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

Entropy as a Service

Unlocking the full potential of cryptography

Apostol Vassilev, NIST
Harold Booth, NIST
Robert Staples, NIST

The Challenge

Imagine…

- My organization's security policy requires strong keys
  - Example: 256-bit AES keys that **are** 256-bit strong

- But, what does that mean? How to measure it?

- The solution: get entropy from a known good source!

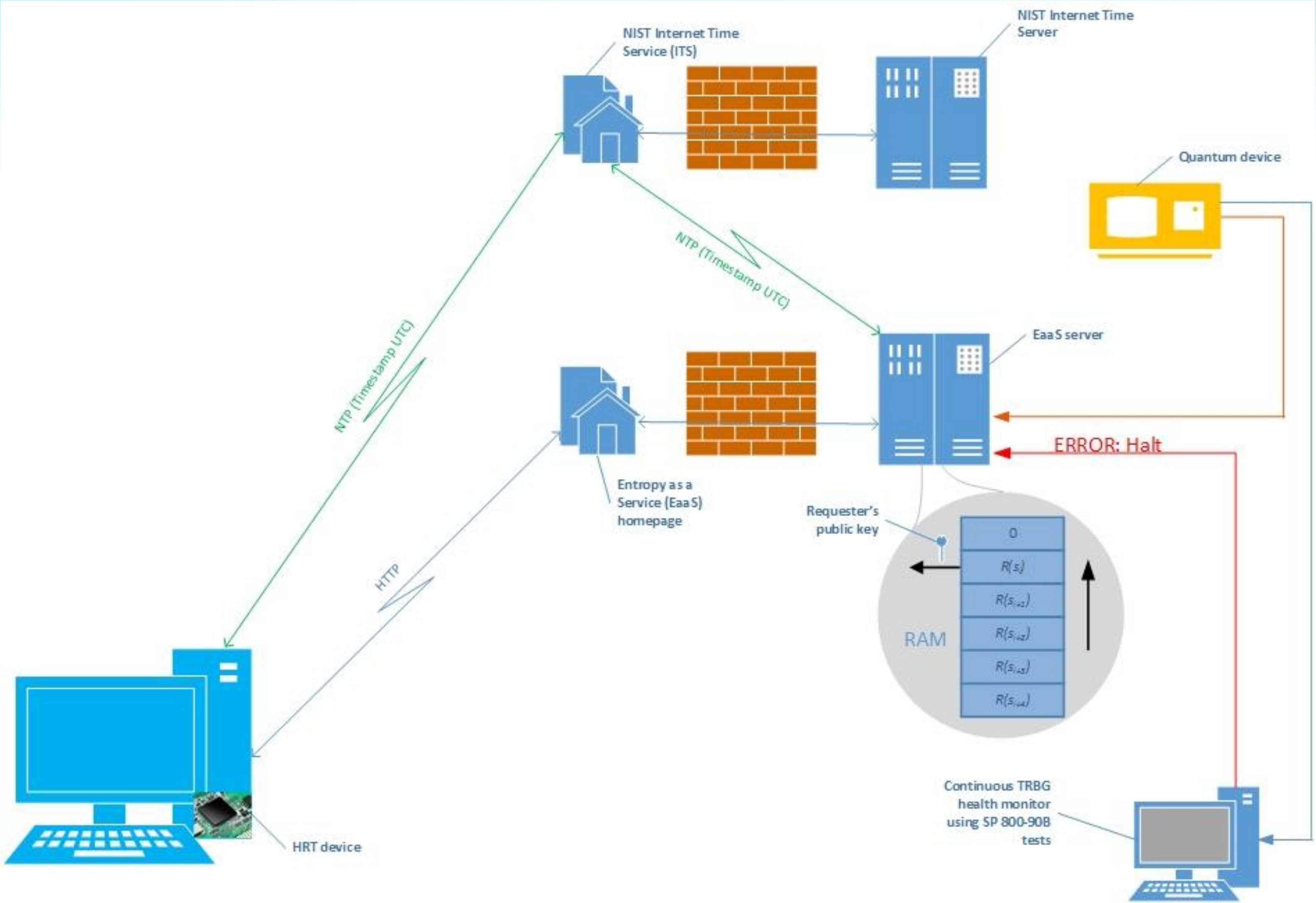
Our Solution

- Public service providing entropy for use in cryptography
  - Delivers entropy upon request from clients
    - clients seed DRBG's after mixing EaaS random data with locally available entropy
    - clients use the DRBG output to generate local keys **independently** from EaaS
  - Delivers entropy securely - no one else can see it
- High-Quality entropy from a provably good source
- Client receives assurance of key strength

What Our Solution is

- **NOT** a key generation service
  - Cryptographic keys are generated **locally** on the client using DRBG's
  - DRBG's are seeded with random data resulting from mixing **several** independent sources, including local entropy
  - Even if an attacker gains full control of one server, he/she will have **no possibility** of gaining meaningful insights into the client keys
- **NOT** similar to the NIST beacon or sharing components with it
  - The service does **NOT** record any incoming or outgoing record
  - The service does **NOT** record any internal quantum random data
  - The service does **NOT** share any components with the beacon

The Solution Architecture



Demo

- Note: We'll be available in Booth 219 in the expo floor after the demo to continue with questions or discussions

Potential Attacks and Defenses

- Replay Attack
  - Messages are timestamped and the signature includes the timestamp
- Man-in-the-Middle
  - Data is encrypted and signed, ensuring both security and authenticity
- DNS Poisoning
  - Messages are signed, ensuring authenticity

Our Demo Implementation

- **Server** – Java JBoss AS 6.1.0 run in Eclipse Luna
- **Client** – C#, written and run in Visual Studio
- **Hardware Root of Trust** – TPM "Trusted Platform Module"
- **Encryption/Signature** – RSA/SHA-256
- **Operating System** – Windows 7 Enterprise SP1
- Proof-of-Concept implementation will be opened for others to review

Future Work

- Collective Authority (Cothority)
  - Developed by Bryan Ford and others
  - Decentralizes authority in a system
  - Entropy created and signed by a network of independently-operated EaaS servers
  - Compromising one node does not compromise the resulting entropy
  - Trust is not in an individual organization, but in the collective of all EaaS hosts

Contents

- The Problem
- Our Solution
- The “Big” Picture
- Potential Attacks and their Defenses
- Our Implementation
- Demo
- Future Work