

RSACConference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN
ELEMENT

SESSION ID: ACB-W01

Entropy as a Service: A Framework for Delivering High-quality Entropy



Ravi Jagannathan

Security Architect
VMware

David Ott

Sr. Staff Researcher and Academic Program Director
VMware

#RSAC

The Problem of Weak Entropy

	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
... using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
... using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
... using default certificates or default keys	670,391	(5.23%)		
... using low-entropy repeated keys	43,852	(0.34%)		
... using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
... using DSA keys we could compromise			105,728	(1.03%)
... using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
... using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
... identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
... model using low-entropy repeated keys	314,640	(2.45%)		

Table 2: Summary of vulnerabilities — We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

The Problem of Weak Entropy

[09] CVE-2000-0357 : ORBit and esound in Red Hat Linux do not use sufficiently random numbers, December 1999.

[10] CVE-2001-0950: ValiCert Enterprise Validation Authority uses insufficiently random data, January 2001.

[11] CVE-2001-1141: PRNG in SSLeay and OpenSSL could be used by attackers to predict future pseudorandom numbers, July 2001.

[12] CVE-2001-1467: mkpasswd, as used by Red Hat Linux, seeds its random number generator with its process ID, April 2001.

[13] CVE-2003-1376: WinZip uses weak random number generation for password protected ZIP files, December 2003.

[14] CVE-2005-3087: SecureW2 TLS implementation uses weak random number generators during generation of the pre-master secret, September 2005.

[15] CVE-2006-1378: PasswordSafe uses a weak random number generator, March 2006.

[16] CVE-2006-1833: Intel RNG Driver in NetBSD may always generate the same random number, April 2006.

[17] CVE-2007-2453: Random number feature in Linux kernel does not properly seed pools when there is no entropy, June 2007.

[18] CVE-2008-0141: WebPortal CMS generates predictable passwords containing only the time of day, January 2008.

[19] CVE-2008-0166: OpenSSL on Debian-based operating systems uses a random number generator that generates predictable numbers, January 2008.

[20] CVE-2008-2108: GENERATE SEED macro in php produces 24 bits of entropy and simplifies brute force attacks against the rand and mt_rand functions, May 2008.

[21] CVE-2008-5162: The arc4random function in FreeBSD does not have a proper entropy source for a short time period immediately after boot, November 2008.

[22] CVE-2009-0255: TYPO3 creates the encryption key with an insufficiently random seed, January 2009.

[23] CVE-2009-3238: Linux kernel produces insufficiently random numbers, September 2009.

[24] CVE-2009-3278: QNAP uses rand library function to generate a certain recovery key, September 2009.

[25] CVE-2011-3599: Crypt::DSA for Perl, when /dev/random is absent, uses the data::random module, October 2011.

[26] CVE-2013-1445: The crypto.random.atfork function in Py-Crypto before 2.6.1 does not properly reseed the

pseudo-random number generator (PRNG) before allowing a child process to access it, October 2013.

[27] CVE-2013-4442: Password generator (aka Pwgen) before 2.07 uses weak pseudo generated numbers when /dev/urandom is unavailable, December 2013.

[28] CVE-2013-5180: The srandomdev function in Libc in Apple Mac OS X before 10.9, when the kernel random-number generator is unavailable, produces predictable values instead of the intended random values, October 2013.

[29] CVE-2013-7373: Android before 4.4 does not properly arrange for seeding of the OpenSSL PRNG, April 2013.

[30] CVE-2014-0016: tunnel before 5.00, when using fork threading, does not properly update the state of the OpenSSL pseudo-random number generator, March 2014.

[31] CVE-2014-0017: The rand bytes function in libssh before 0.6.3, when forking is enabled, does not properly reset the state of the OpenSSL pseudorandom number generator, March 2014.

[32] CVE-2014-4422: The kernel in Apple iOS before 8 and Apple TV before 7 uses a predictable random number generator during the early portion of the boot process, October 2014.

Agenda

- Entropy
- NIST EaaS
- VMware Suggested Improvements
- Experiments
- Future Work

What is entropy?

Entropy: def

- a measure of randomness, unpredictability
- more entropy => harder for adversary to guess

More precisely: Number of information bits not known to an adversary

- k bits of entropy
- Max 2^k guesses needed to find a value (assuming uniform distribution)

Example: $k = 8$ bits

Max 256 guesses to find value

Example: $k = 32$ bits

Max 4,294,967,296 guesses to find value

Example: $k = 64$ bits

Max 18,446,744,073,709,551,616 guesses to find value

Entropy Sources

Entropy Source:

- a noise source
- sampling and quantization
- minimal conditioning (e.g., unbiasing)

Key questions:

- Does the entropy show statistical bias?
- Is it unpredictable and without pattern?
- Has enough entropy been collected per sampled output?

Physical processes

- thermal noise from a resistor
- ring oscillators

Quantum

- photons

Clock drift

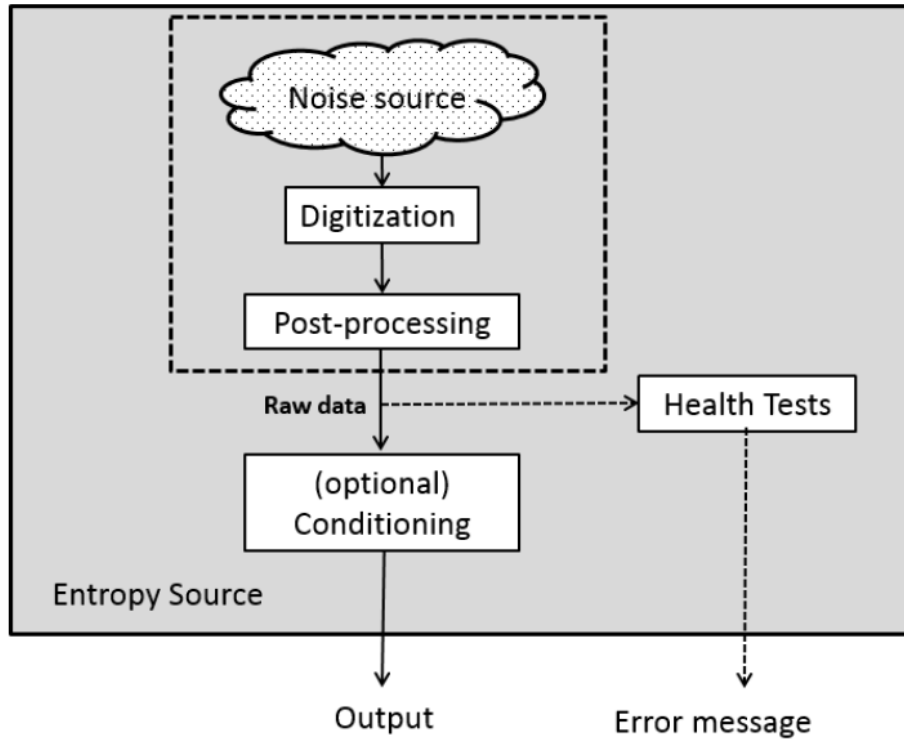
OS subsystems

- network packets
- disk seek times

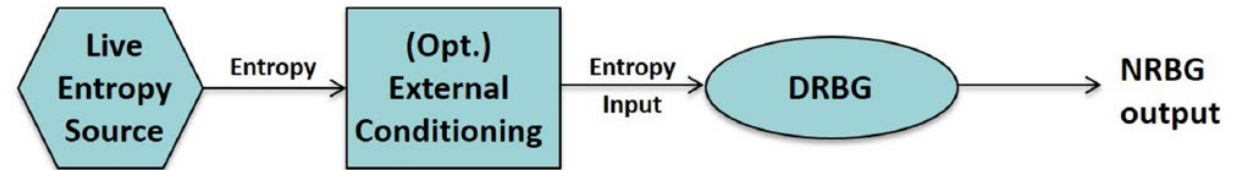
User processes

- mouse movements
- keyboard stroke timings

RBGs: Conditioning and Testing



NIST Entropy Source Model

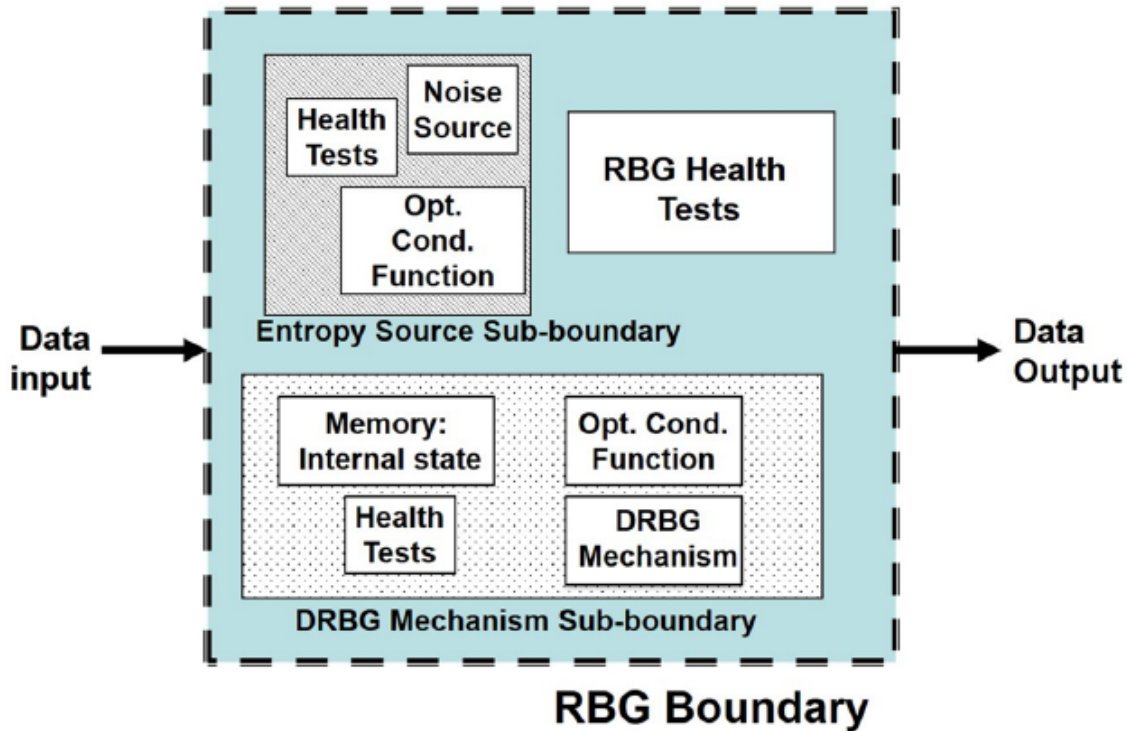


NIST Oversampling NRBG Construction

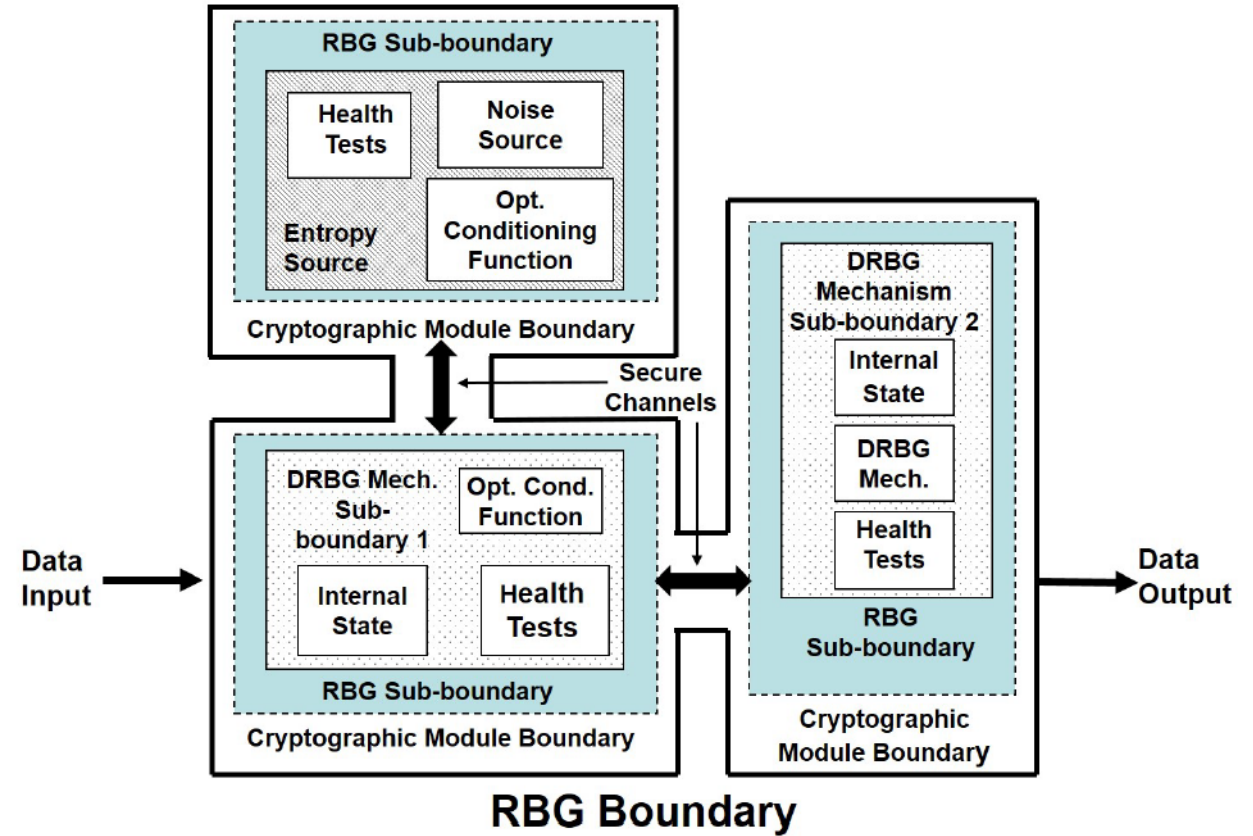
DRBG = Deterministic Random Bit Generator

- a.k.a., Pseudorandom Number Generator (PRNG)
- Input: random seed value
- Output: sequence of random values with uniform distribution
- Sequence is deterministic (same for given seed)
- E.g., hash function, block cipher

RBGs: Conditioning and Testing



NIST Random Bit Generator (RBG)



NIST Distributed RBG

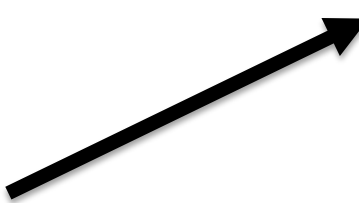
NIST SP 800-90B, "Recommendation for the Entropy Sources Used for Random Bit Generation". January 2018.

NIST SP 800-90C, "Recommendation for Random Bit Generator (RBG) Contstructions. April 2016.

Validation: How do you know you have it right?


Health Tests:

- Startup tests
- Continuous health tests
- On-demand health tests

- 
- Repetition Count Test
 - $P(\text{identical values}) \text{ from } C \text{ consecutive samples} \leq \alpha$
 - Adaptive Proportion Test
 - Test if value frequency within window size $W \leq \text{threshold}$

Applied to:

- Entropy source
- Output of conditioning function

- 
- IID Tests
 - Test whether values are independent and identically distributed
 - Min-Entropy Estimation
 - Determine how much entropy in a noise source sample

Special cases

What's hard?

Entropy Source

- What to use as an entropy source?
- Is the entropy rate adequate?
- How do you know?

Continuous health tests

- Do you have enough samples?
- Is the device capable of doing the calculations?

Problematic Platforms:

- IoT devices
- Cloud computing – lack sources of non-determinism
- Systems administrated by non-experts
- Platforms/applications requiring a large amount of entropy



Entropy as a Service (EaaS)



Proposal:

- High entropy random data available as service over the network.
- Provably robust entropy source
- Secure delivery
- Serves large number of needy devices

Entropy Server

- Quantum entropy source provides continuous random data to FIFO buffer in memory
- Responds to client requests by removing random values, encrypting, sending to client

Client Devices

- Request and consume entropy (key establishment, nonces, authentication)
- Dedicated software protected by trusted hardware (e.g., TPM, Arm TrustZone)

EaaS Architecture



IoT client w/
network
capability,
i.e. device on the
IoT



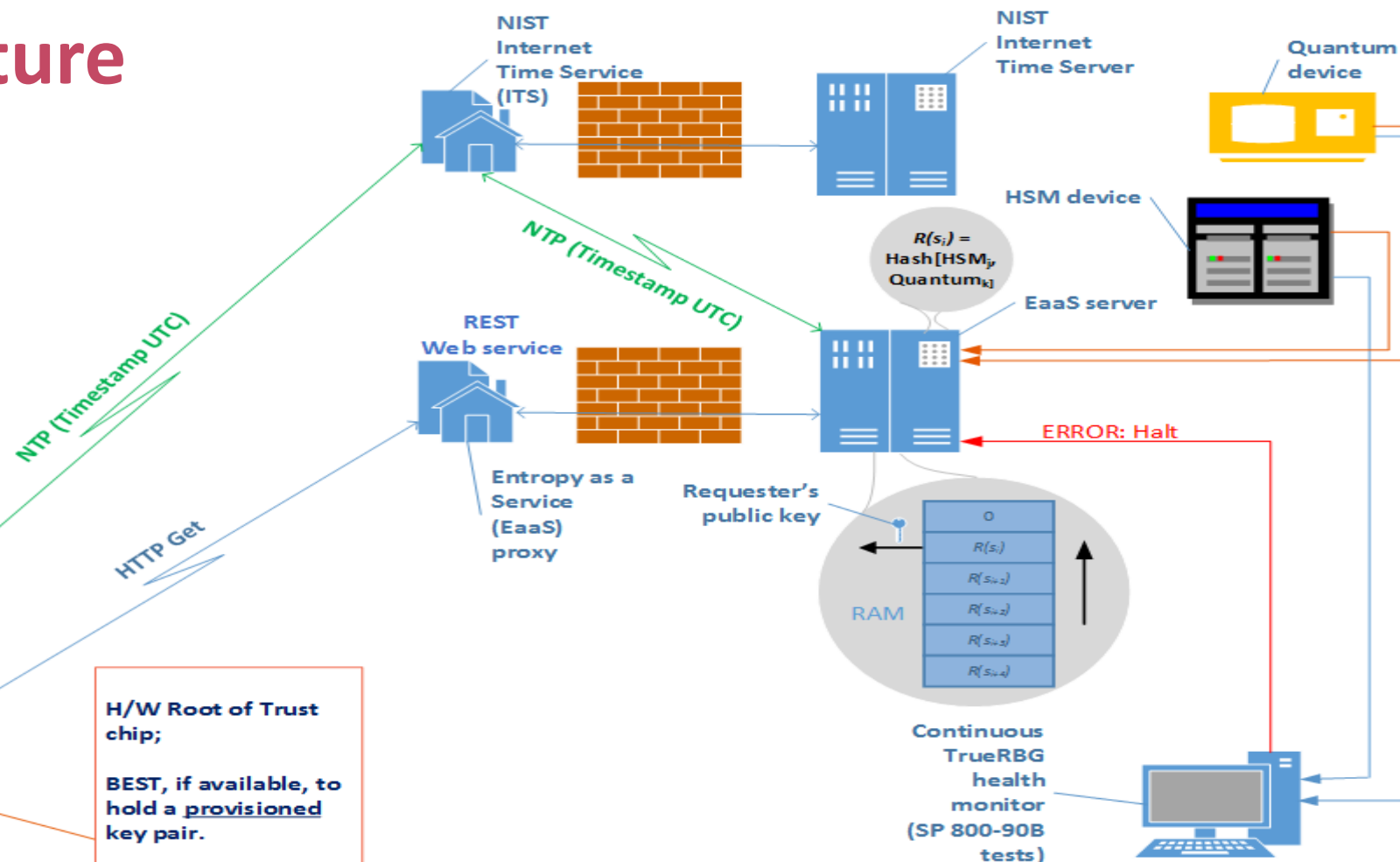
seed =
Hash[EaaS₁, ...,
EaaS_n, local];
Key =
DRBG(seed)

**H/W Root of Trust
chip;**

**BEST, if available, to
hold a provisioned
key pair.**

**Otherwise, a
protected memory/
file location may be
used**

NOTE: EaaS₁, ..., EaaS_n above indicate data from *n*
different EaaS server instances;
local indicates locally available random data, if any



EaaS: Request/Response Protocol



NIST:

- HTTP GET request
- XML response with encrypted, signed payload
- NTP timestamps prevents replay attacks

Our suggestion:

- HTTP over TLS (HTTPS) GET request
- Eliminate NTP
- JSON response from server



EaaS Req/Rsp: HTTP over TLS (HTTPS)

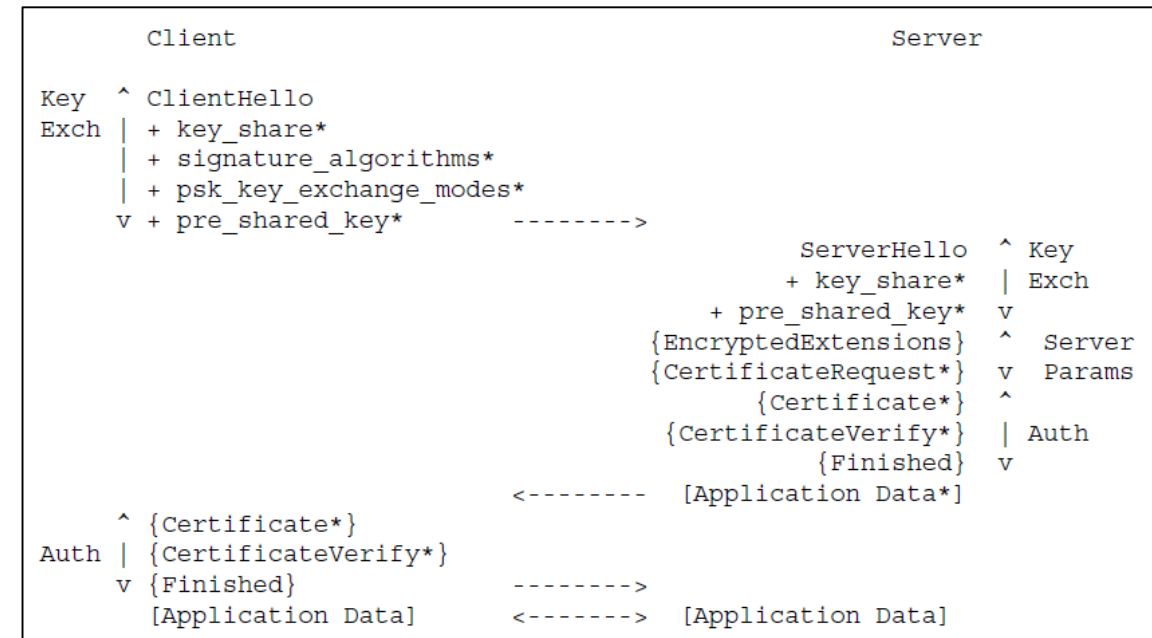
TLS 1.3 (or 1.2 for now)

- Authentication
- Encryption
- Replay protection

- Leverage standard implementation
- Connection-level encryption handles need to encrypt entropy
- TLS 1.3 reused for secure data exchange and EaaS.

Bootstrapping? Solutions:

- Pre-configured symmetric key (AES)
- Pre-configured entropy bits
- Weak entropy source, but used only for brief initial time window



TLS 1.3 handshake

EaaS: JSON Response

JSON Advantages (over XML)

- Open standard (RFC 8259) while XML schemas tend to be custom/proprietary
- Widely implemented and portable
- Support for array objects
- Easier to parse, less computation, highly optimized these days
- Shorter (less verbose)

XML Example:

```
<response>
  <entropy>
    encrypted base64-encoded
  </entropy>
  <timestamp>
    ts here
  </timestamp>
  <digital sig>
    sig here
  </digital sig>
</response>
```

JSON Example:

```
{
  "Client Knock": [
    { "client Identity": "VMware" },
    { "Entropy need": "2048 bits" },
    { "Number of Requests": "4" }
  ]
}
```

EaaS: Entropy Source

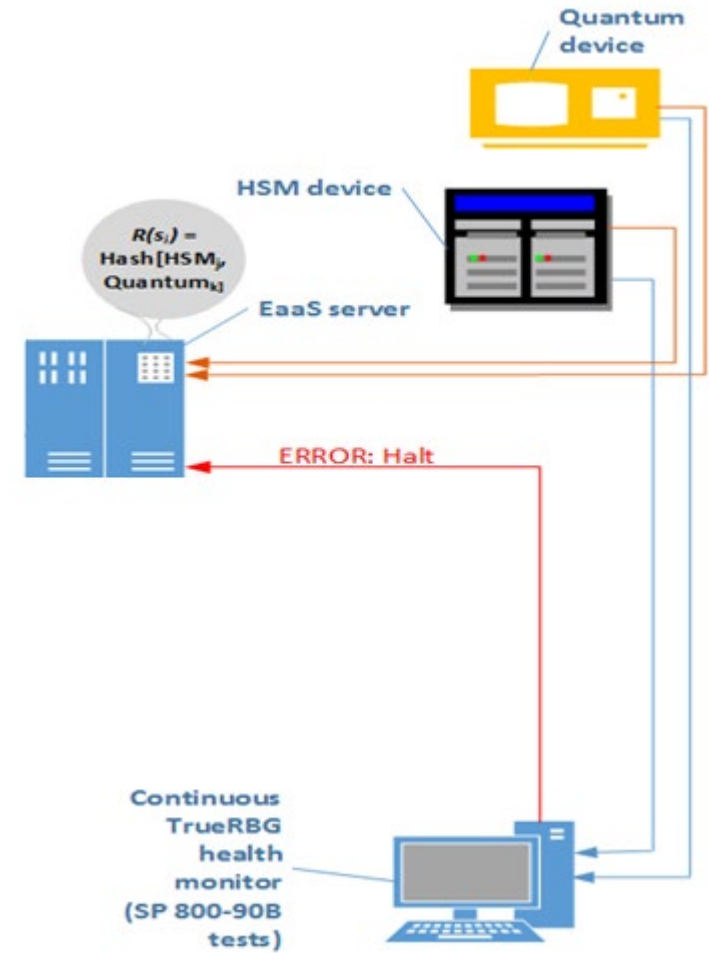
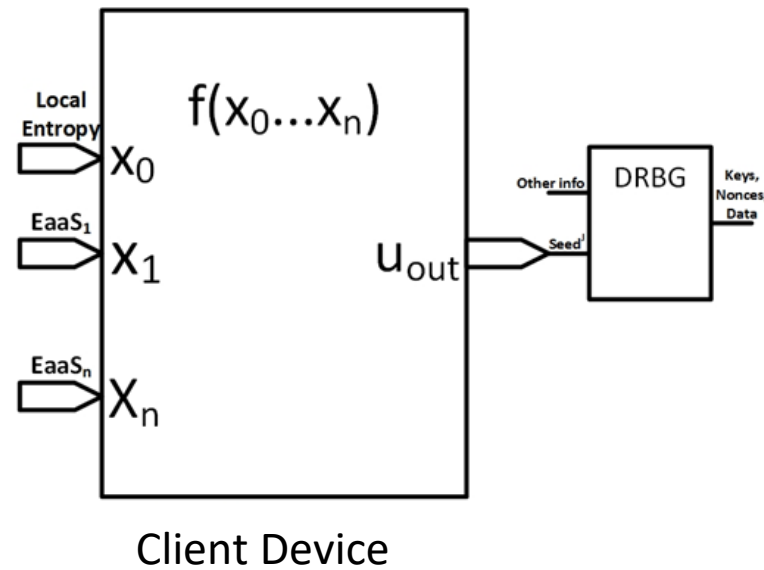


NIST:

- True RBG = True Random Bit Generator (e.g., quantum device)
- SP 800-90B compliant
- Continuous monitoring solution

Client Entropy Usage:

- IoT device: May not trust underlying entropy source
- VMs/Containers in cloud: Cloning replicates DRBG state, requires reseeding
- Mixing function can be used to combine weak entropy with high-quality EaaS entropy, or to mix entropy sources

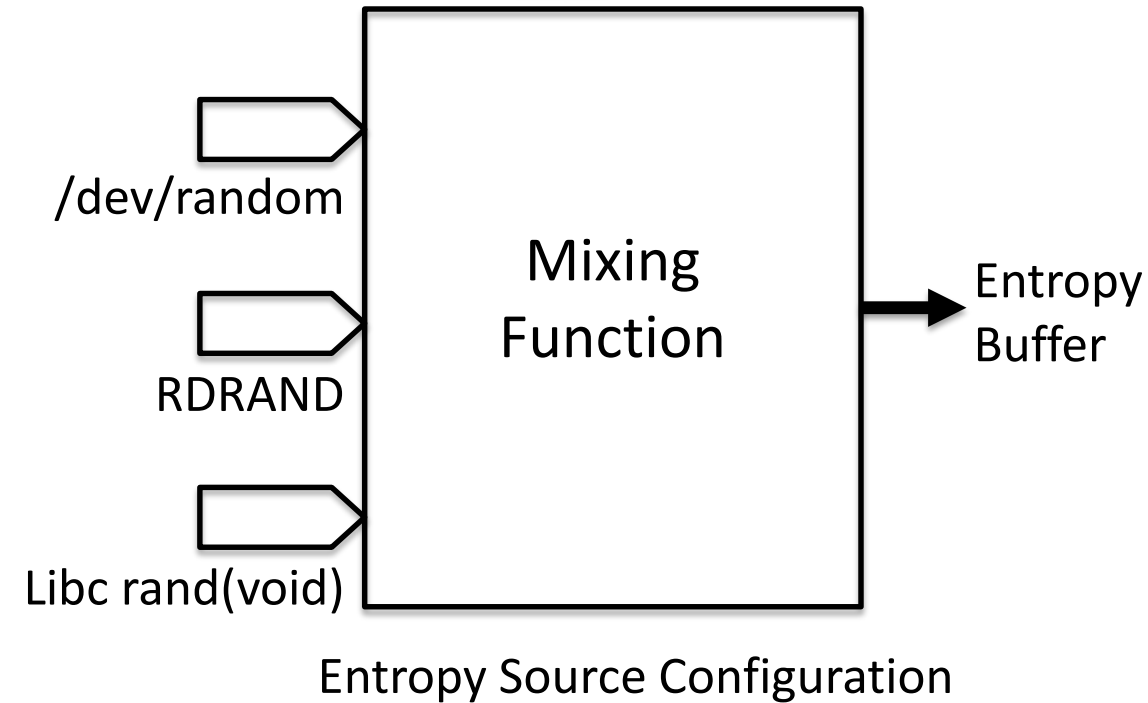
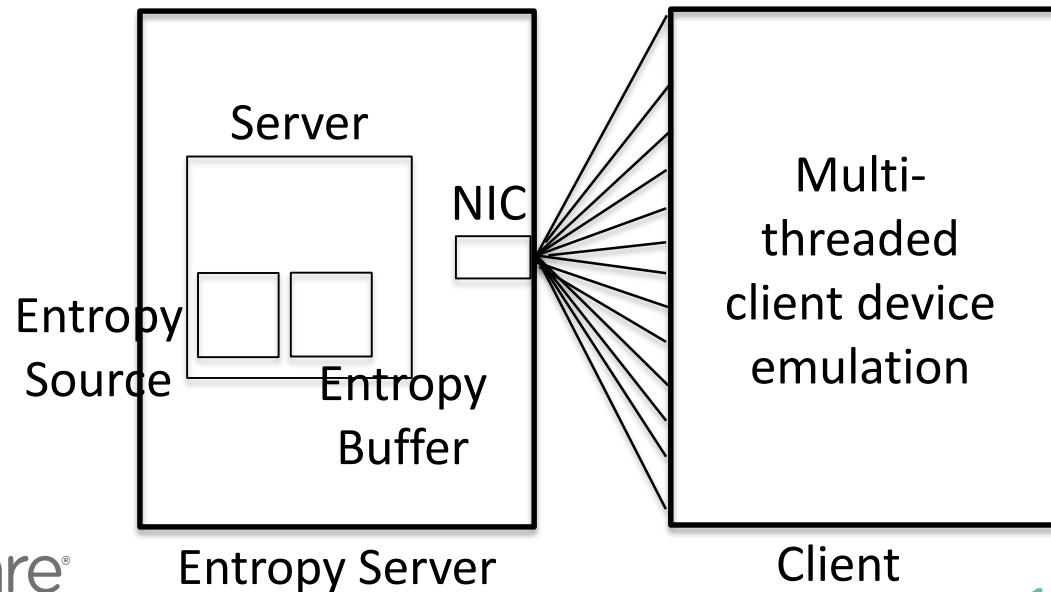


Entropy Server

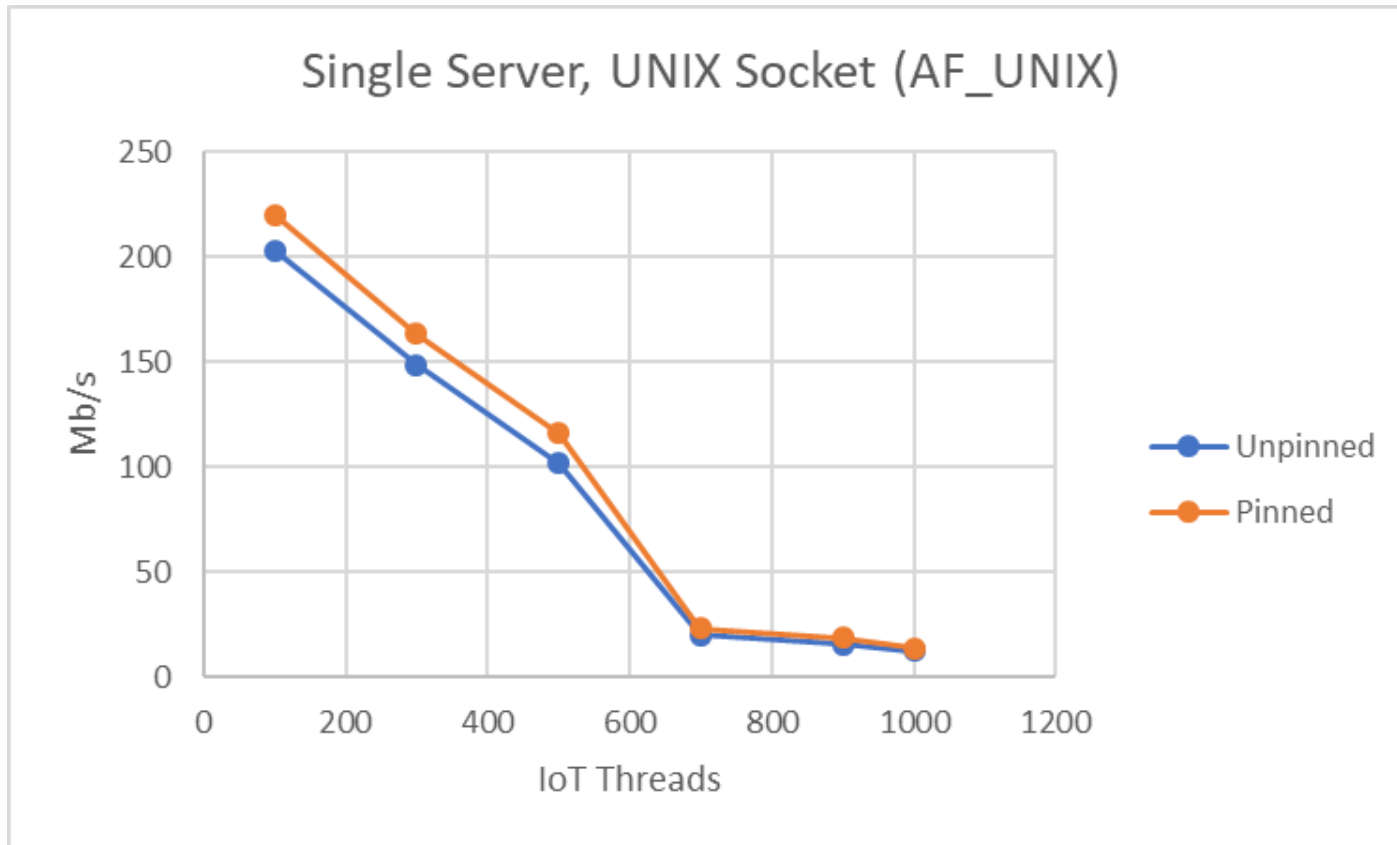
RSAC Conference 2020

EaaS Scaling Experiments

- Dell Power Edge R940 (56 core, 256 GB DRAM, 100 GB ethernet)
- Ubuntu Linux 18.04



EaaS Scaling Experiments: Single Server



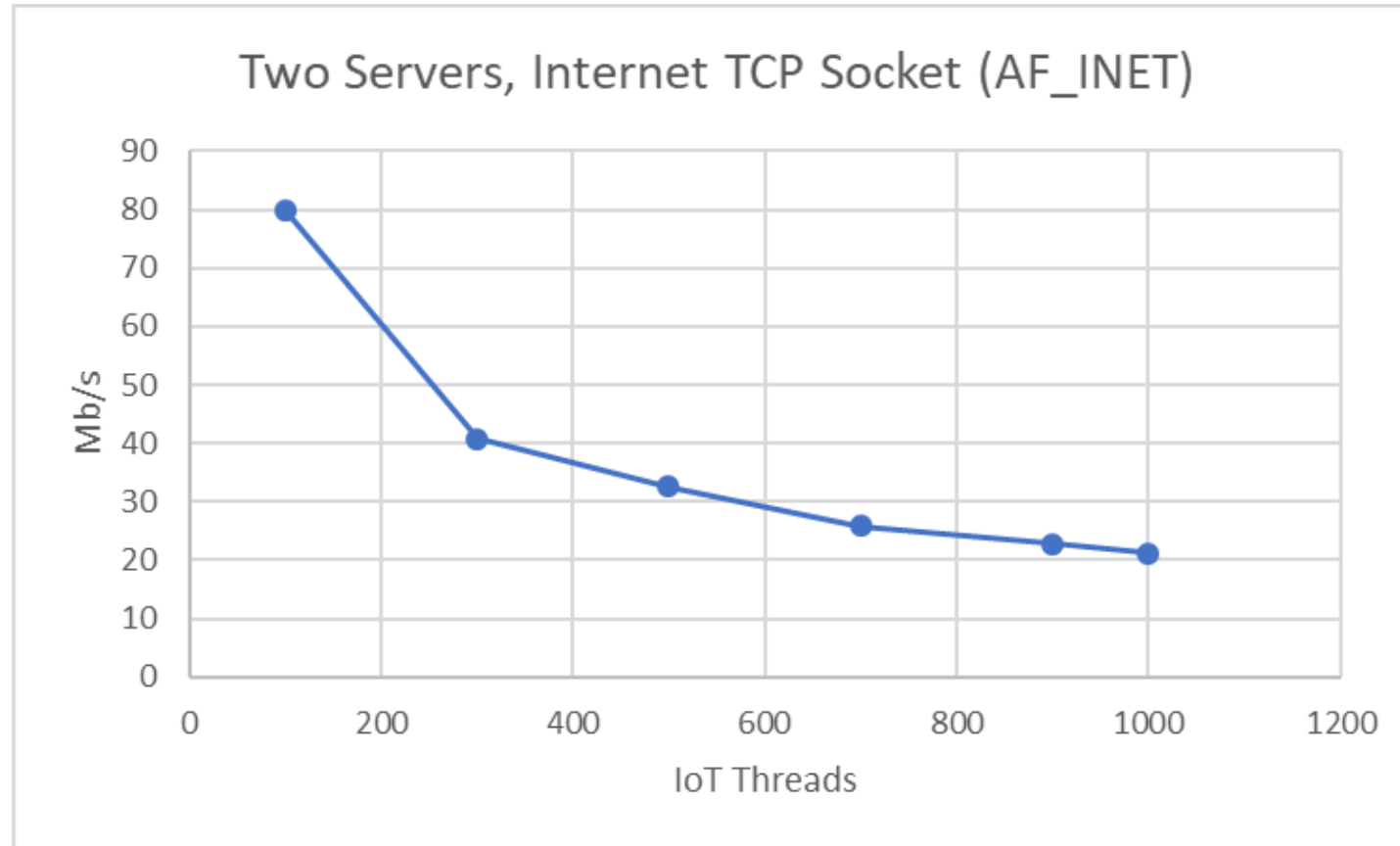
Shows:

- Number of IoT devices that entropy source can support
- Performance impact of CPU pinning

Setup notes:

- Memory buffering between emulated devices and EaaS server.
- All network effects removed

EaaS Scaling Experiments: 2 Servers, Network Connection

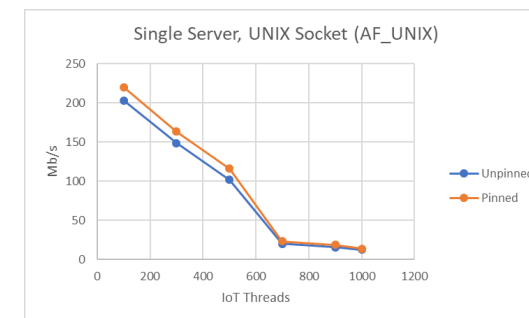


Shows:

- Number of IoT devices that can be comfortably supported over a network

Setup notes:

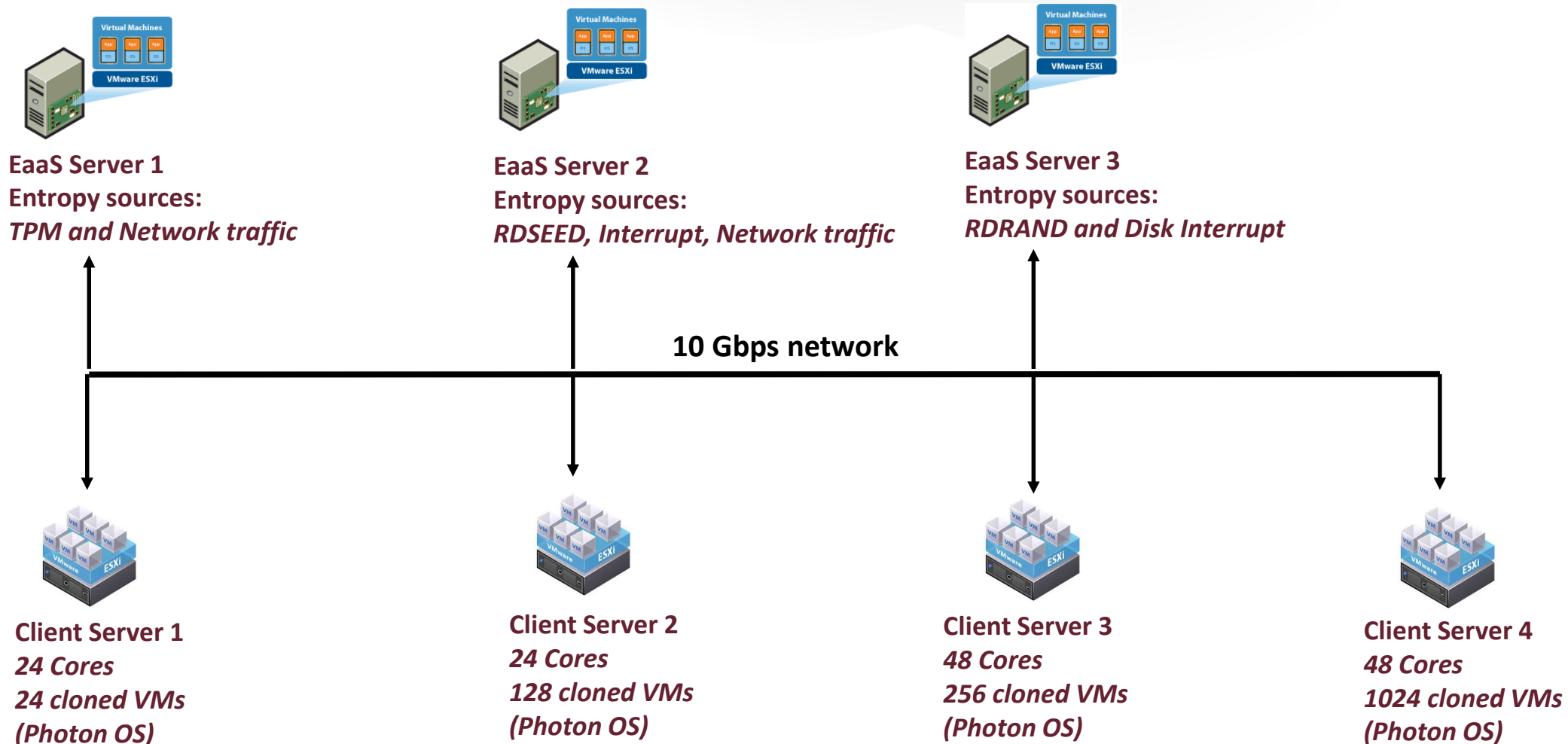
- Includes 100 Gbps network between EaaS server and emulated IoT devices
- But, no appreciable transmission time, queuing delay, congestion effects



EaaS Scaling Experiments: Client boot time entropy

Setup Notes:

- EaaS Server: running VMware ESXi
- Clients: VMware Photon OS running on ESXi



EaaS Scaling Experiments: VM clone boot time entropy

Time to accumulate 256 bits of entropy after boot

	EaaS Entropy Source	Single Entropy Source (Interrupts)	Dual Entropy Sources (Interrupts, network)
<i>Cloned VMs on 2.6 GHz core</i>	<i>Time required for entropy accumulation from EaaS</i>	<i>Time required for entropy accumulation from interrupts</i>	<i>Time required for e.a. from interrupts and network traffic</i>
24 cores / 24 clones	65 ms – 300 ms	1.0 – 1.8 sec	600 ms – 900 ms
24 cores / 48 clones	72 ms – 800 ms	1.2 sec – 2.6 sec	714 ms – 1.9 sec
48 cores / 256 clones	96 ms – 1.2 sec	1.7 sec – 9.0 sec	1.2 sec – 7.0 sec
48 cores / 1024 clones	147 ms – 5.1 sec	5 sec – 220 sec	12 sec – 140 sec

Q: How long does it take for a PhotonOS Cloned VM to get 256-bit entropy seed?

Future Work

Error Conditions

- Need more comprehensive treatment

Testing

- Production environments

IoT devices

- Testing scheme with real IoT devices

Apply What You Have Learned

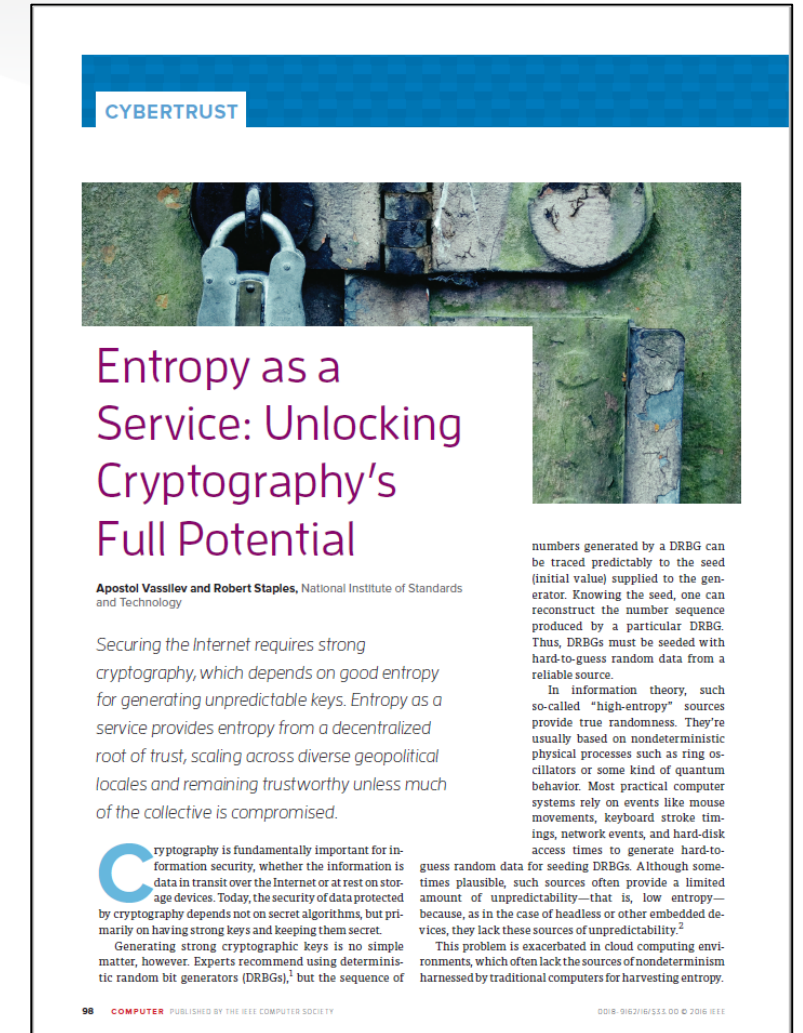
- What are your device case studies?
 - Especially in 5G and cloud environments
- Critical steps for EaaS adoption:
 - Updating 800-90B in recognizing EaaS as an entropy source
 - Formal procedure to validate EaaS service
 - Or existing entropy test & justification procedure is good enough?
 - Experience in running EaaS service
 - This experiment falls under this bucket
 - Select a security system which allows proactive policy to be set according to your organization's needs
 - Drive an implementation project to protect all critical databases
- Can you help with EaaS JSON command set definitions?

Acknowledgment



Apostol Vassilev
Research Team Lead
Security Test, Validation and Measurement Group
NIST

Robert Staples
Security Test, Validation and Measurement Group
NIST



IEEE Computer, vol 49, no 9. September 2016.

RSA[®]Conference2020

Thank You!

Abstract

Weak or poorly designed entropy sources undermine the strength of cryptographic protections in IoT communications and cloud. Entropy as a Service (EaaS), a framework developed in collaboration with NIST, offers a practical and effective solution. We discuss how the solution works to make high-quality entropy widely available to devices or systems lacking robust entropy sources.