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

N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman,
USENIX Sec'12

20203590 Hyunsik Jeong

Intro

- Detecting weak keys/signatures in the wild
- Collected public keys/certificates

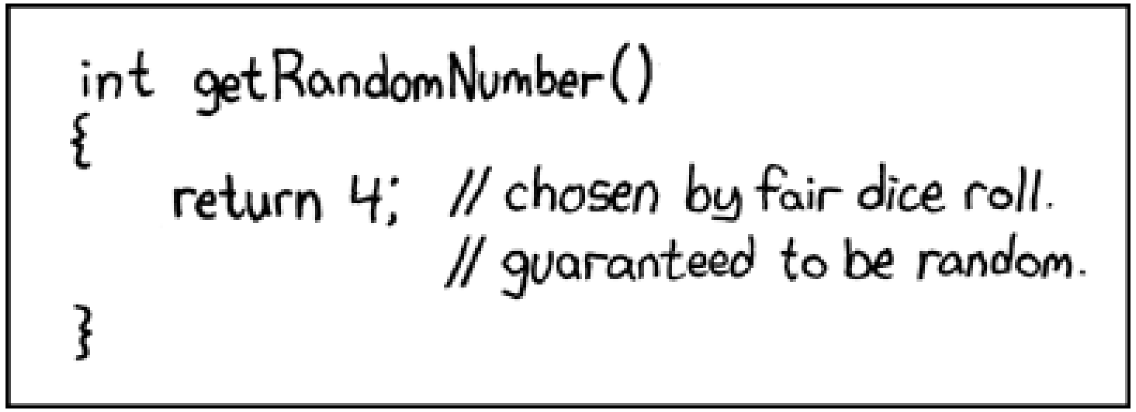
- Tried to figure out how weak keys/signatures were generated

Public keys and Randomness

- Public key cryptography used everywhere!
    - TLS (used in HTTPS), SSH, …
- Based on *randomly* generated secret keys

Public keys and Randomness

- Public key cryptography used everywhere!
  - TLS (used in HTTPS), SSH, …
- Based on *randomly* generated secret keys
- What if they are not random?



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

from: xkcd (https://www.explainxkcd.com/wiki/index.php/221:_Random_Number)

Collecting public keys

Finding Hosts
Nmap from EC2
25 hosts, ~25 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

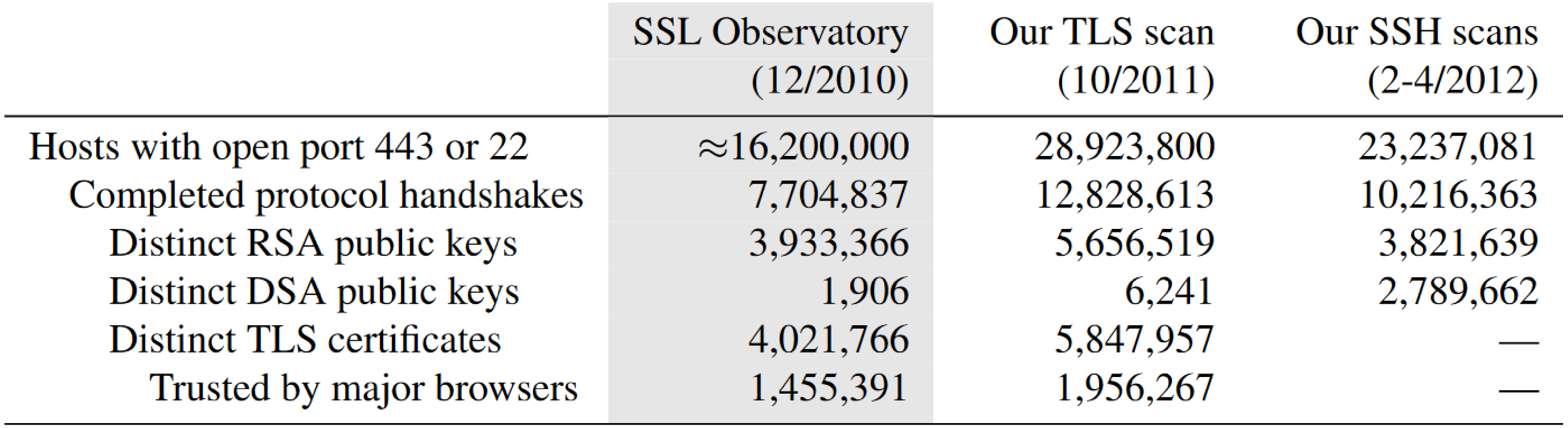
Parsing Certs
OpenSSL, database

Certificates
6 million certificates (2 million browser-trusted)

What could go wrong?

1. Repeated keys

Repeated keys



	SSL Observatory (12/2010)	Our TLS scan (10/2011)	Our SSH scans (2-4/2012)
Hosts with open port 443 or 22	≈16,200,000	28,923,800	23,237,081
Completed protocol handshakes	7,704,837	12,828,613	10,216,363
Distinct RSA public keys	3,933,366	5,656,519	3,821,639
Distinct DSA public keys	1,906	6,241	2,789,662
Distinct TLS certificates	4,021,766	5,847,957	—
Trusted by major browsers	1,455,391	1,956,267	—

- TLS: 7,770,232 hosts (61%)
- SSH: 6,642,222 hosts (65%)

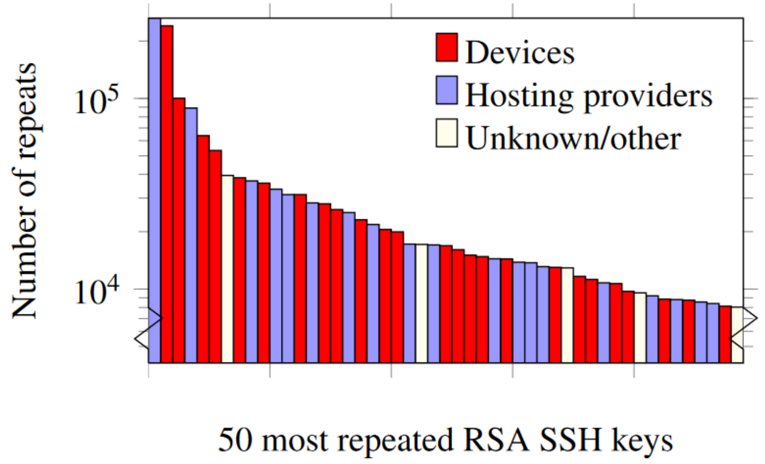
Shared keys

Non-vulnerable reasons for shared keys

- Corporations shared keys across certificates
- Shared hosting providers

Vulnerable reasons for shared keys

- Default certificates and keys
- Low entropy problems



What could go wrong?

1. Repeated keys

2. Repeated factors in RSA keys

RSA revisited

- Generate two random prime numbers p, q

- Public key: (e, N), $N = pq$ (Usually $e = 65537$)
- Private key: $d = e^{-1} \pmod{\phi}, \phi = (p-1)(q-1)$

- Why is it difficult to break?

RSA revisited

- Generate two random prime numbers p, q

- Public key: (e, N), $N = pq$ (Usually $e = 65537$)
- Private key: $d = e^{-1} \pmod{\phi}, \phi = (p-1)(q-1)$

- Why is it difficult to break?
    1. Hard to factorize N, so difficult to get ϕ and calculate d
    2. For given encrypted message $m^e \pmod{N}$, it's hard to recover m (DLP)

Repeated factors

- What if $N_1 = pq, N_2 = pr$?
  The greatest common divisor (GCD) is p.
- Euclidean method! (from 300 BC)
  - Takes 15μs for two 1024-bit numbers
- For multiple Ns, Bernstein's algorithm can be used.

https://www.worldhistory.org/image/4139/euclid-of-alexandria/

Result?

- 11,170,883 RSA keys
- 1.3 hours on EC2 Cluster Compute Eight Extra Large Instance
    - only $5!

- Got 2,134 prime factors
- Computed private keys for 64,081 TLS hosts (0.50%)

https://i.insider.com/5c7967b3eb3ce8763f505bf5?width=700&format=jpeg&auto=webp

What could go wrong?

1. Repeated keys

2. Repeated factors in RSA keys

3. Repeated DSA signature randomness

DSA revisit

- Pick two random prime numbers: p, q
- Private key: x / Public key: $y = g^x \bmod p$

- Signature (r, s):

$$\text{For random nonce } k\text{:}$$
$$r = \left(g^k \bmod p\right) \bmod q$$
$$s = k^{-1}(H(m) + xr) \bmod q$$

Ephemeral key is shared

$$r = \left(g^{k} \bmod p\right) \bmod q$$

$$\begin{tikzcd}
s = k^{-1}(H(m) + xr) \bmod q \\
k = s^{-1}(H(m) + xr) \bmod q
\end{tikzcd}$$

$$\begin{tikzcd}
s \arrow[dr] & k^{-1} \arrow[dl] \\
k & s^{-1}
\end{tikzcd}$$

Ephemeral key is shared

$$r = \left(g^k \bmod p\right) \bmod q$$
$$s = k^{-1}(H(m) + xr) \bmod q$$

$$\begin{tikzcd} s \arrow[dr] & k^{-1} \arrow[dl] \\ k & s^{-1} \end{tikzcd}$$

$$k = s^{-1}(H(m) + xr) \bmod q$$

$$s_1^{-1}(H(m_1) + xr) = k = s_2^{-1}(H(m_2) + xr) \pmod q$$

Result?

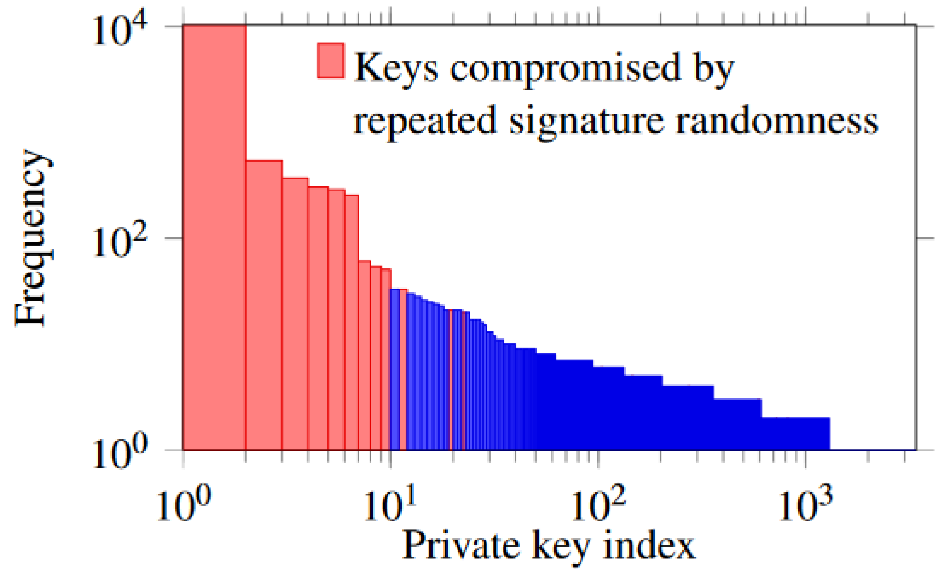
- 9,114,925 DSA signatures from SSH

- 4,094 signatures with same public key and r
- Recovered 281 distinct private keys
- These keys are used in 105,728 hosts (1.6%)

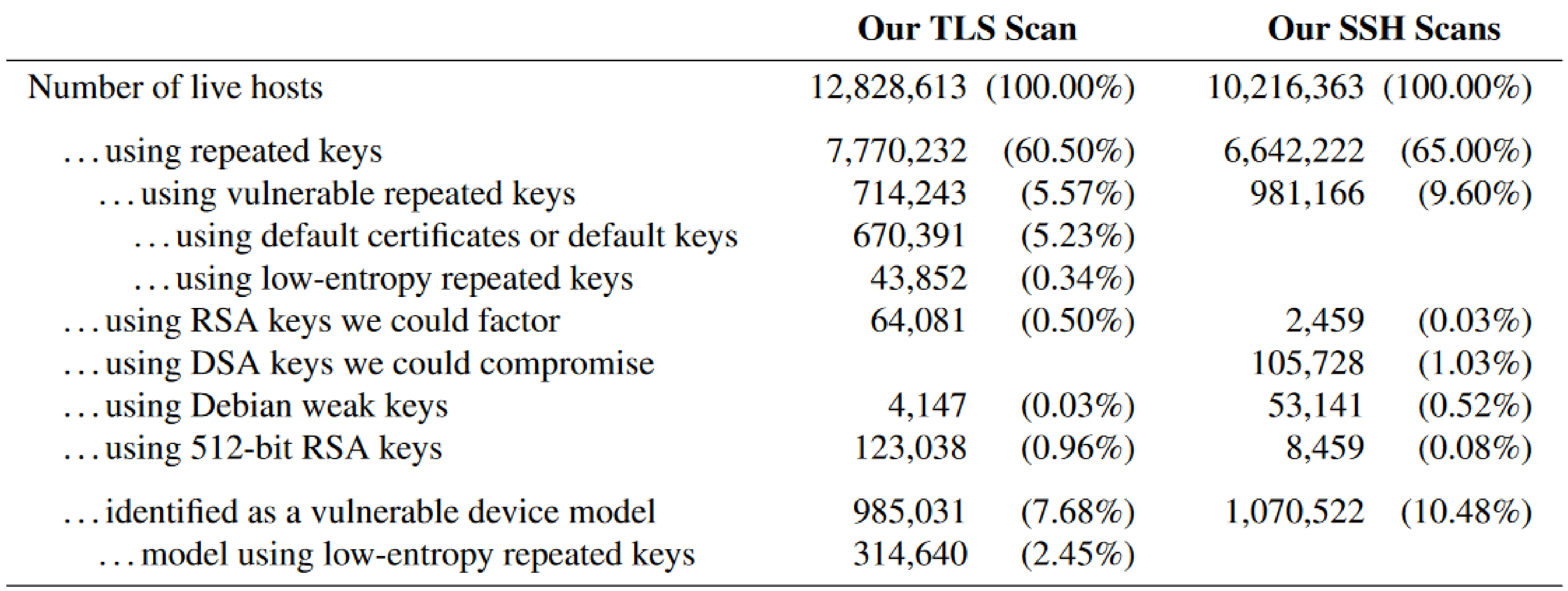
Result?

- Clustered vulnerable signatures by r values manufacturers

- 75.8% of the cases were from two manufacturers



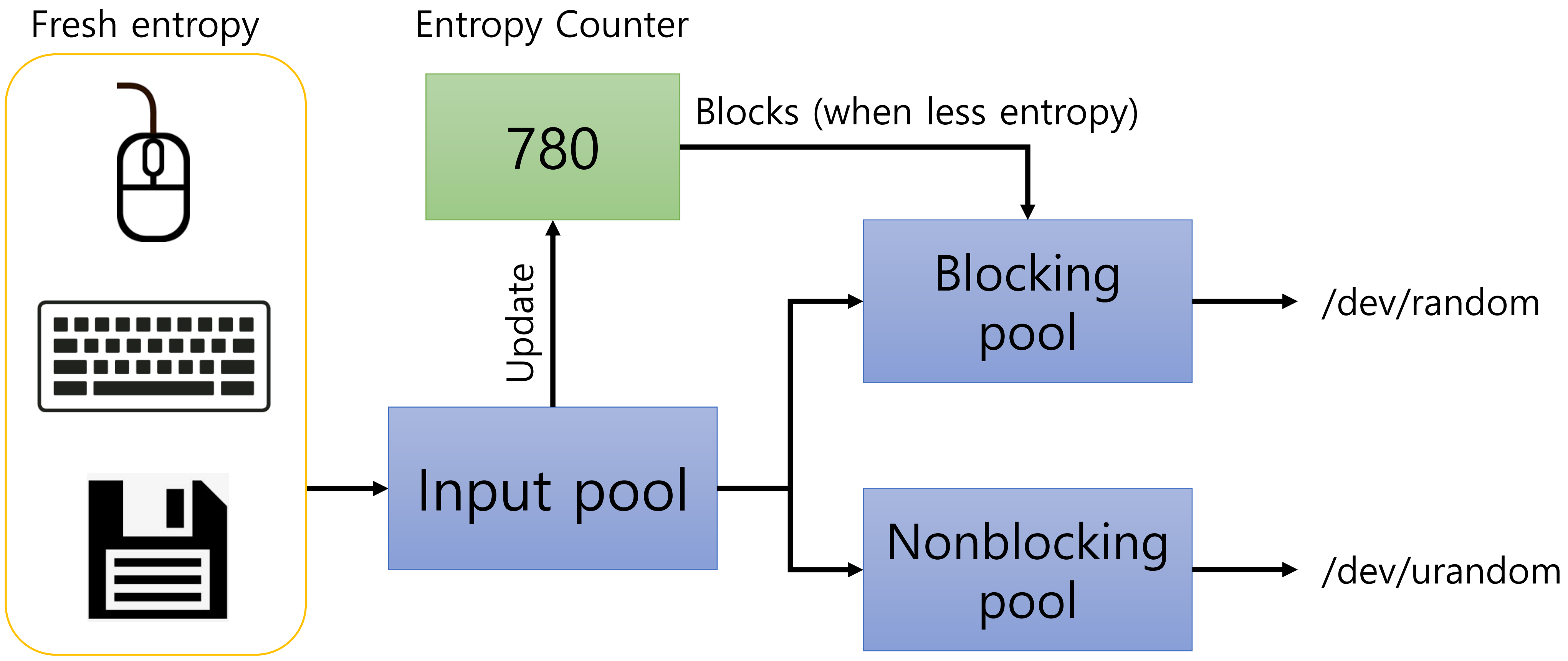
Final result



	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
. . . using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
. . . using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
. . . using default certificates or default keys	670,391	(5.23%)		
. . . using low-entropy repeated keys	43,852	(0.34%)		
. . . using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
. . . using DSA keys we could compromise			105,728	(1.03%)
. . . using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
. . . using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
. . . identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
. . . model using low-entropy repeated keys	314,640	(2.45%)		

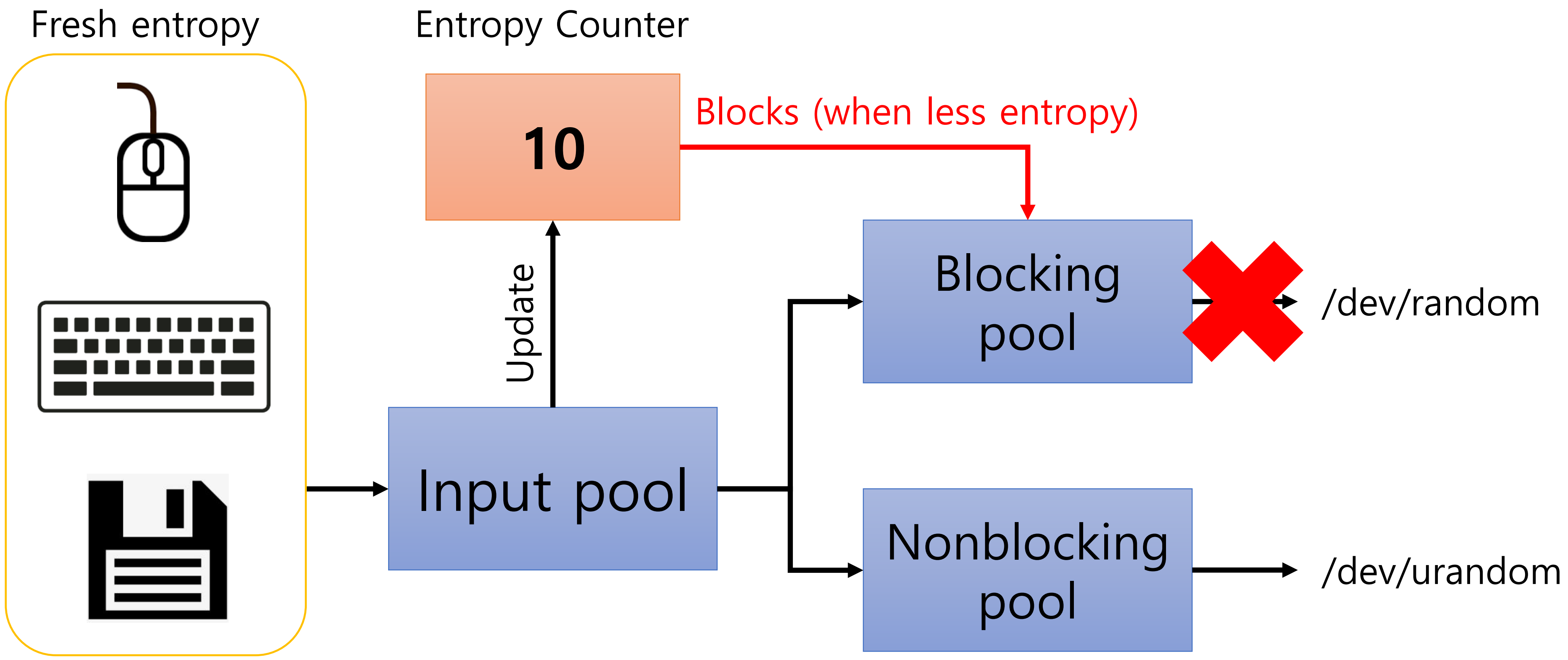
Weak entropy and the Linux RNG

- Nearly everything uses /dev/urandom

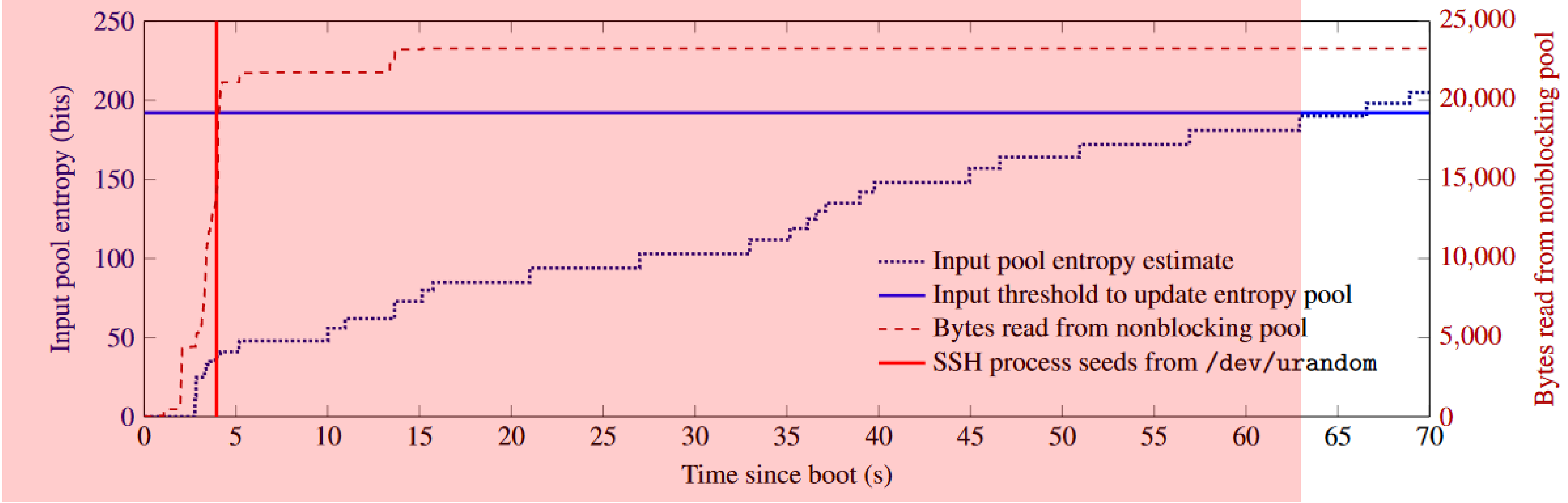


Weak entropy and the Linux RNG

- Nearly everything uses /dev/urandom

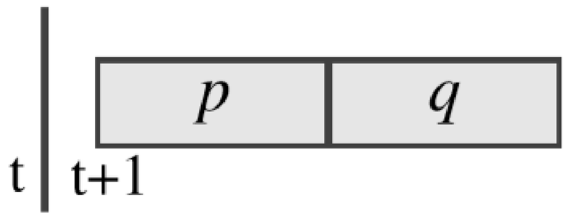


Weak entropy and the Linux RNG

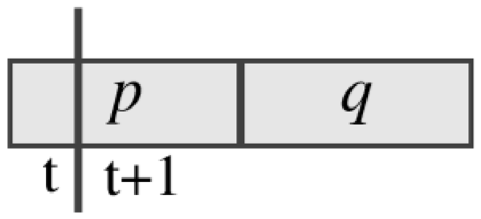


Ubuntu 10.04 test system (typical boot)

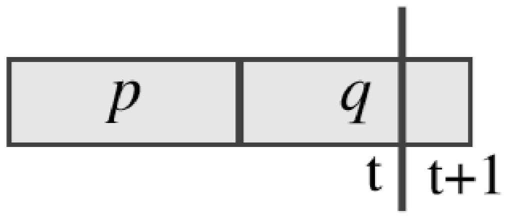
Factorable RSA keys



Both factor repeated
Not bad



Both factor different
Very good



Only one different
Problematic!

Defense

- **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
- **Create public key check service for end users**

Conclusion

- Studied entropy via global perspective on public keys
- Found widespread vulnerabilities
  - 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 is still difficult

Related works

Problems with random number generation

- “Randomness and the Netscape browser”, Dr. Dobb’s Journal 21 (1996)
- DSA-1571-1 OpenSSL—Predictable random number generator (2008)
- “Analysis of the Linux random number generator”, SP ’06

Weak entropy and cryptography

- Console hacking 2010: PS3 epic fail, Talk at 27th Chaos Communication Congress (2010)
- “When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography”, NDSS ‘10

Follow-up works

Other cryptographic vulnerabilities

- Unsecure ECDSA key

  “Elliptic curve cryptography in practice.”, FC ‘14
- Diffie-Hellman algorithm

  “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”, CCS ‘15

Malfunction of RNG

- “Security Analysis of Pseudo-Random Number Generators with Input: /dev/random is not Robust”, CCS ‘13
- “Not-so-random numbers in virtualized Linux and the Whirlwind RNG”, SP ‘14

Good Questions

From Youngjin Jin (Best question!)

Q: Does the randomly generated number have to come from the operating system like /dev/urandom?

In other words, would it be possible to use something like a hardware random number generator instead of OS-based pseudorandom number generator?

A:

- Why not?
- But not every device

Good Questions

From Tuan Hoang Dinh

Q: In this paper, the author used the vocabulary “conjecture” many times. But there is no clear evidence from the result that heads them to the possible reason for vulnerabilities. Did they know the vulnerabilities in advance?

A:

- Those are basic vulnerabilities!

Good Questions

From Yeongbin Hwang

Q: There is a description of the randomness of urandom used in Linux. How is randomness managed in other operating systems?

A:

- Windows – the poor OS
- CNG

Bad Questions

Q: Are there any vulnerabilities in Diffie-Hellman or ECDSA?

A: There’s no vulnerabilities in Diffie-Hellman and ECDSA.

Q: Even though the repeated keys are a problem, it is hard to memorize all the different keys. So most of all use repeated passwords and key. So making the password or key pattern policy that the user can memorize well and secure is important.
A: There’s no reason to memorize keys, as the key is big.

Bad Questions

Q: Is there a study comparing the randomness between PRNG libraries such as arc4random?

A: This paper is not about PRNG, and there are various PRNGs which is not able to compare.

Q: Is any new operation added to make randomness in OS?

A: /dev/urandom is not a problem, what do you mean?