The mechanics of compromising low entropy RSA keys

Austin Allshouse

This talk is about...

Nominally:

Recovering private keys from a subset of vulnerable RSA certificates

Functionally:

Calculating shared factors across large batches of integers

“...using our scalable GCD algorithm for shared factors...”

“...batch GCD on RSA keys, using a custom distributed version...”

“...we adapted the batch GCD implementation…”

Hello darkness, my old friend...

$$p \times q = n$$

random prime | random prime | public modulus

When primes are reused...

$$p \times q_1 = n_1; \quad p \times q_2 = n_2$$

$$\gcd(n_1, n_2) = p$$

$$n_1/p = q_1; \quad n_2/p = q_2$$

Select past research...

2012

"Mining your Ps and Qs..."

- Discovered widespread prime reuse in certificates

- Demonstrated flaws in pseudorandom number generation

2016

"Weaks Keys Remain Widespread..."

- Greatly expanded scope of keys evaluated (81 million)

- Detail a method of parallelizing modulus factorization

2018

"Reaping and breaking keys at scale..." @DEF CON 26

- Industrialized key acquisition and factoring on a massive scale from diverse sources (hundreds of millions)

GCD circa 300 BC (Euclid)

Prime products: (7 x 67) = 469; (11 x 61) = 671; (7 x 59) = 413; (17 x 53) = 901

```python
from itertools import combinations
products = [469, 671, 413, 901]
def gcd(a, b):
  if a == 0:
      return b
  return gcd(b%a, a)

for pair in combinations(products, 2):
   print(f'gcd{pair} = {gcd(*pair)}')
```

```
gcd(469, 671) = 1
gcd(469, 413) = 7
gcd(469, 901) = 1
gcd(671, 413) = 1
gcd(671, 901) = 1
gcd(413, 901) = 1
```

Batch GCD circa 2004 AD (Bernstein)

Product Tree → Remainder Tree

Building:

`child1 * child2 = parent`

Decomposing:

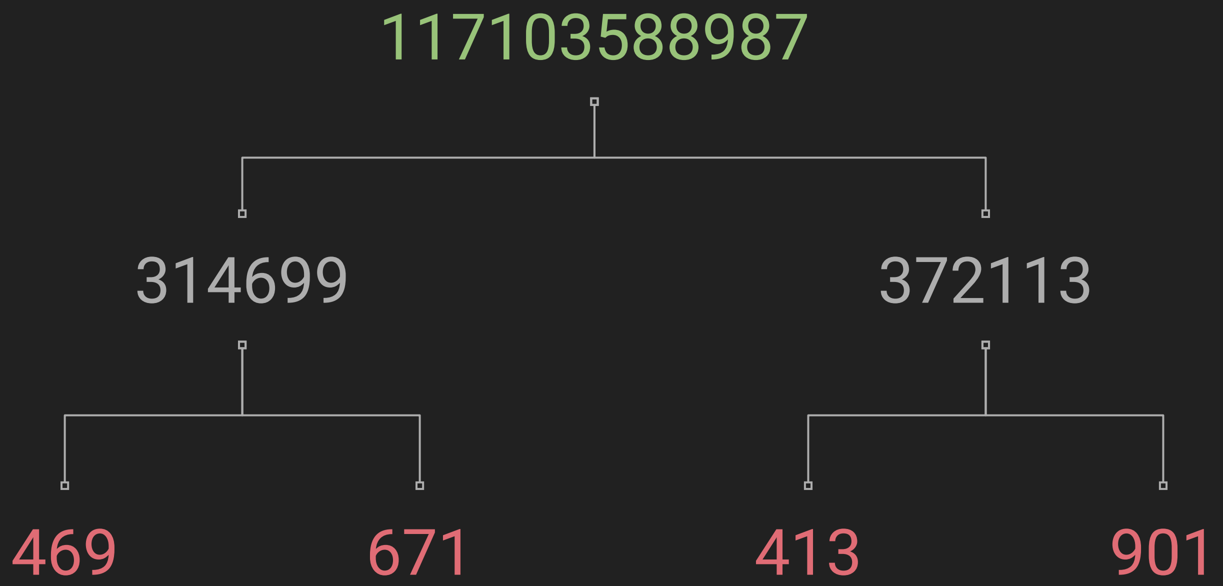
parent mod $child^2$ = child

Remainder Tree Leaves

gcd(remainder/product, product) = shared_factor

Product Tree

Prime products: (7 x 67) = 469; (11 x 61) = 671; (7 x 59) = 413; (17 x 53) = 901



Remainder Tree

Prime products: (7 x 67) = 469; (11 x 61) = 671; (7 x 59) = 413; (17 x 53) = 901

117103588987

117103588987 mod $(314699)^2$ = 18068128386

117103588987 mod $(372113)^2$ = 117103588987

18068128386 mod $(469)^2$ = 91924

18068128386 mod $(671)^2$ = 407297

117103588987 mod $(413)^2$ = 124313

117103588987 mod $(901)^2$ = 482936

gcd(91924 / 469, 469) = 7

gcd(407297 / 671, 671) = 1

gcd(124313 / 413, 413) = 7

gcd(482936 / 901, 901) = 1

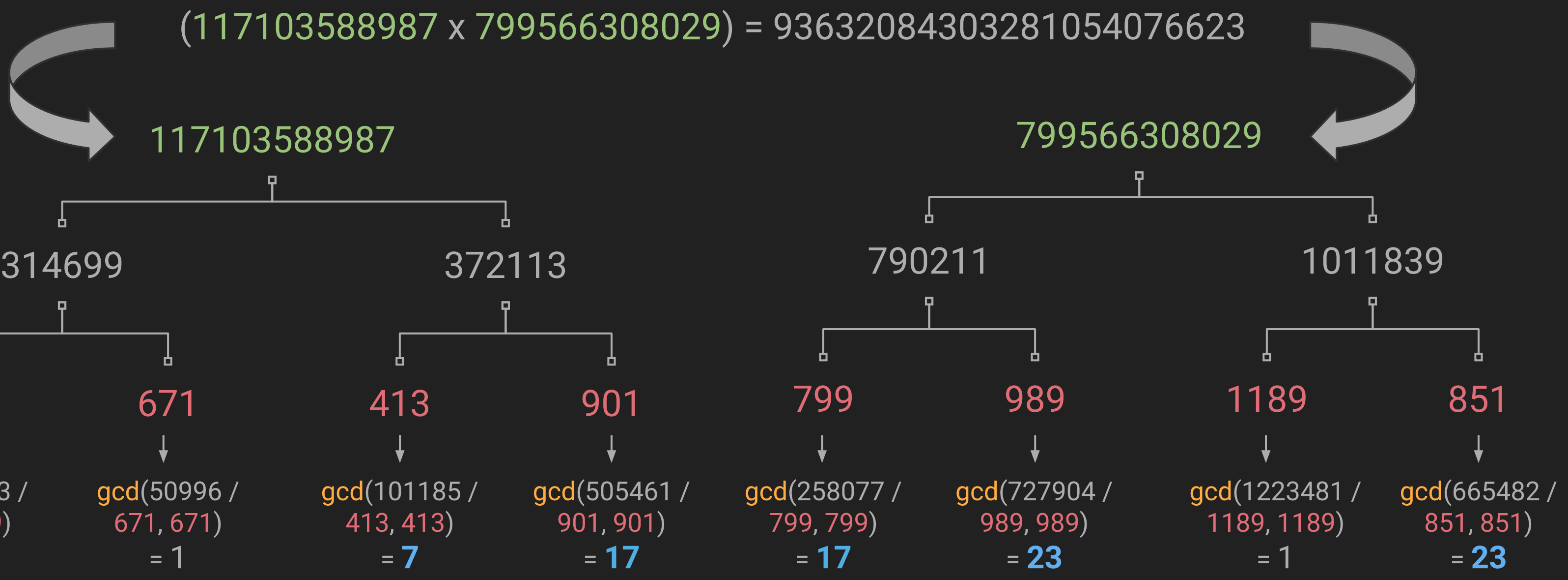
Parallelization - 150 million 2048-bit moduli

Batch Count	**1**	**5**
Batch Size	150 million	30 million
Product Tree Size	> 1 terabyte	~ 180 gigabytes
Tree Permutations	1	20

Tree permutation

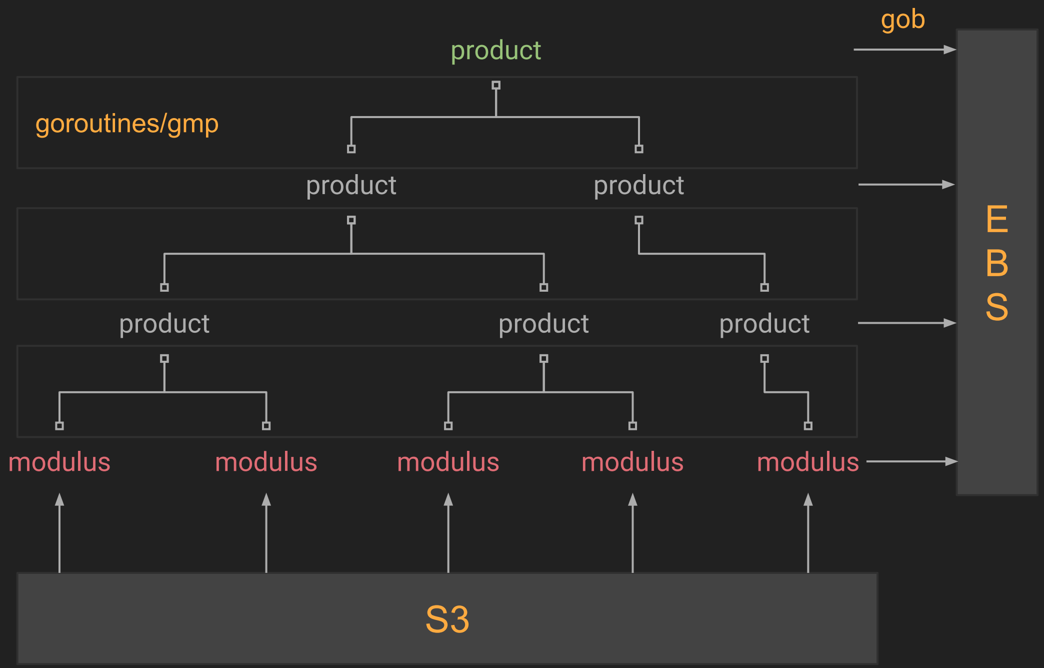
Batch 1: (7 x 67) = 469; (11 x 61) = 671; (7 x 59) = 413; (17 x 53) = 901

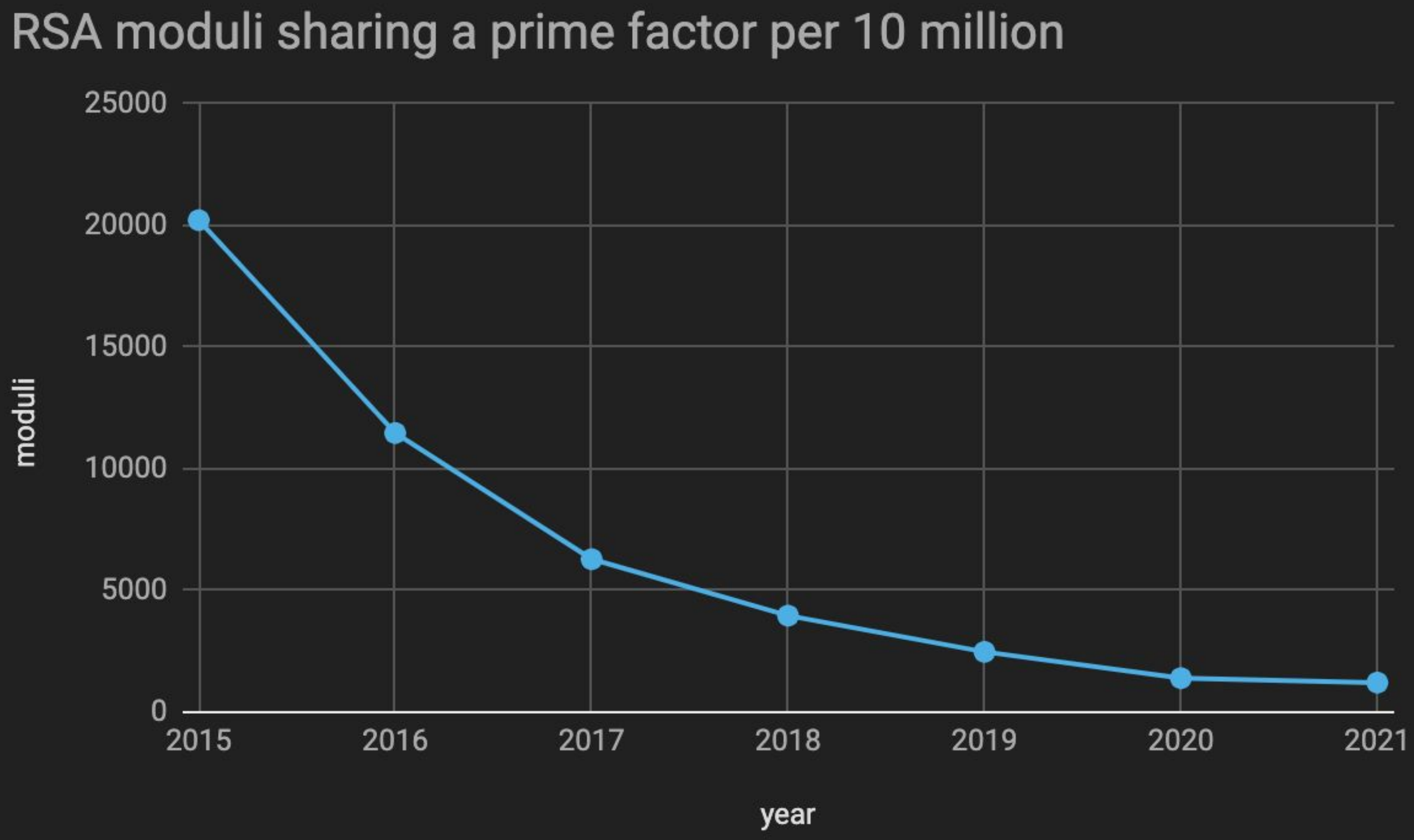
Batch 2: (17 x 47) = 799; (23 x 43) = 989; (29 x 41) = 1189; (23 x 37) = 851



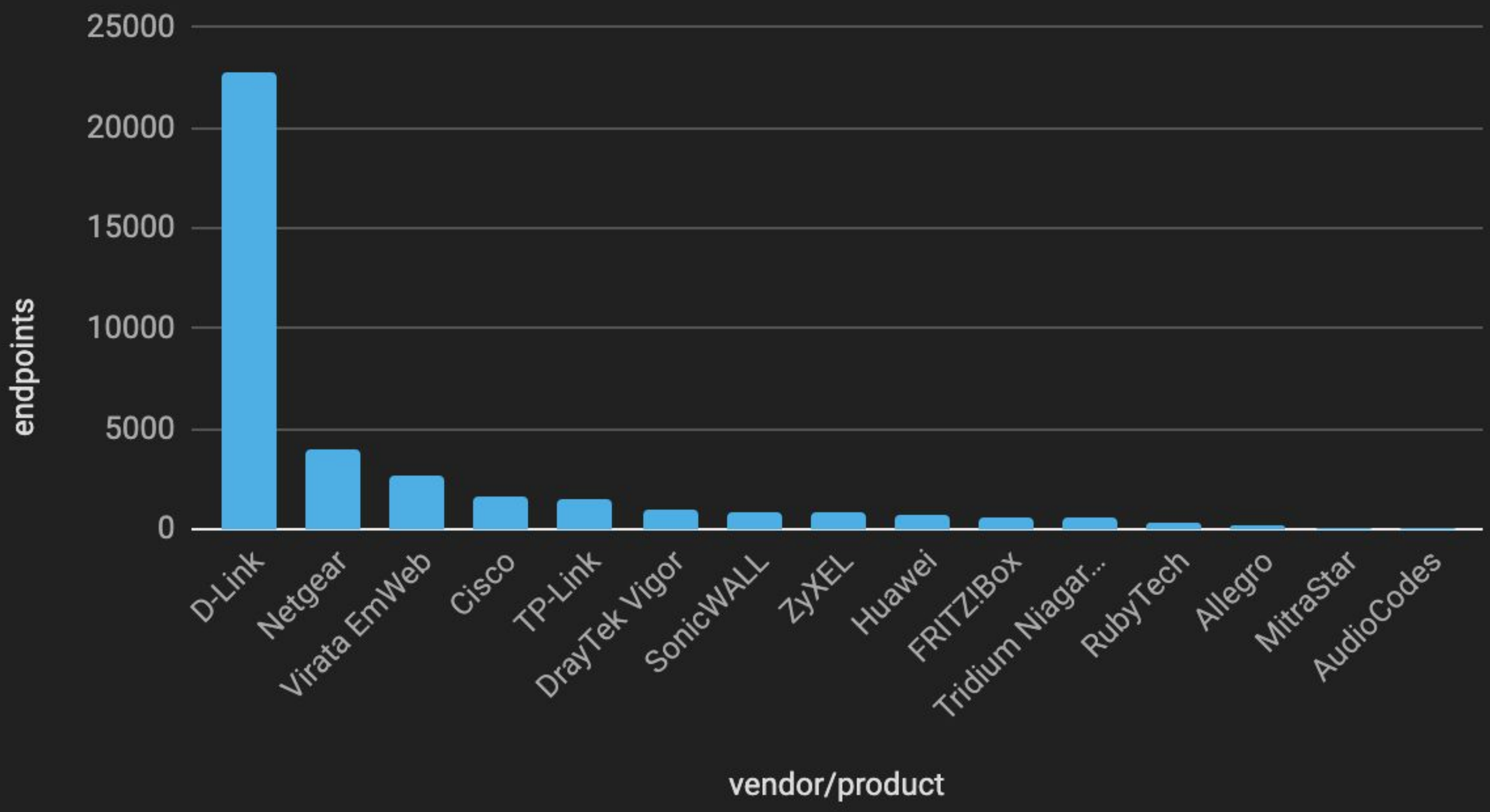
Implementation tech stack

Language	golang
Arithmetic	github.com/ncw/gmp
Storage	S3 / EBS
Serialization	gob
Concurrency	goroutines
Orchestration	bash

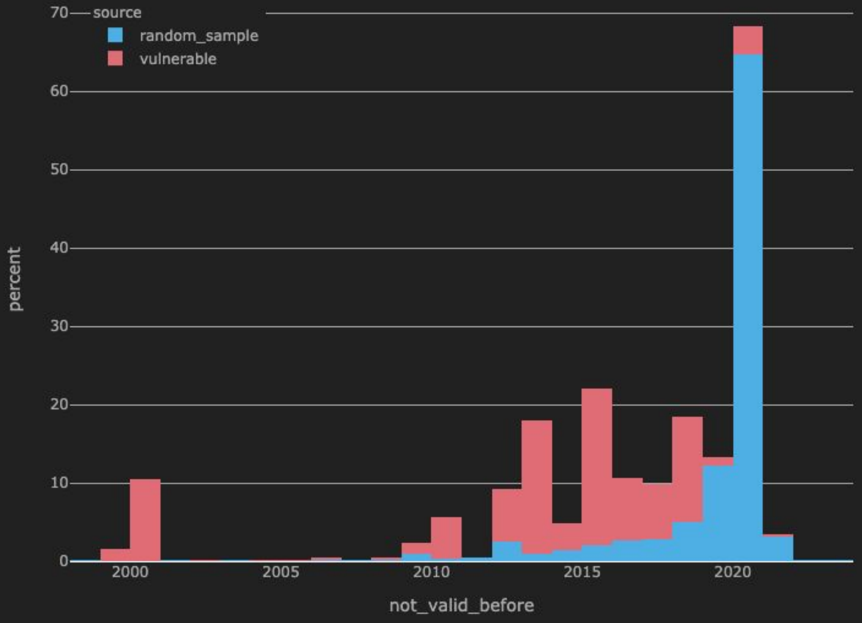


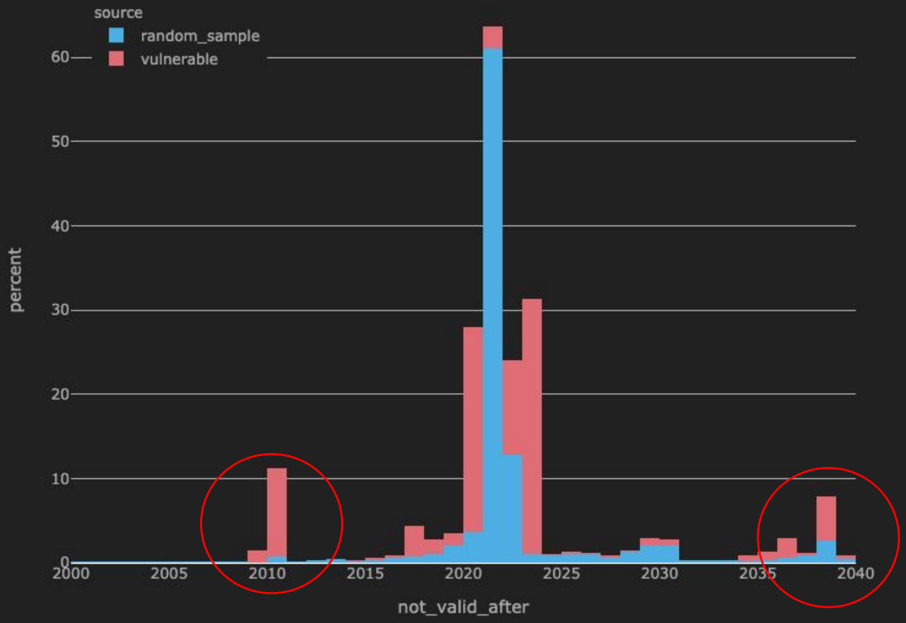

RSA moduli sharing a prime factor per 10 million
25000
20000
15000
10000
5000
0
moduli
2015
2016
2017
2018
2019
2020
2021
year

Top 15 endpoints with factorable keys by vendor



Old and busted certificates



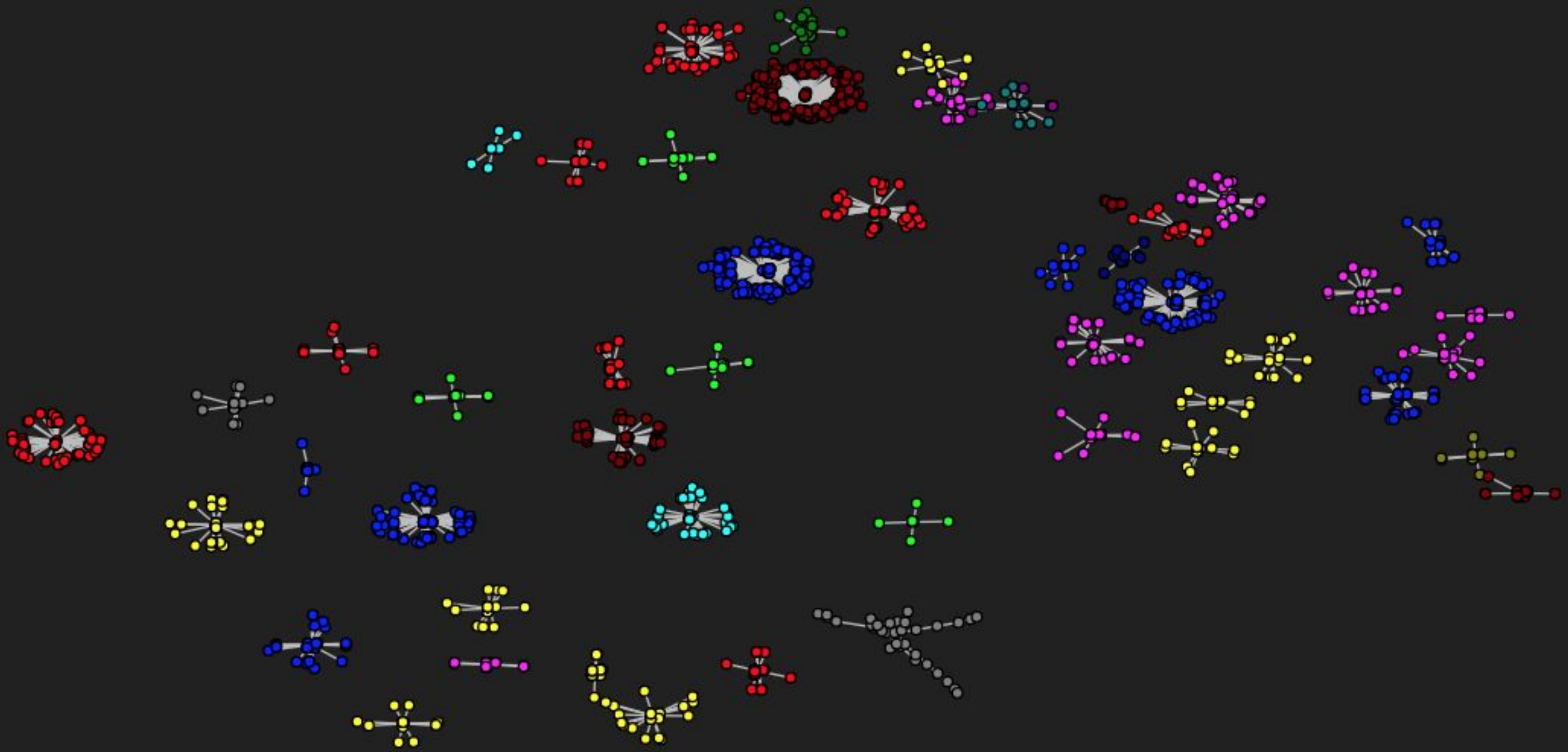


At risk...

- Vendor auto-generated device certificates
- Old, unmanaged devices (i.e. shadow IT)

Industry Sectors	Relative Likelihood of Vulnerability
Finance, Insurance, Legal	**1x**
Business Services, Engineering	**3x**
Government, Manufacturing, Hospitality	**4x**
Defense, Entertainment, Real Estate	**6x**
Utilities	**10x**

Shared primes are device-specific; disjoint



In conclusion...

- Vendors have largely addressed this vulnerability
  - doesn’t matter if old keys are still in use
- Isolated to self-signed/non-public CA signed certificates
- Massive scale of key acquisition is not necessary
  - limit batches to keys from specific devices

Reference Implementation (Python)

https://github.com/austinallshouse/defcon29-key-factorization-reference

BITSIGHT®
The Standard in SECURITY RATINGS