Mining Your *P*s and *Q*s
Detection of Widespread Weak Keys in Embedded Devices

Nadia Heninger

UC San Diego

↓

Microsoft Research New England

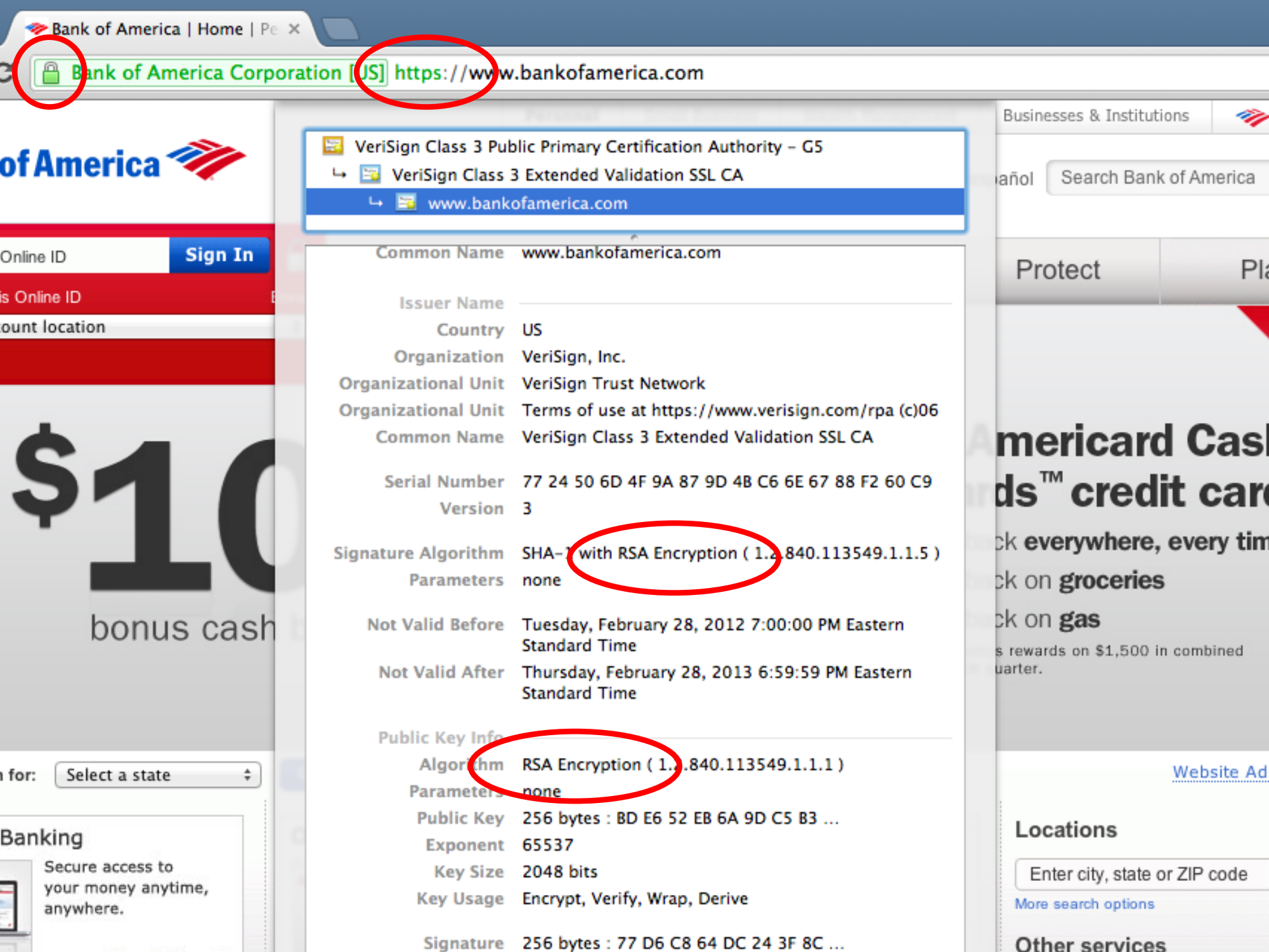
Zakir Durumeric Eric Wustrow Alex Halderman

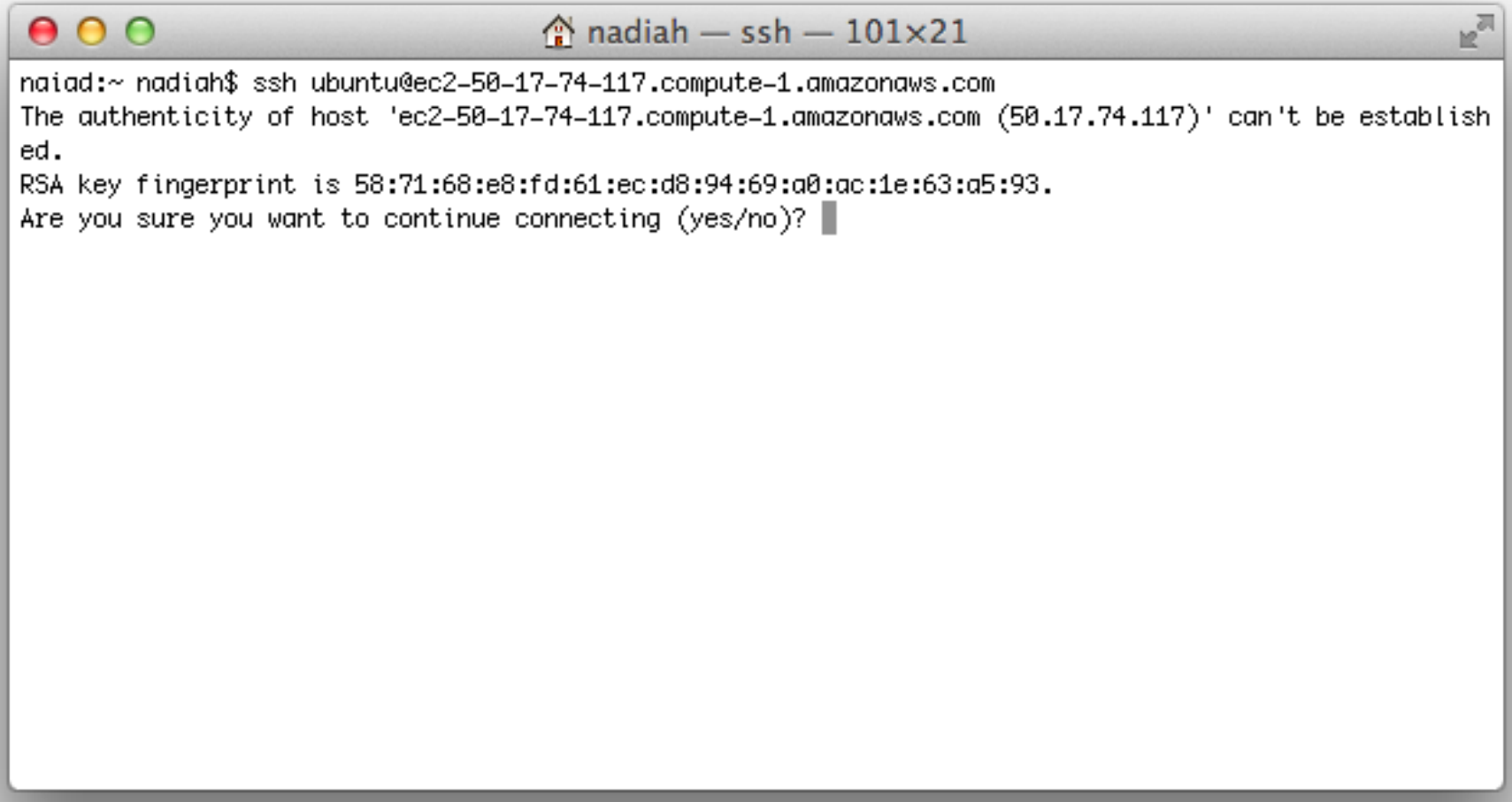
University of Michigan

February 26, 2012

Flashback: RSA 2012 Keynote Cryptographers' Panel

\So Whit, do you feel vindicated?"


Bank of America | Home
Bank of America Corporation [US] https://www.bankofamerica.com
VeriSign Class 3 Public Primary Certification Authority - G5
VeriSign Class 3 Extended Validation SSL CA
www.bankofamerica.com
Common Name www.bankofamerica.com
Issuer Name
Country US
Organization VeriSign, Inc.
Organizational Unit VeriSign Trust Network
Organizational Unit Terms of use at https://www.verisign.com/rpa (c)06
Common Name VeriSign Class 3 Extended Validation SSL CA
Serial Number 77 24 50 6D 4F 9A 87 9D 4B C6 6E 67 88 F2 60 C9
Version 3
Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters none
Not Valid Before Tuesday, February 28, 2012 7:00:00 PM Eastern Standard Time
Not Valid After Thursday, February 28, 2013 6:59:59 PM Eastern Standard Time
Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters none
Public Key 256 bytes : BD E6 52 EB 6A 9D C5 B3 ...
Exponent 65537
Key Size 2048 bits
Key Usage Encrypt, Verify, Wrap, Derive
Signature 256 bytes : 77 D6 C8 64 DC 24 3F 8C ...
Businesses & Institutions
Search Bank of America
Sign In
Protect
bonus cash
Banking
Secure access to your money anytime, anywhere.
Locations
Enter city, state or ZIP code
More search options
Other services


nadiah — ssh — 101×21
naiad:~ nadiah$ ssh ubuntu@ec2-50-17-74-117.compute-1.amazonaws.com
The authenticity of host 'ec2-50-17-74-117.compute-1.amazonaws.com (50.17.74.117)' can't be establish
ed.
RSA key fingerprint is 58:71:68:e8:fd:61:ec:d8:94:69:a0:ac:1e:63:a5:93.
Are you sure you want to continue connecting (yes/no)?

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

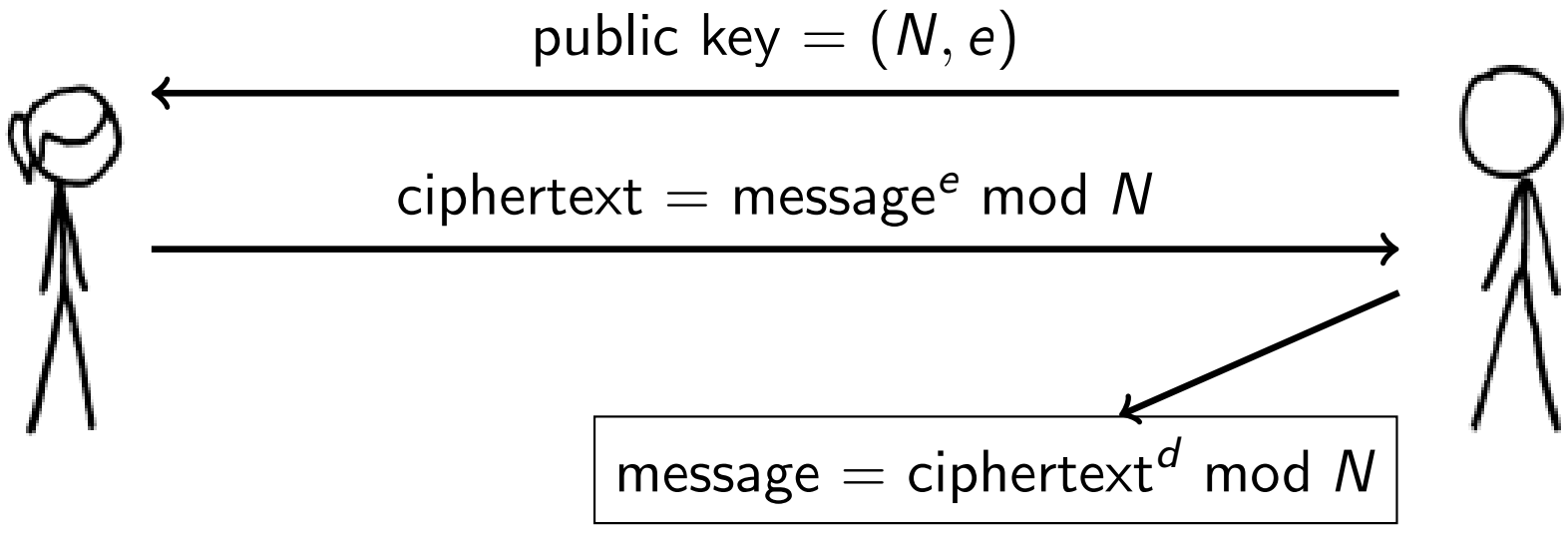
$N = pq$ modulus

e encryption exponent

Private Key

p, q primes

d decryption exponent
(compute from e,p,q)



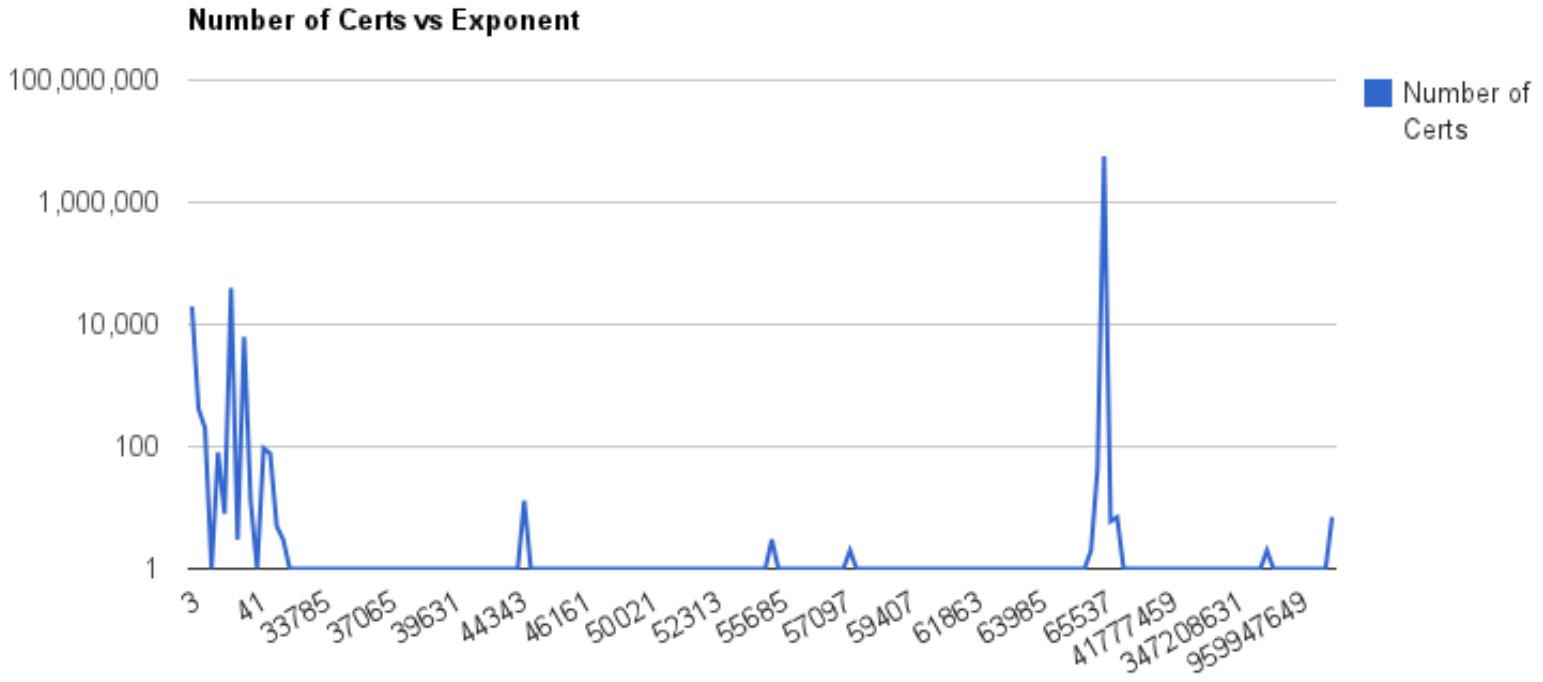
What could go wrong: Repeated keys

RSA Public Keys

$N = pq$ modulus

e encryption exponent

- Two hosts share e: not a problem.



What could go wrong: Repeated keys

RSA Public Keys

$N = pq$ modulus

e encryption exponent

- Two hosts share e: not a problem.
- Two hosts share N: $\rightarrow$ both know private key of the other.

  Security Implications:

  - Either host could man-in-the-middle the other.
  - Either host could decrypt traffic from TLS RSA key exchange.

Looking for problems: Repeated keys

$>$ 60% of hosts served non-unique public keys.

Many valid (and common) reasons to share keys:

- Shared hosting situations. Virtual hosting.
- A single organization registers many domain names with the same key.

Looking for problems: Repeated keys

$> 60\%$ of hosts served non-unique public keys.

Common (and unwise) reasons to share keys:

- Device default certificates/keys.
- Apparent entropy problems in key generation.

Looking for problems: Repeated keys

> 60% of hosts served non-unique public keys.

Common (and unwise) reasons to share keys:

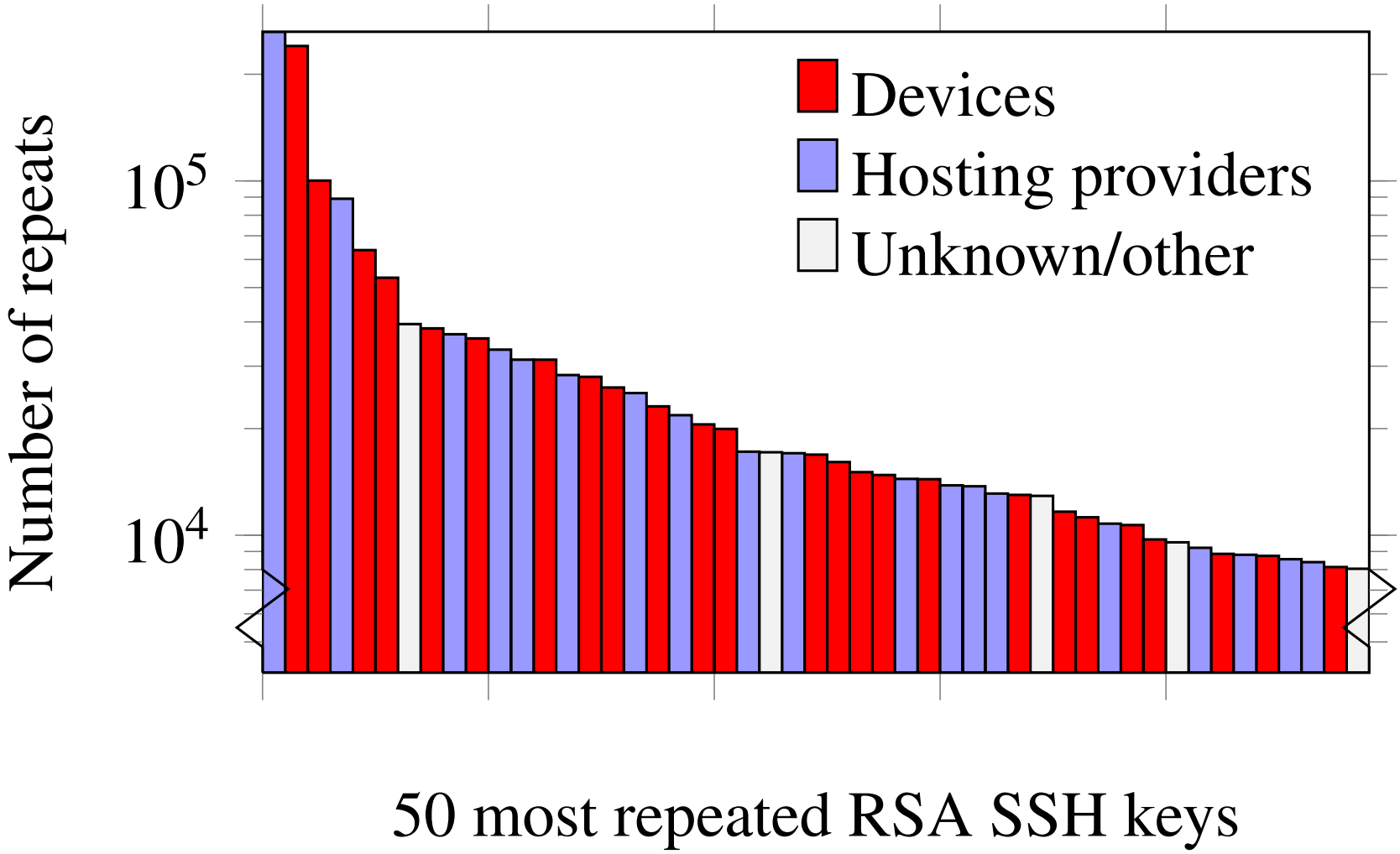
- Device default certificates/keys.
- Apparent entropy problems in key generation.

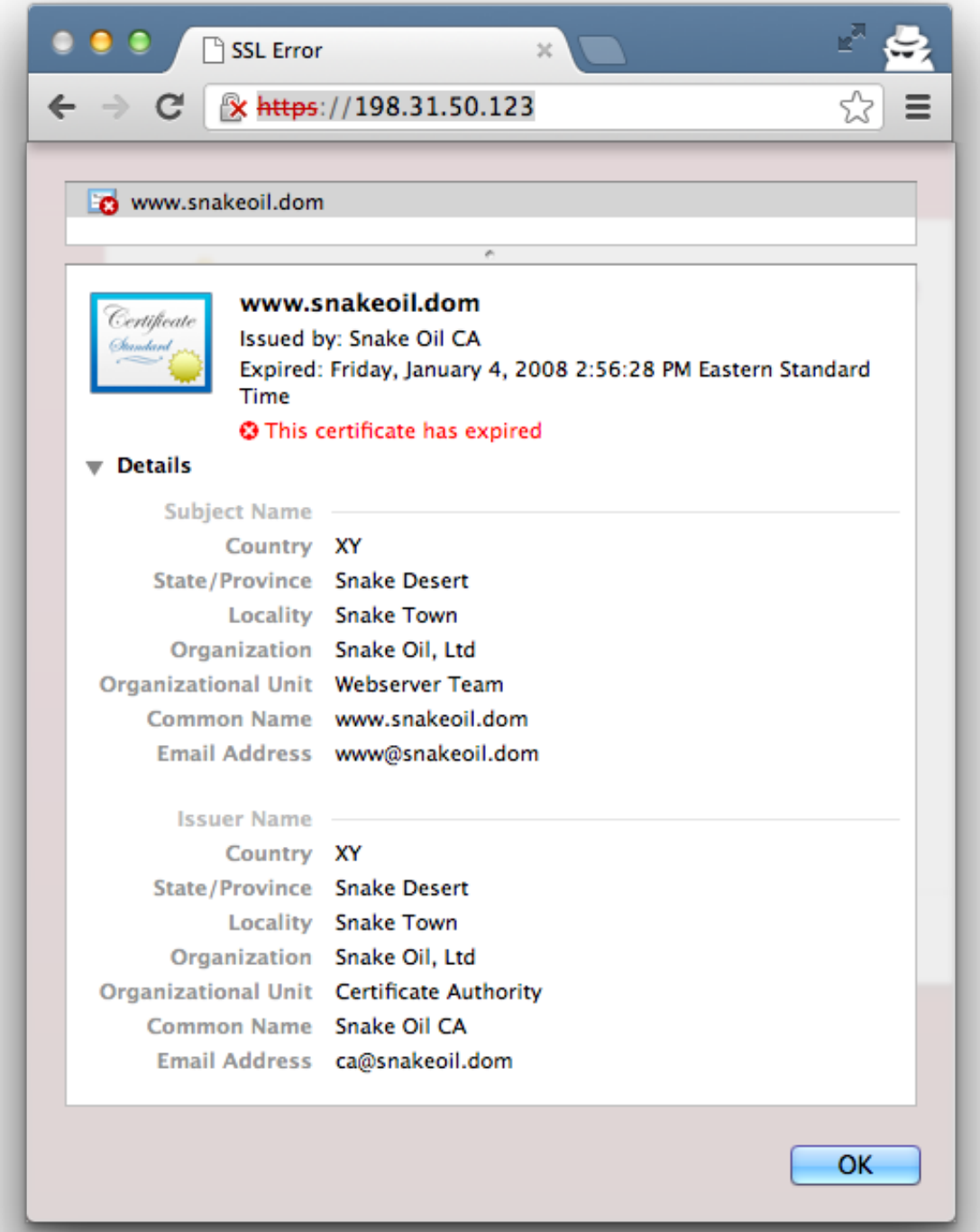
TLS:
default certificates/keys:
670,000 hosts (5%)

low-entropy repeated keys:
40,000 hosts (0.3%)

SSH:
default or low-entropy keys:
1,000,000 hosts (10%)

Looking for problems: Repeated keys




SSL Error
https://198.31.50.123
www.snakeoil.dom
www.snakeoil.dom
Issued by: Snake Oil CA
Expired: Friday, January 4, 2008 2:56:28 PM Eastern Standard Time
This certificate has expired
Details
Subject Name
Country XY
State/Province Snake Desert
Locality Snake Town
Organization Snake Oil, Ltd
Organizational Unit Webserver Team
Common Name www.snakeoil.dom
Email Address www@snakeoil.dom
Issuer Name
Country XY
State/Province Snake Desert
Locality Snake Town
Organization Snake Oil, Ltd
Organizational Unit Certificate Authority
Common Name Snake Oil CA
Email Address ca@snakeoil.dom
OK

What could go wrong: RSA common factors

If two RSA moduli share a prime factor in common
$\rightarrow$ outside observer can factor both keys with GCD algorithm.

$$N_1 = pq_1 \qquad N_2 = pq_2$$

$$\gcd(N_1, N_2) = p$$

Time to factor
768-bit RSA modulus:
2.5 calendar years
[Kleinjung et al. 2010]

Time to calculate GCD
for 1024-bit RSA moduli:
15μs

What could go wrong: RSA common factors

If two RSA moduli share a prime factor in common
$\rightarrow$ outside observer can factor both keys with GCD algorithm.

$$N_1 = pq_1 \qquad N_2 = pq_2$$

$$\gcd(N_1, N_2) = p$$

Security Implications:

- Anyone could man-in-the-middle vulnerable hosts.
- Anyone can decrypt traffic from TLS RSA key exchange.

What should happen if we GCD some RSA keys?

Experiment: Compute GCD of each pair M moduli randomly chosen from P randomly primes.

What *should* happen? **Nothing.**

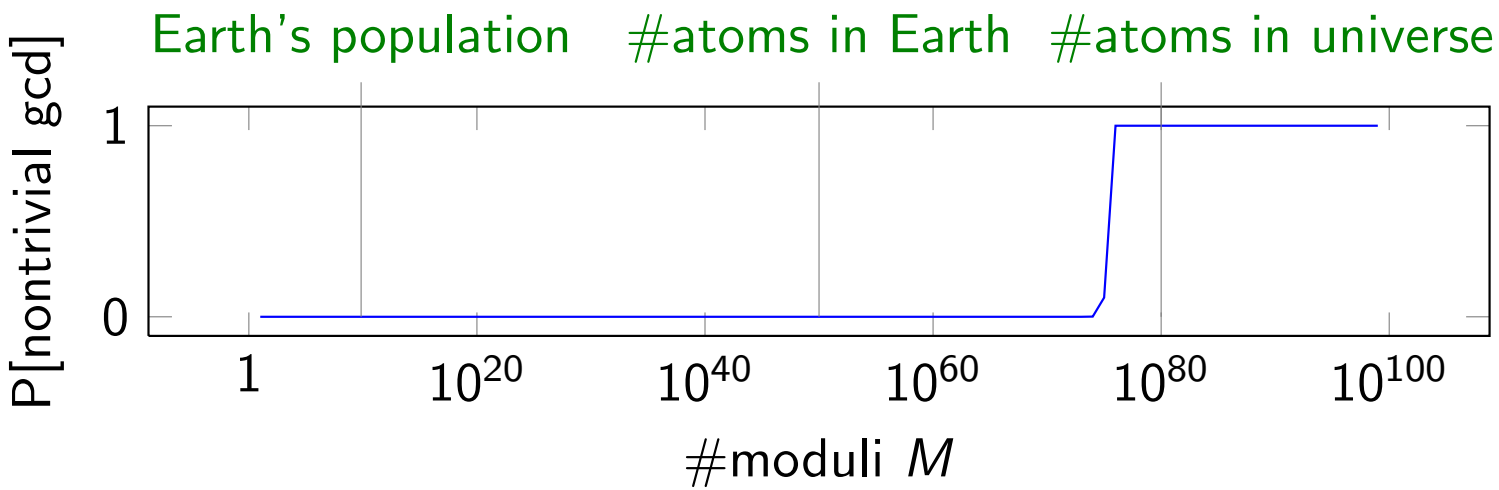
What should happen if we GCD some RSA keys?

Experiment: Compute GCD of each pair M moduli randomly chosen from P randomly primes.

What *should* happen? **Nothing.**

Prime Number Theorem:
$\sim 10^{150}$ 512-bit primes

Birthday bound:
$\Pr[\text{nontrivial gcd}] \approx 1 - e^{-2M^2/P}$



Efficiently computing pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ for our dataset would take

$$15\mu\text{s} \times \binom{11 \times 10^6}{2} \text{pairs} \approx 30 \text{ years}$$

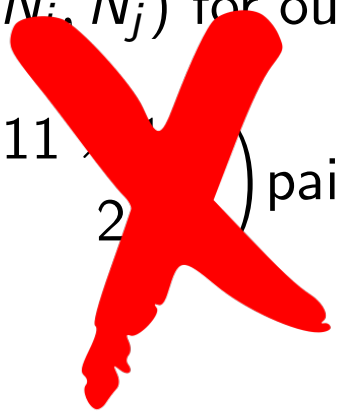
of computation time.

Efficiently computing pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ for our dataset would take

$$15\mu\text{s} \times \binom{11}{2} \text{pairs} \approx 30 \text{ years}$$

of computation time.

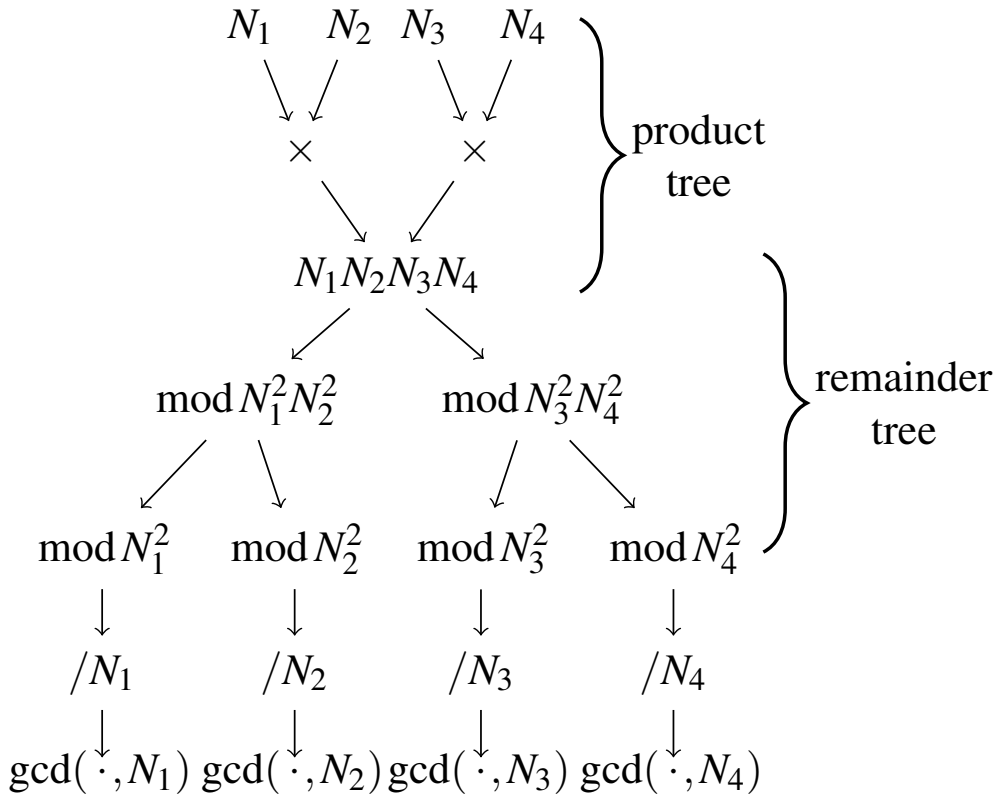


Efficiently computing pairwise GCDs

Adapted efficient algorithm due to [Bernstein 2004].

Ran on 11,170,883 distinct moduli in SSL, SSH, and EFF Observatory datasets

- 1.5 hours on 16 cores.
- $5 of Amazon ec2 time.

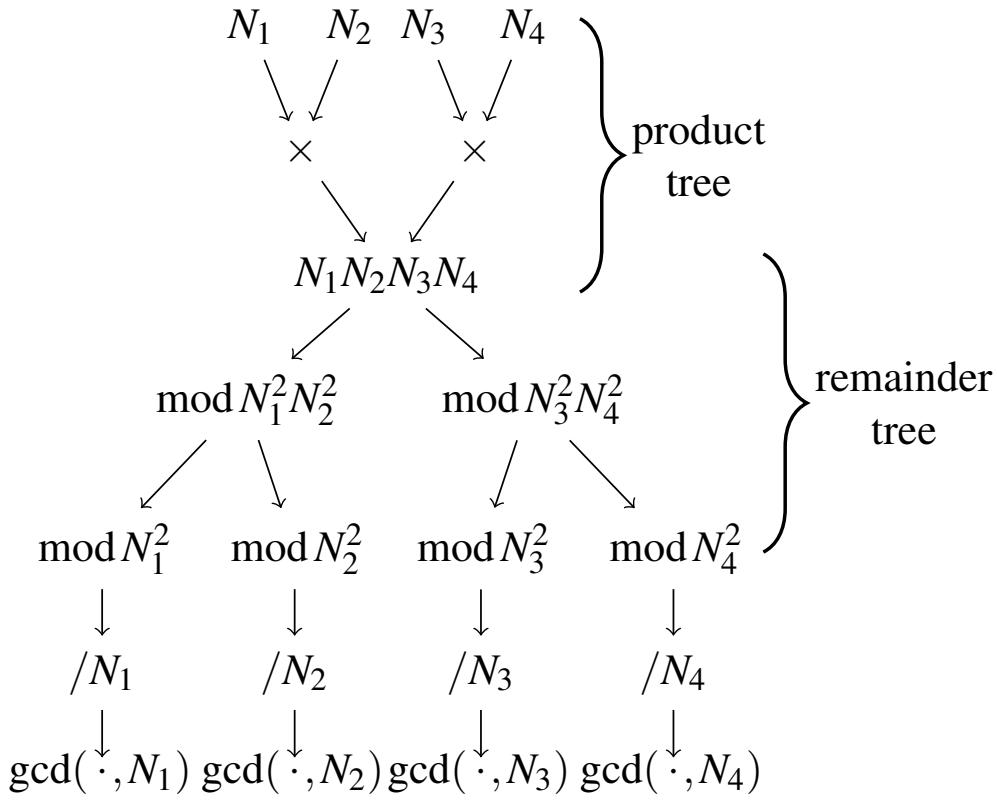


Efficiently computing pairwise GCDs

Adapted efficient algorithm due to [Bernstein 2004].

Ran on 11,170,883 distinct moduli in SSL, SSH, and EFF Observatory datasets

- 1.5 hours on 16 cores.
- $5 of Amazon ec2 time.



Results:

- 2,314 distinct prime factors factored 16,717 moduli
- Private keys for 64,081 TLS (0.50%) and 2,459 SSH (0.03%) hosts in our scan

FIPS PUB 186-3

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

Digital Signature Standard (DSS)

CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**

What could go wrong: Weak DSA signature nonce

DSA Public Key

p, q, g domain parameters

$y = g^x \bmod p$

Private Key

x private key

DSA signatures require a random nonce.

- If the nonce is predictable
  $\rightarrow$ can easily compute long-term private key.
- Two distinct signatures use same nonce and private key
  $\rightarrow$ can easily compute nonce
  $\rightarrow$ can easily compute long-term private key.

Security Implications:

- Anyone could man-in-the-middle vulnerable hosts.

Computing weak DSA keys

- Collected 9,114,925 DSA signatures from two SSH scans.
- 4,365 contained the same nonce as another signature.
- Computed 281 distinct private DSA keys.
- Keys were served by 105,728 (1.03%) of SSH DSA hosts.

0.5% of all TLS hosts vulnerable! 1% of SSH hosts!

ING DIRECT

IRONKEY MODEL S200
THE WORLD'S FIRST FIPS 140-2 LEVEL 3 FLASH DRIVE
AES 256-BIT HARDWARE ENCRYPTION
LEARN MORE

Advertise on NYTimes.com

Flaw Found in an Online Encryption Method

By JOHN MARKOFF
Published: February 14, 2012

SAN FRANCISCO — A team of European and American mathematicians and cryptographers have discovered an unexpected weakness in the encryption system widely used worldwide for online shopping, banking, e-mail and other Internet services intended to remain private and secure.

RECOMMEND
TWITTER
LINKEDIN
COMMENTS (127)
SIGN IN TO E-MAIL
PRINT
REPRINTS
SHARE

SOUND OF MY VOICE
IN THEATRES 04.27.2012
Click to View

Readers' Comments

Readers shared their thoughts on this article.

Read All Comments (127) »

The flaw — which involves a small but measurable number of cases — has to do with the way the system generates random numbers, which are used to make it practically impossible for an attacker to unscramble digital messages. While it can affect the transactions of individual Internet users, there is nothing an individual can do about it. The operators of large Web sites will need to make changes to ensure the security of their systems, the researchers said.

The potential danger of the flaw is that even though the number of users affected by the flaw may be small, confidence in the security of Web transactions is reduced, the authors said.

The system requires that a user first create and publish the product of two large prime numbers, in addition to another number, to generate a public "key." The original numbers are kept secret. To encrypt a message, a second person employs a formula that contains the public number. In practice, only someone with knowledge of the original prime numbers can decode that

Log in to see what your friends are sharing on nytimes.com.
Privacy Policy | What's This?
Log In With Facebook

What's Popular Now

Why I Am Leaving Goldman Sachs

The Benefits of Bilingualism

Gazzang

Advertise on NYTimes.com

Get the TimesLimited E-Mail

Sign Up
Privacy Policy

... only two of the factored certificates were signed by a CA, and both are expired. The web pages aren't active.

... only two of the factored certificates were signed by a CA, and both are expired. The web pages aren't active.

Look at subject information for certificates:

```
CN=self-signed, CN=system generated, CN=0168122008000024
CN=self-signed, CN=system generated, CN=0162092009003221
CN=self-signed, CN=system generated, CN=0162122008001051
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+1145D5C30089/emailAddres
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+139819C30089/emailAddres
CN=self-signed, CN=system generated, CN=0162072011000074
CN=self-signed, CN=system generated, CN=0162122009008149
CN=self-signed, CN=system generated, CN=0162122009000432
CN=self-signed, CN=system generated, CN=0162052010005821
CN=self-signed, CN=system generated, CN=0162072008005267
C=US, O=2Wire, OU=Gateway Device/serialNumber=360617088769, CN=Gateway Authentication
CN=self-signed, CN=system generated, CN=0162082009008123
CN=self-signed, CN=system generated, CN=0162072008005385
CN=self-signed, CN=system generated, CN=0162082008000317
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+3F5878C30089/emailAddres
CN=self-signed, CN=system generated, CN=0162072008005597
CN=self-signed, CN=system generated, CN=0162072010002630
CN=self-signed, CN=system generated, CN=0162032010008958
CN=109.235.129.114
CN=self-signed, CN=system generated, CN=0162072011004982
CN=217.92.30.85
CN=self-signed, CN=system generated, CN=0162112011000190
CN=self-signed, CN=system generated, CN=0162062008001934
CN=self-signed, CN=system generated, CN=0162112011004312
CN=self-signed, CN=system generated, CN=0162072011000946
C=US, ST=Oregon, L=Wilsonville, CN=141.213.19.107, O=Xerox Corporation, OU=Xerox Office Business Group,
CN=XRX0000AAD53FB7.eecs.umich.edu, CN=(141.213.19.107|XRX0000AAD53FB7.eecs.umich.edu)
CN=self-signed, CN=system generated, CN=0162102011001174
CN=self-signed, CN=system generated, CN=0168112011001015
CN=self-signed, CN=system generated, CN=0162012011000446
```

Why do we find vulnerable keys?

Evidence strongly suggested *widespread implementation problems*.

Clue #1: Vast majority generated by network devices:

- Juniper network security devices
- Cisco routers
- IBM server management cards
- Intel server management cards
- Innominate industrial-grade firewalls
- ...

Identified devices from > 50 manufacturers

Why? Random number generation.

crypto keys

Why? Random number generation.

crypto keys

To generate random keys, we need a source of randomness.

Why? Random number generation.

crypto keys

↑

application pseudorandom number generator

To generate random keys, we need a source of randomness.

Why? Random number generation.

crypto keys

↑

application pseudorandom number generator

↗ ↑ ↖

time pid

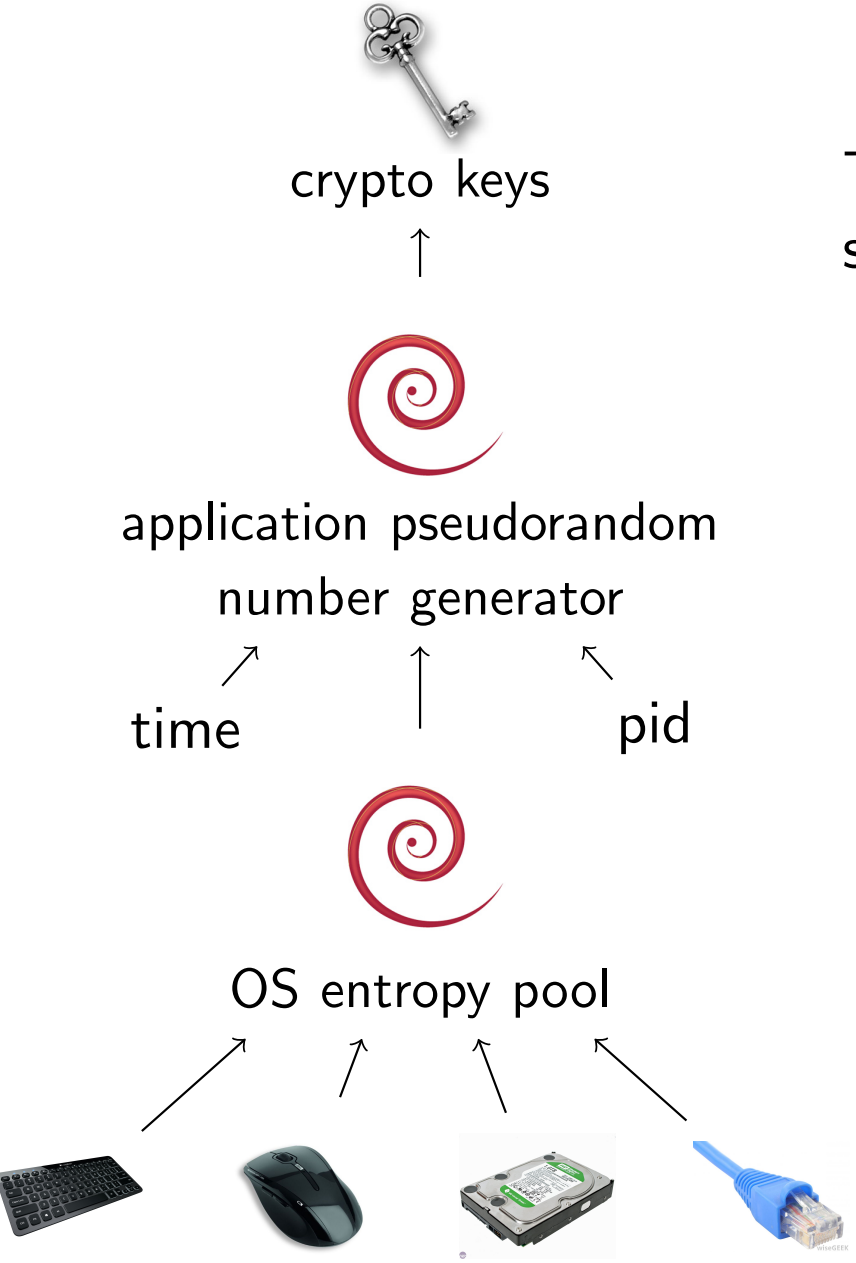
OS entropy pool

To generate random keys, we need a source of randomness.

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."

–John von Neumann

Why? Random number generation.

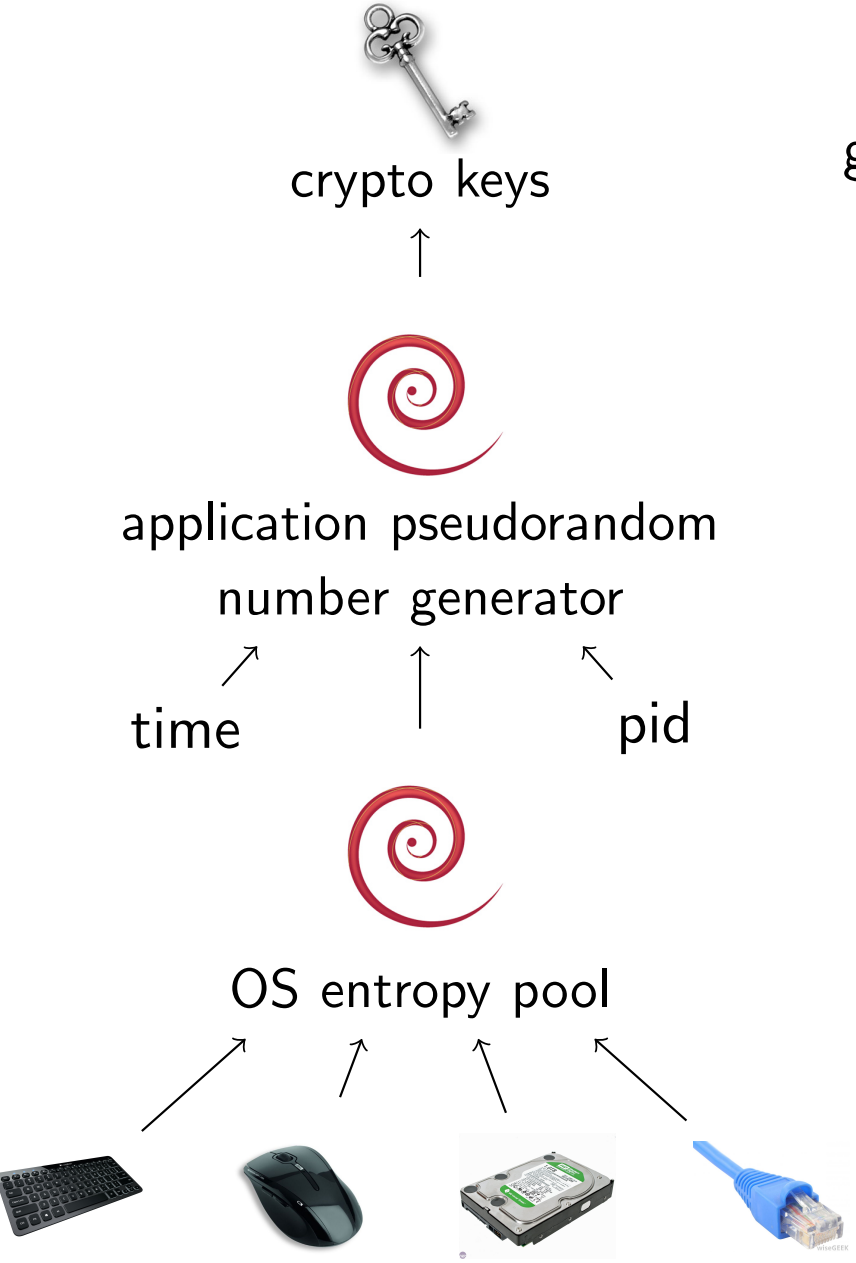


To generate random keys, we need a source of randomness.

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."

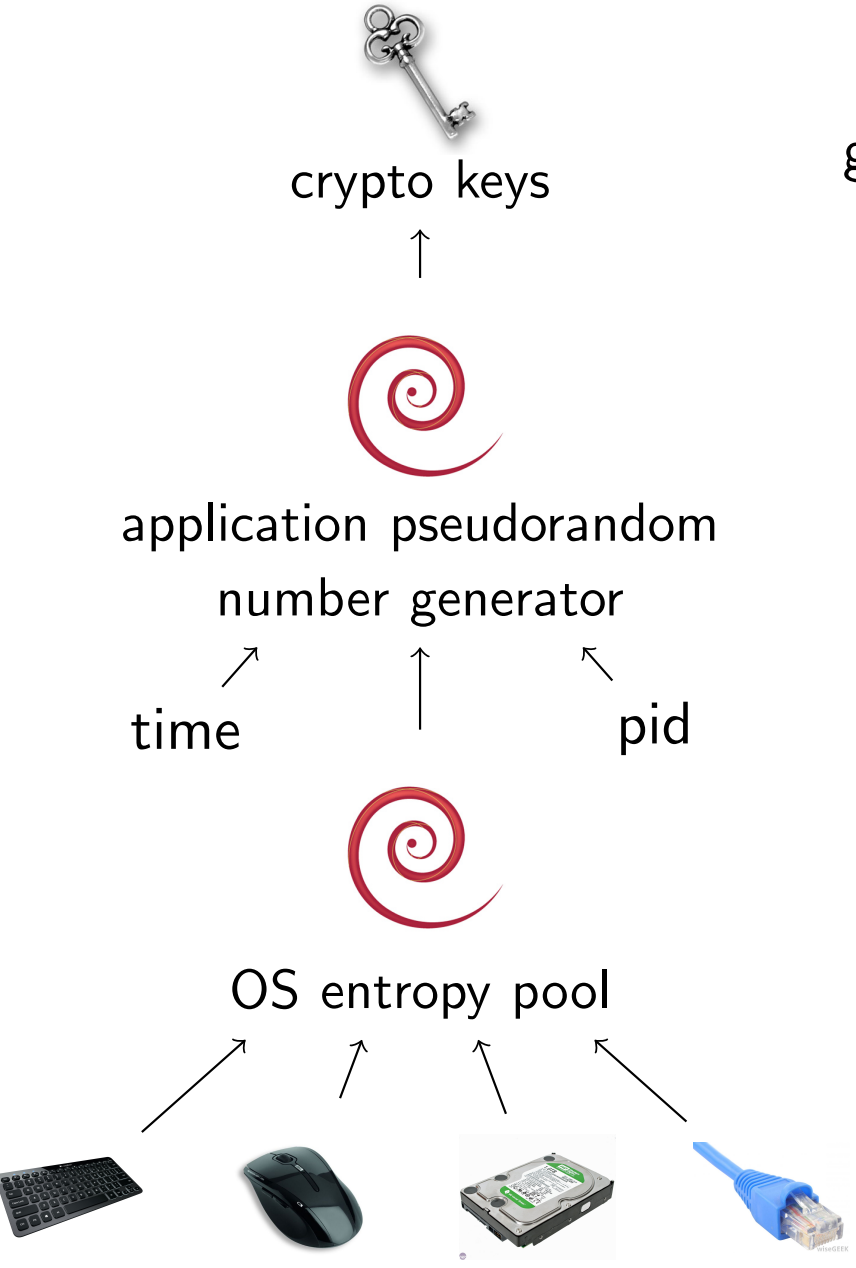
–John von Neumann

Why? Random number generation.



Hypothesis: Devices automatically generate crypto keys on first boot.

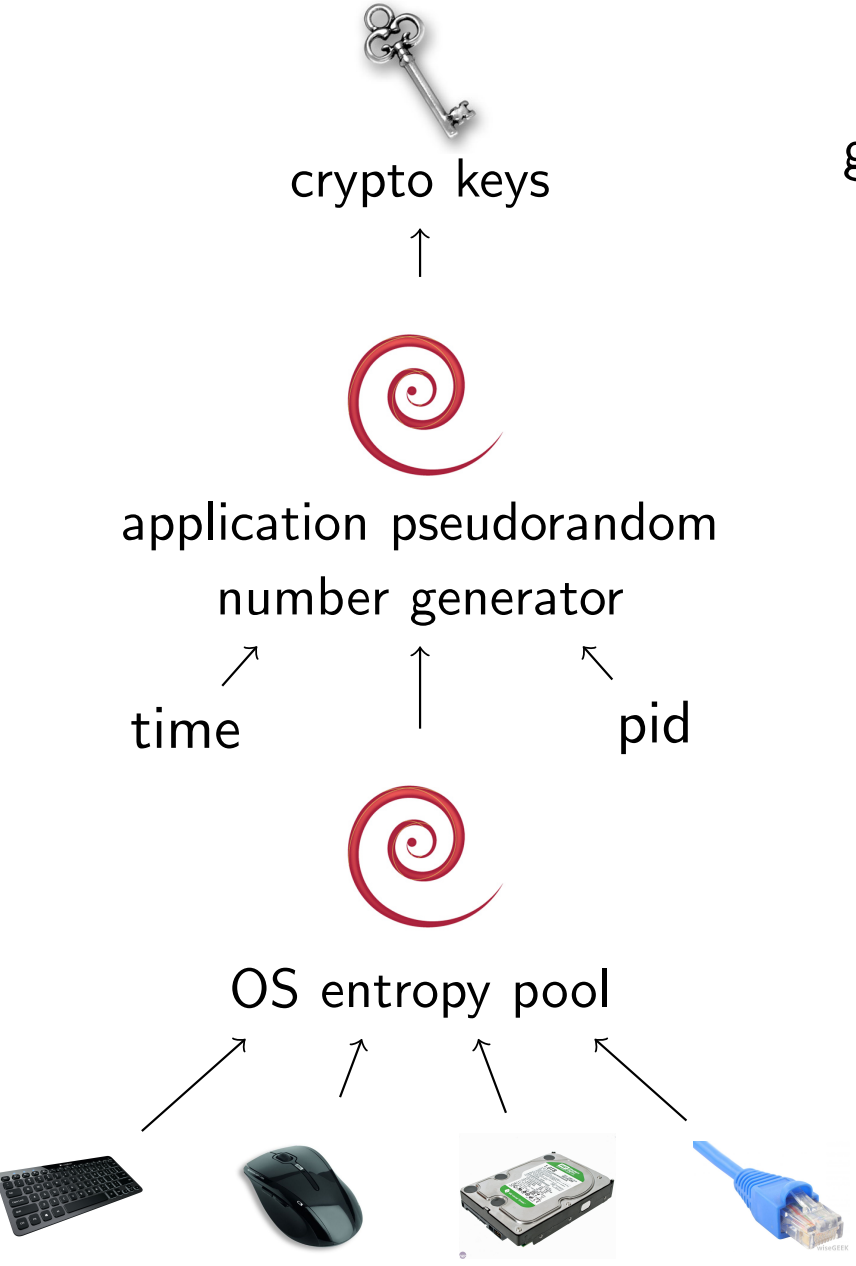
Why? Random number generation.



Hypothesis: Devices automatically generate crypto keys on first boot.

- Headless or embedded devices may lack these entropy sources.

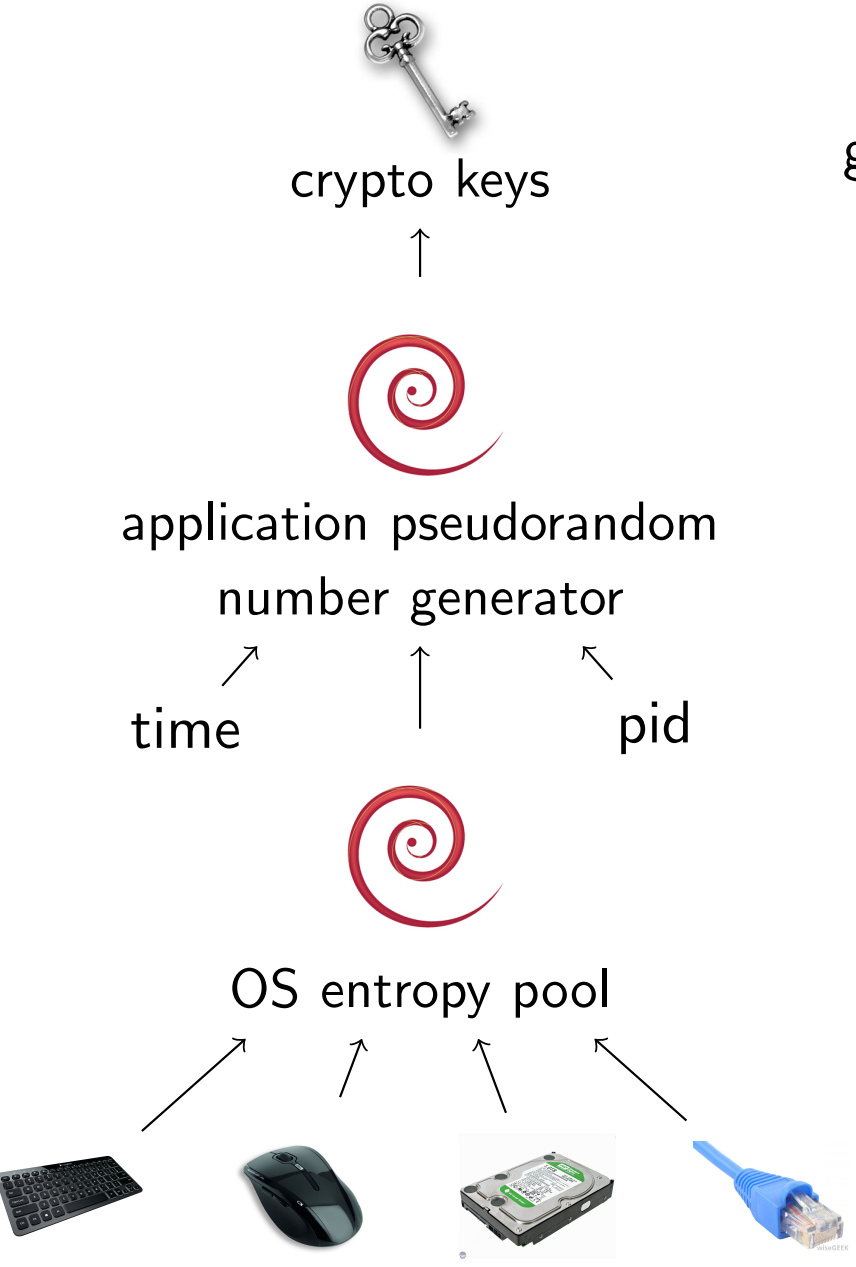
Why? Random number generation.



Hypothesis: Devices automatically generate crypto keys on first boot.

- OS random number generator may not have incorporated any entropy when queried by software.
- Headless or embedded devices may lack these entropy sources.

Why? Random number generation.



Hypothesis: Devices automatically generate crypto keys on first boot.

- OS random number generator may not have incorporated any entropy when queried by software.

  Experimentally verified Linux "boot-time entropy hole"

- Headless or embedded devices may lack these entropy sources.

Linux random number generators

`/dev/random`	`/dev/urandom`
"high-quality" randomness	pseudorandomness
blocks if insufficient entropy available	never blocks

As a general rule, `/dev/urandom` should be used for everything except long-lived GPG/SSL/SSH keys.—`man random`

`random`'s conservative blocking behavior is a usability problem.

This results in many developers using `urandom` for cryptography.

Generating vulnerable RSA keys in software

- Insufficiently random seeds for pseudorandom number generator $\Longrightarrow$ we should see repeated keys.

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

- We do:
  - $>$ 60% of hosts share keys
  - At least 0.3% due to bad randomness.
- Repeated keys may be a sign that implementation is vulnerable to a targeted attack.

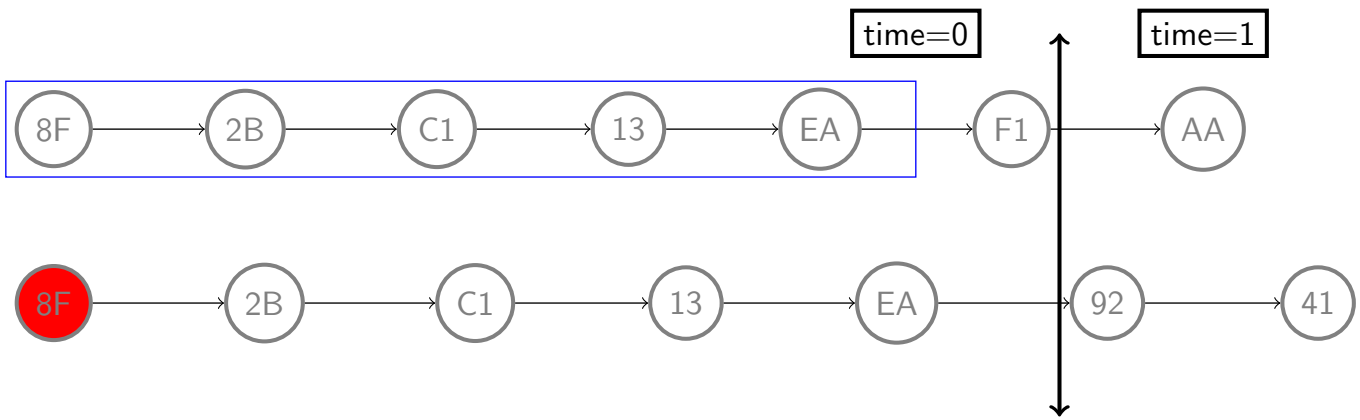
But why do we see factorable keys?

Generating factorable RSA keys in software

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

OpenSSL adds time in seconds

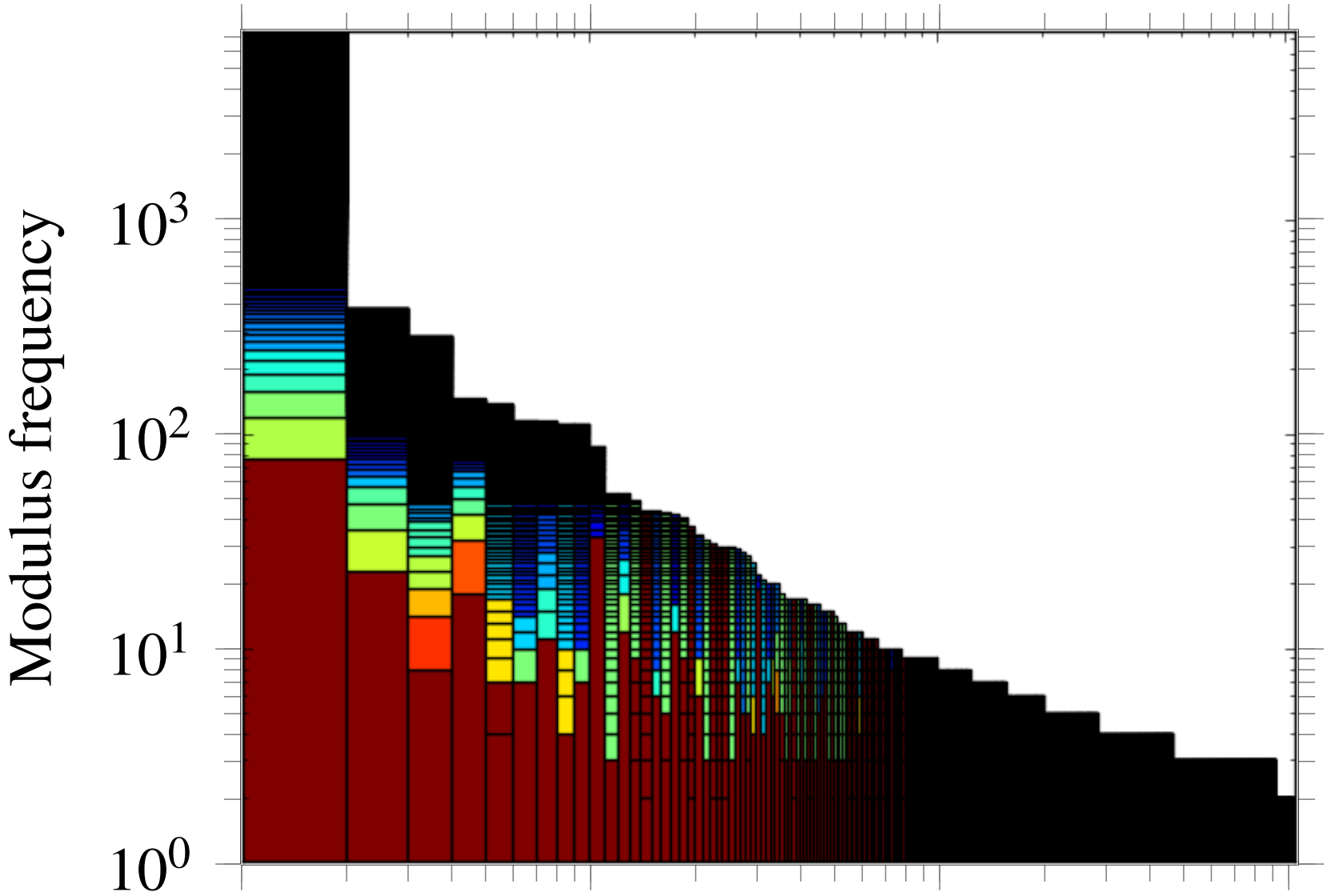
Insufficient randomness can lead to factorable keys.



Experimentally verified OpenSSL generates factorable keys in this situation.

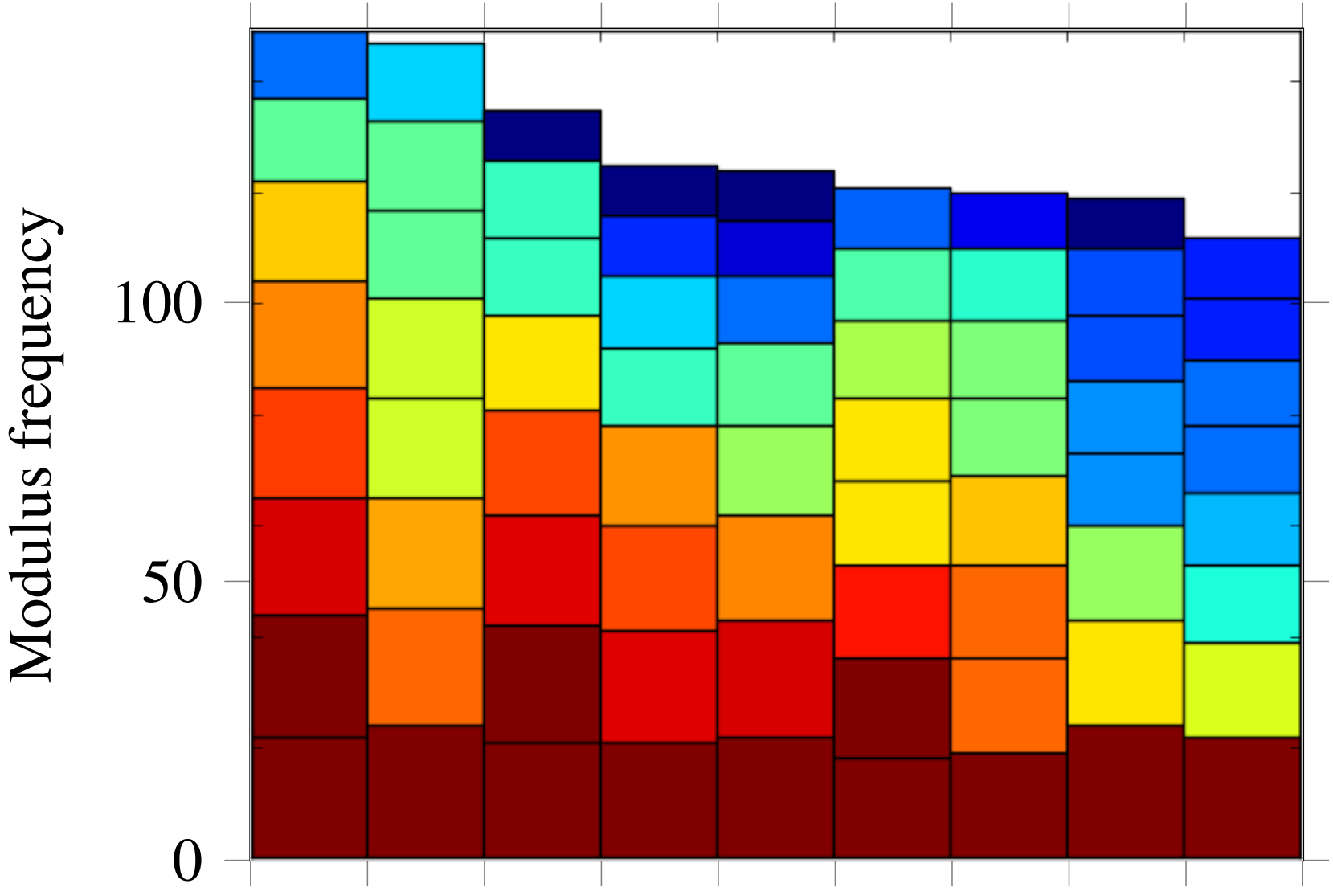
Distribution of prime factors

Juniper SRX branch devices



Distribution of prime factors

IBM Remote Supervisor Adapter II and Bladecenter Management Module



Why do we see weak DSA signatures?

Step 1: DSA keys generated using insufficient entropy, so many hosts shared keys.

Step 2: Signature nonces generated from PRNG counter seeded once using insufficient entropy and never updated.

Two counters in same state $\rightarrow$ keys revealed.

Dropbear (popular SSH implementation) generates nonces this way.

Disclosure

- Wrote disclosures to 61 companies.
- 13 had Security Incident Response Team contact information available.
- Have gotten responses from 28.
- 13 have told us they have fixed the problem
- 5 have informed us of security advisories
- Coordinating through US-CERT, ICS CERT, JP-CERT

Linux kernel has been patched.

More examples of bad randomness

RSA

- **PGP** database: factored 2 of 700,000 keys [Lenstra et al. 2012]
- **Taiwan Citizen Digital Certificates**: 103 factored out of 2.26 million [Chou et al. 2012]

ECDSA

- **Bitcoin**: 133 of 187,000 addresses compromised from nonrandom ECDSA signatures. [Moore Wustrow 2013]

More examples of bad randomness

RSA

- **PGP** database: factored 2 of 700,000 keys [Lenstra et al. 2012]
- **Taiwan Citizen Digital Certificates**: 103 factored out of 2.26 million [Chou et al. 2012]

ECDSA

- **Bitcoin**: 133 of 187,000 addresses compromised from nonrandom ECDSA signatures. [Moore Wustrow 2013]

Possible targeted attack?

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

  Not observed to be exploitable (so far)

Practical mitigations

Developers and manufacturers:

- Collect entropy more aggressively, add hardware sources
- Provide seed in factory
- Run for a while before generating cryptographic keys.

CAs:

- Test for repeated, factorable, and other weak keys.

Users:

- Check against known weak keys.
- Regenerate keys.

Lessons

Systems:

- New insights from taking a macroscopic view of crypto practice.
- Cryptographic entropy is hard to get right.

Cryptography:

- Need to design cryptosystems resilient to random number generation problems. (e.g. DSA nonce deterministically generated from message, salt.)

Contributions

New methodology for discovering vulnerabilities without having to reverse-engineer or study particular implementations.

In-depth analysis of root causes of weak keys in widely deployed software.

Mitigations and suggestions for developers, manufacturers, users, and researchers.

Mining your Ps and Qs: Widespread Weak Keys in Network Devices Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman *Usenix Security 2012*

https://factorable.net