# Mining Your Ps and Qs:
# Detection of Widespread Weak Keys in Network Devices

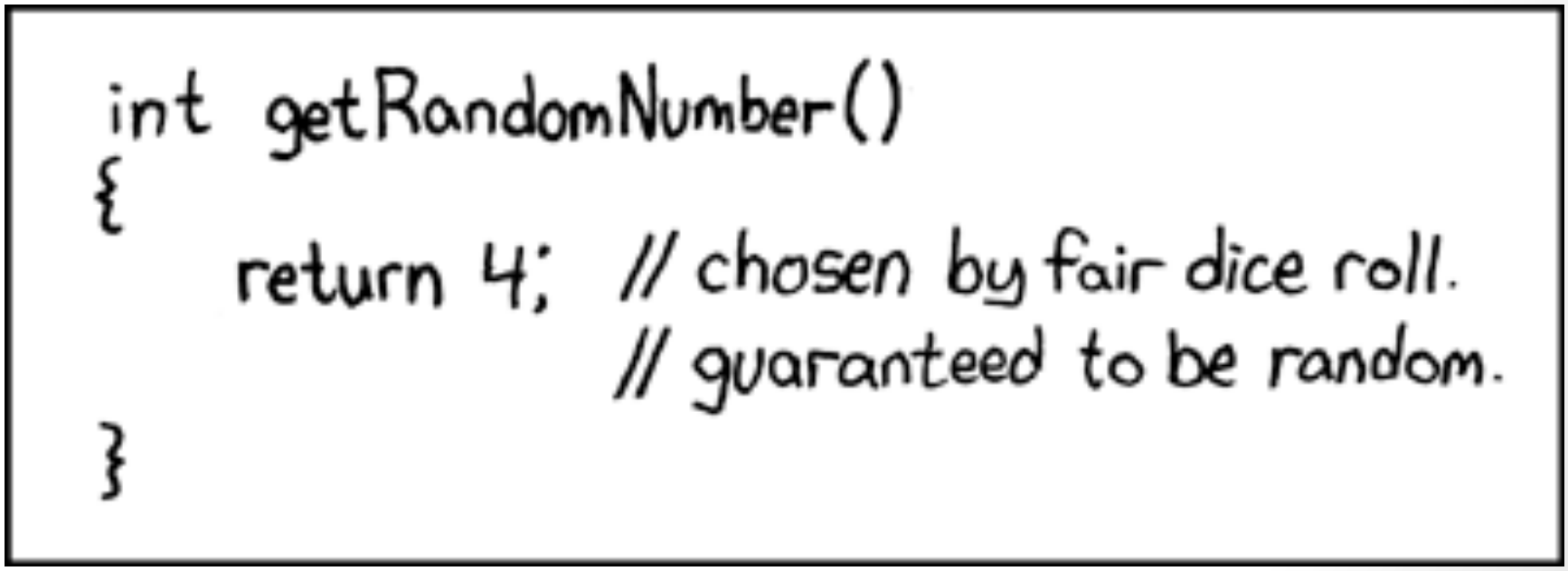
Nadia Heninger **Zakir Durumeric**, Eric Wustrow, Alex Halderman

UCSD Jacobs School of Engineering

COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY of MICHIGAN ■ COLLEGE of ENGINEERING

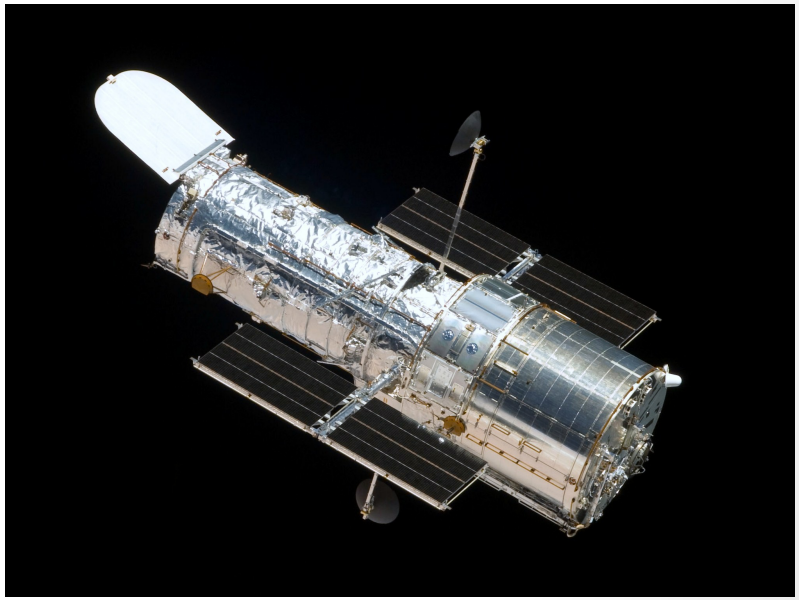
# Public Keys and Randomness

- Public keys secure Internet communications; e.g. SSL, SSH
- Security requires good randomness



# Research Agenda

1) Collect keys
2) Look for specific vulnerabilities
3) Investigate causes



# Collecting Public Keys

**Finding Hosts**

Nmap from EC2

25 hosts, <24 hours

| Port 443 (HTTPS) | Port 22 (SSH) |
|---|---|
| 29 million hosts | 23 million hosts |

**Retrieving Keys**

Event Driven Process

3 hosts, <48 hours

| Port 443 (HTTPS) | Port 22 (SSH) |
|---|---|
| 13 million hosts | 10 million hosts |

**Parsing Certs**

OpenSSL, database

| Certificates |
|---|
| 6 million certificates<br>(2 million browser-trusted) |

# What could go wrong?

1. Repeated keys

# Repeated Keys

| | Port 443 (HTTPS) | Port 22 (SSH) |
|---|---|---|
| **Live Hosts** | 12.8 million | 10.2 million |
| **Distinct RSA public keys** | 5.6 million | 3.8 million |
| **Distinct DSA public keys** | 6,241 | 2.8 million |

Why are so many keys shared?

# Investigating Shared Keys

Manually investigated hosts sharing keys

Non-vulnerable reasons for shared keys

- Corporations share keys across certificates
- Shared hosting providers

# Investigating Shared Keys

Manually investigated hosts sharing keys

Vulnerable reasons for shared keys

- Default certificates and keys
- Apparent entropy problems
- 714,000 (5.6%) of TLS hosts
- 981,000 (9.6%) of SSH hosts

# Snake-oil Keys

Apache ships default certificates with installation

Found 22 CA-signed certificates with keys copied from snake-oil certificate!

*Lesson:* Users are only going to follow your instructions approximately…

# What could go wrong?

1. Repeated keys

2. Repeated factors in RSA keys

# RSA Keys

Public key modulus **N = pq**

Factoring **N** reveals the private key

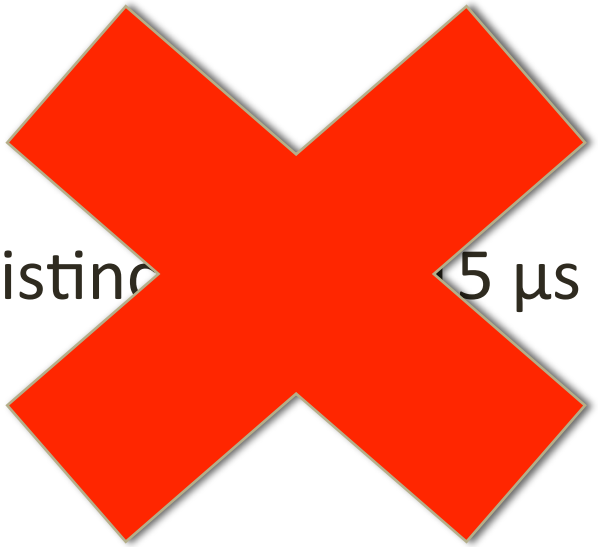
Factoring 1024-bit RSA not known to be feasible

However…

For $N_1 = pq_1$ and $N_2 = pq_2$

we can efficiently compute $p = GCD(N_1, N_2)$

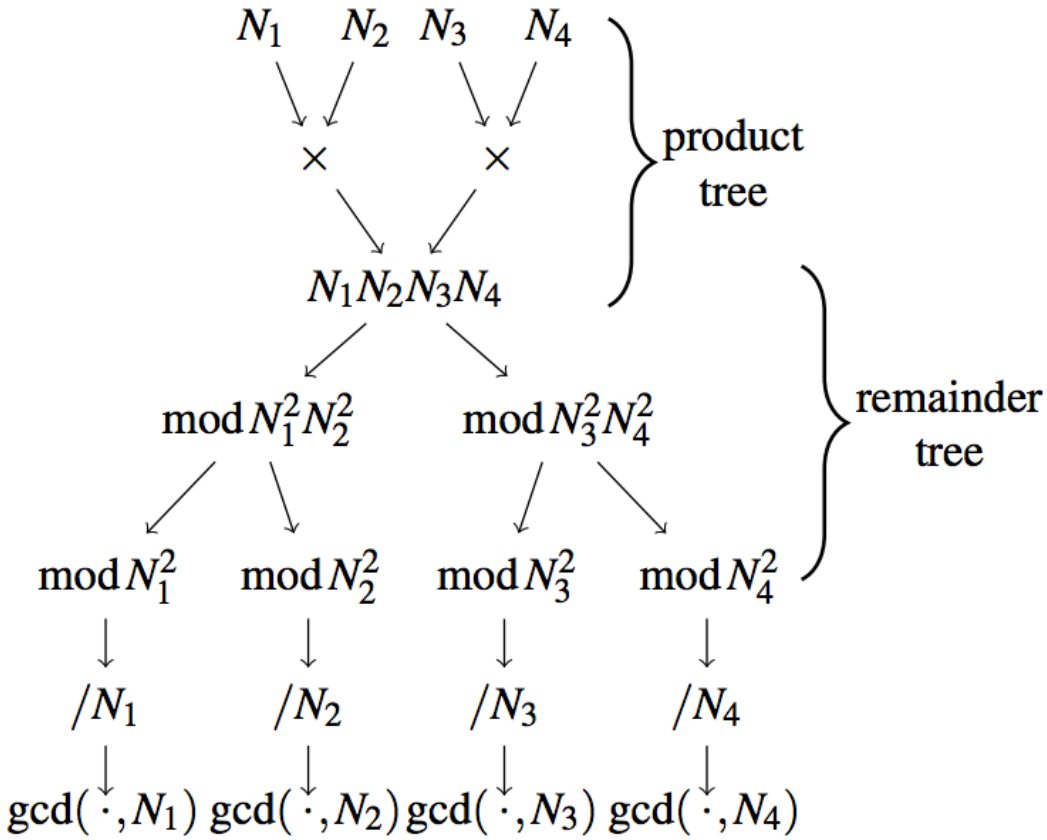
# Looking for Shared RSA Factors



$6 \times 10^{13}$ distinc … 5 µs ≈ 30 years

# Looking for Shared RSA Factors

All Pairs GCD
(*algorithm due to Bernstein*)




Our Implementation

- 1.3 hours on EC2
- $5.00

2,134 prime factors

Computed private keys for 64,081 TLS hosts (0.50%)

# What could go wrong?

1. Repeated keys

2. Repeated factors in RSA keys

3. Repeated DSA signature randomness

# DSA Signatures

Standard  Digital Signature Algorithm

Each signature contains a random ephemeral key

If predictable

⇒ can easily compute private key

Two different signatures with same ephemeral and long-term keys

⇒ can easily compute randomness

⇒ can easily compute private key

# Looking for shared randomness

We collected DSA signatures during SSH key exchange

4,365 signatures used shared ephemeral keys

Computed private long-term keys for
105,728 (1.03%) of SSH hosts

# Vulnerable Devices

Vast majority of compromised keys generated by headless or embedded network devices

- Routers, Firewalls, Switches, Server Management Cards, Cable Modems, Voice-Over-IP devices
- Automatically generate keys

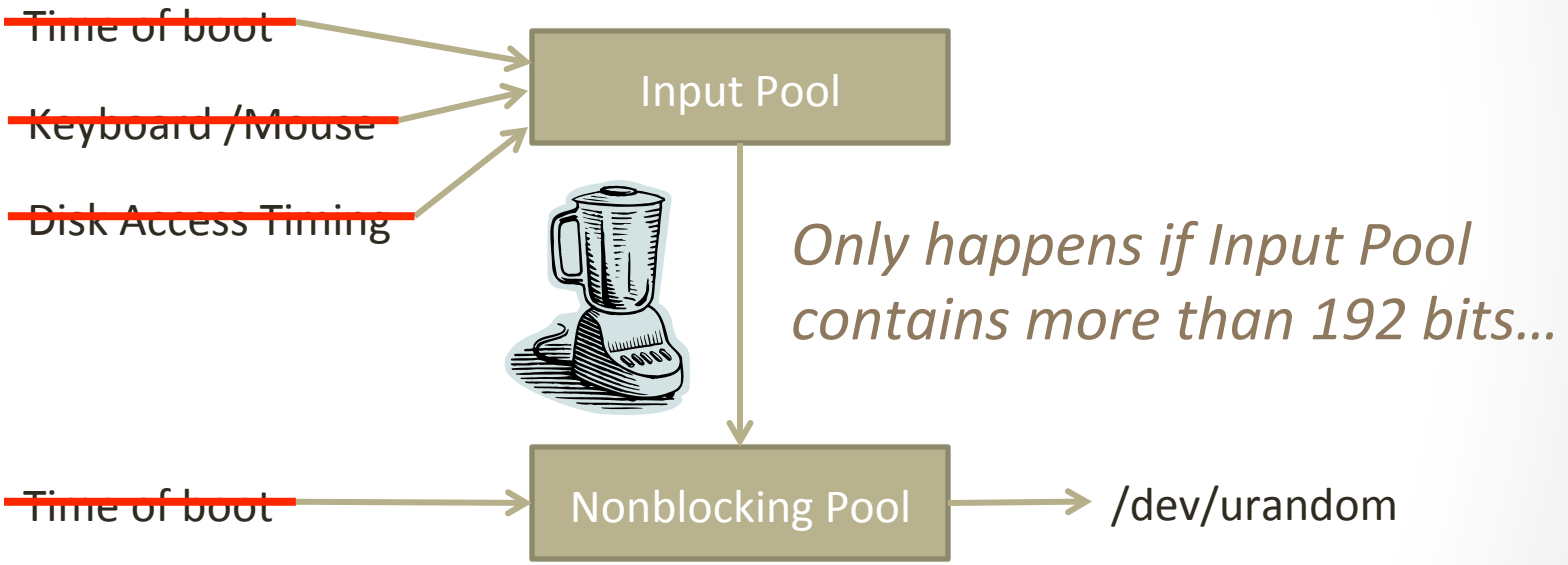
Identified devices from 41 manufacturers

# Research Agenda

1) Collect keys
2) Look for specific vulnerabilities
3) Investigate causes

# Linux /dev/urandom

Nearly everything uses /dev/urandom

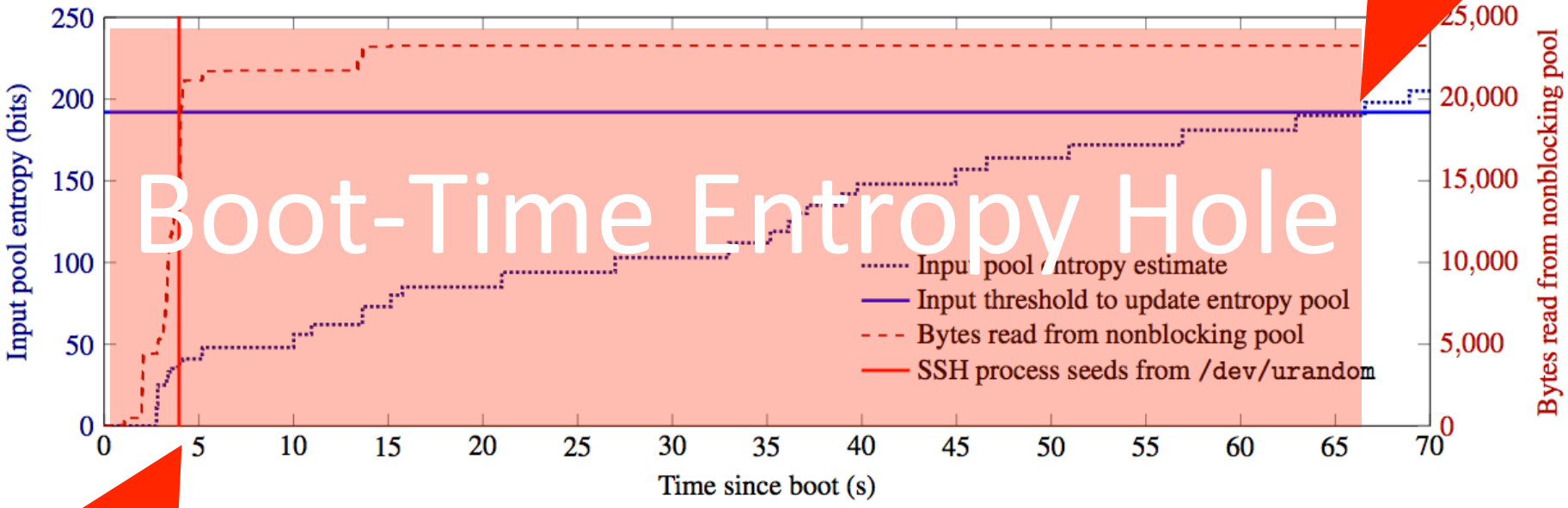



**Problem 1:** Embedded devices may lack all these sources

**Problem 2:** /dev/urandom can take a long time to "warm up"

# Ubuntu Server 10.04 Test System
## (Typical boot)

First Input entropy mixed into /dev/urandom




OpenSSH seeds from /dev/urandom

Dev /dev/urandom may be predictable for a period after boot.

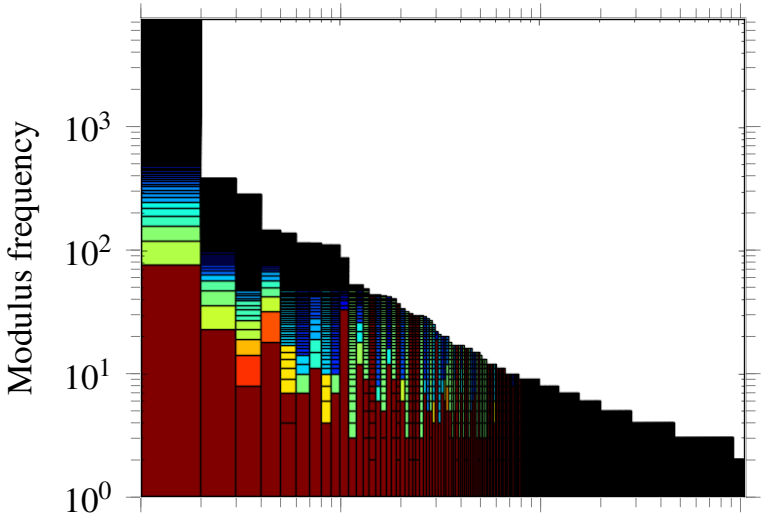
# Why are keys factorable?

```
prng.seed(seed)
p = prng.generate_random_prime()
q = prng.generate_random_prime()
N = p*q
```

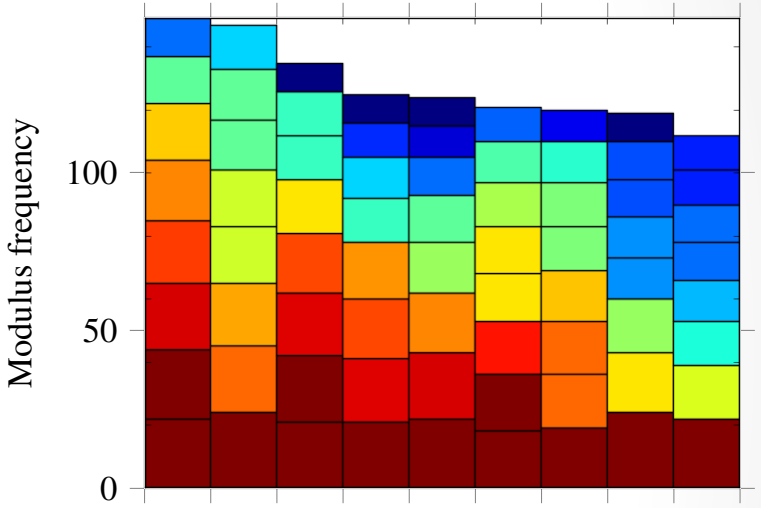
# Why are keys factorable?

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

# One unusual case…




Typical factor distribution for one device model




Factor distribution for a particular IBM Device

9 possible primes
36 total possible moduli!

# Disclosure

Wrote disclosures to about 60 companies

- About 10 had security contact information
- Approximately 20 responded
- 3 have informed us of security advisories
- US-CERT is helping us coordinate

Linux Kernel has been patched

# Disclosure to end-users

- Found a number of Citrix remote-access devices using CA-signed certs with keys copied from default certs
- Certs belonged to Fortune 500 companies, insurance providers, law firms, a major public transit authority, and the US Navy.
- I tried to contact these companies…

# Mitigations

**Lessons** for OS developers, crypto library developers, app developers, device makers, certificate authorities, end users, security and crypto researchers

**More entropy sources**

Add hardware sources

Kernel collects more aggressively

**Better communication between applications and OS**

/dev/urandom isn't providing the service people need

**Created public key check service for end users**

# Is this the tip of the iceberg?

Probably mainly see devices *without* real-time clocks

- RTC may mask serious entropy problems

Possible targeted attack

- Guess time of first boot and compute key

On traditional PCs, margin of safety for keys generated on first boot is slim

- Not observed to be exploitable (so far)

# Future Work

Further cryptographic vulnerabilities

- Diffie-Hellman, ECDSA
- IMAPS, DNSSEC

Further impacts of boot-time entropy hole

- TCP sequence numbers
- ASLR

Further applications of top-down methodology

# Conclusion

Studied entropy via global perspective on public keys

Found widespread vulnerabilities in embedded devices

- Shared keys (5.6% of TLS hosts; 9.6% of SSH)
- Factorable RSA keys (0.5% of TLS hosts; 0.03% of SSH)
- Repeated DSA randomness (1.0% of SSH hosts)

Secure random number generation still difficult

# **Mining Your Ps and Qs:** Detection of Widespread Weak Keys in Network Devices

Our website: https://factorable.net

Nadia Heninger **Zakir Durumeric**, Eric Wustrow, Alex Halderman

UCSD Jacobs School of Engineering

COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY of MICHIGAN ■ COLLEGE of ENGINEERING