Tag You're It!

Revisiting the Reality of DNSSEC Keytags

Roland van Rijswijk-Deij

NLNETLABS

Introduction

- **DNSSEC validation** requires a resolver to **match signatures to keys**
  → it is **inefficient to check** a signature **against every DNSKEY** in a zone

- To enable **fast matching**, DNSSEC has the notion of **keytags**
  (first introduced in draft 01 of what became RFC 2535 in August 1997)

- These are **16-bit values** included in RRSIG records and can **help** a resolver **find a matching key**

- They are **only a hint**; the **idea is** that it is **unlikely for keytag collisions to occur** in a DNSKEY set, so they **help** match keys to signatures **in most cases**

NLNETLABS

Introduction

- **Some years ago**, Roy Arends presented at **DNS OARC** on the **curious case of the unused keytags** [1]
- He generated **large numbers of keys**, and found that **only between a quarter and half out of the 65536 possible keytags occurred**
- **Due to mathematical properties of RSA keys and** how the **keytag algorithm** computes the tag from the key's RDATA

[1] https://slideplayer.com/slide/10329289/

NLNETLABS

In this talk

- While - with a generous community effort - Roy managed to explain **why certain keytags do not occur in theory**
- In this talk we look at **what keytags occur in practice**
- And **draw lessons** from this **for protocol design** (or: why we should have picked a different, simple algorithm for keytag computation)

Quick refresher: keytag algorithm

- Assuming "self" is a Python DNSKEY object, this is the algorithm:

```
def keytag(self):
  acc = int(0)

  wire = self.towire()

  for i in range(0, len(wire)):
    if i & 1 == 1:
      acc += wire[i]
    else:
      acc += wire[i] << 8

    acc += (acc >> 16) & 0xffff

  return acc & 0xffff
```

- Basically accumulate even bytes in the lower 8 bits, odd bytes in the high 8 bits, and do some fiddling with carry bits

- The outcome of this algorithm highly depends on the amount of structure and predictability in the input!

RSA keys have a lot of structure

- **Skipping** the **details** (which can be found in Roy Arends' OARC presentation), **RSA keys have a lot of structure**
- E.g. due to the **modulus always** being **odd**, **use of safe primes** in key generation, strong **preference for** certain **public exponents**, ...
- Also: **flags, protocol version and algorithm** are **also included** in the computation!
- In the **OARC presentation**, there was talk of **either 16384 or 32768 possible keytags** (but **did not take every case into consideration**)

What happens in practice?

- **Experiment** discussed **at OARC relied on generating** lots of **keys**
- What we wanted to know: **what happens in the wild?**
- We took one recent day of **data from OpenINTEL for .com and .nl** and looked at two things:
  - **Keytags in the wild** for RSA and ECDSA
  - **Keytag collisions in the wild** for RSA and ECDSA

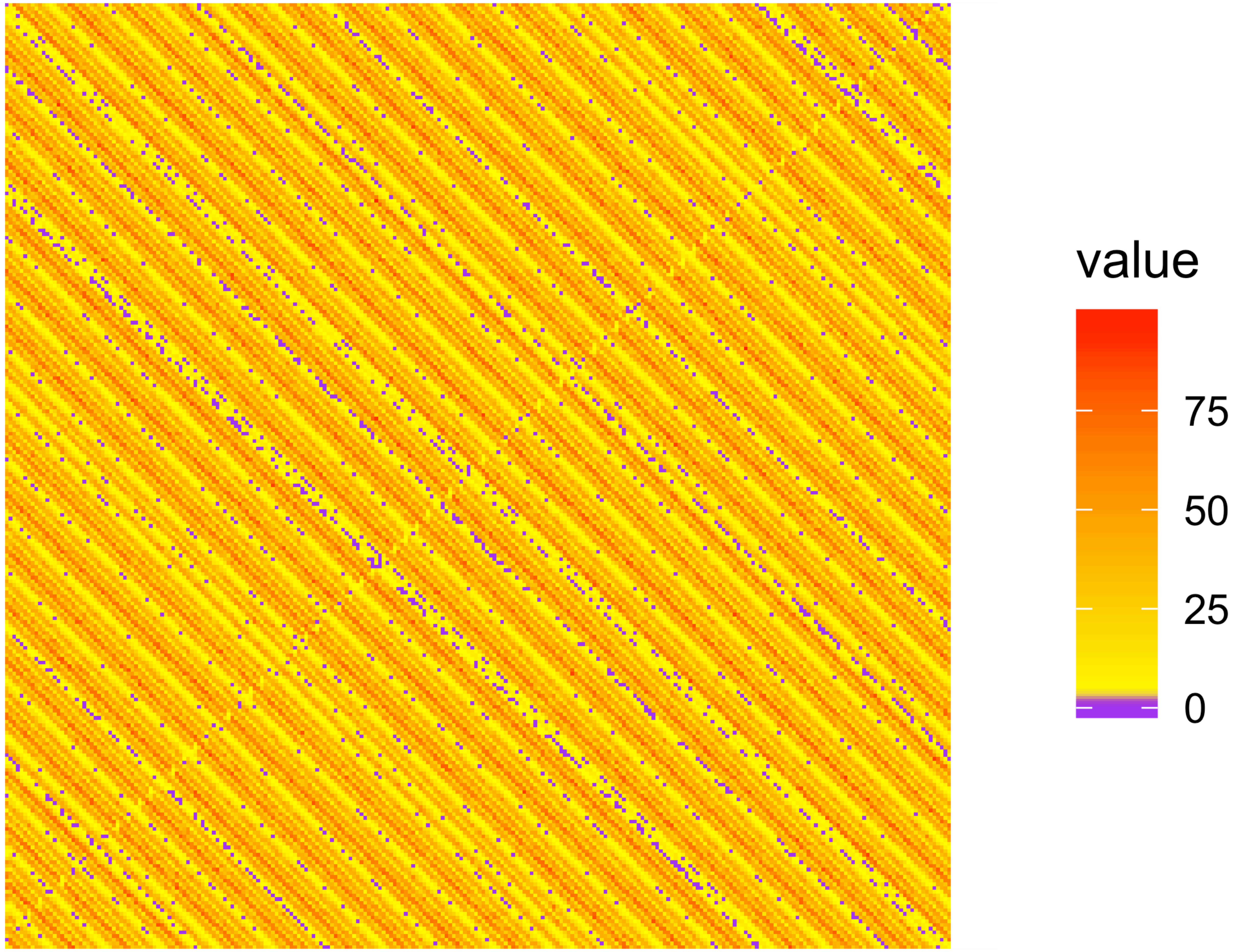
NLNETLABS

What we expect to find

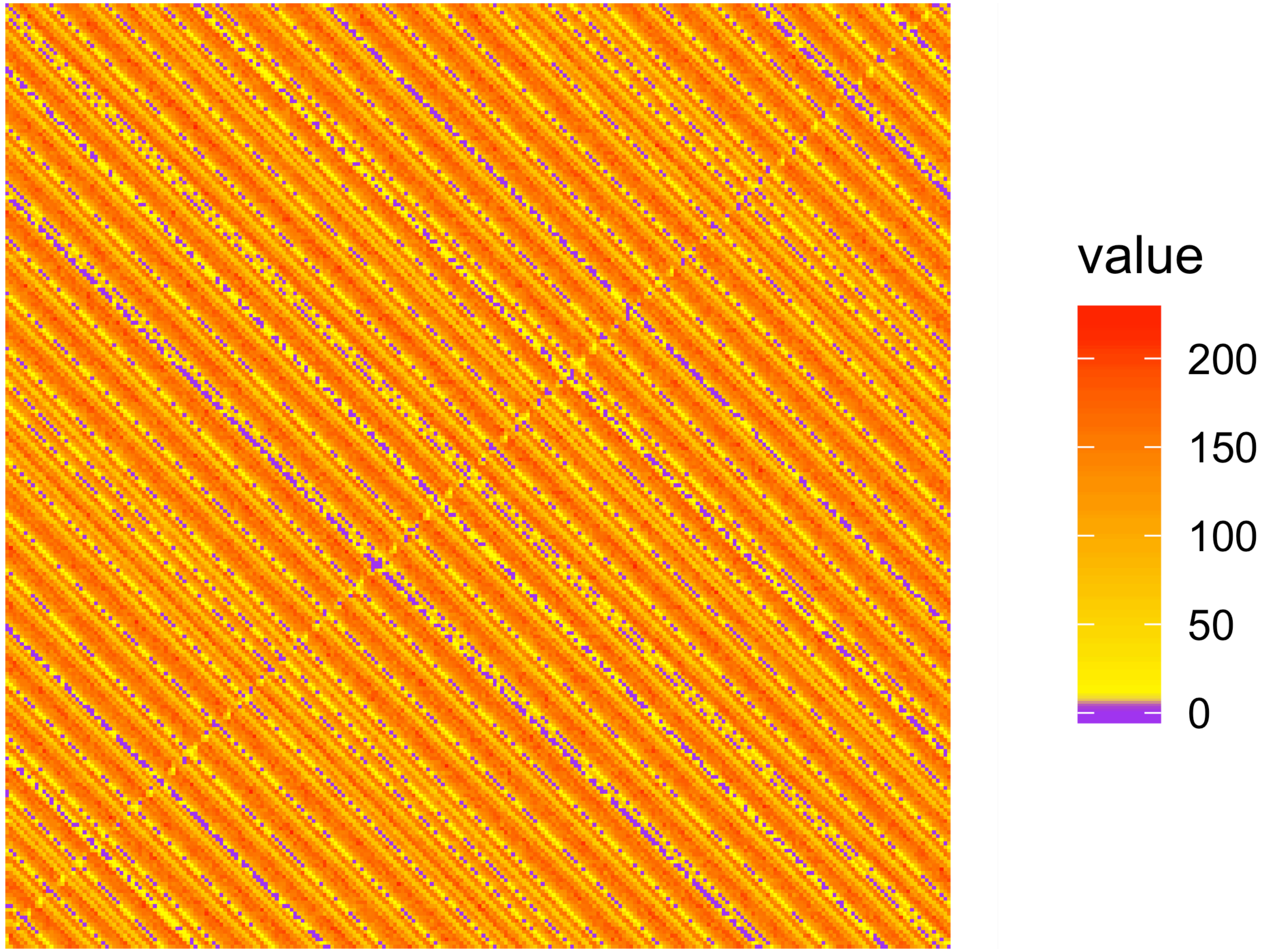
- As stated, RSA keys contain a lot of structure, and the keytag algorithm output is influenced by structure
- We **expect to find:**
  - That **certain keytags** are **much more common for RSA** keys
  - That **keytag collisions occur in the wild for RSA** keys
  - That the **occurrence of keytags for ECDSA** is much more **uniformly distributed** due to a lack of structure in ECDSA keys

NLNETLABS

Heatmaps for RSA keytags



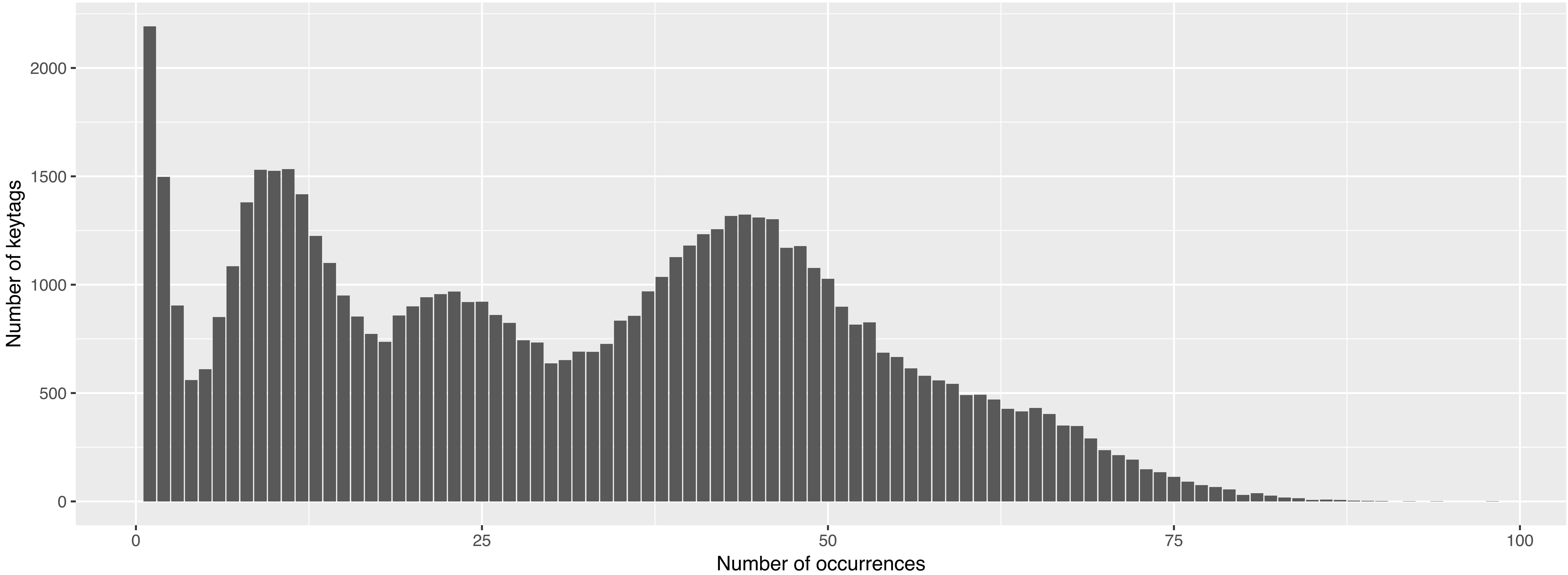
.com



.nl

NLNETLABS

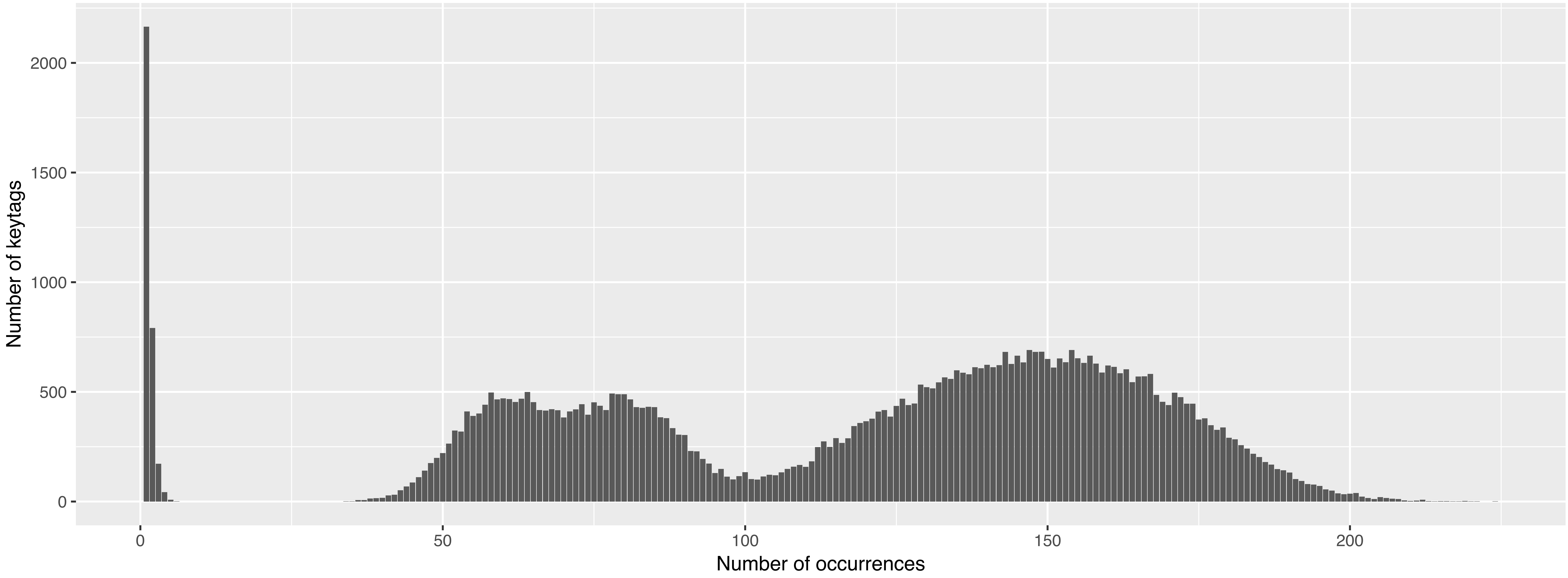
Distribution of RSA keytags in .com



- Takeaway: some keytags occur only once, a few keytags occur over 75 times more often than that

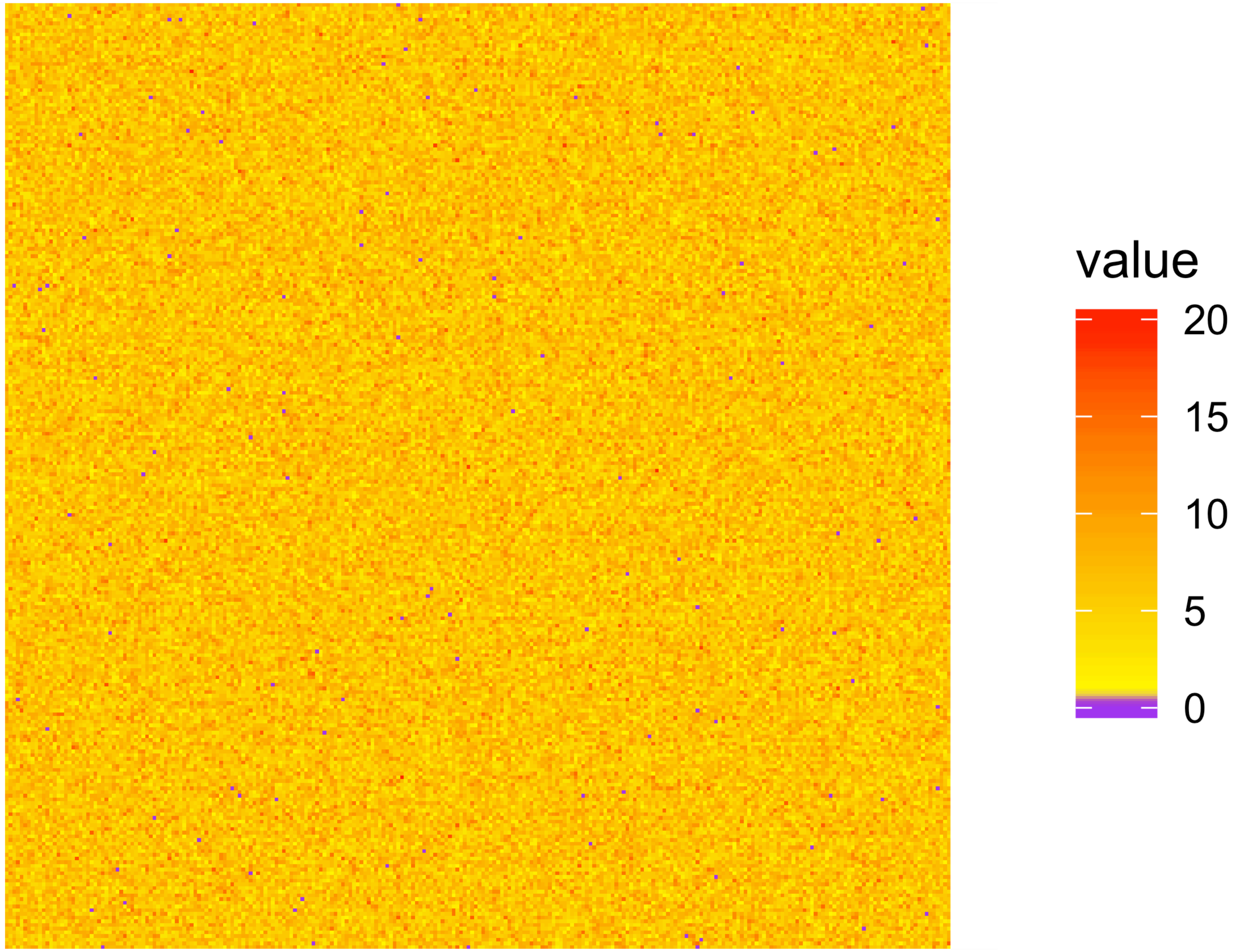
NLNETLABS

Distribution of RSA keytags in .nl

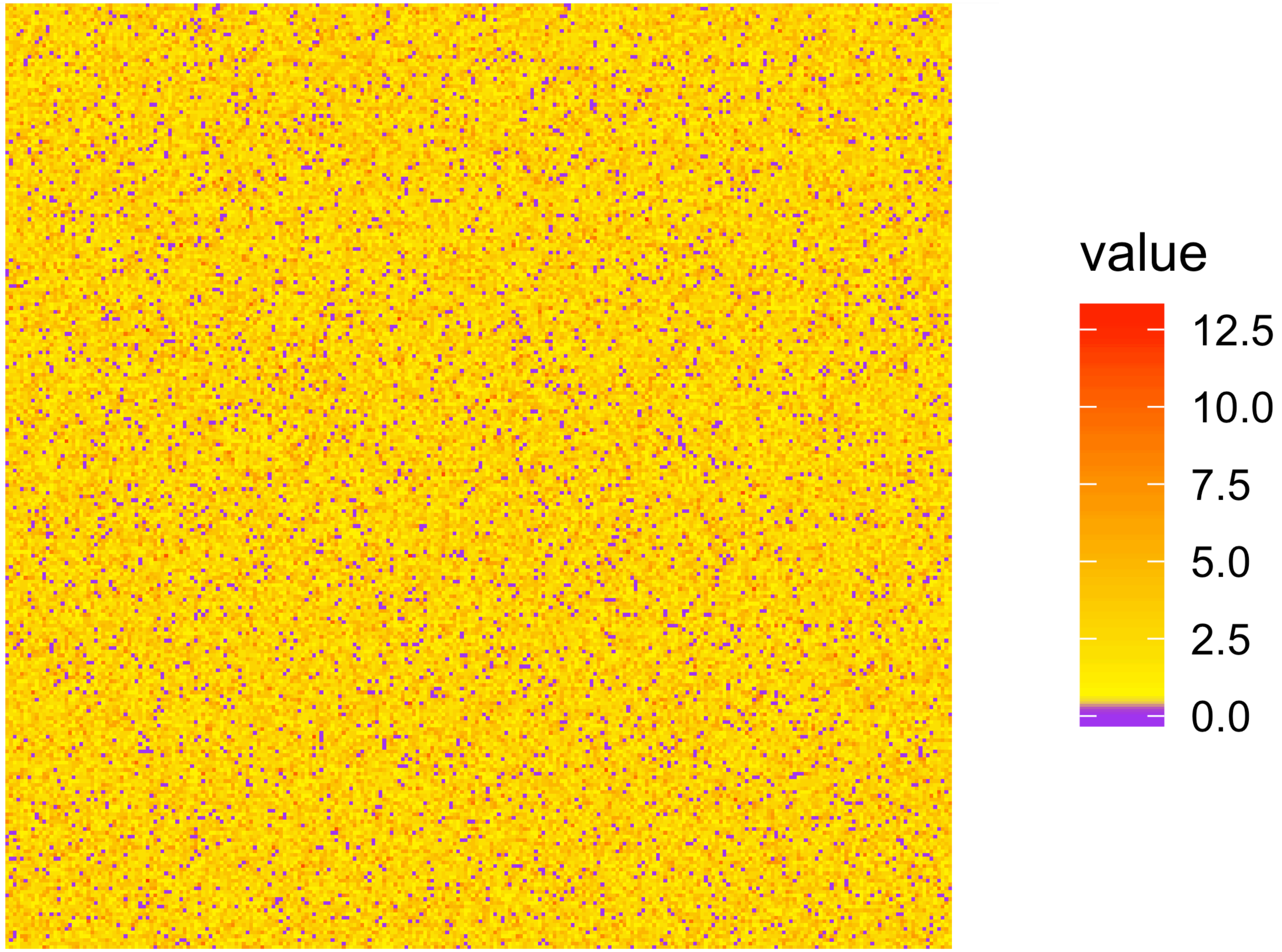


- Why the difference? Different distribution in algorithms and key sizes!

So what about ECDSA?

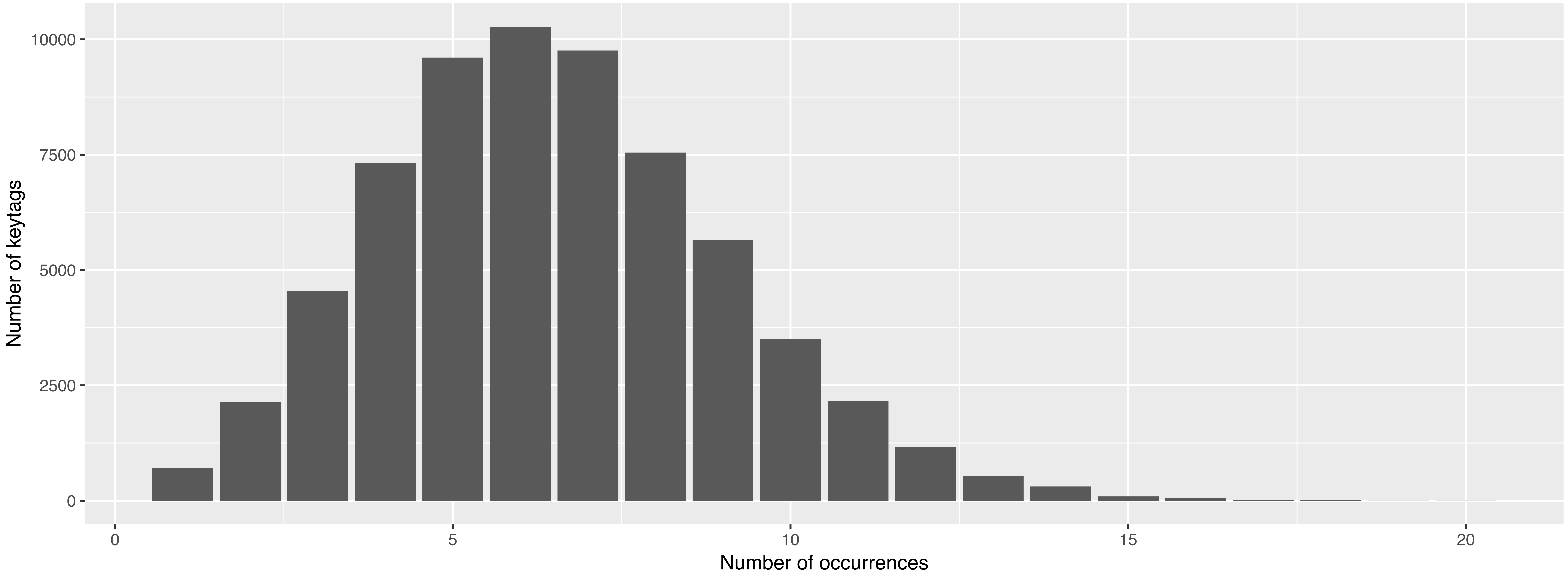


.com



.nl

ECDSA keytag distribution



- This **looks much more like a Gaussian** distribution (but isn't quite)

NLNETLABS

Collisions

- **Searched for collisions in** OpenINTEL, by computing keytags for DNSKEY RRsets for all signed domains in **.nl** over **3 years of data**

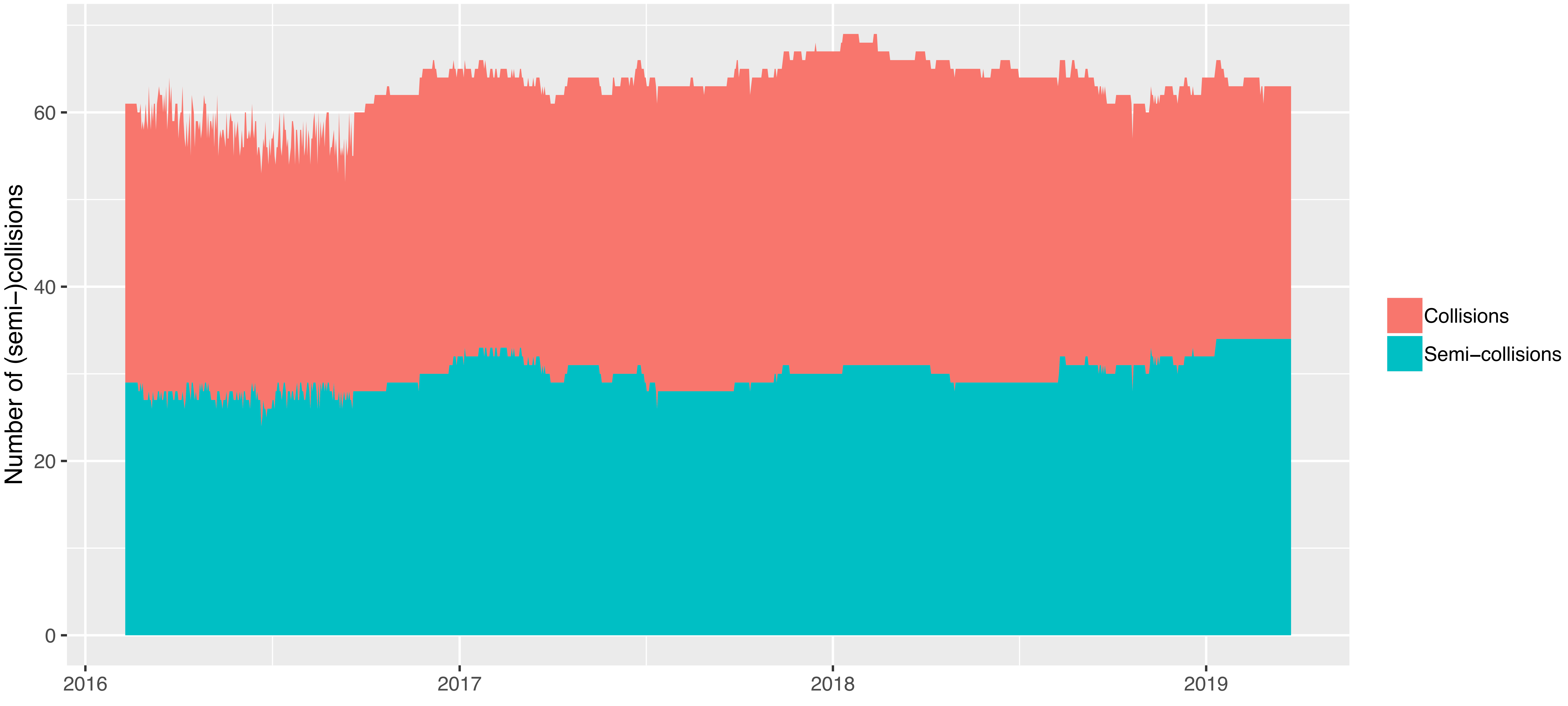
- Distinguish between **two types of collisions:**

  - "**Real"** collisions, where two or more **different keys** in the keyset of the **same algorithm and size have the same keytag**

  - **"Semi"** collisions, where two or more different keys in the keyset have the **same key tag, but** have a **different size or algorithm**

NLNETLABS

Collisions over time



NLNETLABS

How rare are collisions?

- If keytags were uniformly distributed, we would expect the birthday paradox to apply, with *n* the number of keys and *d* the number of possible keytags; table shows this theoretical vs. actual probability

#keys in DNSKEY set	Theoretical probability	Observed probability
2	0,00153%	0,00122%
3	0,00458%	0,00322%
4	0,00915%	0,01616%
5 or more	0,01526%	0,00148%

Real-world impact

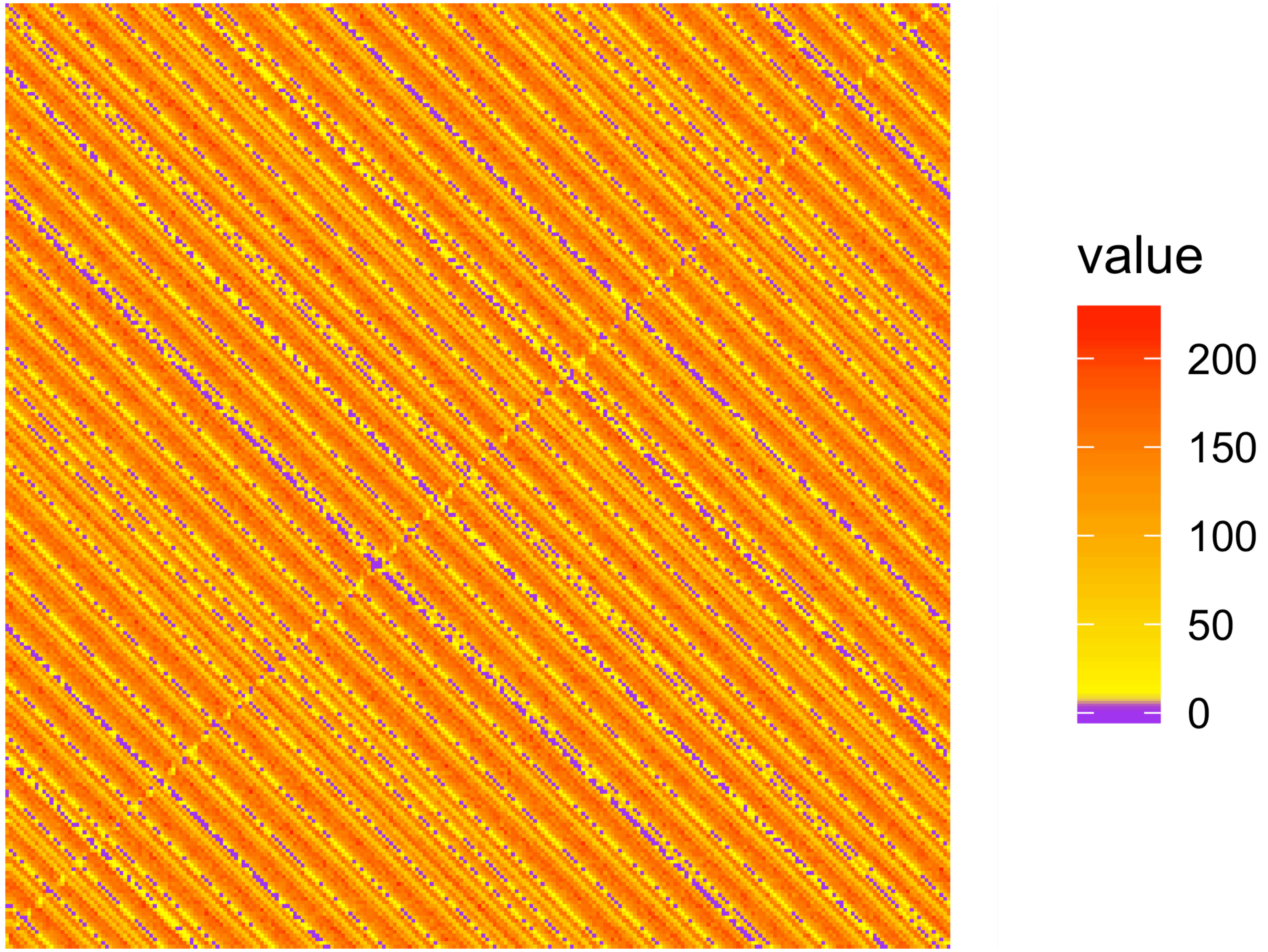
- **Keytags** are **only intended as a hint** for resolvers; they should **never solely** rely on them to **identify** the **correct key for** signature **validation**
- While very rare in practice, **collisions have a real impact**
- A **collision forces resolvers to validate signatures against all keys** that have a matching key tag until the correct key is found
- This **will lead to extra** CPU intensive **cryptographic operations**, that cause a small, but **quantifiable increase in load** on validating resolvers

NLNETLABS

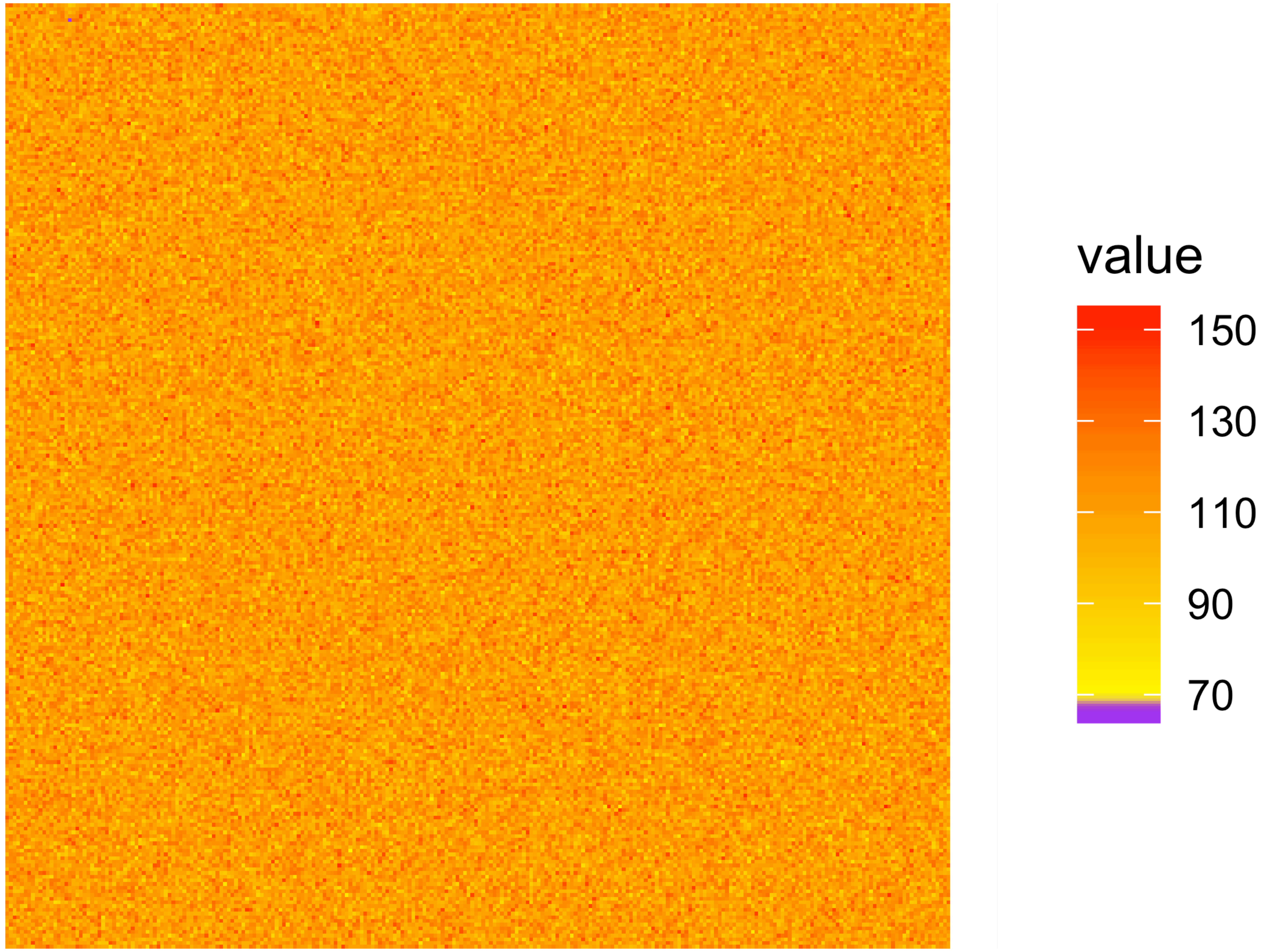
What could we do differently?

- Clearly, the **keytag algorithm appears not optimal** for its purpose, **but** appears to make **collisions less likely than expected**
- So **what** would happen **if** we used something that produces a **random uniform output**, regardless of any structure in the input?
- (Cryptographic) **hash functions have this property**, **but** are likely **much more computationally intensive**
- Yet there is a **middle ground:** what if we used **CRC16?**

Heatmap: keytag vs. CRC16 for .nl

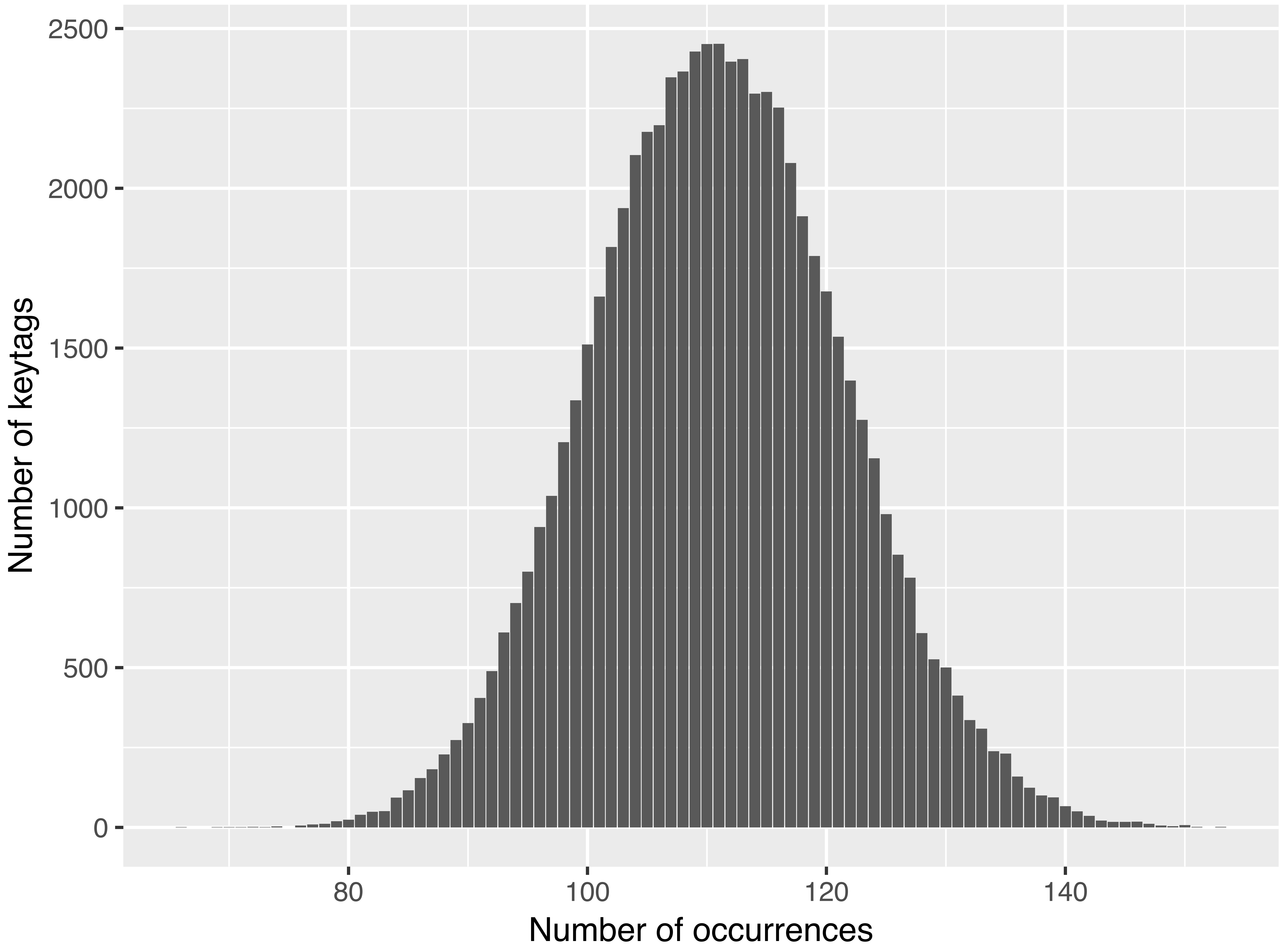


keytags



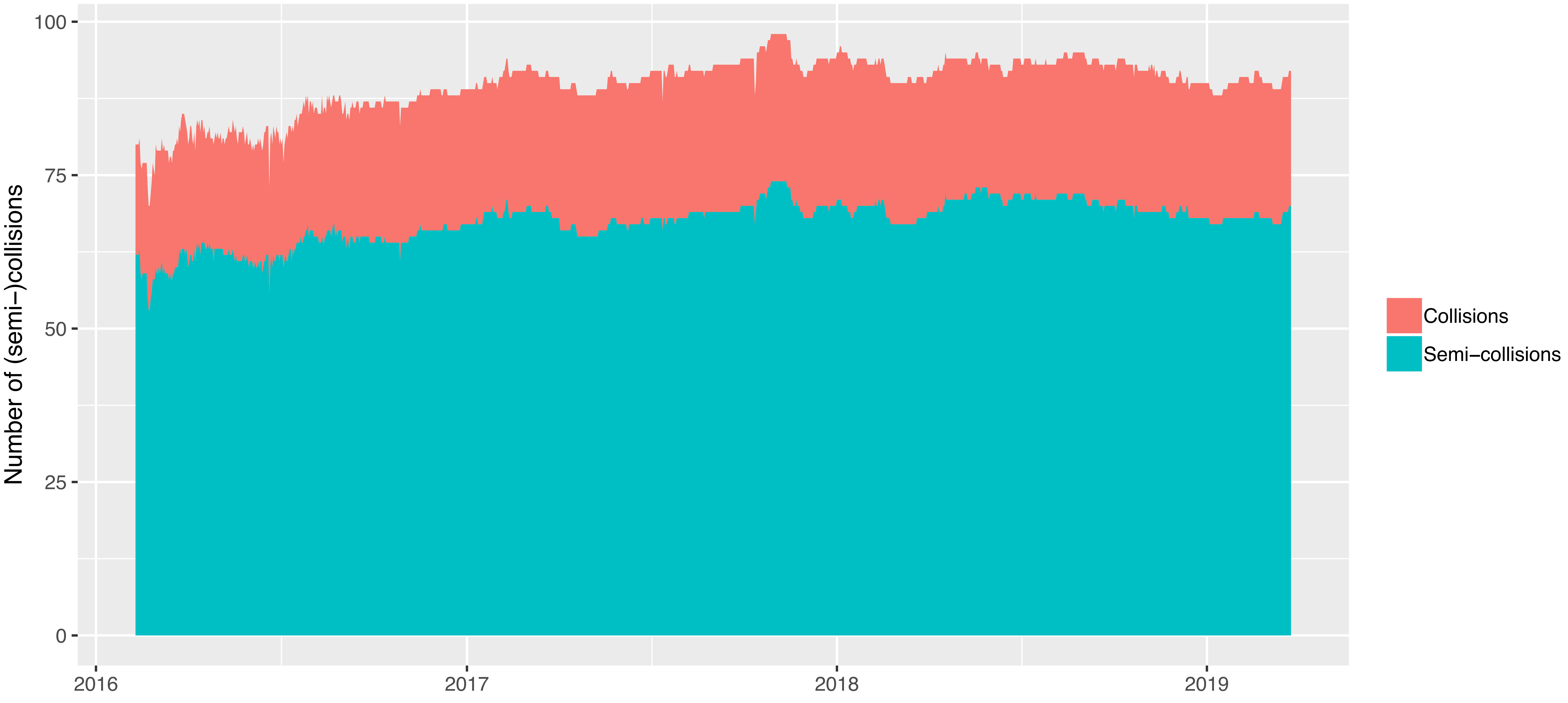
CRC16

.nl RSA CRC16 distribution



NLNETLABS

Is it actually better?



NLNETLABS

Why is CRC16 not better?

- Random uniform (-like) distribution means; any keytag is equally likely
- Due to the Generalised Birthday Paradox, the likelihood of a collision increases with the number of keys in a keyset

#keys in DNSKEY set	Theoretical probability	Observed probability
2	0,00153%	0,00165%
3	0,00458%	0,00465%
4	0,00915%	0,02349%
5 or more	0,01526%	0,00853%

Open questions

- The unspoken assumption for the empirical data is that implementations don't already prevent collisions from occurring
- For some, this assumption may not hold (e.g. LDNS, BIND)
- Can we fingerprint the crypto libraries used based on what we know from Roy's presentation and the distribution we observe?

Conclusion: what did we learn?

- At first glance, the original keytag algorithm seems suboptimal
- Yet choosing something "better" in terms of the likelihood of values occurring turns out to be worse!
- This is at least an interesting lesson in protocol design; without really meaning to, the writers of the DNSSEC RFCs picked a "better" algorithm
- If we literally translate a Dutch phrase for this, they were: "Unknowingly capable" ;-)

Operator and implementer advice

- Operators: if your domain is important and likely to be queried (very) frequently, then make sure you have no keytag collisions in your DNSKEY set -- the probabilities show that the chances of requiring more than one extra key generation is vanishingly small
- Implementers: consider (optionally) checking DNSKEY sets for keytag collisions and regenerate key(s) if a collision occurs
- Arguably: follow Postel's Law!

NLNETLABS

Thank you! Questions?

nl.linkedin.com/in/rolandvanrijswijk

@reseauxsansfil

roland@nlnetlabs.nl