What if encrypted communications are not as secure as we think?

Enrico Branca

CanSecWest

March 17, 2017

The plan

1. Get cryptographic keys from open data sources.
2. Decode the key material and keep only key-related data.
3. Parse all the data and produce statistics.
4. Identify which keys are vulnerable to attacks.
5. Find if any key can be broken.
6. Get insights and see what can be done next

Why we keep only key-related data?

- We focus only on key generation and this means dealing only with numbers

- Not interested in finding who or what made a mistake

- Not willing to deal with legal or privacy related issues associated with massive data correlation (EU privacy laws)

Get lots of crypto keys

- Internet-Wide Scan Data Repository

Website: https://scans.io

Downloaded all files that may contain crypto keys

- Suricata IDS/IPS to collect keys from services
  - Rule: *alert tls any any -> any any (msg:"No Alert TLS Store"; tls.subject:"CN="; tls.store; noalert; classtype:policy-violation; sid:5216010; rev:3;)*

- Custom parser to get PGP keys from SKS key dumps

Sample the crypto keys

We start with: HTTPS – POP3S – SMTPS – IMAPS – SSH - VPN

Collected from 2012 to end 2016.

From a total dataset of more than 500 million keys:

- Sample 250 million keys.
- Convert keys to PEM (asci armoured) files.
- Remove duplicate certificates, keys, sessions.

We remain with **74.220.631** certificates.

Decode the certificates

For each certificate we keep **only**:

- Public Key Algorithm
- Signature Algorithm
- Key Size
- Modulus
- Exponent

And for each certificate we generated its ‘thumbprint’:

- sha1 hash of file containing SSL certificate in DER format

Data parsing

1. Convert keys to PEM armoured ASCII
2. Convert to DER then calculate thumbprint
2. If duplicate is found discard the certificate
3. Convert DER back to PEM then to ASCII
6. Process Data

Public Key Algorithms

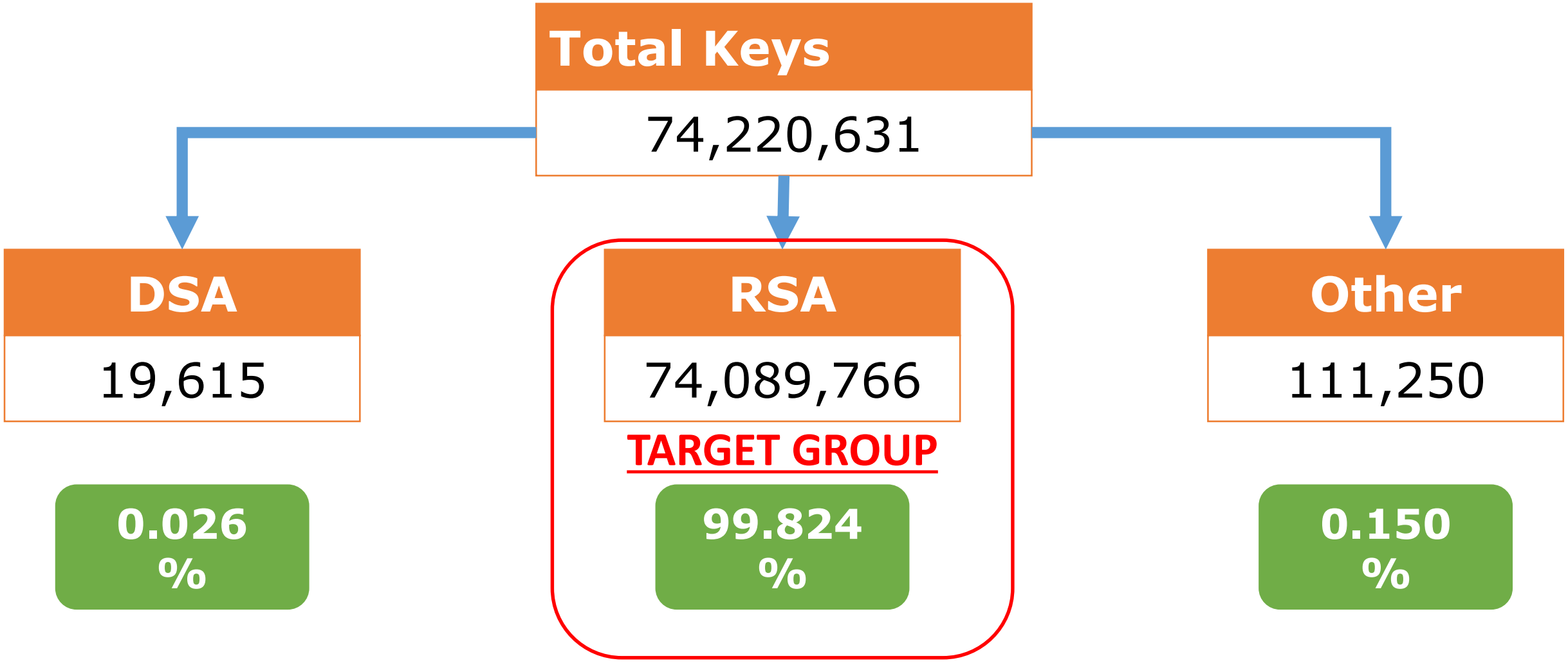
Public Key Algorithm	Algorithm Identifier	Keys Found	Percentage	O.I.D
RSA	rsa	3	0.00004 %	2.5.8.1.1
	rsaencryption	74.089.763	99.82368 %	1.2.840.113549.1.1.1
DSA	dsaencryption-old	3	0.00004 %	1.3.14.3.2.12
	dsaencryption	19.612	0.02642 %	1.2.840.10040.4.1
Other	NULL	283	0.00038 %	???
	id-ecpublickey	110.967	0.14951 %	1.2.840.10045.2.1

Top Signature Algorithms

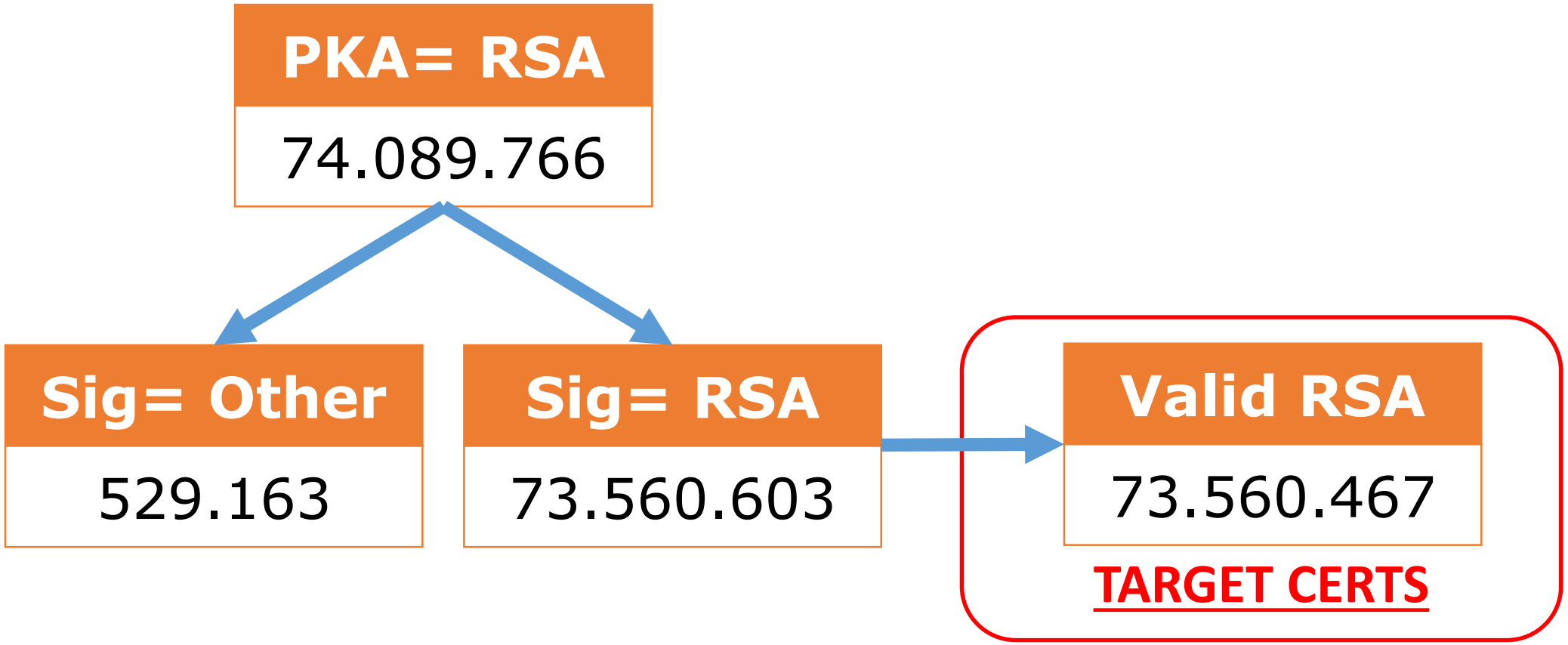
RSA - Signature Algorithms (TOP 5)	Count	%
sha1withrsaencryption	56.143.606	76.3229
md5withrsaencryption	12.151.179	16.5186
sha256withrsaencryption	5.109.937	6.9466
sha1withrsa	95.745	0.1302
sha512withrsaencryption	47.451	0.0645

DSA - Signature Algorithms (TOP 5)	Count	%
dsawithsha1	18911	96.5241
sha1withrsaencryption	537	2.7409
md5withrsaencryption	64	0.3267
dsa_with_sha256	59	0.3011
sha256withrsaencryption	10	0.0510

Public Key Algorithms



Target Certificates



We are analysing **ONLY** certificates using **RSA**.

RSA keys to test

From all the PEM files we exclude any certificate that uses unknown types of public-key algorithm

Certificates using RSA algorithm = 74.089.766

- using RSA Signature = 73.560.603
  - in non standard certificates = 136
  - in standard certificates = **73.560.467**

The number of RSA keys that will be tested is **73.560.467**

RSA key tests

In total we have **73.560.467** keys to test.

Each key will be tested for the following conditions:

1. Is the **public key** (modulus/exponent) unique?
2. Is the **modulus** unique?
3. Is the **modulus** a product of two large primes?
4. Is the **exponent** valid?

For each question we will count how many keys are impacted.

Is the public-key unique?

RSA public key: the pair of values (n, e), where 'n' is the modulus and 'e' is the exponent.

- Distinct Certificates: 73.560.467
- Individual public-keys: 47.725.709
- Distinct public keys: 45.208.601
- Certificates using same public-key: 25.834.758 **(38.54 %)**

Certificates can have different metadata but can also share the same pare of values (n, e).

Is the modulus unique?

RSA modulus: the product of two large primes ‘p’ and ‘q’

- Individual public-keys: 47.725.709
- Individual moduli: 45.990.010
- Distinct moduli: 43.459.140
- keys using same moduli: 2.530.870 **(5.50 %)**

Keys can have different modulus-exponent combinations (public-key) but share the same modulus.

Is the modulus unique?

keys using same modulus: 2.530.870 **(5.50 %)**

In theory there keys should always be unique, no two keys should **EVER** share the same modulus (no collision).

Keys can have the same modulus only when both have used, in the key generation, the same values for primes ‘p’ and ‘q’.

When this happens both keys should be considered insecure.

RSA key tests

Certificates using same public-key: 25.834.758 **(38.54 %)**

keys using same modulus: 2.530.870 **(5.50 %)**

First two tests seems suggest that not only there is a large portion of keys that share the same set of prime numbers, but also that the large amount of shared moduli cannot be entirely associated with broken RSA implementations.

Different systems, at different times, generated the same set of primes resulting in key collisions.

Modulus divisors

Is the **modulus** a product of two large primes?

For this test we are going to check if any of the moduli shares a prime with any other moduli.

Assume a key X has a modulus created from prime A and B.

Assume a key Y has a modulus created from prime A and C.

(As mentioned before this, in theory, should NEVER happen)

We want to check if any two keys, for example key X and Y, share any prime/divisor (in this case they share prime A).

Modulus divisors

To test for common divisors we are going to use the software provided by the team of the project “factorable.net”.

Site: https://factorable.net/

Software: “Fast pairwise GCD computation: **fastgcd-1.0.tar.gz**”

https://factorable.net/resources.html

For this test the set has been divided in chunks and each chunk has been tested for the presence of shared divisors.

Modulus divisors

- Individual public-keys: 47.725.709
- Distinct moduli: 3.459.140
- Moduli sharing a divisor: 169.709 **(0.39 %)**

Found in keys smaller than 2048 bits:167.543 (0.3855 %)
- Impacting how many distinct certificates: 756.529

Found in keys between 2048 and 4095 bits: 2.102 (0.0048 %)
- Impacting how many distinct certificates: 2.319

Found in keys bigger than 4095 bits: 64 (0.00015 %)
- Impacting how many distinct certificates: 64

Modulus divisors

- Certificates using same public-key: 25.834.758 **(38.54 %)**
- keys using same modulus: 2.530.870 **(5.50 %)**
- Moduli sharing a divisor: 169.709 **(0.39 %)**
  - Distinct certificates impacted: 758.912 (1.031 % of all certs)

A large portion of keys use the same modulus or has at least one divisor in common with other moduli, condition that should not be associated with a broken RSA implementation.

Different systems at different times can generate the same set of primes resulting in key collisions.

Modulus small primes

For this test we are going to test the assumption that the modulus is a product of two large primes.

To do so we check that the modulus cannot be divided by any of the first 10.000 primes (small primes).

- Individual public-keys: 47.725.709
- Distinct moduli: 43.459.140
- Moduli divisible by small primes: 7.912 **(0.182 %)**
  - Distinct certificates impacted: 9.098 (0.0124 % of all certs)

Modulus small primes

- Certificates using same public-key: 25.834.758 **(38.54 %)**
- keys using same modulus: 2.530.870 **(5.50 %)**

Moduli sharing a divisor: 169.709 **(0.39 %)**

- Distinct certificates impacted: 758.912 (1.031 % of all certs)

Moduli divisible by small primes: 7.912 **(0.182 %)**

- Distinct certificates impacted: 9.098 (0.0124 % of all certs)

The presence of moduli that are divisible by small primes can suggest the presence of broken RSA implementations, as no software should generate a key without this validation.

RSA Key exponent

In theory any RSA implementation should validate exponent values but if we check the top 20 values something is wrong.

Exponent Value	Count	Exponent Value	Count
65537	72.407.807	**11**	922
3	812.692	**19**	280
35	231.051	**37**	200
17	79.860	**13**	174
4097	8.142	**65535**	85
47	7.424	**35353**	40
5	4.905	**1**	**28**
7	2.643	**44611**	23
59	2.239	**65543**	22
257	1.510	**65539**	19

Exponent value of “1” should not be used, EVER.

RSA Key exponent

The enciphered text is the same as the message text, no encryption is happening even if RSA crypto is used, and no error message will be generated by either parties.

Any key that uses exponent ‘1’ should be considered BROKEN and regenerated using proper exponent values.

- Individual public-keys: 47.725.709
- Public-key with exponent ‘1’: 28 **(0.00006 %)**
  - Distinct certificates impacted: 28 (0.00004 % of all certs)

RSA Key exponent

Real Example:

https://docs.saltstack.com/en/latest/topics/releases/0.15.1.html#rsa-key-generation-fault

release: 2013-05-08

RSA KEY GENERATION FAULT

RSA key generation was done incorrectly, leading to very insecure keys.

It is recommended to regenerate all RSA keys.

…

This issue affects all known versions of Salt.

Specific commits:

https://github.com/saltstack/salt/commit/5dd304276ba5745ec21fc1e6686a0b28da29e6fc

```
-    gen = RSA.gen_key(keysize, 1, callback=lambda x, y, z: None)
+     gen = RSA.gen_key(keysize, 65537, callback=lambda x, y, z: None)
```

Test Summary

- Certificates using same public-key: 25.834.758 **(38.54 %)**
- keys using same modulus: 2.530.870 **(5.50 %)**
- Moduli sharing a divisor: 169.709 **(0.39 %)**
  - Distinct certificates impacted: 758.912 (1.031 % of all certs)
- Moduli divisible by small primes: 7.912 **(0.182 %)**
  - Distinct certificates impacted: 9.098 (0.0124 % of all certs)
- Public-key with exponent '1': 28 **(0.00006 %)**
  - Distinct certificates impacted: 28 (0.00004 % of all certs)

Test Summary

Should we panic? **NO**, test results are in line with the results of previous research, most notably:

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

Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman

USENIX Security Symposium (USENIX Security), August 2012

https://factorable.net/weakkeys12.extended.pdf

February 2012 - Ron was wrong, Whit is right

Arjen K. Lenstra(1), James P. Hughes(2), Maxime Augier(1), Joppe W. Bos(1), Thorsten Kleinjung(1), and Christophe Wachter(1)

(1)EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland

(2)Self, Palo Alto, CA, USA

https://eprint.iacr.org/2012/064.pdf

What more to check?

We tested for issues that can arise after a key has been generated, but what about what happens before?

If we assume that the amount of keys found to be broken is not entirely related to broken RSA implementations, then we can assume that ‘chance’ has something to do with it.

In theory no two keys should EVER share a prime or be the same, but we have seen that is not the case, so we can try to generate a collision.

Keys with shared divisors

To generate key collisions nothing special is needed, we just have to generate LOTS of keys.

For this purpose two systems have been configured.

- PC-1: AMD Opteron, Centos 7 x64, no FIPS mode, HAVEGE (PRNG)
- PC-2: AMD Opteron, Centos 7 x64, FIPS mode, HAVEGE (PRNG)

(PRNG: pseudo random number generator)

Each system is set to generate 100.000.000 keys, 3 times, using a local installation of openssl (version 1.0.1e-60)

Keys with shared divisors

All keys have been tested, in chunks of 100.000 keys, against the keys in the dataset to check for shared primes.

PC-1: AMD Opteron 6166 HE, Centos 7 x64, no FIPS mode

- Run 1: found shared divisor at 7.000.000 keys
- Run 2: found shared divisor at 1.500.000 keys
- Run 3: found shared divisor at 54.700.000 keys

PC-2: AMD Opteron 6166 HE, Centos 7 x64, FIPS mode

- Run 1: found shared divisor at 47.300.000 keys
- Run 2: no shared divisor found
- Run 3: found shared divisor at 21.600.000 keys

Keys with shared divisors

Both systems have no hardware RNG, entropy levels were always above 2048bit, key generated in 3 distinct processes.

The system in FIPS mode was reporting, on average, 4 key errors every 100.000 keys due to keys failing primality tests.

The system not configured in FIPS mode produced, on average, keys with a shared prime more often than the system configured in FIPS mode, and never reported primality errors.

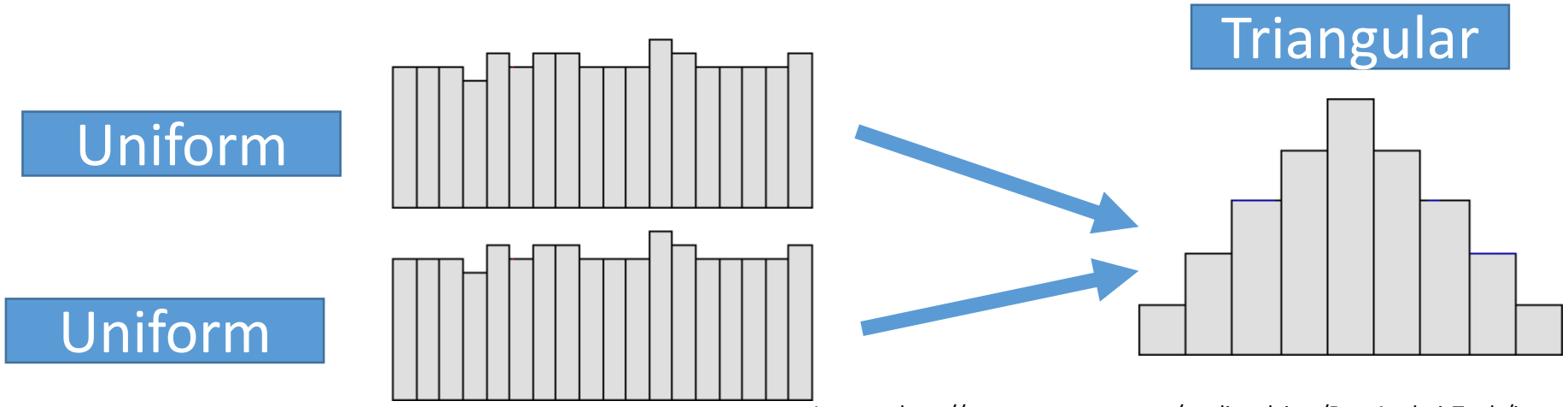
Keys with shared divisors

The systems I used were low end (really low end).

- CPU without AES-NI instructions
- No internal hardware random number generator
- No external hardware random number generator
- No pause between key generation
- Low system activity

Even if conditions were not optimal, it was indeed possible to generate keys that share primes with keys found in the wild (with patience…).

What more to check?

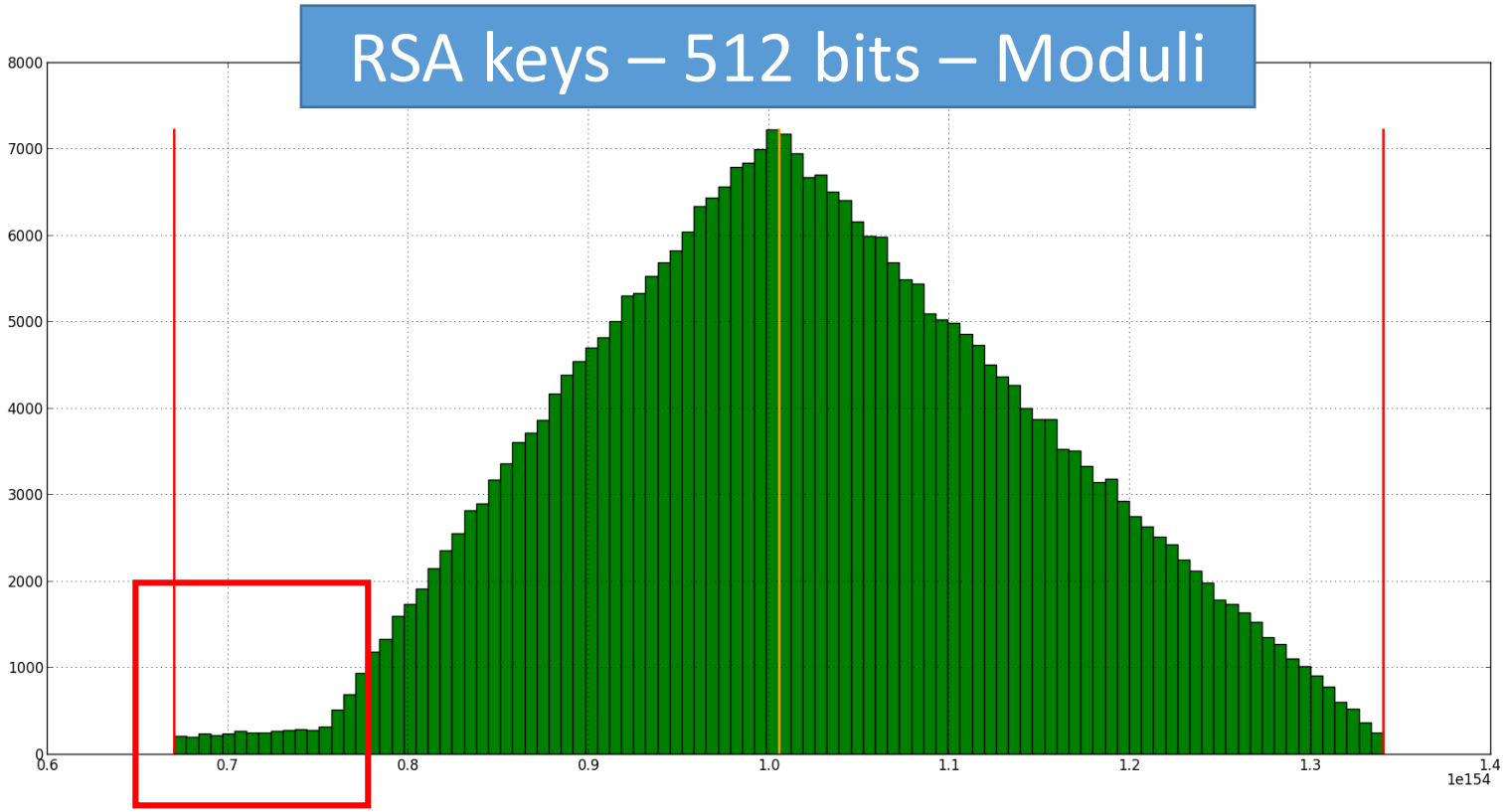


Images : http://www.pqsystems.com/qualityadvisor/DataAnalysisTools/interpretation/histogram_shape.php

To create an RSA key we need to generate large primes at random, over time the prime generation should, for a large number of iterations, be able to generate primes that cover the entire key space (uniform distribution).

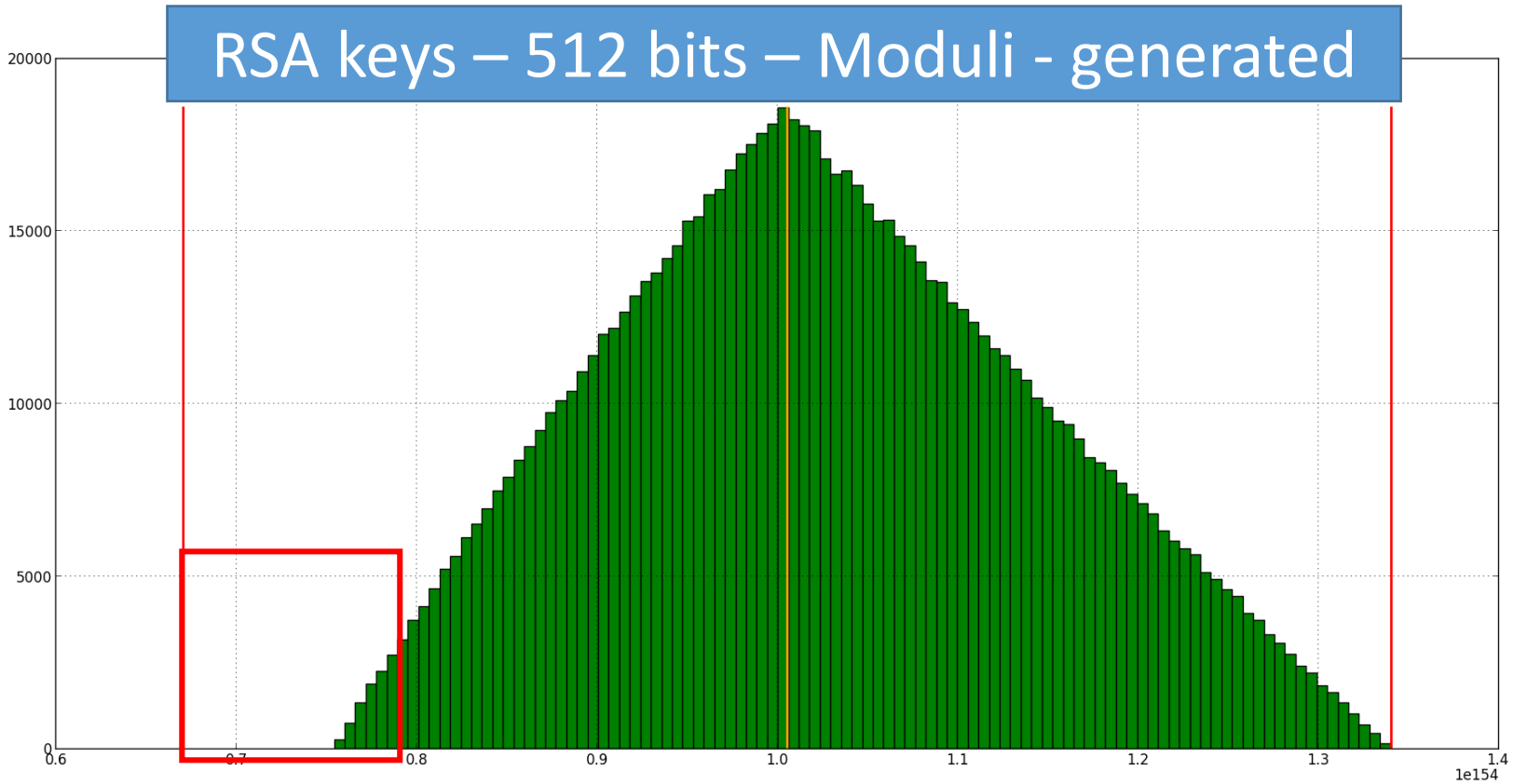
If all is fine, the product of these two group of primes should produce a triangular distribution covering the entire key space.

Key space



In reality if we plot the distribution of the moduli we have collected, we don't have a triangular distribution, and we have something different at the beginning of the numeric interval.

Key space



We can see the difference by generating some RSA keys using something like openssl or gnutls, to see that keys are not covering the whole numeric interval as there is a gap.

Key space

From an analysis of the moduli collected from the wild and the one generated locally, a difference was found.

If the system that generates the RSA has FIPS mode enabled:

- Generated keys cover the whole numeric interval

If the system is not configured in FIPS mode:

- Generated keys always have the first 4 bits set as ‘1001’
- A gap is always present at the beginning of the numeric space

Key space

Testing with openssl, gnutls, ssh-keygen, gpg and gpg2, the difference in the numeric interval used for key generation will be present.

The reason seems to be related to an optimization in the generation of primes.

If both prime ‘p’ and ‘q’ are generated by setting first, second and last bit to 1, the resulting modulus will have the first four bits set as ‘1001’.

Key space

A beautiful explanation that cover key generation and the logic behind has been created by professor “Avinash Kak” (Purdue University, USA) and is available online.

Computer and Network Security

https://engineering.purdue.edu/kak/compsec/NewLectures/

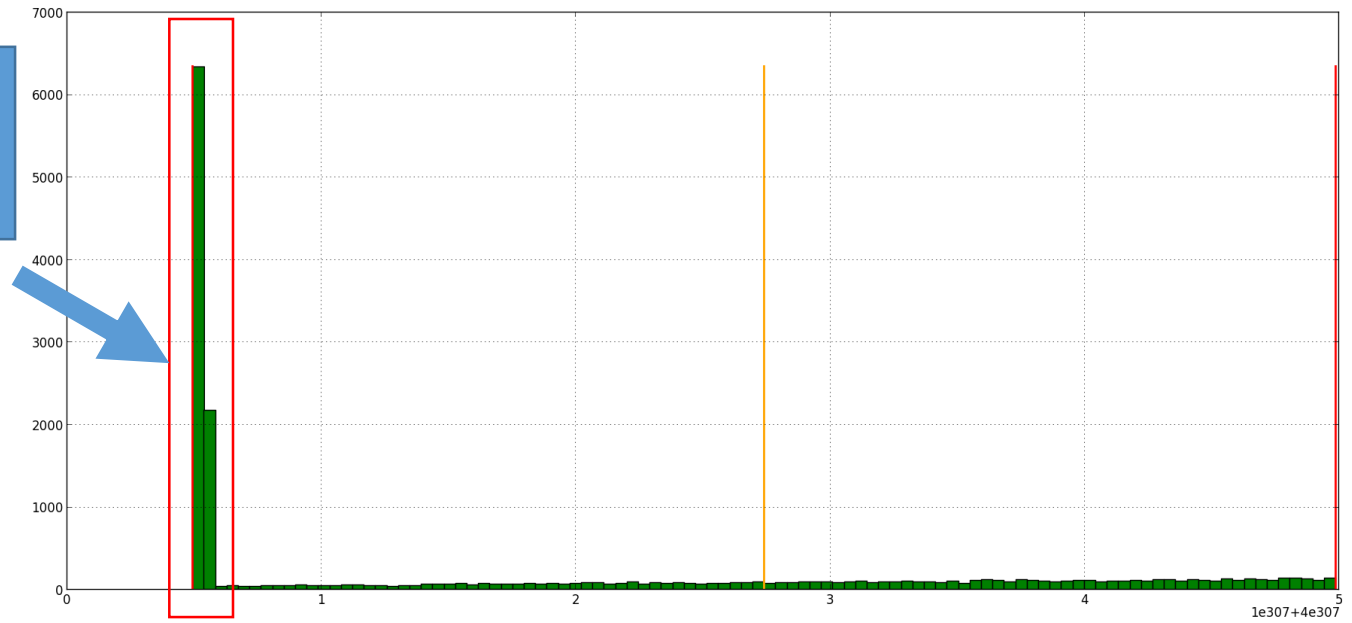
Public-Key Cryptography and the RSA Algorithm

https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf

Key space

RSA keys – 1023 bits – Moduli

Broken RSA implementation



Why is this information useful?

Because it is possible to identify broken implementations without performing complicated math tests.

Communications using TLS

In theory we should have no duplicated key, no key divisible by small primes, no key sharing primes with other keys and no key using exponent ‘1’.

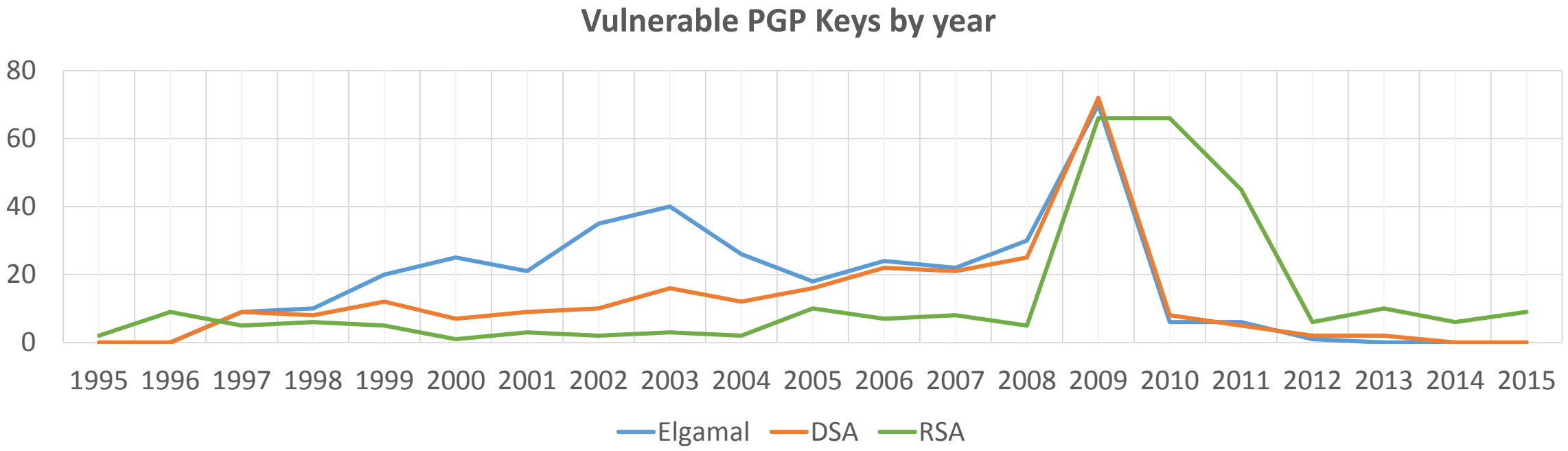
With long term key collection, the use of temporary TLS keys does not help as the more keys are generated the higher is the probability or finding a key collision or a key sharing a prime.

Using larger keys does not help either as resourceful attacker can just generate keys locally and perform offline attacks later.

PGP key sets

Does it change something if we use PGP public-key crypto?

The answer is no, as we have the same issues as with TLS crypto used in web communications.



PGP key sets

	Elgamal	DSA	RSA		Elgamal	DSA	RSA
1995	0	0	2	**2006**	24	22	7
1996	0	0	9	**2007**	22	21	8
1997	9	9	5	**2008**	30	25	5
1998	10	8	6	**2009**	70	72	66
1999	20	12	5	**2010**	6	8	66
2000	25	7	1	**2011**	6	5	45
2001	21	9	3	**2012**	1	2	6
2002	35	10	2	**2013**	0	2	10
2003	40	16	3	**2014**	0	0	6
2004	26	12	2	**2015**	0	0	9
2005	18	16	10	**2016**	0	0	0

A dataset of 8.932.412 PGP keys has been tested for all the conditions described in the previous tests and we found a total of **895** (0.01 %) broken keys that should be regenerated.

PGP key sets

PGP keys with shared primes	
Elgamal	0
DSA	256
RSA	261

PGP Keys with small divisors	
Elgamal	362
DSA	262
RSA	266

Test for keys with a shared divisor was positive for both DSA and RSA keys.

Test for keys divisible by small factors was positive for all key types.

Test for RSA keys with exponent ‘1’ returned 10 bad keys.

PGP key sets

Test regarding the distribution of moduli in the numeric interval confirmed that RSA keys generated for PGP systems follow the same distribution as the keys used for TLS.

And key size does not help in preventing key compromise.

Modulus bit length	
Bits	Count
768	2
1024	179
2048	216
3072	7
4096	134
16384	1

Top 5 Divisors	
Divisor	Count
641	434
6700417	387
3	287
5	126
2	109

Thanks

Achim, OWASP
Johanna, OWASP
Jim, OWASP
Glenn, OWASP

Mozilla Security Team

CanSecWest

Thank You

Enrico Branca

enrico.branca@awebof.info

enrico.branca@owasp.org