"Major key alert!" Anomalous keys in Tor relays

George Kadianakis*
The Tor Project

Claudia V. Roberts*
Princeton University

Laura M. Roberts*
Princeton University

Philipp Winter*
Princeton University

ABSTRACT

In its more than ten years of existence, the Tor network has seen hundreds of thousands of relays come and go. Each relay maintains several RSA keys, amounting to millions of keys, all archived by The Tor Project. In this paper, we analyze 3.7 million RSA public keys of Tor relays. We (*i*) check if any relays share prime factors or moduli, (*ii*) identify relays that use non-standard exponents, and (*iii*) characterize malicious relays that we discovered in the first two steps. Our experiments revealed that ten relays shared moduli, and 3,557 relays—almost all part of a research project—shared prime factors, allowing adversaries to reconstruct private keys. We further discovered 122 relays that used non-standard RSA exponents, presumably in an attempt to attack onion services. By simulating how onion services are positioned in Tor's distributed hash table, we identified four onion services that were likely targeted by these malicious relays.

Keywords

Tor, RSA, onion service, cryptography, factorization

1. INTRODUCTION

Having seen hundreds of thousands of relays come and go over the last decade, the Tor network is one of the largest volunteer-run anonymity networks. To implement onion routing, all the relays maintain several RSA key pairs, the most important of which are a medium-term key that rotates occasionally and a long-term key that ideally never changes. Most relays run The Tor Project's reference C implementation on Linux systems, but some run third-party implementations or run on constrained systems such as Raspberry Pis which raises the question of whether these machines are managing to generate safe keys upon bootstrapping. Past work has investigated the safety of keys in TLS and SSH servers [1] and nation-wide databases [2], as well as POP3S, IMAPS, and SMTPS servers [3]. In this work, we study the Tor network.

Relays with weak cryptographic keys can pose a significant threat to Tor users. The exact impact depends on the type of key that is vulnerable. In the best case, an attacker only manages to compromise the TLS layer that protects Tor cells, which are also encrypted. In the worst case, an attacker compromises a relay's long-term "identity key," allowing her to impersonate the relay. To better protect Tor users, we need methods to quickly find relays with vulnerable keys and remove them from the network before adversaries can exploit them.

Drawing on a publicly-archived dataset of 3.7 million RSA public keys [4], we set out to analyze these keys for weaknesses and anomalies: we looked for shared prime factors, shared moduli, and non-standard RSA exponents. To our surprise, we found more than 3,000 keys with shared prime factors, most belonging to a 2013 research project [5]. Ten relays in our dataset shared a modulus, suggesting manual interference with the key generation process. Finally, we discovered 122 relays whose RSA exponent differed from Tor's hard-coded exponent. We believe that most of these relays were meant to manipulate Tor's distributed hash table (DHT), presumably in an attempt to attack onion services as we discuss in Section 5.4. To learn more, we implemented a tool that simulates how onion services are placed on the DHT, revealing four onion services that were likely targeted.

The entities responsible for the incidents we uncovered are as diverse as the incidents themselves: researchers, developers, and actual adversaries were all involved in generating key anomalies. By looking for information that relays had in common, such as similar nicknames, IP address blocks, uptimes, and port numbers, we were able to group the relays we discovered into clusters that were likely operated by the same entities.

We publish all our source code and data, allowing third parties such as The Tor Project to continuously check the keys of new relays and alert developers if any of these keys are vulnerable or non-standard.[1] Tor developers can then take early action and remove these re-

*All four authors contributed substantially and share first authorship. The names are ordered alphabetically.

[1]Our project page is available online at https://nymity.ch/anomalous-tor-keys/.

lays from the network before adversaries get the chance to take advantage of them. In summary, we make the following two main contributions:

- We analyze a dataset consisting of 3.7 million RSA public keys for weak and non-standard keys, revealing thousands of affected keys.
- We characterize the relays we discovered, show that many were likely operated by a single entity, and uncover four onion services that were likely targeted

The rest of this paper details our project. In Section 2, we provide background information, followed by Section 3 where we discuss related work. In Section 4, we describe our method, and Section 5 presents our results. We discuss our work in Section 6 and conclude in Section 7.

2. BACKGROUND

We now provide brief background on the RSA cryptosystem, how the Tor network employs RSA, and how onion services are implemented in the Tor network.

2.1 The RSA cryptosystem

The RSA public key cryptosystem uses key pairs consisting of a public encryption key and a privately held decryption key [6]. The encryption key, or "RSA public key," is comprised of a pair of positive integers: an exponent e and a modulus N. The modulus N is the product of two large, random prime numbers p and q. The corresponding decryption key, or "RSA private key," is comprised of the positive integer pair d and N, where $N = pq$ and $d = e^{-1} \bmod (p-1)(q-1)$. The decryption exponent d is efficient to compute if e and the factorization of N are known.

The security of RSA rests upon the difficulty of factorizing N into its prime factors p and q. While factorizing N is impractical given sufficiently large prime factors, the greatest common divisor (GCD) of *two moduli* can be computed in mere microseconds. Consider two distinct RSA moduli $N_1 = pq_1$ and $N_2 = pq_2$ that share the prime factor p. An attacker could quickly and easily compute the GCD of N_1 and N_2, which will be p, and then divide the moduli by p to determine q_1 and q_2, thus compromising the private key of both key pairs. Therefore, it is crucial that both p and q are determined using a strong random number generator with a unique seed.

Even though the naive GCD algorithm is very efficient, our dataset consists of more than 3.7 million keys and naively computing the GCD of every pair would take more than three years of computation (assuming 15 μs per pair). Instead, we use the fast pairwise GCD algorithm by Bernstein [7] which can perform the computation at hand in just a few minutes.

2.2 The Tor network

The Tor network is among the most popular tools for digital privacy and anonymity. As of March 2017, the Tor network consists of almost 7,000 volunteer-run relays [8].

Each of these relays maintains RSA, Curve25519, and Ed25519 key pairs to authenticate and protect client traffic [9, § 1.1]. In this work, we analyze the RSA keys. We leave the analysis of the other key types for future work. Each Tor relay has the following three 1024-bit RSA keys:

Identity key Relays have a long-term identity key that they use only to sign documents and certificates. Relays are frequently referred to by their fingerprints, which are hashes over their identity keys. The compromise of an identity key would allow an attacker to impersonate a relay by publishing spoofed descriptors signed by the forged identity key.

Onion key Relays use medium-term onion keys to decrypt cells when circuits are created. The onion key is only used in the Tor Authentication Protocol that is now superseded by the ntor handshake [10]. A compromised onion key allows the attacker to read the content of cells until the key pair rotates.

Connection key The short-term connection keys protect the connection between relays using TLS and are rotated at least once a day. The TLS connection provides defense in depth. If compromised, an attacker is able to see the encrypted cells that are exchanged between Tor relays.

In our work we consider the identity keys and onion keys that each relay has because the Tor Project has been archiving the public part of the identity and onion keys for more than ten years, allowing us to draw on a rich dataset [4]. The Tor Project does not archive the connection keys because they have short-term use and are not found in the network consensus or relay descriptors.

2.3 Onion services

In addition to client anonymity, the Tor network allows operators to set up anonymous servers, typically called "onion services."[2] The so-called "hidden service directories," or "HSDirs," are a subset of all Tor relays and comprise a distributed hash table (DHT) that stores the information necessary for a client to connect to an onion service. These HSDirs are a particularly attractive target to adversaries because they get to learn

[2]The term "hidden services" was used in the past but has been discontinued.

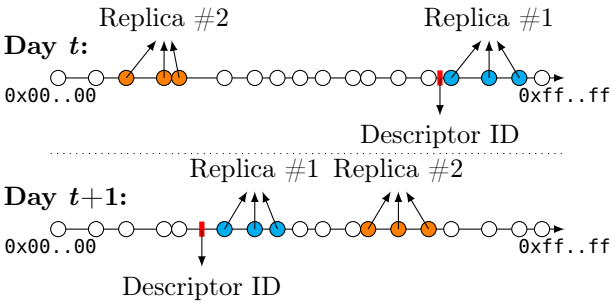

Figure 1: Each day, an onion service places its descriptor ID at a pseudorandom location in Tor's "hash ring," which consists of all HSDir relays (illustrated as circles).

about onion services that are set up in the Tor network. An onion service's position in the DHT is governed by the following equations:

$$\begin{aligned} \textit{secret-id-part} &= \textit{SHA-1}(\textit{time-period} \,| \\ &\qquad \textit{descriptor-cookie} \,| \\ &\qquad \textit{replica}) \\ \textit{descriptor-id} &= \textit{SHA-1}(\textit{permanent-id} \,| \\ &\qquad \textit{secret-id-part}) \end{aligned} \tag{1}$$

Secret-id-part depends on three variables: *time-period* represents the number of days since the Unix epoch; *descriptor-cookie* is typically unused and hence empty; and *replica* is set to both the values 0 and 1, resulting in two hashes for *secret-id-part*. The concatenation of both *permanent-id* (the onion service's hashed public key) and *secret-id-part* is hashed, resulting in *descriptor-id*, which determines the position in the DHT. When arranging all HSDirs by their fingerprint in ascending order, the three immediate HSDir neighbors in the positive direction constitute the first replica while the second replica is at another, pseudorandom location, as shown in Figure 1. The onion service's descriptor ID and hence its two replicas changes every day when *time-period* increments.

3. RELATED WORK

In 2012, Lenstra et al. [11] and Heninger et al. [1] independently analyzed a large set of RSA public keys used for TLS, SSH, and PGP. Both groups discovered that many keys shared prime factors, allowing an attacker to efficiently compute the corresponding private keys. The researchers showed that the root cause was weak randomness at the time of key generation: Many Internet-connected devices lack entropy sources, resulting in predictable keys.

One year later, Bernstein et al. [2] showed similar flaws in Taiwan's national "Citizen Digital Certificate"

First key published	2005-12
Last key published	2016-12
Number of relays (by IP address)	1,083,805
Number of onion keys	3,174,859
Number of identity keys	588,945
Total number of public keys	3,763,804

Table 1: An overview of our RSA public key dataset.

database. Among more than two million 1024-bit RSA keys, the authors discovered 184 vulnerable keys, 103 of which shared prime factors. The authors could break the remaining 81 keys by applying a Coppersmith-type partial-key-recovery attack [12, 13].

Valenta et al. [14] optimized popular implementations for integer factorization, allowing them to factor 512-bit RSA public keys on Amazon EC2 in under four hours for only $75. The authors then moved on to survey the RSA key sizes that are used in popular protocols such as HTTPS, DNSSEC, and SSH, discovering numerous keys of only 512 bits.

Most recently, in 2016, Hastings et al. [3] revisited the problem of weak keys and investigated how many such keys were still on the Internet four years after the initial studies. The authors found that many vendors and device owners never patched their vulnerable devices. To make matters worse, the number of vulnerable devices has actually *increased* since 2012.

4. METHOD

In this section, we discuss how we drew on a publicly-available dataset (Section 4.1) and used Heninger and Halderman's fastgcd [15] tool to analyze the public keys that we extracted from this dataset (Section 4.2).

4.1 Data collection

The Tor Project archives data about Tor relays on its CollecTor platform [4], allowing researchers to learn what relays were online at any point in the past. Drawing on this data source, we compiled a set of RSA keys by downloading all server descriptors from December 2005 to December 2016 and extracting the identity and onion keys with the Stem Python library [16]. Table 1 provides an overview of the resulting dataset—approximately 200 GB of unzipped data. Our 3.7 million public keys span eleven years and were created on one million IP addresses.

4.2 Finding vulnerable keys

To detect weak, potentially factorable keys in the Tor network, we used Heninger and Halderman's tool, fastgcd [15], which takes as input a set of moduli from public keys and then computes the pair-wise greatest

common divisor of these moduli. Fastgcd's C implementation is based on a quasilinear-time algorithm for factoring a set of integers into their co-primes. We used the PyCrypto library [17] to turn Tor's PKCS#1-padded, PEM-encoded keys into fastgcd's expected format, which is hex-encoded moduli. Running fastgcd over our dataset took less than 20 minutes on a machine with dual, eight-core 2.8 GHz Intel Xeon E5 2680 v2 processors with 256 GB of RAM.

Fastgcd benefits from having a moduli pool as large as possible because it allows the algorithm to draw on a larger factor base to use on each key [1]. To that end, we reached out to Heninger's group at the University of Pennsylvania, and they graciously augmented their 129 million key dataset with our 3.6 million keys and subsequently searched for shared factors. The number of Tor weak keys did not go up, but this experiment gave us more confidence that we had not missed weak keys.

5. RESULTS

We present our results in four parts, starting with shared prime factors (Section 5.1), followed by shared moduli (Section 5.2), unusual exponents (Section 5.3), and finally targeted onion services (Section 5.4).

5.1 Shared prime factors

Among all 588,945 identity keys, fastgcd found that 3,557 (0.6%) moduli share prime factors. We believe that 3,555 of these keys were all controlled by a single research group, and upon contacting the authors of the Security & Privacy 2013 paper entitled "Trawling for Tor hidden services" [5], we received confirmation that these relays indeed were run by their research group. The authors informed us that the weak keys were caused by a shortcoming of their key generation utility. The issue stemmed from the fact that their tool first generated thousands of prime numbers and then computed multiple moduli using combinations of those prime numbers in a greedy fashion without ensuring that the same primes were not reused.

Because of the following shared properties, we are confident that all relays were operated by the same group:

1. All relays were online either between November 11, 2012 and November 16, 2012 or between January 14, 2013 and February 6, 2013, suggesting two separate experiments. We verified this by checking how long the relays stayed in the Tor network consensus. The Tor consensus is updated hourly and documents which relays are available at a particular time. This data is archived by The Tor Project and is made publicly available on the CollecTor platform [4].
2. All relays exhibited a predictable port assignment scheme. In particular, we observed ports {7003, 7007, . . . , 7043, 7047} and {8003, 8007, . . . , 8043, 8047}.
3. Except for two machines that were located in Russia and Luxembourg, all machines were hosted in Amazon's EC2 address space. All machines except the one located in Luxembourg used Tor version 0.2.2.37.
4. All physical machines had multiple fingerprints. 1,321 of these 3,557 relays were previously characterized by Winter et al. [18, § 5.1].

The remaining two keys belonged to a relay named "DesasterBlaster," whose origins we could not determine. Its router descriptor indicates that the relay has been hosted on a MIPS machine which might suggest an embedded device with a weak random number generator:

```
router DesasterBlaster 62.226.55.122 9001 0 0
platform Tor 0.2.2.13-alpha on Linux mips
```

To further investigate, we checked whether the relay "DesasterBlaster" shares prime factors with any other relays. It appears that the relay has rotated multiple identity keys, and it only shares prime factors with its own keys.

5.2 Shared moduli

In addition to finding shared prime factors, we discovered relays that share a *modulus*, giving them the ability to calculate each other's private keys. With p, q, and each other's e's in hand, the two parties can compute each other's decryption exponent d, at which point both parties now know the private decryption keys.

Table 2 shows these ten relays with shared moduli, clustered into four groups. The table shows the relays' truncated, four-byte fingerprint, IP addresses, and RSA exponents. Note that the Tor client hard-codes the RSA exponent to 65,537 [9, § 0.3], a recommended value that is resistant to attacks against low public exponents [19, § 4]. Any other value indicates non-standard key generation. All IP addresses were hosted by OVH, a popular French hosting provider, and some of the IP addresses hosted two relays, as our color coding indicates. Finally, each group shared a four- or five-digit prefix in their fingerprints. We believe that a single attacker controlled all these relays with the intention to manipulate the distributed hash table that powers onion services [5]—the shared fingerprint prefix is an indication. Because the modulus is identical, we suspect that the attackers iterated over the relays' RSA exponents to come up with the shared prefix. The Tor Project informed us that it discovered and blocked these relays in August 2014 when they first came online.

Short fingerprint	IP address	Exponent
838A296A	188.165.164.163	1,854,629
838A305F	188.165.26.13	718,645
838A71E2	178.32.143.175	220,955
2249EB42	188.165.26.13	4,510,659
2249EC78	178.32.143.175	1,074,365
E1EFA388	188.165.3.63	18,177
E1EF8985	188.165.138.181	546,019
E1EF9EB8	5.39.122.66	73,389
410BA17E	188.165.138.181	1,979,465
410BB962	5.39.122.66	341,785

Table 2: Four groups of relays that have a shared modulus. All relays further share a fingerprint prefix in groups of two or three, presumably to manipulate Tor's distributed hash table.

5.3 Unusual exponents

The Tor source code hard-codes the public RSA exponent to 65,537, which is best practice [19, § 4]. Interested in non-standard key generation, we checked if our dataset featured relays with different exponents. Non-standard exponents may indicate that a relay was after a specific fingerprint in order to position itself in Tor's hash ring.[3] To obtain a fingerprint with a given prefix, an adversary has to modify the underlying key material p, q, and e until they result in the desired prefix. Repeated modification of e is significantly more efficient than modifying p or q because it is costly to verify if a large number is prime. Leveraging this method, the tool Scallion [20] generates vanity onion service domains by iterating over the service's public exponent.

Among all of our 3.7 million keys, 122 possessed an exponent other than 65,537.[4] One relay had both non-standard identity *and* onion key exponents while all remaining relays only had non-standard identity key exponents. Ten of these relays further had a shared modulus, which we discuss in Section 5.2. Assuming that these relays positioned themselves in the hash ring to attack an onion service, we wanted to find out what onion services they targeted. One can identify the victims by first compiling a comprehensive list of onion services and then determining each service's position in the hash ring at the time the malicious HSDirs were online.

5.4 Identifying targeted onion services

We obtained a list of onion services by augmenting the list of the Ahmia search engine [21] with services that we discovered via Google searches and by contacting researchers who have done similar work [22]. We ended up with a list of 17,198 onion services that were online at some point in time. Next, we developed a tool that takes as input our list of onion services and the malicious HSDirs we discovered.[5] The tool then determines all descriptors these onion services ever generated and checks if any HSDir shared five or more hex digits in its fingerprint prefix with the onion service's descriptor.

It is difficult to identify all targeted onion services with certainty. First, our list of onion services does not tell us when a service was online. Second, an HSDir could be responsible for an onion service simply by chance, rather than on purpose, resulting in a false positive. Third, our list of onion services is not exhaustive, so we are bound to miss some potential victims. Our tool identified the following four onion services for which we have strong evidence that they were purposely targeted. Because none of these four services seem to have been intended for private use, we are comfortable publishing them.

22u75kqyl666joi2.onion The service appears to be offline today, so we were unable to see for ourselves what it hosted. According to cached index pages we found online, the onion service used to host a technology-focused forum in Chinese. A subset of relays from Table 4 targeted the onion service on both August 14 and 15, 2015 by providing nine out of the total of twelve responsible HSDirs.

n3q7l52nfpm77vnf.onion As of February 2017, the service is still online, hosting the "Marxists Internet Archive," an online archive of literature.[6] Figure 2 shows a screenshot of the service's index page. A subset of relays from Table 3 targeted the onion service from November 27 to December 4, 2016. The malicious HSDirs acted inconsistently, occasionally targeting only one replica.

silkroadvb5piz3r.onion The onion service used to host the Silk Road marketplace, whose predominant use was a market for narcotics. The service was targeted by a subset of relays from Table 6, from May 21 to June 3, 2013. The HSDirs were part of a measurement experiment that resulted in a blog post [23].

thehub7gqe43miyc.onion The onion service used to host a discussion forum, "The Hub," focused on darknet markets. A subset of relays from Table 4 targeted both of The Hub's replicas from August 22, 2015.

[3] A different approach to detecting relays that position themselves in the hash ring is to determine how often they change their fingerprints. [18, § 4.3.3]

[4] We list all relays in detail in Appendix A.

[5] Both the tool and our list of onion services are available online at https://nymity.ch/anomalous-tor-keys/.

[6] The onion service seems to be identical to the website https://www.marxists.org.

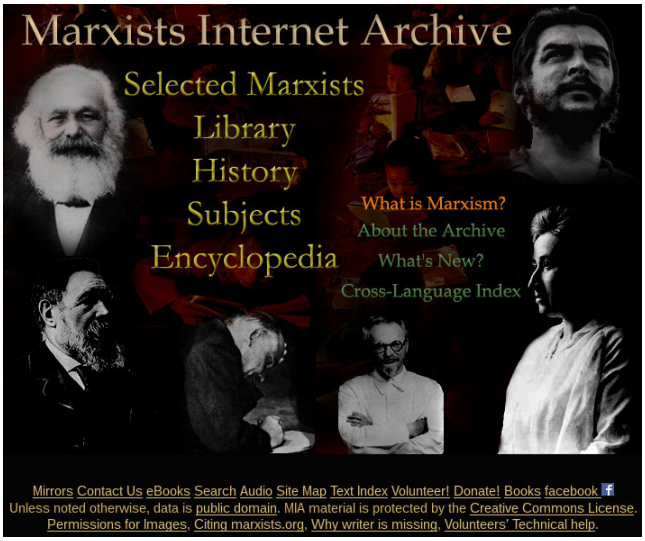

Figure 2: A screenshot of the index page of the onion service `n3q7l52nfpm77vnf.onion`, taken on February 13, 2017.

It is not clear what the HSDirs did once they controlled the replicas of the onion services they targeted. The HSDirs could have counted the number of client requests, refused to serve the onion service's descriptor to take it offline, or correlate client requests with guard relay traffic in order to deanonymize onion service visitors [24].

6. DISCUSSION

Implications of anomalous Tor keys.

As touched on earlier in Section 2.2, the main use of the identity key in Tor is to sign the relay's descriptor, which includes various information about the relay, e.g., its IP address, contact information, etc. Relays publish their public identity keys in their descriptor. The network consensus acts as the public key infrastructure of Tor. Signed by the directory authorities whose public keys are hard-coded in Tor's source code, the network consensus points to the descriptors of each Tor relay that is currently online. If an attacker were to break the identity key of a relay (as we demonstrated), she could start signing descriptors in the relay's name and publishing them. The adversary could publish whatever information she wanted in the descriptor, e.g. her own IP address, keys, etc., in order to fool Tor clients.

Preventing non-standard exponents.

Recall that the Tor reference implementation hard-codes its public RSA exponent to 65,537 [9, § 0.3]. The Tor Project could prevent non-standard exponents by having the directory authorities reject relays whose descriptors have an RSA exponent other than 65,537, thus slowing down the search for fingerprint prefixes. Adversaries would then have to iterate over the primes p or q instead of the exponent, rendering the search more computationally expensive. Given that we discovered only 122 unusual exponents in over ten years of data, we believe that rejecting non-standard exponents is a viable defense in depth.

Analyzing onion service public keys.

Future work should shed light on the public keys of onion services. Onion services have an incentive to modify their fingerprints to make them both recognizable and easier to remember. Facebook, for example, was lucky to obtain the easy-to-remember onion domain `facebookcorewwwi.onion` [25]. The tool Scallion assists onion service operators in creating such vanity domains [20]. The implications of vanity domains on usability and security are still poorly understood [26]. Unlike the public keys of relays, onion service keys are not archived, so a study would have to begin with actively fetching onion service keys.

In vivo Tor research.

Caution must be taken when conducting research using the live Tor network. Section 5.1 showed how a small mistake in key generation led to many vulnerable Tor relays. To keep its users safe, The Tor Project has recently launched a research safety board whose aim is to assist researchers in safely conducting Tor measurement studies [27]. This may entail running experiments in private Tor networks that are controlled by the researchers, or using network simulators such as Shadow [28].

7. CONCLUSION

Previous research has studied the problem of weak RSA keys in different systems, and we wondered if there might be weak keys in the Tor network too that have the potential to compromise Tor users' safety. Thus, the goal of our work was to look for weak and anomalous keys in Tor and investigate their origins in order to address the problem. We achieved our goal by gathering all the archived RSA keys used in Tor since 2005 and examining them for common prime factors as previous work had done. Additionally, we looked for keys that shared the same moduli and for keys that had non-standard public exponents.

We found indications that entities had purposely created anomalous keys in order to attack Tor's onion services. We also found that researchers inadvertently created weak keys while conducting experiments on Tor. Our work demonstrates that the presence of weak and anomalous RSA keys in Tor is often a sign of malicious activity that should be paid attention to, and indeed, our findings motivated The Tor Project to develop

scripts to look for non-standard RSA exponents, which go against Tor's specification.

Acknowledgements

We want to thank Nadia Heninger and Josh Fried for augmenting their database with our moduli and attempting to find factors in them. We also want to thank Ralf-Philipp Weinmann, Ivan Pustogarov, Alex Biryukov from the Trawling research team and Donncha O'Cearbhaill from The Tor Project for providing us with additional information that helped us in our analysis of the weak keys. Finally, we want to thank Edward W. Felten for providing valuable feedback on an earlier version of our paper. This research was supported in part by the Center for Information Technology Policy at Princeton University and the National Science Foundation Awards CNS-1540066 and CNS-1602399.

References

[1] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices". In: *USENIX Security*. USENIX, 2012. URL: https://factorable.net/weakkeys12.extended.pdf (cit. on pp. 1, 3, 4).

[2] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. "Factoring RSA keys from certified smart cards: Coppersmith in the wild". In: *ASIACRYPT*. Springer, 2013. URL: https://smartfacts.cr.yp.to/smartfacts-20130916.pdf (cit. on pp. 1, 3).

[3] Marcella Hastings, Joshua Fried, and Nadia Heninger. "Weak Keys Remain Widespread in Network Devices". In: *IMC*. ACM, 2016. URL: https://www.cis.upenn.edu/~nadiah/papers/weak-keys/weak-keys.pdf (cit. on pp. 1, 3).

[4] The Tor Project. *CollecTor*. URL: https://collector.torproject.org (cit. on pp. 1–4).

[5] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *Security and Privacy*. IEEE, 2013. URL: http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf (cit. on pp. 1, 4).

[6] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. URL: https://people.csail.mit.edu/rivest/Rsapaper.pdf (cit. on p. 2).

[7] Daniel J. Bernstein. *How to find smooth parts of integers*. 2004. URL: https://cr.yp.to/factorization/smoothparts-20040510.pdf (cit. on p. 2).

[8] The Tor Project. *Servers – Tor metrics*. URL: https://metrics.torproject.org/networksize.html (cit. on p. 2).

[9] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. URL: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt (cit. on pp. 2, 4, 6).

[10] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. "Anonymity and one-way authentication in key exchange protocols". In: *Designs, Codes and Cryptography* 67.2 (2013), pp. 245–269. URL: https://nymity.ch/anomalous-tor-keys/pdf/Goldberg2013a.pdf (cit. on p. 2).

[11] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. "Public Keys". In: *CRYPTO*. Springer, 2012. URL: https://nymity.ch/anomalous-tor-keys/pdf/Lenstra2012a.pdf (cit. on p. 3).

[12] Don Coppersmith. "Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known". In: *EUROCRYPT*. Springer, 1996, pp. 178–189. URL: https://nymity.ch/anomalous-tor-keys/pdf/Coppersmith1996a.pdf (cit. on p. 3).

[13] Don Coppersmith. "Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities". In: *Journal of Cryptology* 10.4 (1997), pp. 233–260. URL: https://www.di.ens.fr/~fouque/ens-rennes/coppersmith.pdf (cit. on p. 3).

[14] Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, and Nadia Heninger. "Factoring as a Service". In: *Financial Cryptography*. ACM, 2016. URL: https://eprint.iacr.org/2015/1000.pdf (cit. on p. 3).

[15] Nadia Heninger and J. Alex Halderman. *fastgcd*. URL: https://factorable.net/fastgcd-1.0.tar.gz (cit. on p. 3).

[16] Damian Johnson. *Stem Docs*. URL: https://stem.torproject.org (cit. on p. 3).

[17] Dwayne Litzenberger. *PyCrypto – The Python Cryptography Toolkit*. URL: https://www.dlitz.net/software/pycrypto/ (cit. on p. 4).

[18] Philipp Winter, Roya Ensafi, Karsten Loesing, and Nick Feamster. "Identifying and characterizing Sybils in the Tor network". In: *USENIX Security*. USENIX, 2016. URL: https://nymity.ch/sybilhunting/pdf/sybilhunting-sec16.pdf (cit. on pp. 4, 5).

[19] Dan Boneh. "Twenty Years of Attacks on the RSA Cryptosystem". In: *Notices of the American Mathematical Society* 46.2 (1999). URL: http://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf (cit. on pp. 4, 5).

[20] Eric Swanson. *Scallion – GPU-based Onion Hash generator*. URL: https://github.com/lachesis/scallion (cit. on pp. 5, 6).

[21] Juha Nurmi. *Ahmia – Search Tor Hidden Services*. URL: https://ahmia.fi/onions/ (cit. on p. 5).

[22] Srdjan Matic, Platon Kotzias, and Juan Caballero. "CARONTE: Detecting Location Leaks for Deanonymizing Tor Hidden Services". In: *CCS*. ACM, 2015. URL: https://software.imdea.org/~juanca/papers/caronte_ccs15.pdf (cit. on p. 5).

[23] Donncha O'Cearbhaill. *Trawling Tor Hidden Service – Mapping the DHT*. 2013. URL: https://donncha.is/2013/05/trawling-tor-hidden-services/ (cit. on pp. 5, 9).

[24] Roger Dingledine. *Tor security advisory: "relay early" traffic confirmation attack*. July 2014. URL: https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack/ (cit. on p. 6).

[25] Roger Dingledine. *Facebook brute forcing hidden services*. Oct. 2014. URL: https://lists.torproject.org/pipermail/tor-talk/2014-October/035412.html (cit. on p. 6).

[26] Philipp Winter. *Are vanity onion domains a good idea?* Oct. 2015. URL: https://moderncrypto.org/mail-archive/messaging/2015/001928.html (cit. on p. 6).

[27] The Tor Project. *Tor Research Safety Board*. URL: https://research.torproject.org/safetyboard.html (cit. on p. 6).

[28] Rob Jansen and Nicholas Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". In: *NDSS*. Internet Society, 2012. URL: http://www.robgjansen.com/publications/shadow-ndss2012.pdf (cit. on p. 6).

APPENDIX

A. RELAYS WITH UNUSUAL EXPONENTS

Section 5.3 discussed 122 relays whose public RSA exponents differed from Tor's hard-coded 65,537. Tables 3 to 7 together list these 122 relays. In particular, we show each relay's truncated four-byte fingerprint, nickname, IP address, and public RSA exponent. Many of these 122 relays exhibit partial fingerprint collisions (highlighted in gray), presumably to obtain a specific position in Tor's hash ring.

Fingerprint	Nickname	IP address	Exponent
0759DB6F	schedule	150.95.142.186	229,995,765
0759DB6F	dengue	133.130.122.105	954,990,355
0759DB6F	Almond	150.95.143.40	626,061,519
09A54DB0	epitome	45.76.128.202	1,678,745,769
09A54DB3	euler	163.44.153.177	1,108,857,575
09A54DB3	niche	163.44.113.116	1,745,865,531
264EA128	guanguan	163.44.167.158	220,632,011
264EA12B	shift	45.32.105.219	132,219,129
264EA12B	youknow	45.76.113.171	1,798,607,207
26597E61	cocoa	104.238.191.196	1,379,799,401
26597E62	victuals	45.63.31.68	32,576,079
26597E62	gauge	128.199.217.46	131,552,757
2D148D3D	bury	150.95.143.40	266,644,331
2D148D3E	zebra	150.95.142.186	241,957,811
2E25D844	direct	45.76.113.171	17,621,801
2E25D845	myapple	45.32.105.219	1,044,674,813
2E25D846	comeback	104.238.165.134	689,575,361
2E25D846	restart	107.191.47.191	7,561,442,929
2E25D847	bluesky	163.44.167.158	1,394,906,153
32E71B90	zebra	150.95.142.186	2,843,570,039
3968276F	pizza	104.238.165.134	2,684,303,857
5404DC16	schedule	150.95.142.186	2,703,265,981
59E415D4	police	159.203.254.220	1,931,398,351
59E415D4	ethyl	178.62.245.218	256,546,481
59E415D7	porsche	162.243.164.151	473,803,601
6761D2BC	salou	128.199.59.20	112,319,419
6761D2BC	caker	95.85.24.170	1,865,406,823
6761D2BE	chauvinism	138.197.200.144	900,199,685
71363B81	dessert	133.130.122.105	68,863,333
71363B83	bury	150.95.143.40	285,526,435
7CDB224E	coupon	138.68.154.127	47,223,567
7CDB224F	capris	45.63.31.68	263,219,313
905CC77C	sour	45.63.20.92	2,746,565,943
A0E83AA0	bowl	163.44.113.116	253,332,051
A0E83AA1	truth	163.44.153.177	2,109,850,307
A0E83AA2	develop	45.76.128.202	672,985,205
D271E035	victuals	45.63.31.68	2,724,738,511
DE2702F4	pizza	104.238.165.134	184,370,845
DE2702F4	sour	45.63.20.92	488,020,267
DE2702F4	genre	107.191.47.191	1,933,228,135
EBF154D8	quinoa	45.63.20.92	12,242,179,243
EBF154D9	quote	107.191.47.191	310,366,091
EBF154DA	monk	104.238.165.134	588,652,521
F5079E2D	Almond	150.95.143.40	226,820,943
F5079E2D	schedule	150.95.142.186	1,561,992,849
F5079E2E	dengue	133.130.122.105	713,479,109

Table 3: A seemingly-related cluster of 46 relays that went online in November 2016.

Fingerprint	Nickname	IP address	Exponent
2DFDC2BA	America0823	98.158.107.61	773,161,427
325CAC0A	America0816	98.158.107.61	320,165,239
325CAC0A	Britain0816	176.56.180.162	739,276,705
325CAC0B	NSingapore0816	98.158.108.55	1,178,204,265
37D5E568	America02	172.82.166.25	17,733,657,127
37D5E568	America03	172.82.166.23	4,794,439,555
37D5E568	HongKong02	103.37.1.128	1,781,593,029
816FEE14	NSingapore0823	98.158.108.55	994,060,627
816FEE15	America0823	98.158.107.61	686,629,695
816FEE16	Britain0823	176.56.180.162	254,007,767
90645A9B	Britain0816	176.56.180.162	2,236,171,913
A5C59B3D	NSingapore0817	98.158.108.55	558,846,429
A5C59B3F	America0817	98.158.107.61	1,792,382,161
A5C59B3F	Britain0817	176.56.180.162	342,841,829
BC79109C	HongKong02	103.37.1.128	711,846,827
BC79109C	HongKong01	103.37.1.129	5,113,334,139
E5E77830	Hongkong0817	98.158.109.184	1,323,419,807
E5E77831	Singapore0817	98.158.108.58	1,897,601,619
E5E77832	NewYork0817	98.158.100.102	577,107,197
F6961286	Hongkong0823	98.158.109.184	7,219,330,549
F6961286	NewYork0823	98.158.100.102	9,757,773,095
F6961286	Singapore0823	98.158.108.58	25,384,989,341
F6961286	Singapore0829	98.158.108.58	35,994,097,691
FA0BDA0E	HongKong03	103.37.1.38	817,251,045
FA0BDA0E	HongKong01	103.37.1.129	1,961,759,239
FA0BDA0E	America01	104.247.221.69	15,741,212,315
FA256740	Singapore0816	98.158.108.58	264,349,871
FA256741	Hongkong0816	98.158.109.184	751,944,245
FA256743	NewYork0816	98.158.100.102	623,959,941

Table 4: A seemingly-related cluster of 29 relays that went online in between August and October 2015.

Fingerprint	Nickname	IP address	Exponent
2249E809	Unnamed	188.165.164.163	4,611,991
2249EB42	Unnamed	188.165.26.13	4,510,659
2249EC78	Unnamed	178.32.143.175	1,074,365
410BA17E	Unnamed	188.165.138.181	1,979,465
410BB962	Unnamed	5.39.122.66	341,785
410BCDC1	Unnamed	188.165.3.63	667,165
838A296A	Unnamed	188.165.164.163	1,854,629
838A305F	Unnamed	188.165.26.13	718,645
838A71E2	Unnamed	178.32.143.175	220,955
E1EF8985	Unnamed	188.165.138.181	546,019
E1EF9EB8	Unnamed	5.39.122.66	73,389
E1EFA388	Unnamed	188.165.3.63	18,177

Table 5: A seemingly-related cluster of twelve relays that went online in August 2014.

Fingerprint	Nickname	IP address	Exponent
0E483850	totalimpact	176.31.119.209	31,329,919
1272B9A3	totalimpact	176.31.119.209	61,205,961
1AC4DA51	totalimpact	176.31.119.209	66,377,251
38FCC604	totalimpact	176.31.119.209	18,143,565
51FC178D	totalimpact	176.31.119.209	20,752,399
52820131	totalimpact	94.23.163.185	11,999,175
59529817	totalimpact	94.23.163.185	34,481,629
6B6B1DBC	Unnamed	176.31.119.209	14,876,273
712CA45A	totalimpact	94.23.163.185	31,093,293
8E1600DD	curiousDHTRelay	178.238.140.54	13,103,911
B81B43C0	totalimpact	176.31.119.209	61,669,007
BC89A92F	totalimpact	176.31.119.209	2,069,559
BCB33286	totalimpact	176.31.119.209	22,077,013
BE6FFBA5	nobbey	128.199.34.37	35,843,231
BE6FFBDA	nobbey	128.199.34.37	39,143,575
DE15299D	totalimpact	94.23.163.185	41,454,569
E038CACD	Unnamed	176.31.119.209	8,141,357
E038CBE7	Unnamed	176.31.119.209	19,231,119
E9F25C48	totalimpact	176.31.119.209	36,446,771
F52D83EF	totalimpact	94.23.163.185	49,938,863
FF0BF54F	totalimpact	176.31.119.209	45,295,357

Table 6: A cluster of 21 relays that a Tor developer used to run experiments [23].

Fingerprint	Nickname	IP address	Exponent
13225B74	venlafaxin	217.20.118.12	35
739758B1	snooker	66.57.36.178	1,497,483
739758B1	snooker	66.57.36.178	418,223
63A8D7D5	cheezeblaster	68.4.225.95	106,533
8808B01B	sginne	80.223.97.109	860,345,903
A574E18E	zeratulemperator	178.32.58.139	345,084,801
D4073DE6	caker000	208.92.93.82	47,124,401
FC360E29	DaeshTorBlock	201.175.4.83	176,127
36043967	CAundisclosedio	159.203.25.149	15,514,301
8FF3ED2E	conoha1	133.130.126.71	1,173,127,861
8FF3ED2F	conoha2	133.130.125.198	724,859,007
CF45F18F	USundisclosedio	173.236.251.43	11,753,943
C722E7E9	Tokyo01	133.130.126.71	2,022,320,527
2D04AF7C	shallot	45.32.185.135	1,341,827

Table 7: Fourteen relays that do not seem to have been part of a cluster. The relay groups "snooker," and "conoha1" and "conoha2" do seem related, though.