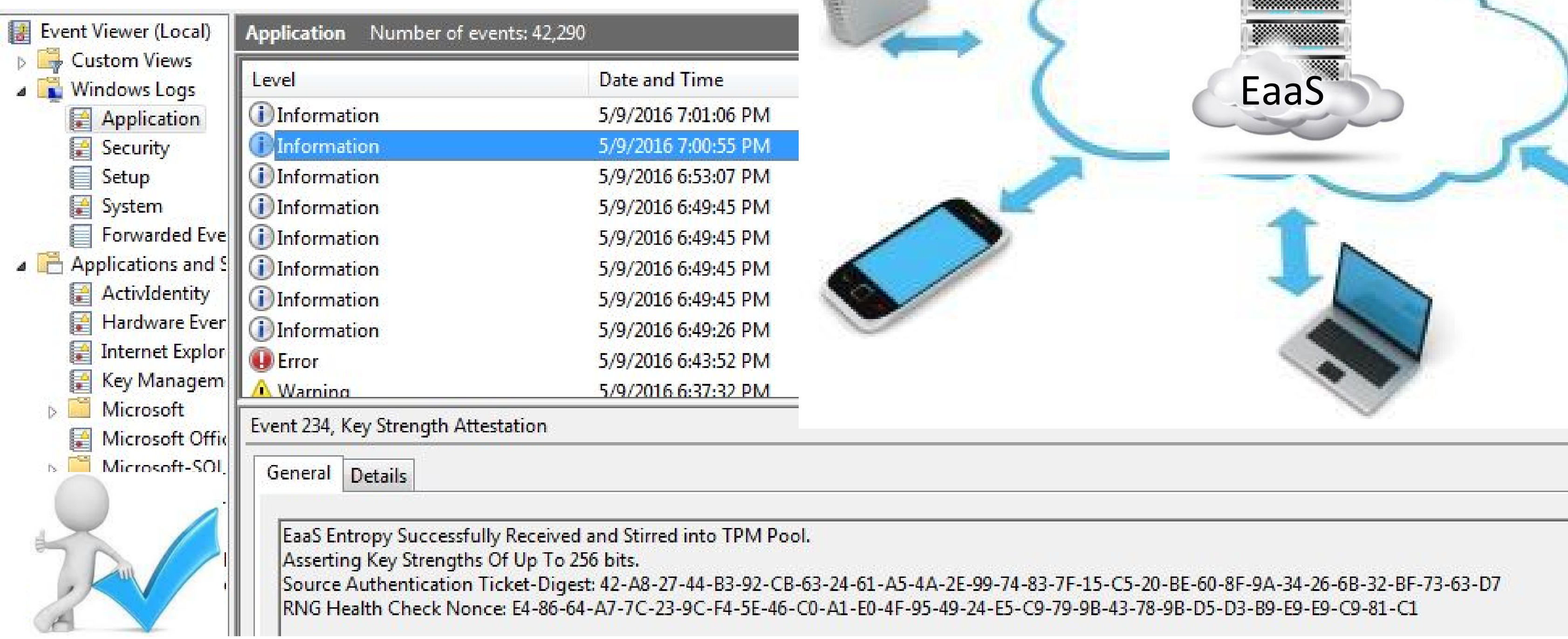
Linux Kernel Revisited

Instrumented to access EaaS and seed local entropy pool



C program, using "RNDADDENTROPY" ioctl to add entropy.

Enterprise key strength attestation



Potential attacks and mitigation

Standard attacks on web service and protocol

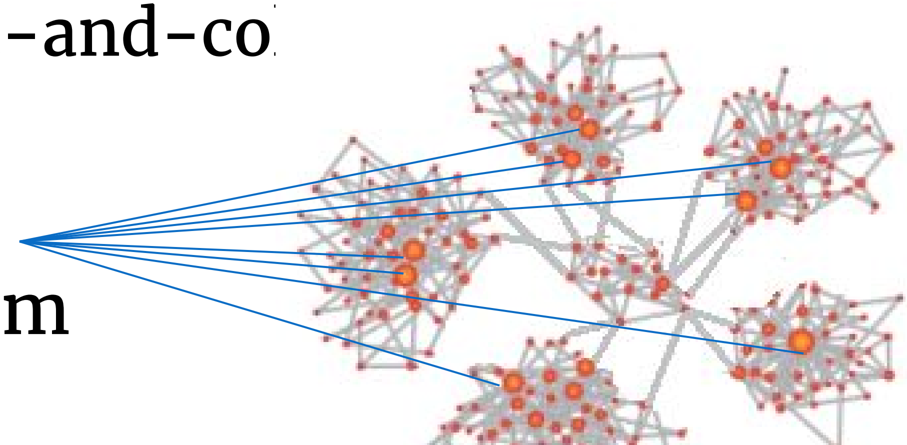
- Message replay
- Man-In-The-Middle
- DNS poisoning

Protocol features and out-of-band provisioning mitigate these attacks

Trust-related attacks and mitigation

EaaS-specific attacks on the web service

- Honest-but-curious EaaS instance
- Dishonest-but-non-colluding EaaS instances
- Dishonest-and-co es



EaaS ecosystem

Image Courtesy: Cornell Univ. Networks Course Blog
https://blogs.cornell.edu/info2040/2012/09/26/7720/

Status and next steps

See project page at : http://csrc.nist.gov/projects/eaas/

Now: Functional prototype implemented; demoed at
- CIF 2015 in Washington, DC
- DRBG Workshop 2016, NIST

Next:
- Stand-up publicly accessible NIST EaaS in Q2, 2016
- publish client and server sample code on GitHub

Questions?