Entropy as a Service

Unlocking the full potential of cryptography

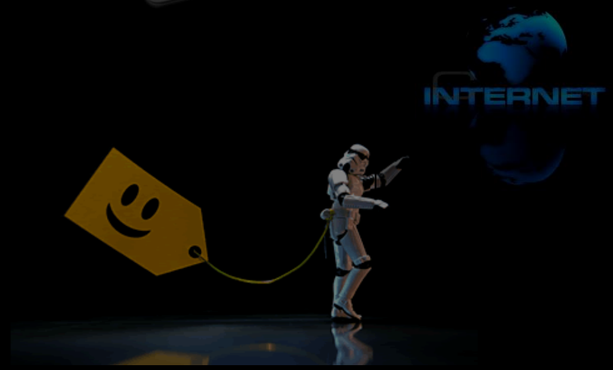
Apostol Vassilev
Robert Staples

STVM/CSD/NIST

<u>A perspective</u>: cryptography evolves very fast to provide security in cyberspace

Emerging crypto technologies

- **lightweight crypto**
- **lighter versions of legacy protocols**
  - **tinyDTLS, lightweight DTLS**
- **post-quantum cryptography**



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

Courtesy of XKCD

- **In modern cryptography the algorithms are well-known**
- **Security depends largely on the secrecy of the keys**
- Cryptographic keys must be (nearly) impossible to guess
- Key generation and key management govern the strength and security of keys
- Key generation is strongly dependent on entropy

The big question

Where are the keys coming from?

Real World Examples 2013

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

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

<u>Problem</u>: Low-quality hardware RNG, stuck in a short cycle:

0 0 1 etc.

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

- able to obtain RSA private keys for 0.50% of TLS hosts and 0.03% of SSH hosts

- **public keys shared nontrivial common factors due to entropy problems**

- able to obtain DSA private keys for 1.03% of SSH hosts

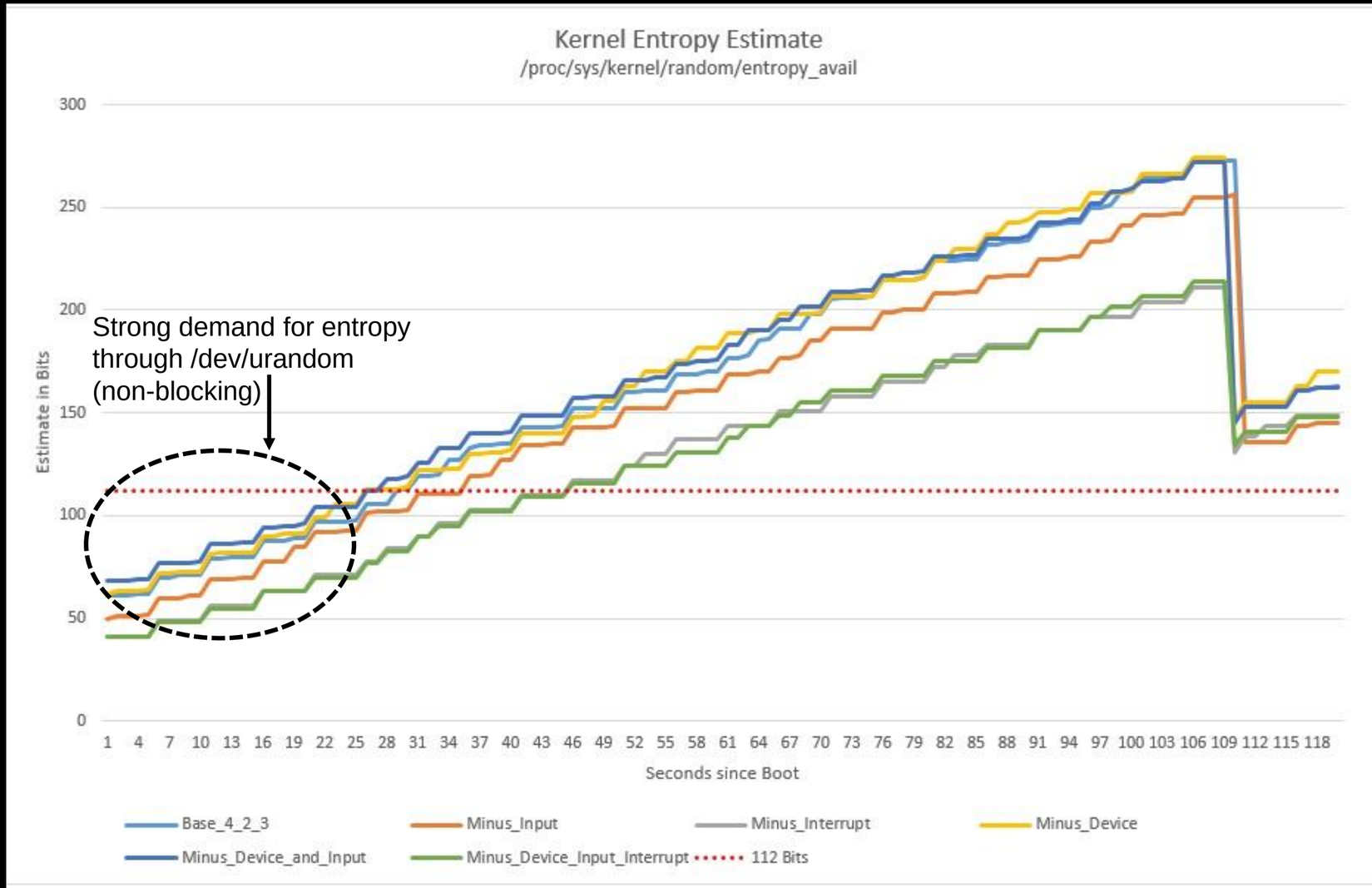
- **due to insufficient signature randomness**

Real World Examples 2016

The Linux kernel dissected - <u>four</u> sources of kernel 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: expand a well-known irrational number, e.g. π, and test the output bit sequence for randomness. Chances are, it will be reported as random.

Using the statistical test approach 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**
- **clients seed DRBG's after mixing EaaS random data with local random data**
- **clients use the DRBG output to generate local keys independently from EaaS**

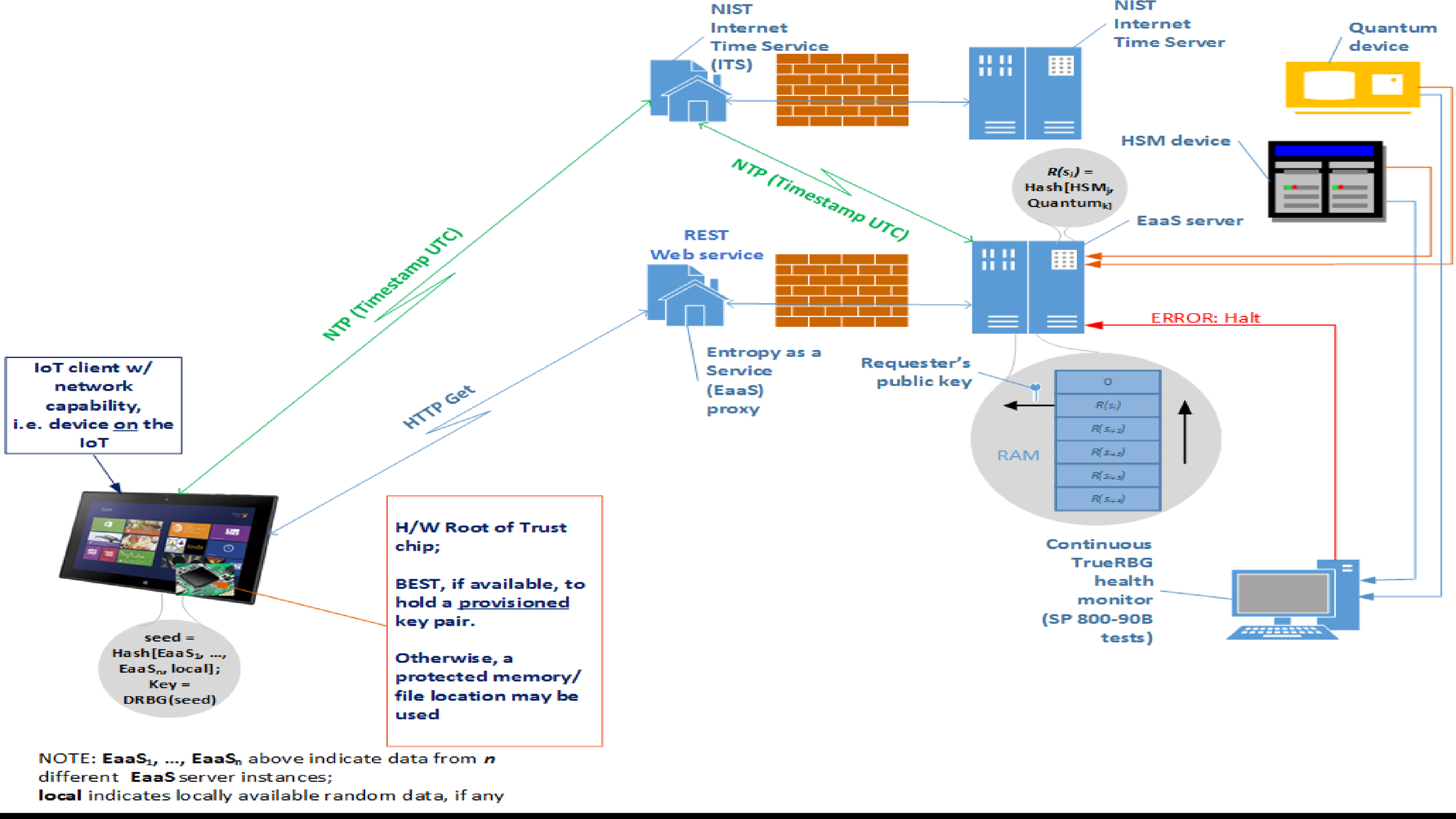
Our solution is

NOT a key generation service

- **cryptographic keys are generated locally on the client using DRBG's**
- **clients DRBG's are seeded with random data resulting from mixing several independent sources, including local random data**
- **even if an attacker gains full control of one server, he/she will have no possibility of gaining meaningful insights into the clients' keys**

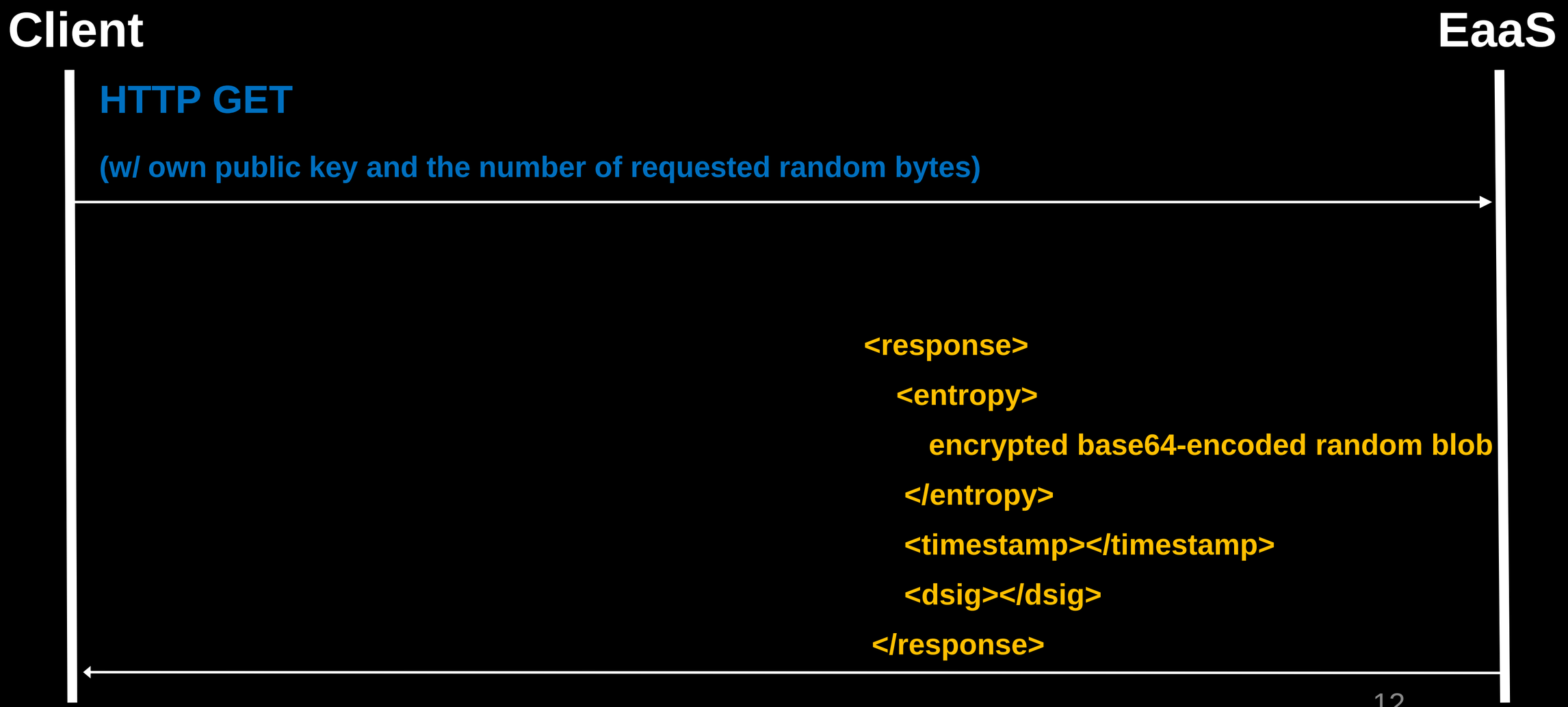
NOT similar to the NIST beacon

- **EaaS does NOT record any incoming or outgoing record**
- **EaaS does NOT record any internally generated random data**



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



Potential attacks and defenses

Attacks mitigated by protocol features and provisioning

- Message replay

- Man-In-The-Middle

- DNS poisoning

Potential attacks and defenses

Attacks mitigated by mixing data from multiple EaaS instances

- Honest-but-curious EaaS instance

- Dishonest but non-colluding EaaS instances

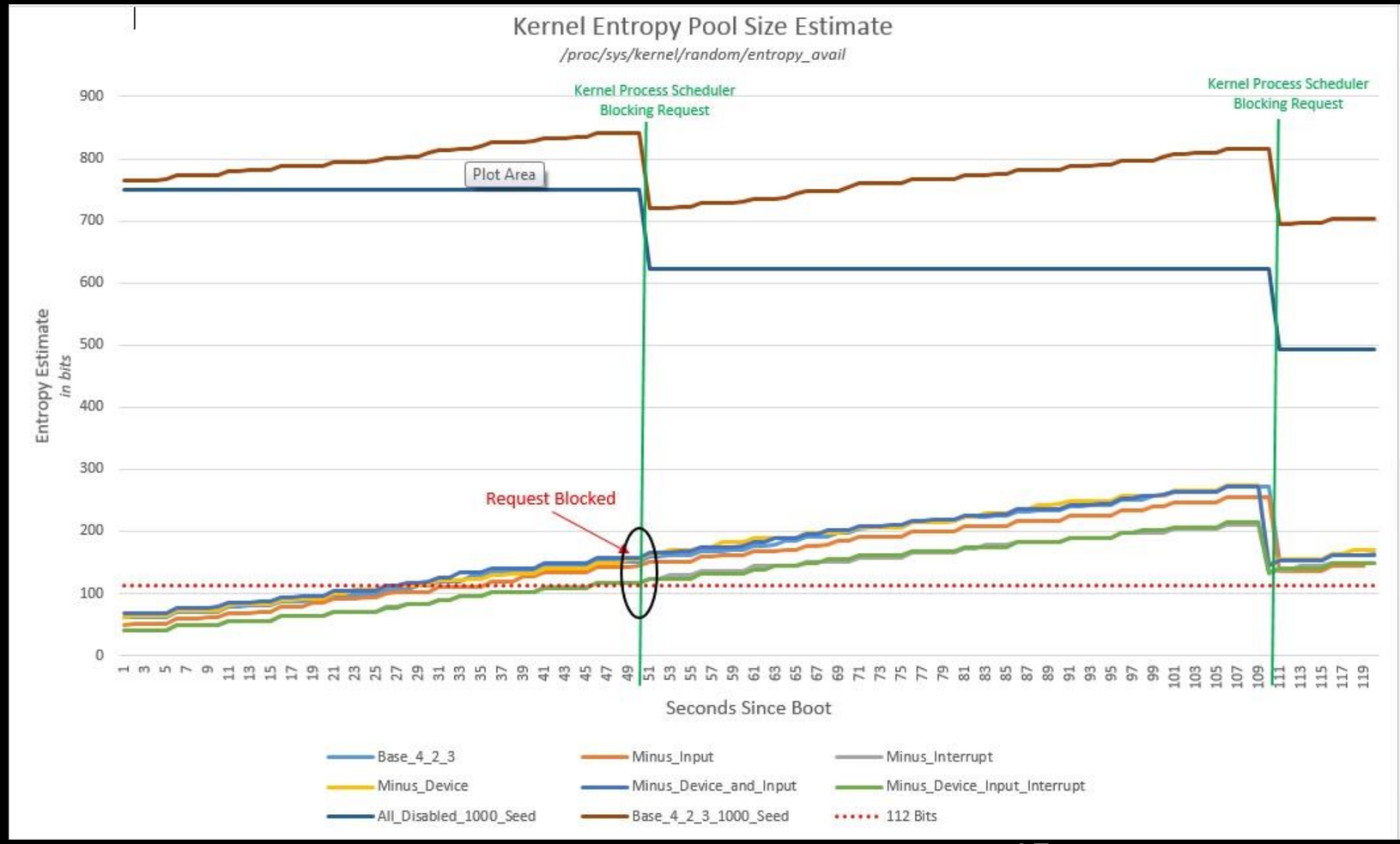
- Dishonest and colluding EaaS instances

Status and next steps

Linux Kernel Entropy Revisited:

- **uses EaaS client to request random bits from EaaS server and seeds the image pool**

- **C program, using the "`RNDADDENTROPY`" `ioctl` to add the entropy.**



Status and next steps

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

Functional prototype implemented

- **demonstrated in September at the CIF 2015 in Washington, DC**

Planning to stand-up a publicly accessible NIST EaaS in Q2, 2016

- **publish server and client sample code on GitHub**
  - **allow people to look, enhance, adopt as they please**

Conceive a public criteria for reputable EaaS hosts

Questions?