Entropy as a Service
By
Ravi Jagannathan
Senior Security Architect, VMware

National Institute of Standards and Technology
U.S. Department of Commerce

The Problems of Weak Entropy are well known

[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.

H. Corrigan-Gibbs and S. Jana. “Recommendation for Randomness in the Operating System, or, How to Keep Evil Children Out of Your Pool and Other Random Facts”. *HotOS 2015.*

Entropy as a Service (EaaS)

Proposal:

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

Entropy Server

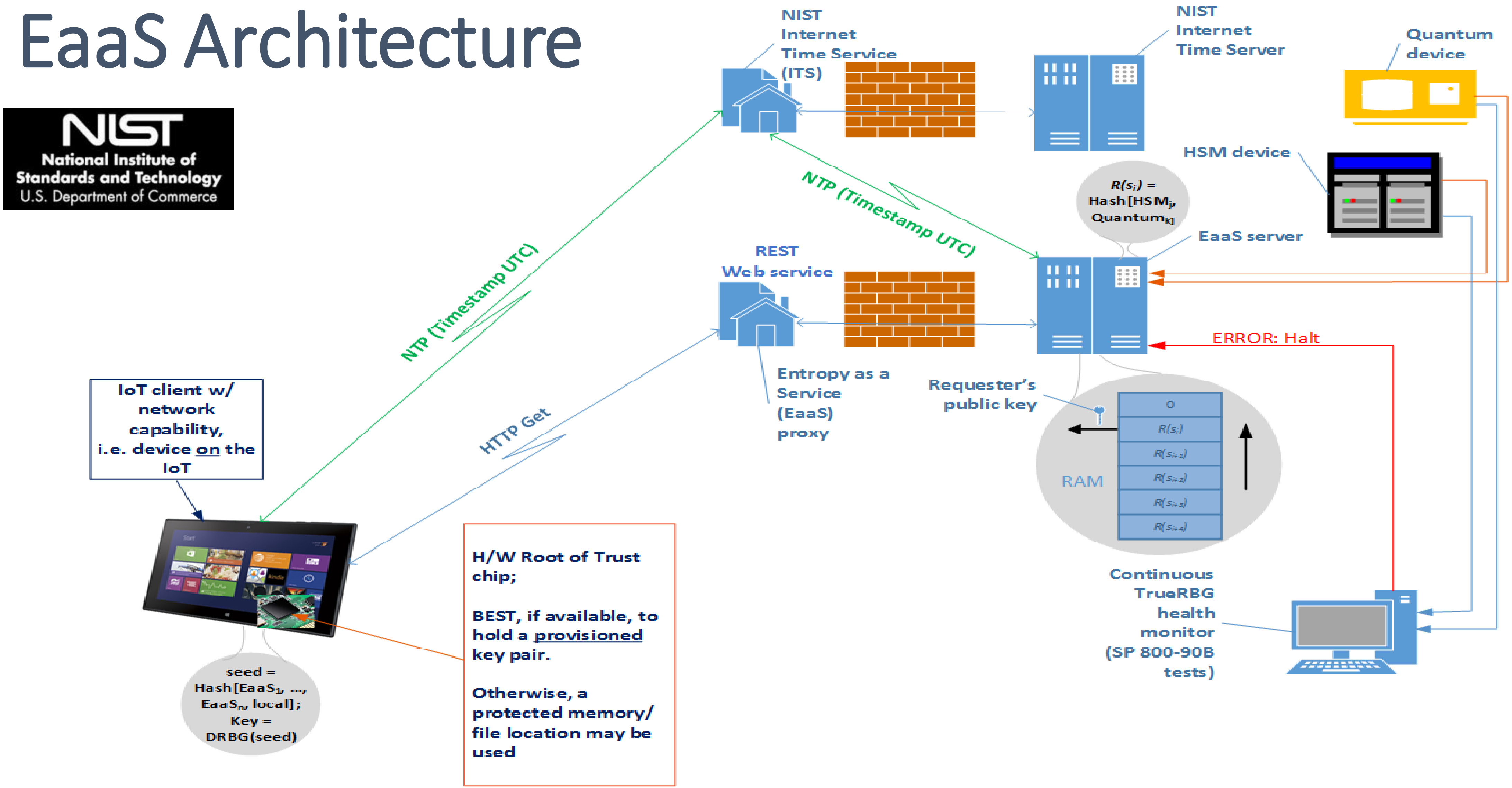
- Quantum entropy source provides continuous random data to FIFO buffer in memory
- Responds to client requests by removing random values, encrypting, sending to client

Client Devices

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

A. Vassilev and R. Staples. “Entropy-as-a-Service: Unlocking the Full Potential of Cryptography”. *IEEE Computer*. September 2016.

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. Vassilev and R. Staples. "Entropy-as-a-Service: Unlocking the Full Potential of Cryptography". *IEEE Computer*. September 2016.

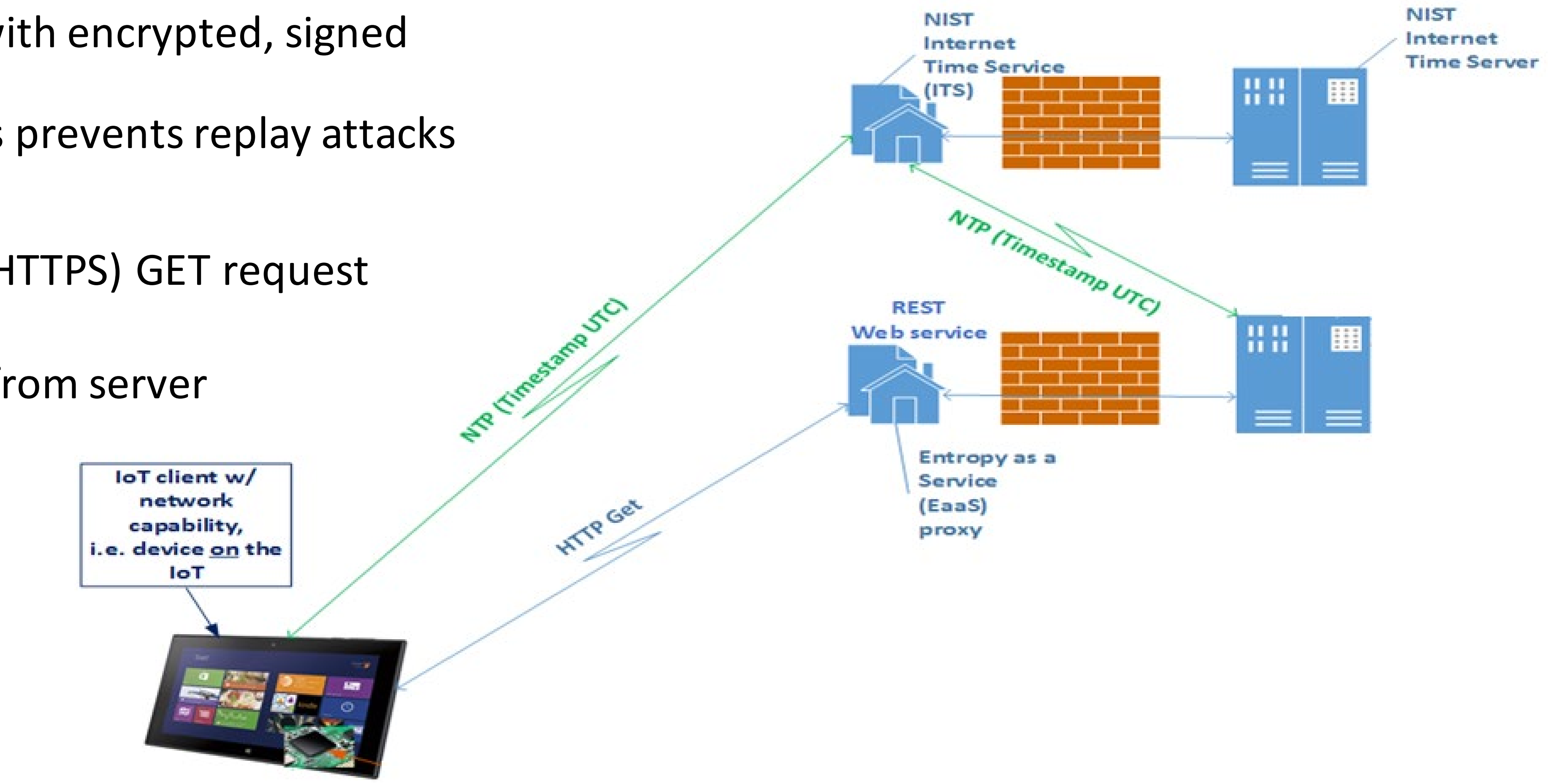
EaaS: Request/Response Protocol

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



A. Vassilev and R. Staples. "Entropy-as-a-Service: Unlocking the Full Potential of Cryptography". *IEEE Computer*. September 2016.

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
- Generate quality entropy bits through longer boot time
- Weak entropy source, but used only for brief initial time window
  - Secure Enclave

```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                            + pre_shared_key*  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]
```

TLS 1.3 handshake

E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". IETF RFC 8446. August 2018.

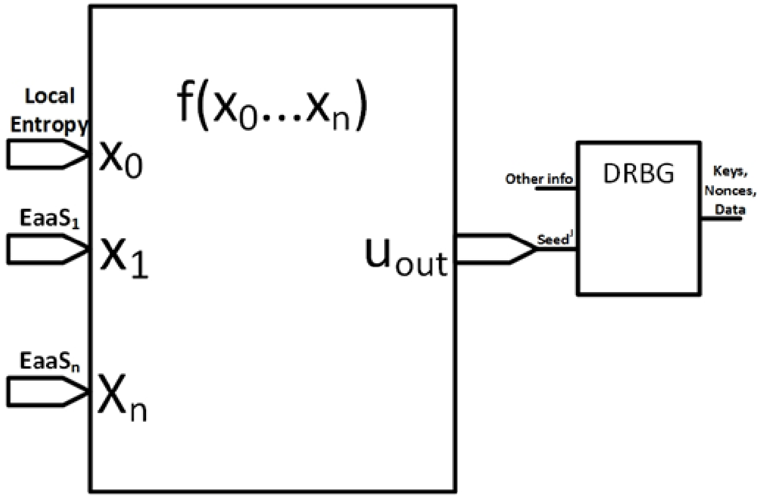
EaaS: Entropy Source

NIST:

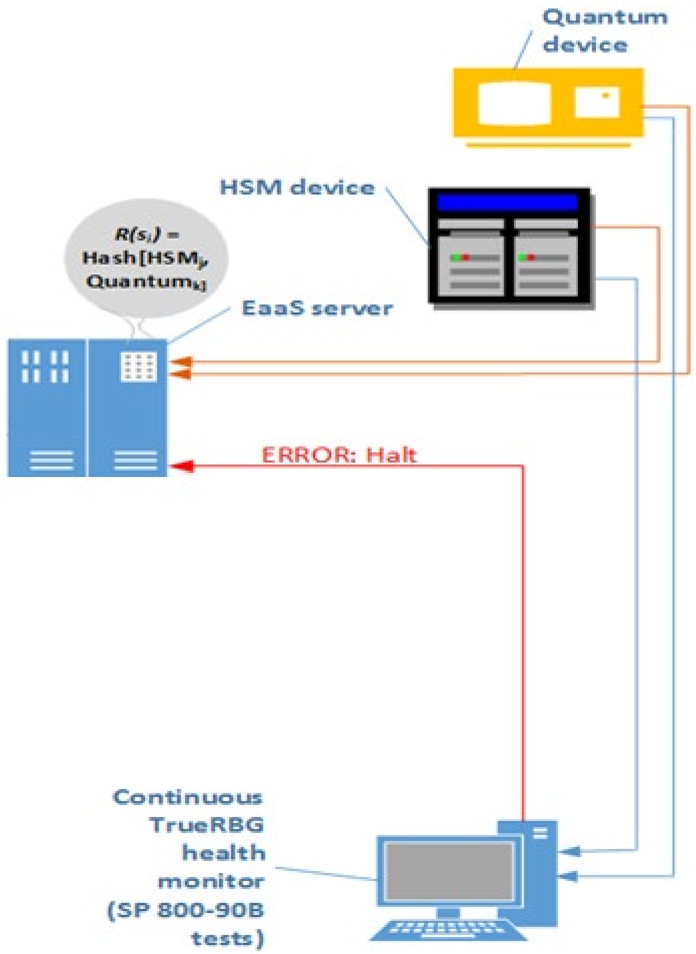
- True RBG = True Random Bit Generator (e.g., quantum device)
- SP 800-90B compliant
- Continuous monitoring solution

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



Client Device

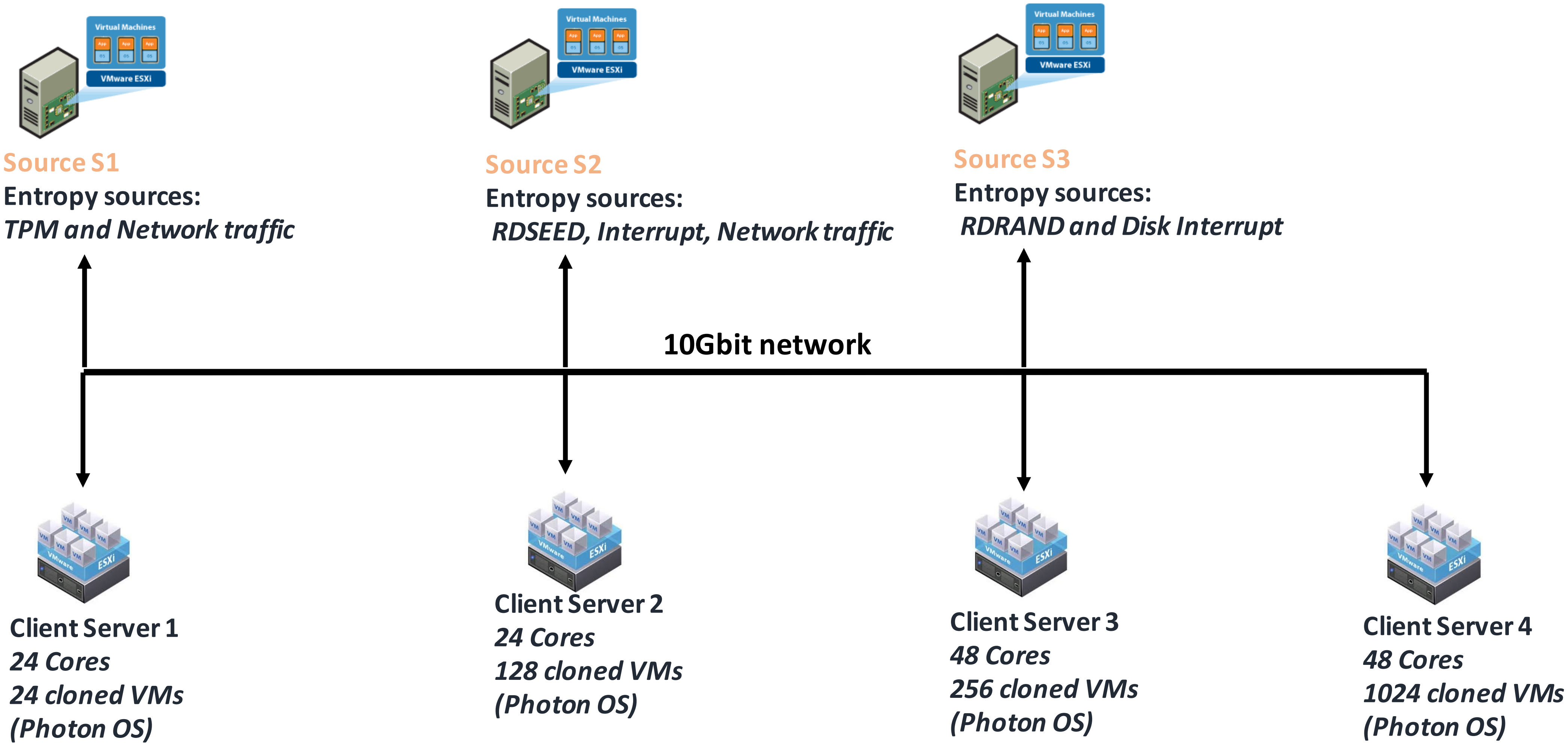


Entropy Server

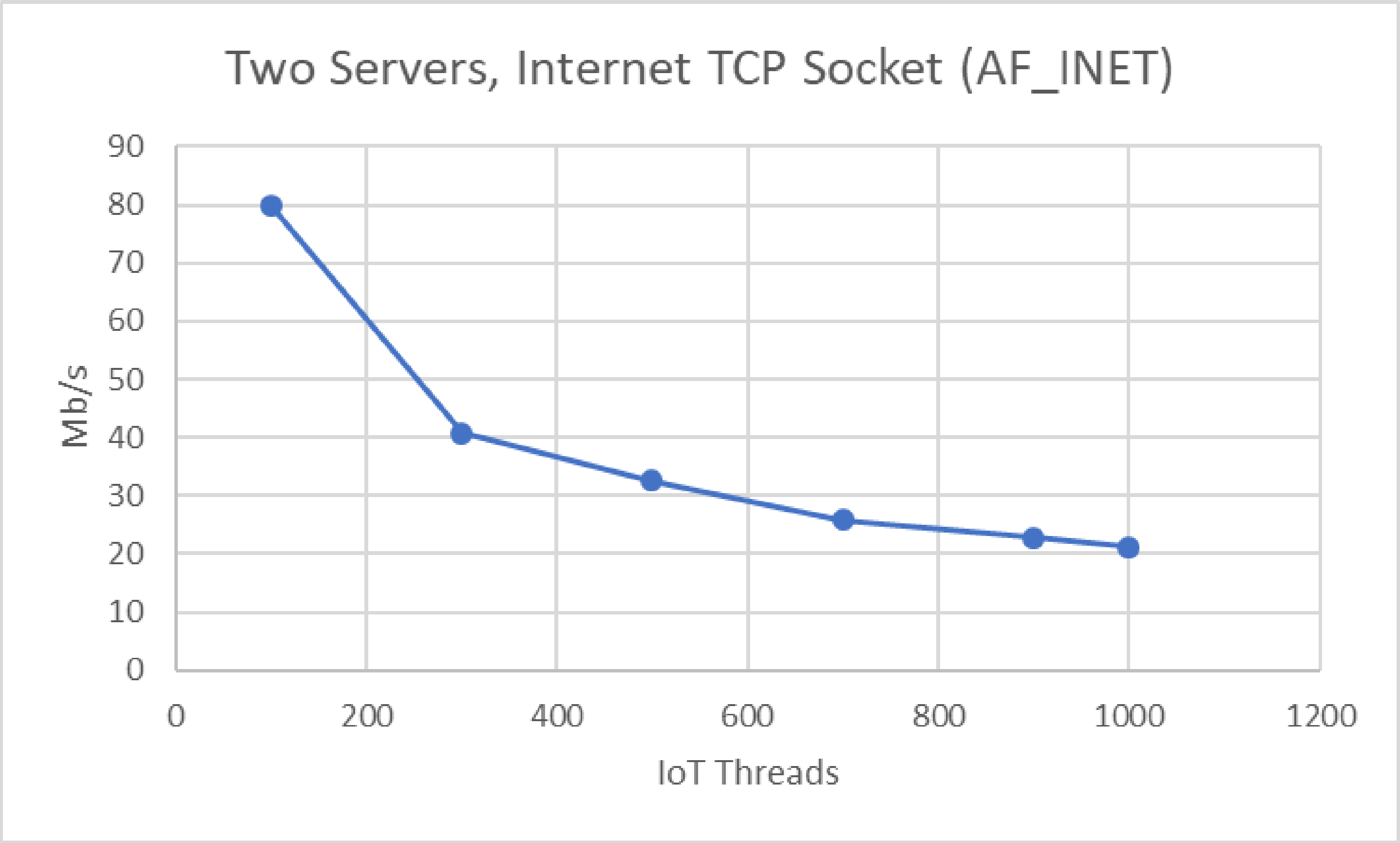
EaaS Scaling: JSON command set Prototype

Setup Notes:

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



EaaS Scaling: Two Servers, Network Connection

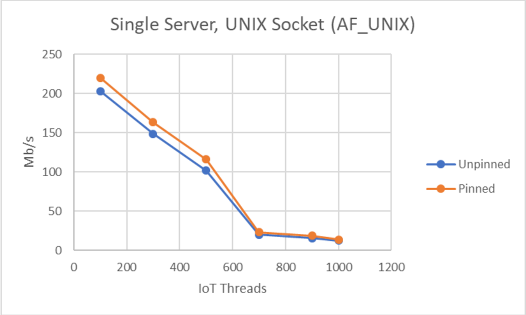


Shows:

- 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



Call to Action

- 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?
- What are your device case studies?
    - Especially in 5G and cloud environments

Wanna help EaaS JSON protocol Development?

1 Get in touch

Ravi Jagannathan

jravi@vmware.com

2 Interested in prototyping?

Need people in Cloud environment with Containers

3 IoT / 5G devices?

Interested? Contact Ravi Jagannathan, jravi@vmware.com

4 Any other thoughts?

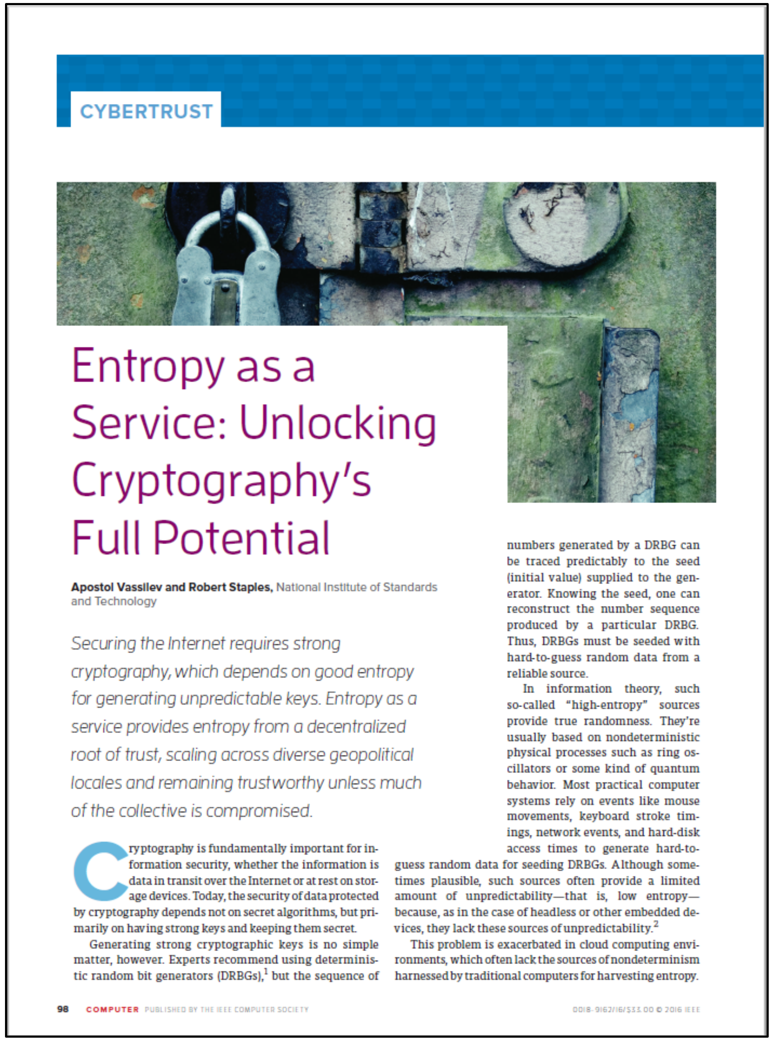
Please get in touch.

Acknowledgment

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

Robert Staples
Security Test, Validation and Measurement Group
NIST

National Institute of Standards and Technology
U.S. Department of Commerce



CYBERTRUST

Entropy as a Service: Unlocking Cryptography's Full Potential

Apostol Vassilev and Robert Staples, National Institute of Standards and Technology

Securing the Internet requires strong cryptography, which depends on good entropy for generating unpredictable keys. Entropy as a service provides entropy from a decentralized root of trust, scaling across diverse geopolitical locales and remaining trustworthy unless much of the collective is compromised.

Cryptography is fundamentally important for information security, whether the information is data in transit over the Internet or at rest on storage devices. Today, the security of data protected by cryptography depends not on secret algorithms, but primarily on having strong keys and keeping them secret.

Generating strong cryptographic keys is no simple matter, however. Experts recommend using deterministic random bit generators (DRBGs),[1] but the sequence of numbers generated by a DRBG can be traced predictably to the seed (initial value) supplied to the generator. Knowing the seed, one can reconstruct the number sequence produced by a particular DRBG. Thus, DRBGs must be seeded with hard-to-guess random data from a reliable source.

In information theory, such so-called "high-entropy" sources provide true randomness. They're usually based on nondeterministic physical processes such as ring oscillators or some kind of quantum behavior. Most practical computer systems rely on events like mouse movements, keyboard stroke timings, network events, and hard-disk access times to generate hard-to-guess random data for seeding DRBGs. Although sometimes plausible, such sources often provide a limited amount of unpredictability—that is, low entropy—because, as in the case of headless or other embedded devices, they lack these sources of unpredictability.[2]

This problem is exacerbated in cloud computing environments, which often lack the sources of nondeterminism harnessed by traditional computers for harvesting entropy.

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

Thank You!

vmware®

Thank You!

vmware

Thank You!