Entropy-as-a-Service: Unlocking the Full Potential of Cryptography

Apostol Vassilev and Robert Staples, NIST

Abstract: *Securing the Internet requires strong cryptography, which depends on the availability of good entropy for generating unpredictable keys and accurate clocks. Attacks abusing weak keys or old inputs portend challenges for the Internet. EaaS is a novel architecture providing entropy and timestamps from a decentralized root of trust, scaling gracefully across diverse geopolitical locales and remaining trustworthy unless much of the collective is compromised.*

Disclaimer
The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

Introduction

Cryptography is fundamentally important for information security, whether 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. Experts recommend using the output from deterministic random bit generators (DRBGs), to generate keys for cryptographic applications [2]. However, 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 sequence of numbers a particular DRBG produces. Thus, DRBGs must be seeded with hard-to-guess random data from a reliable source.

In information theory, such sources are referred to as "high-entropy" sources that provide true randomness. Usually they are based on nondeterministic physical processes such as ring oscillators or some kind of quantum behavior. Most practical computer systems rely on events such as mouse movements, keyboard stroke timings, network events and hard disk access times to generate hard-to-guess random data for seeding DRBG's. Although sometimes plausible, such sources often provide a limited amount of unpredictability, i.e. low entropy, because, as in the case of headless or embedded devices, they lack these sources of unpredictability, cf. [1]. This problem is exacerbated in cloud computing. Cloud computing environments often lack the sources of non-determinism harnessed by traditional computers for harvesting entropy. Cloud service providers typically use a single reference image of a guest virtual machine (a "golden" image) in order to create multiple instances of it in response to user demand. Each instance often has very limited ability to harvest randomness.

Another domain where the demand for good cryptographic keys is strong is the Internet of Things, or "IoT". IoT devices tend to be small, resource-constrained, headless or embedded but their functional capabilities span a wide range. Although characterizing the ability of IoT devices to generate good cryptographic keys is out of scope for this paper, it is reasonable to think that some types of IoT devices with network connectivity and modest computational power to perform asymmetric cryptographic operations may also benefit from the proposed service architecture.

There is a growing need for strong entropy on the Internet and the Entropy-as-a-Service (EaaS) system is designed to help.

1 Recent Findings of Poor Entropy

The concerns about the potential weakness of cryptographic keys are far from just theoretical. This section briefly reviews real-life examples of catastrophic security breaches resulting from poorly constructed or predictable cryptographic keys.

1.1 Mining Your Ps and Qs

A recent study [1] provided one of the most comprehensive Internet-wide searches of SSH and TLS servers to date, checking 12.8 million TLS and 23 Million SSH hosts.

In this survey, the researchers discovered alarming results: 5% of HTTPS and 10% of SSH hosts shared keys because of insufficient entropy from the source used, allowing them to actually calculate the private keys of 0.5% of HTTPS hosts and 1% of SSH hosts.

Almost all vulnerable hosts were headless or embedded network devices like routers or firewalls. Such hosts often run a pared-down Linux kernel and do not have the usual random events from input devices that a desktop computer would have. As a result, there exists an "entropy hole" in which the output of `/dev/urandom` could be constant across multiple boots for a period of time early in the boot process. In one case, the same key was generated in over 25% of boots.

1.2 Entropy-starvation in Embedded Devices

We built on the approaches from the study discussed above [1] to investigate the strength of the Linux kernel entropy sources. In particular, we simulated the behavior of an embedded or headless device, without a hard drive or a keyboard/mouse, always starting with empty pool (i.e. no "seed"). We built a pared-down Linux kernel with different combinations of kernel entropy input devices disabled. Depending on the combination of entropy sources disabled, the entropy count in the pool took anywhere from 20 to 45 seconds to generate the bare-minimum threshold of 112 bits – see Figure 1. More time was needed to reach the threshold when some contributing sources were turned off, simulating environments with constrained resources.

This experiment illustrates the potential weakness of Linux kernel entropy sources in embedded/headless Internet deployments, e.g., in cloud environments. We observed particularly strong demand for entropy through the unblocking /dev/urandom interface with requests as high as 4096-bits shortly after boot when little random data is accumulated, highlighted with the black dashed circle in Figure 1. This behavior of Linux in fact opened the door for the exploits described in [1].

2 Entropy-as-a-Service

The security issues resulting from the effects of poor entropy discussed so far illustrate the fundamental importance of good randomness for security – see also "The Importance of Entropy to Information Security", A. Vassilev and T. Hall, IEEE Computer, 47(2), pp.79-81, 2/2014.

The existing technological headwinds that hinder the implementation of robust random bit generation capabilities in conventional computing devices make apparent the need for alternative means of providing high-quality entropy to devices that cannot produce their own in sufficient quantity or quality.

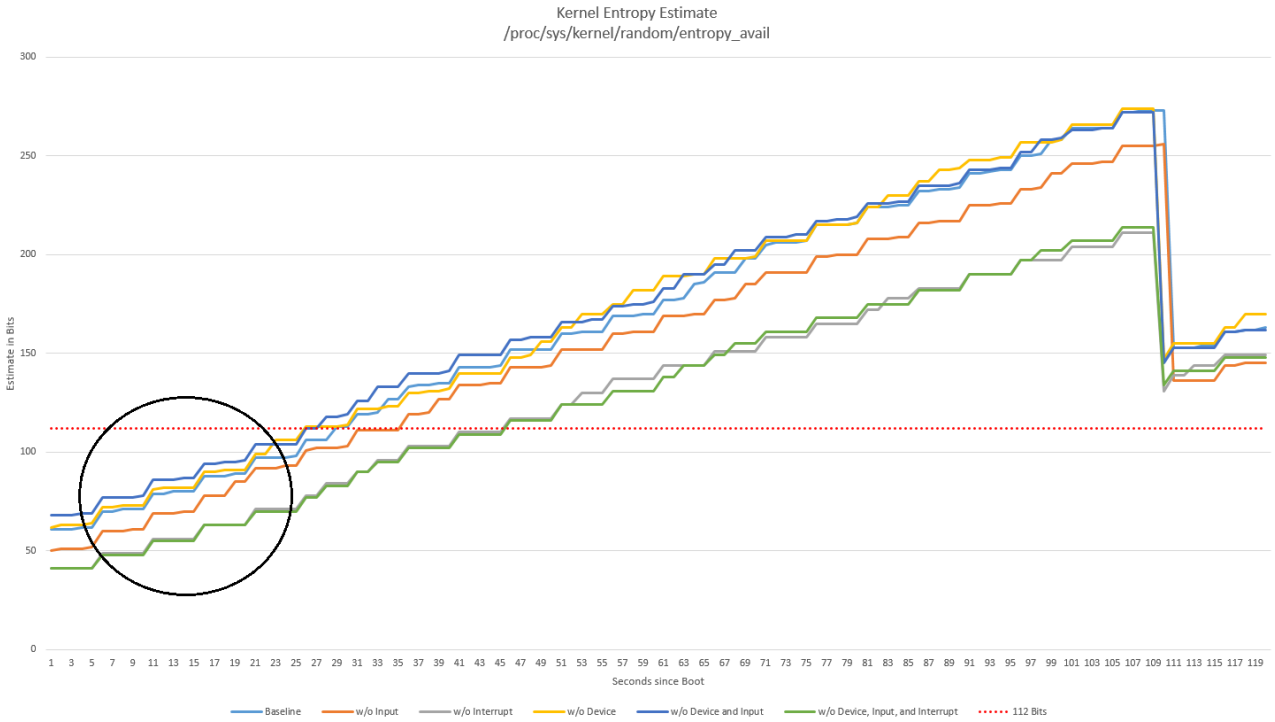

Figure 1 Linux kernel entropy accumulation

The widely available and highly redundant nature of the Internet creates an effective medium by which to provide good random data to needy clients, in this case using a REST (cf., http://www.restapitutorial.com/) interface, i.e. an Entropy-as-a-Service (EaaS) solution. Instead of relying solely on weak pseudo-random data in classical computers, EaaS provides a novel and secure way of delivering high-entropy data to requesting devices. EaaS leverages existing protocols and technologies, which makes adoption easy.

EaaS uses HTTP to transfer entropy payloads from the service to clients. To secure this transmission, the server encrypts the data using the client's provided public key and digitally signs the payload with the server's own private key.

2.1 A sketch of a protocol

The client makes a HTTP GET request to the EaaS server, with the number of bytes of random data to return, and its own public key, which is used to encrypt the returned payload.

The structure of the server XML response is shown below.

Successful Response

```
<response>
  <entropy>
     encrypted base64-encoded
     random data
  </entropy>
  <timestamp></timestamp>
  <dsig></dsig>
</response>
```

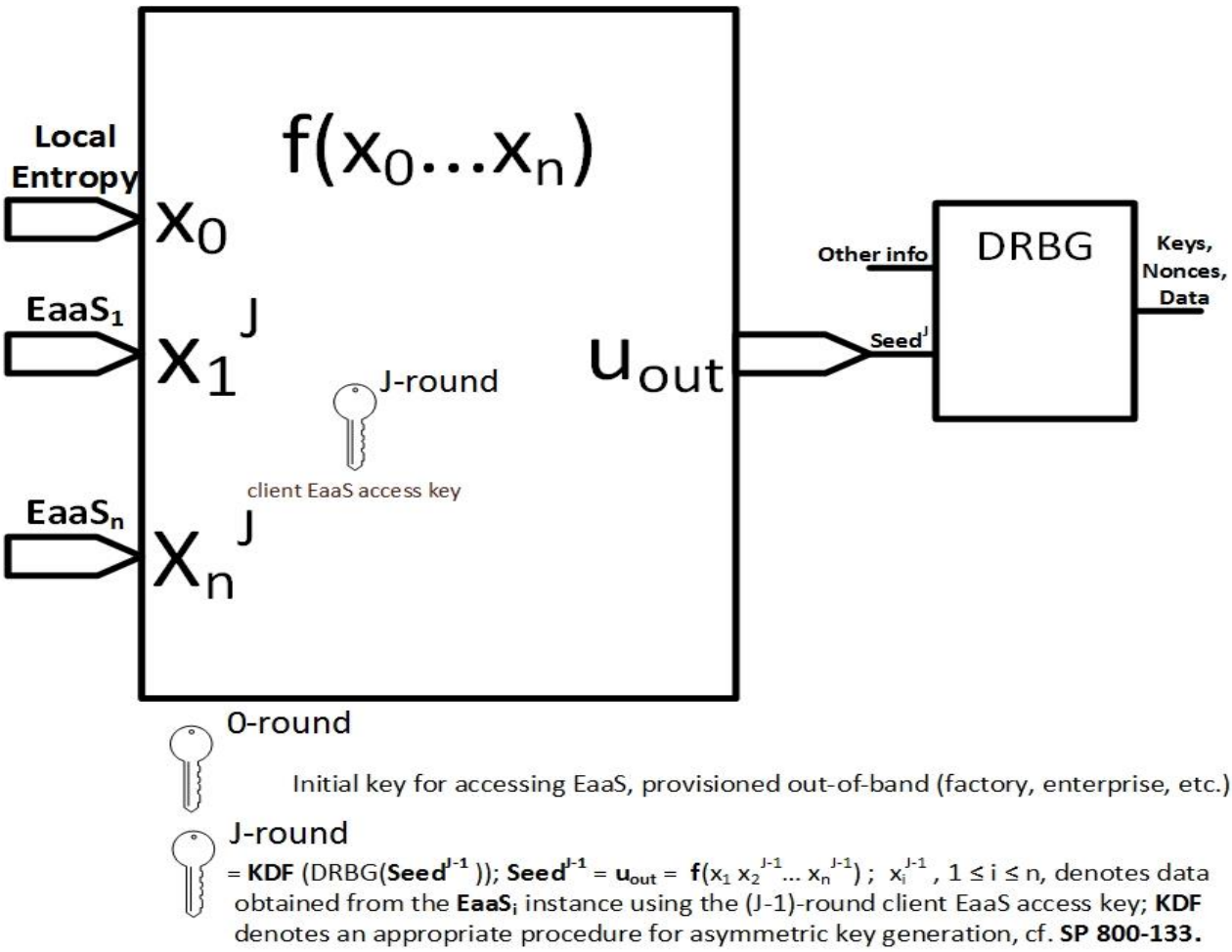

Figure 2 Client key management

Here, the tags are: *Entropy* – the payload, encoded in base64 format; *Timestamp*; *Signature (dsig)*.

2.2 Resolving the chicken-and-egg conundrum

Clients need a public key to access EaaS and request high-entropy data to strengthen their key generation capabilities. How then, can a client have a strong public key to come to the EaaS system in the first place?

The critical observation here is that it is much easier and cheaper to generate strong keys out-of-band than to implement robust random bit generation capabilities in conventional devices. Manufacturers of devices can, and often do, generate strong keys in the factory and provision them on devices. This model of key provisioning is well known and widely used in many industries, including smart card and TPM manufacturing. When a customer receives a shipment of devices to deploy, they also receive through independent means the secrets required to change the factory keys on each device and assume ownership of the devices.

In order for an adversary to break this model of provisioning and take control of a deployed device, they must penetrate the manufacturer factory security and record every device key issued. In addition, the adversary must monitor every single interaction between a device and EaaS. Missing just one such interaction would render the attacker defeated. This sets a very high security bar for attackers.

Another mitigation against such potential attacks is to always mix the externally-obtained random data with locally-generated pseudo-random data using suitable cryptographic mechanisms, e.g., hashing, and renew the

EaaS access key on each round, as illustrated in Figure 2. Note that the mechanism for updating the client key for accessing EaaS can be shown to provide perfect forward secrecy.

3 The EaaS architecture

The architecture of the Entropy-as-a-Service system consists of two main parts: the client-side and the server-side. The critical components of the system are the entropy source, the EaaS server and a secure device in the client systems capable of providing strong isolation and protection of cryptographic keys stored inside the device and offering a set of basic cryptographic services.

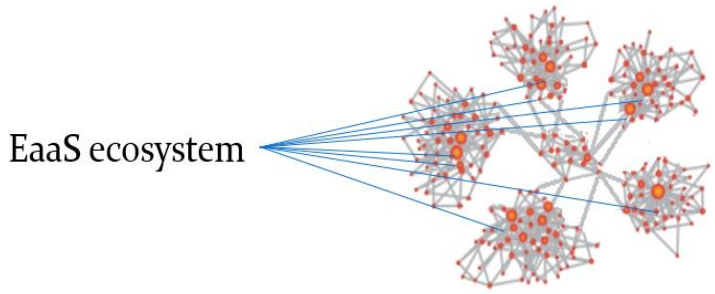

Figure 3 EaaS Ecosystem (Image Courtesy: Cornell Univ. Networks Course Blog, https://blogs.cornell.edu/info2040/2012/09/26/7720/)

3.1 The Server Side

The EaaS server is continuously fed random data from the attached quantum source. The data enters a FIFO-like buffer in the server's RAM, and when a client request comes, the server reads the top value off of the buffer, signs and encrypts it, and then sends it to the requester. The FIFO buffer shifts after every request and when new data comes from the random source. The EaaS server ensures that the FIFO buffer is erased prior to server shutdown and never paged to disk. Open implementations can help provide trust that this does, in fact, occur.

3.2 The Client Side

The client side of the system consists of a classic computing device enabled with a dedicated hardware component capable of storing secret cryptographic keys and seeds. The client system has a dedicated software application bridging the communication between the EaaS and the hardware component. Examples of secure hardware components are the "Trusted Platform Module", or "TPM", TrustZone in ARM processors, and the IPT technology in Intel processors.

Additionally, if a client system or device does not have a secure hardware component, it can still use the EaaS system. The presence of a hardware component simply provides further guarantees to the system or device user, when present.

4 Attacks and their Built-In Mitigations

One important feature of EaaS is that it transfers entropy to clients in a secure fashion.

As can be seen in Figure 4, the protocol sketch, the signature and timestamp of the response allow the client to verify the authenticity of both. Timestamping, in particular, prevents "response replay" attacks.

The digital signature protects against both man-in-the-middle attacks, when a malicious actor intercepts messages and serves as a relay, and DNS poisoning attacks, in which a malicious actor either intercepts DNS requests, or sets up a spoof server near the victim, provided the EaaS public key is provisioned on the client in advance.

Attacks involving dishonest or curious EaaS server instances are mitigated by mixing data from several sources together before use. Thus, even if multiple EaaS instances were somehow colluding against a specific client, if the client can access just one source of non-colluding entropy, including its own weak entropy pool, the efforts

of the malicious instances are mitigated, since they have no way of knowing the input from the other, good sources.

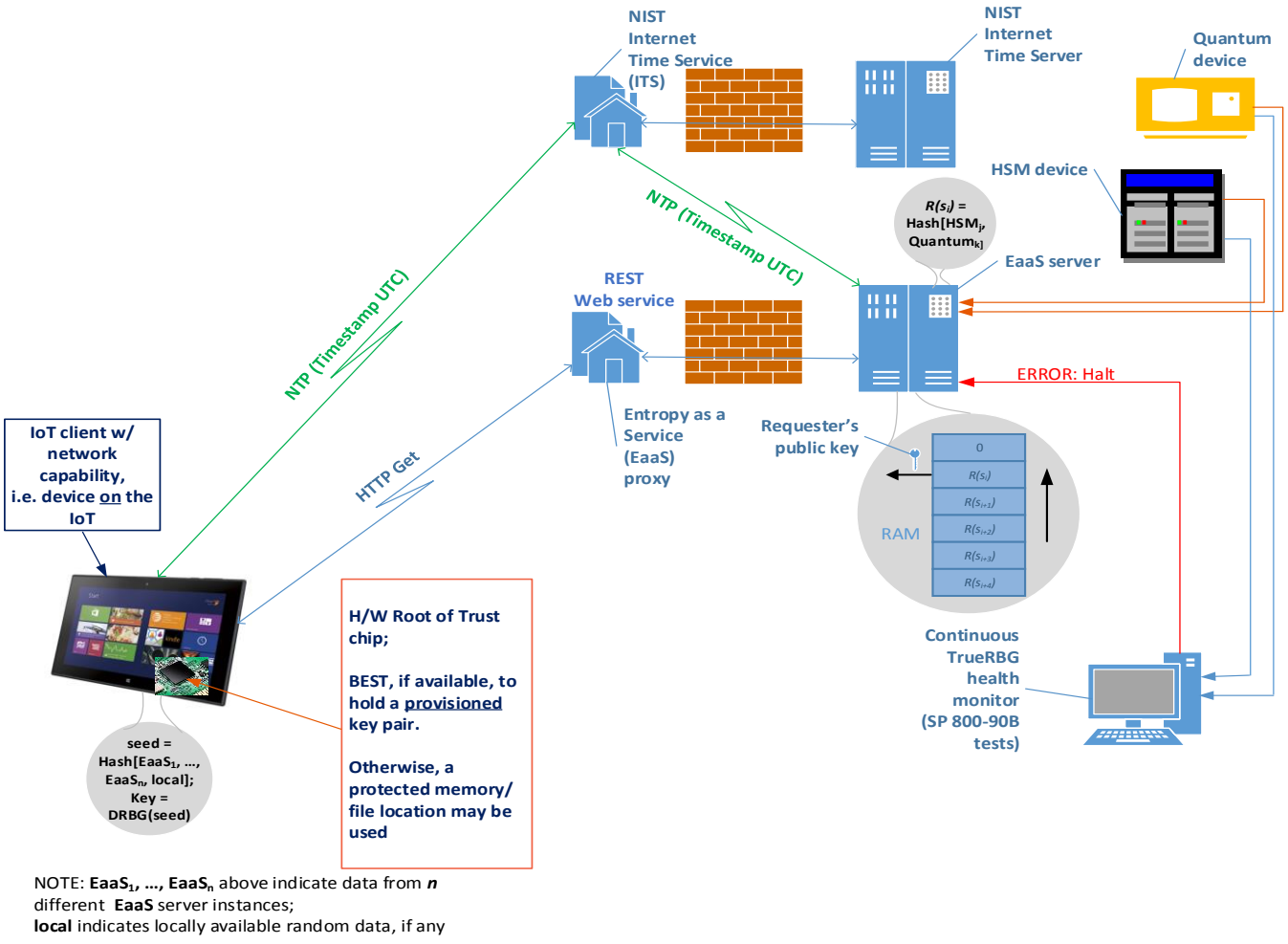

Figure 4 - The Eaas Architecture

5 Real-World Uses

5.1 Real world application: Attestation of cryptographic key strength

One example of the usefulness of this type of system is in assessing the security strength of an enterprise system. Cryptography is fundamentally important for this task and the strength of cryptographic keys being generated at the endpoints of the system is of great importance.

Endpoints using EaaS can attest the strength of keys generated from data coming from a known-good source. Additionally, enterprises could stand up their own internal EaaS, and have complete trust over the entropy in their endpoints.

5.2 Real world application: VM orchestration in cloud computing environments

Today, virtualization and cloud computing have become prevalent in the technology sphere.

Two virtual machine instances instantiated from a common ("golden") image may demonstrate similar or even identical internal state of the local entropy pool so gaining insight in one would allow insight into the other.

However, this is easily remedied by using EaaS to feed unique random data into the image after cloning, or by requesting some EaaS data on boot.

5.3 Helping the security on the Internet

Another important use case is that of headless or other embedded Internet devices that may be entropy-starved, as can be seen in [1].

One way to fix this is to use EaaS to obtain entropy on devices upon boot up. The devices could also store some entropy across boot cycles. Thus, a device is only vulnerable for a few seconds after the initial boot, until the EaaS call is made but simple design decisions may prevent key generation in this small window of time. The greatly improved behavior of Linux seeded with EaaS is shown in Figure 5.

One interesting effect illustrating the benefits of seeding early after boot with EaaS is visible in the change of behavior of the Linux Kernel Process Scheduler (LKPS) – a critical component of the operating system. LKPS needs random data to implement fair and efficient process scheduling. LKPS acquires random data through a blocking interface and can only do that when there is sufficient amount of entropy accumulated in the kernel. Notice that when Linux is seeded with EaaS, LKPS acquires its first random seed after about half the time needed in the case of non-seeded Linux, shown in the lower part of Figure 5. The black oval indicates the missed LKPS seeding due to a lack of sufficient entropy in the kernel. In other words, LKPS reaches its normal operational regime much faster when Linux is seeded from the start, thereby improving the overall stability and performance of the operating system without any additional design or configuration changes.

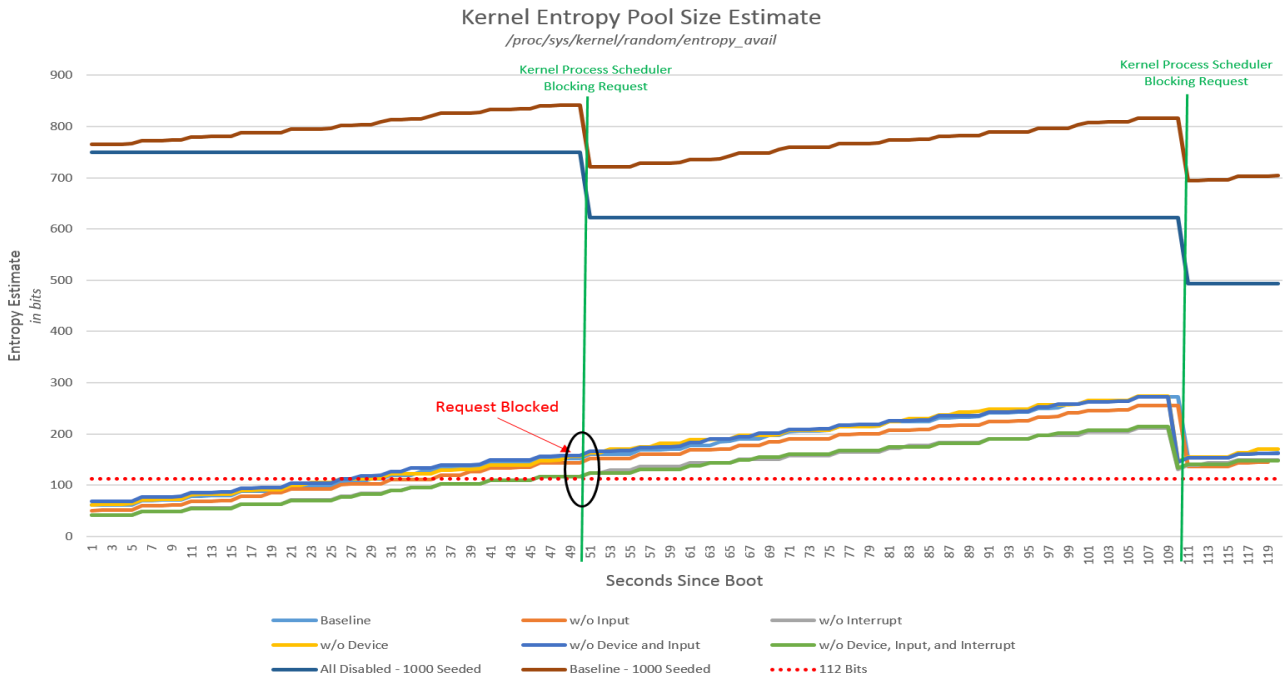

Figure 5 Linux seeded with EaaS

6 Conclusions and future plans

The proverbial "Achilles' Heel" of the assurances from cryptographic security protection is the strength of the keys used to protect critical data.

EaaS stands to create the basis of a future ecosystem of servers which can provide verifiably high-quality entropy to needy clients on request, thereby unlocking the full potential of cryptography. To facilitate the creation of the ecosystem, we plan to share our server implementation, allowing other organizations or entities to review, adopt, and host their own EaaS instances.

We also envision the need to develop criteria for establishing trustworthiness of servers participating in the ecosystem. This, in turn, would allow users of EaaS to select and rely on a subset of servers from the ecosystem that satisfies a desired level of trust/risk.

The authors welcome input and comments regarding EaaS.

7 References

[1] "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," Proc. 21st Usenix Security Symp. [Security 12], 2012; www.usenix.org/system/files/conference/usenixsecurity12/sec12-final228.pdf

[2] NIST SP 800-90A, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf