"Major key alert!" Anomalous keys in Tor relays

George Kadianakis[1⋆], Claudia V. Roberts[2], Laura M. Roberts[2], and Philipp Winter[2]

[1] The Tor Project
[2] 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, (*iii*) characterize malicious relays that we discovered in the first two steps, and (*iv*) develop a tool that can determine what onion services fell prey to said malicious relays. 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 targeted by these malicious relays. Our work provides both The Tor Project and onion service operators with tools to identify misconfigured and malicious Tor relays to stop attacks before they pose a threat to Tor users.

Keywords: Tor, RSA, cryptography, factorization, onion service

1 Introduction

Having seen hundreds of thousands of relays come and go over the last decade, the Tor network is the largest volunteer-run anonymity network. 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 dedicated Linux systems, but some run third-party implementations or operate on constrained systems such as Raspberry Pis which raises the question of whether these machines managed to generate safe keys upon bootstrapping. Past work has investigated the safety of keys in TLS and SSH servers [15], in nation-wide databases [3], as well as in POP3S, IMAPS, and

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

SMTPS servers [14]. In this work, we study the Tor network and pay particular attention to Tor-specific aspects such as onion services.

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 protect Tor users, we need methods to 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 [30], 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 [4]. 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. Most of these relays were meant to manipulate Tor's distributed hash table (DHT) in an attempt to attack onion services as we discuss in Section 5.4. To learn more, we implemented a tool—itos[3]—that simulates how onion services are placed on the DHT, revealing four onion services that were targeted by some of these malicious relays. Onion service operators can use our tool to monitor their services' security; e.g., a newspaper can make sure that its SecureDrop deployment—which uses onion services—is safe [11].

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, shedding light on the anatomy of real-world attacks against Tor.

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.[4] Tor developers can then take early action and remove these relays from the network before adversaries get the chance to take advantage of them. In summary, we make the following three 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.

[3] The name is an acronym for "identifying targeted onion services."
[4] Our project page is available online at `https://nymity.ch/anomalous-tor-keys/`.

– Given a set of malicious Tor relays that served as "hidden service directories," we develop and implement a method that can reveal what onion services these relays were targeting.

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 [27]. 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 [2] 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 December 2017, the Tor network consists of around 7,000 volunteer-run

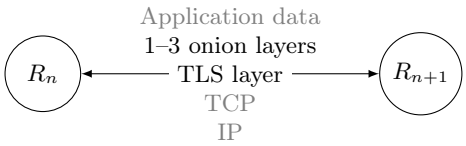

Fig. 1. The protocol stack between two Tor relays R_n and R_n+1. The lowest encryption layer is a TLS connection that contains one (between the middle and exit relay) to three (between the client and guard relay) onion layers. The onion layers protect the application data that the client is sending.

relays [31]. Each hour, information about all relays[5] is summarized in the *network consensus*, which is used by clients to bootstrap a connection to the Tor network. The network consensus is produced by eight geographically-distributed *directory authorities*, machines run by individuals trusted by The Tor Project. For each relay in the consensus, there is a pointer to its *descriptor*, which contains additional, relay-specific information such as cryptographic keys.

Each of the ∼7,000 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 is a hash over their identity key. The compromise of an identity key would allow an attacker to impersonate a relay by publishing spoofed descriptors signed by the compromised 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 [13]. A compromised onion key allows the attacker to read the content of cells until the key pair rotates, which happens after 28 days [32, § 3.4.1]. However, the onion key layer is protected by a TLS layer (see Figure 1) that an attacker may have to find a way around.

Connection key The short-term connection keys protect the connection between relays using TLS and are rotated at least once a day [9, § 1.1]. The TLS connection provides defense in depth as shown in Figure 1. 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 [30]. The

[5] This information includes IP addresses, ports, version numbers, and cryptographic information, just to name a few.

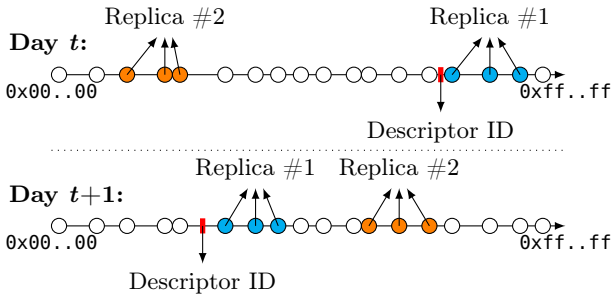

Fig. 2. Each day, an onion service places its descriptor ID at a pseudorandom location in Tor's "hash ring," which consists of all HSDir relays (illustrated as circles).

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."[6] 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 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:

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

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

[6] The term "hidden services" was used in the past but was discontinued, in part because onion services provide more than just "hiding" a web site.

direction constitute the first replica while the second replica is at another, pseudorandom location, as shown in Figure 2. 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. [20] and Heninger et al. [15] 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. [3] showed similar flaws in Taiwan's national "Citizen Digital Certificate" 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 [6,7].

Valenta et al. [36] 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. [14] 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. Surprisingly, 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 [16] 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 [30], 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 [19]. 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.

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

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

Our dataset also contains the keys of Tor's directory authorities. The authorities' keys are particularly sensitive: If an attacker were to compromise more than half of these keys, she could create a malicious network consensus—which could consist of attacker-controlled relays only—that would then be used by Tor clients. Therefore these keys are paramount to the security of the Tor network.

4.2 Finding vulnerable keys

To detect weak, potentially factorable keys in the Tor network, we used Heninger and Halderman's fastgcd [16] tool 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 [21] 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 access to a pool of moduli that is as large as possible because it allows the algorithm to draw on a larger factor base to use on each key [15]. 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 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" [4], we received confirmation that these relays were indeed run by their research group. The authors informed us that the weak keys were caused by a shortcoming in their key generation tool. 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 researchers:

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 [30].
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. [38, § 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. Unfortunately the relay did not have any contact information configured which is why we could not get in touch with its operator.

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 es 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. 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.

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 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 [5, § 4]. Any value other than 65,537 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 [4]—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.

5.3 Unusual exponents

Having accidentally found a number of relays with non-standard exponents in Section 5.2, we checked if our dataset featured more relays with exponents other than 65,537. Non-standard exponents may indicate that a relay was after a specific fingerprint in order to position itself in Tor's hash ring. To obtain a fingerprint with a given prefix, an adversary repeatedly has to modify any of the underlying key material p, q, or 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 [29] 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. 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 [25] with services that we discovered via Google searches and by contacting researchers who have done similar work [23]. 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.[7] The tool then calculates 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. We chose the threshold of five manually because it is unlikely to happen by chance yet easy to create a five-digit collision.

It is difficult to identify all targeted onion services because (*i*) our list of onion services does not tell us when a service was online, (*ii*) an HSDir could be responsible for an onion service simply by chance rather than on purpose, resulting in a false positive, and (*iii*) our list of onion services is not exhaustive, so we are bound to miss victims. Nevertheless our tool identified four onion services (see Table 3) for which we have strong evidence that they were purposely targeted. While HSDirs are frequently in the vicinity of an onion service's descriptor by accident, the probability of being in its vicinity for several days in a row or cover both replicas by chance is negligible. Table 4 shows all partial collisions in detail. 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 set of relays 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.[8] A set of

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

[8] The onion service seems to be identical to the website `https://www.marxists.org` (visited on 2017-05-09).

Table 3. The four onion services that were most likely targeted at some point. The second column indicates if only one or both replicas were attacked while the third column shows the duration of the attack.

Onion service	Replicas	Attack duration
`22u75kqyl666joi2.onion`	2	Two consecutive days
`n3q7l52nfpm77vnf.onion`	2	Six non-consecutive days
`silkroadvb5piz3r.onion`	1	Nine mostly consecutive days
`thehub7gqe43miyc.onion`	2	One day

relays 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 set of relays from May 21 to June 3, 2013. The HSDirs were part of a measurement experiment that resulted in a blog post [26].

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

Our data cannot provide insight into 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 as it was done by the CMU/SEI researchers in 2014 [8]. Since these attacks were short-lived we find it unlikely that they were meant to take offline the respective onion services.

6 Discussion

We now turn to the technical and ethical implications of our research, propose possible future work, and explain how the next generation of onion services will thwart DHT manipulation attacks.

6.1 Implications of anomalous Tor keys

Implications for the network 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 and contact information. 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. In other words, weak keys allow adversaries to obtain the affected relay's reputation which matters because Tor clients make routing decisions based on this reputation.

The Tor protocol's use of forward secrecy mitigates the potential harm of weak keys. Recall that a relay's long-lived identity keys are only used to sign data, so forward secrecy does not apply here. Onion keys, however, are used for decryption and encryption and are rotated by default every 28 days [32, § 3.4.1]. An attacker who manages to compromise a weak onion key is still faced with the underlying TLS layer, shown in Figure 1, which provides defense in depth. The Tor specification requires the keys for the TLS layer to be rotated at least once a day [9, § 1.1], making it difficult to get any use out of compromised onion keys.

Implications for Tor users To understand how Tor users are affected by weak keys we need to distinguish between *targeting* and *hoovering* adversaries.[9] The goal of targeting adversaries is to focus an attack on a small number of users among the large set of all Tor users. Generally speaking, weak keys can be problematic in a targeted setting if they allow an attacker to gain access to a Tor relay she would otherwise be unable to control. This can be the case if the attacker learned the targeted user's guard relay, and the guard happens to have weak keys. However, judging by our experimental results, the probability of an attacker's knowing a targeted user's guard relay *and* said guard relay's having vulnerable keys is very low.

Hoovering adversaries are opportunistic by definition and seek to deanonymize as many Tor users as possible. Recall that Tor clients use a long-lived guard relay as their first hop and two randomly chosen relays for the next two hops.[10] A single compromised relay is not necessarily harmful to users but weak keys can be a problem if a user happens to have a guard relay with weak keys *and* selects an exit relay that also has weak keys, allowing the hoovering adversary to deanonymize the circuit. Again, considering the low prevalence of weak keys and the ease with which The Tor Project could identify and block relays with weak keys, hoovering adversaries pose no significant threat.

6.2 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

[9] We here use Jaggard and Syverson's nomenclature of an adversary that either targets specific Tor users (targeting) or hoovers up all available data to deanonymize as many users as possible (hoovering) [17].

[10] We refer to these relays as randomly chosen for simplicity, but the path selection algorithm is more complicated.

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 computationally more expensive because of the cost of primality tests. 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.

6.3 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` [24]. The tool Scallion assists onion service operators in creating such vanity domains [29]. The implications of vanity domains on usability and security are still poorly understood [37]. 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.

6.4 Investigating the vulnerability of Tor relays to attacks on Diffie-Hellman

Recent work has demonstrated how Diffie-Hellman key exchange is vulnerable to attack [1,35,10]. Because Tor uses Diffie-Hellman, we decided to investigate how it might be affected by those findings. The Tor specification states that the implementation uses the Second Oakley Group for Diffie-Hellman, where the prime number is 1024 bits long [9, § 0.3]. To gather evidence of this usage, we performed an nmap scan on Tor relays using the ssl-dh-params script [12], which confirmed the Tor specification. The use of a 1024-bit prime is concerning because Adrian et al. [1] stated that "1024-bit discrete log may be within reach for state-level actors," and thus, they suggest a move to 2048 bits. The authors also mention that developers should move away from using 1024-bit RSA, as well, which Tor uses.

6.5 *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 [33]. This may entail running experiments in private Tor networks that are controlled by the researchers or using network simulators such as Shadow [18].

As for our own work, we were in close contact with Tor developers throughout our research effort and shared preliminary results as we progressed. Once we wrote up our findings in a technical report, we brought it to the broader Tor

community's attention by sending an email to the tor-dev mailing list [28]. On top of that, we adopted open science practices and wrote both our code and paper in the open, allowing interested parties to follow our progress easily.

6.6 The effect of next-generation onion services

As of December 2017, The Tor Project is testing the implementation of next-generation onion services [22]. In addition to stronger cryptographic primitives, the design fixes the issue of predicting an onion service's location in the hash ring by incorporating a random element. This element is produced by having the directory authorities agree on a random number once a day [34]. The random number is embedded in the consensus document and used by clients to fetch an onion service's descriptor. Attackers will no longer be able to attack onion services by positioning HSDirs in the hash ring; while they have several hours to compute a key pair that positions their HSDirs next to the onion service's descriptor (which is entirely feasible), it takes at least 96 hours to get the HSDir flag from the directory authorities [32, § 3.4.2], so attackers cannot get the flag in time. We expect this design change to disincentivize attackers from manipulating their keys to attack onion services.

7 Conclusion

Inspired by recent research that studied weak keys in deployed systems, we set out to investigate if the Tor network suffers from similar issues. To that end, we drew on a public archive containing cryptographic keys of Tor relays dating back to 2005, which we subsequently analyzed for weak RSA keys. We discovered that (*i*) ten relays shared an RSA modulus, (*ii*) 3,557 relays shared prime factors, and (*iii*) 122 relays used non-standard RSA exponents.

Having uncovered these anomalies, we then proceeded to characterize the affected relays, tracing back the issues to mostly harmless experiments run by academic researchers and hobbyists, but also to attackers that targeted Tor's distributed hash table which powers onion services. To learn more, we implemented a tool that can determine what onion services were attacked by a given set of malicious Tor relays, revealing four onion services that fell prey to these attacks.

The practical value of our work is twofold. First, our uncovering and characterizing of Tor relays with anomalous keys provides an anatomy of real-world attacks that The Tor Project can draw upon to improve its monitoring infrastructure for malicious Tor relays. Second, our work provides The Tor Project with tools to verify the RSA keys of freshly set up relays, making the network safer for its users. In addition, onion service operators can use our code to monitor their services and get notified if Tor relays make an effort to deanonymize their onion service. We believe that this is particularly useful for sensitive deployments such as SecureDrop instances.

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 by the Center for Information Technology Policy at Princeton University and the National Science Foundation Awards CNS-1540066, CNS-1602399, CNS-1111539, CNS-1314637, CNS-1520552, and CNS-1640548.

References

1. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: CCS. ACM (2015), https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf, (visited on 2017-09-22) (Cited on p. 13.)
2. Bernstein, D.J.: How to find smooth parts of integers (2004), https://cr.yp.to/factorization/smoothparts-20040510.pdf, (visited on 2017-05-09) (Cited on p. 3.)
3. Bernstein, D.J., Chang, Y.A., Cheng, C.M., Chou, L.P., Heninger, N., Lange, T., van Someren, N.: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In: ASIACRYPT. Springer (2013), https://smartfacts.cr.yp.to/smartfacts-20130916.pdf, (visited on 2017-05-09) (Cited on pp. 1 and 6.)
4. Biryukov, A., Pustogarov, I., Weinmann, R.P.: Trawling for Tor hidden services: Detection, measurement, deanonymization. In: Security and Privacy. IEEE (2013), http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf, (visited on 2017-05-09) (Cited on pp. 2, 8, and 9.)
5. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Notices of the American Mathematical Society 46(2) (1999), http://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf, (visited on 2017-05-09) (Cited on p. 9.)
6. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: EUROCRYPT. pp. 178–189. Springer (1996), (visited on 2017-05-09) (Cited on p. 6.)
7. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology 10(4), 233–260 (1997), https://www.di.ens.fr/~fouque/ens-rennes/coppersmith.pdf, (visited on 2017-05-09) (Cited on p. 6.)
8. Dingledine, R.: Tor security advisory: "relay early" traffic confirmation attack (Jul 2014), https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack/, (visited on 2017-05-09) (Cited on p. 11.)
9. Dingledine, R., Mathewson, N.: Tor protocol specification, https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt, (visited on 2017-05-09) (Cited on pp. 4, 9, 12, and 13.)
10. Dorey, K., Chang-Fong, N., Essex, A.: Indiscreet logs: Diffie-Hellman backdoors in TLS. In: NDSS. Internet Society (2017), https://wp.internetsociety.org/ndss/

wp-content/uploads/sites/25/2017/09/ndss2017_04A-2_Dorey_paper.pdf, (visited on 2017-09-19) (Cited on p. 13.)
11. Freedom of the Press Foundation: SecureDrop, https://securedrop.org, (visited on 2017-09-19) (Cited on p. 2.)
12. Gajek, J.: ssl-dh-params, https://nmap.org/nsedoc/scripts/ssl-dh-params.html, (visited on 2017-09-22) (Cited on p. 13.)
13. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Designs, Codes and Cryptography 67(2), 245–269 (2013), (visited on 2017-05-09) (Cited on p. 4.)
14. Hastings, M., Fried, J., Heninger, N.: Weak keys remain widespread in network devices. In: IMC. ACM (2016), https://www.cis.upenn.edu/~nadiah/papers/weak-keys/weak-keys.pdf, (visited on 2017-05-09) (Cited on pp. 2 and 6.)
15. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: USENIX Security. USENIX (2012), https://factorable.net/weakkeys12.extended.pdf, (visited on 2017-05-09) (Cited on pp. 1, 6, and 7.)
16. Heninger, N., Halderman, J.A.: fastgcd, https://factorable.net/fastgcd-1.0.tar.gz, (visited on 2017-05-09) (Cited on pp. 6 and 7.)
17. Jaggard, A.D., Syverson, P.: Oft target. In: HotPETs (2017), https://petsymposium.org/2017/papers/hotpets/oft-target-1707.pdf (Cited on p. 12.)
18. Jansen, R., Hopper, N.: Shadow: Running Tor in a box for accurate and efficient experimentation. In: NDSS. Internet Society (2012), http://www.robgjansen.com/publications/shadow-ndss2012.pdf, (visited on 2017-05-09) (Cited on p. 13.)
19. Johnson, D.: Stem docs, https://stem.torproject.org, (visited on 2017-05-09) (Cited on p. 6.)
20. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: CRYPTO. Springer (2012), (visited on 2017-05-09) (Cited on p. 6.)
21. Litzenberger, D.: PyCrypto – the Python cryptography toolkit, https://www.dlitz.net/software/pycrypto/, (visited on 2017-05-09) (Cited on p. 7.)
22. Mathewson, N.: Next-generation hidden services in Tor, https://gitweb.torproject.org/torspec.git/tree/proposals/224-rend-spec-ng.txt, (visited on 2017-08-01) (Cited on p. 14.)
23. Matic, S., Kotzias, P., Caballero, J.: CARONTE: Detecting location leaks for deanonymizing Tor hidden services. In: CCS. ACM (2015), https://software.imdea.org/~juanca/papers/caronte_ccs15.pdf, (visited on 2017-05-09) (Cited on p. 10.)
24. Muffett, A.: Facebook brute forcing hidden services (Oct 2014), https://lists.torproject.org/pipermail/tor-talk/2014-October/035413.html, (visited on 2017-05-09) (Cited on p. 13.)
25. Nurmi, J.: Ahmia – search Tor hidden services, https://ahmia.fi/onions/, (visited on 2017-05-09) (Cited on p. 10.)
26. O'Cearbhaill, D.: Trawling Tor hidden service – mapping the DHT (2013), https://donncha.is/2013/05/trawling-tor-hidden-services/, (visited on 2017-05-09) (Cited on p. 11.)
27. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978), https://people.csail.mit.edu/rivest/Rsapaper.pdf, (visited on 2017-05-09) (Cited on p. 3.)
28. Roberts, L.M.: "anomalous keys in Tor relays" technical report now available (Apr 2017), https://lists.torproject.org/pipermail/tor-dev/2017-April/012161.html, (visited on 2017-08-02) (Cited on p. 14.)

29. Swanson, E.: Scallion – GPU-based onion hash generator, https://github.com/lachesis/scallion, (visited on 2017-05-09) (Cited on pp. 9 and 13.)
30. The Tor Project: CollecTor, https://collector.torproject.org, (visited on 2017-05-09) (Cited on pp. 2, 4, 6, and 8.)
31. The Tor Project: Servers – Tor metrics, https://metrics.torproject.org/networksize.html, (visited on 2017-05-09) (Cited on p. 4.)
32. The Tor Project: Tor directory protocol, version 3, https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt, (visited on 2017-08-02) (Cited on pp. 4, 12, and 14.)
33. The Tor Project: Tor research safety board, https://research.torproject.org/safetyboard.html, (visited on 2017-05-09) (Cited on p. 13.)
34. The Tor Project: Tor shared random subsystem specification, https://gitweb.torproject.org/torspec.git/tree/srv-spec.txt, (visited on 2017-08-02) (Cited on p. 14.)
35. Valenta, L., Adrian, D., Sanso, A., Cohney, S., Fried, J., Hastings, M., Halderman, J.A., Heninger, N.: Measuring small subgroup attacks against Diffie-Hellman. In: NDSS. Internet Society (2017), https://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/ndss2017_04A-1_Valenta_paper_0.pdf, (visited on 2017-09-19) (Cited on p. 13.)
36. Valenta, L., Cohney, S., Liao, A., Fried, J., Bodduluri, S., Heninger, N.: Factoring as a service. In: Financial Cryptography. ACM (2016), https://eprint.iacr.org/2015/1000.pdf, (visited on 2017-05-09) (Cited on p. 6.)
37. Winter, P.: Are vanity onion domains a good idea? (Oct 2015), https://moderncrypto.org/mail-archive/messaging/2015/001928.html, (visited on 2017-05-09) (Cited on p. 13.)
38. Winter, P., Ensafi, R., Loesing, K., Feamster, N.: Identifying and characterizing Sybils in the Tor network. In: USENIX Security. USENIX (2016), https://nymity.ch/sybilhunting/pdf/sybilhunting-sec16.pdf, (visited on 2017-05-09) (Cited on p. 8.)

A Potentially targeted onion services

Table 4. The details of the attacks on four onion services. The second column shows the fingerprints of the HSDirs that were participating in the attack. The third column shows the affected onion service descriptors, followed by the date of the attack in the last column.

Onion service	Truncated HSDir fingerprint	Truncated onion service descriptor	Date
22u75kqyl666joi2	325CAC0B7FA8CD77E39D	325CAC08B0A3180B590E	2015-08-14
	325CAC0AB1AAD27493B9	325CAC08B0A3180B590E	2015-08-14
	325CAC0A43B2121B81CD	325CAC08B0A3180B590E	2015-08-14
	FA256741ED22FD96AF5A	FA256740740356704AB8	2015-08-14
	FA256743ACFCA9B7C85D	FA256740740356704AB8	2015-08-14
	E5E778326AF0FF0A634A	E5E7783245096FB554A1	2015-08-15
	A5C59B3D0FFBDE88405E	A5C59B3CD34802FC4AC3	2015-08-15
	A5C59B3FCCD2FA8FAD42	A5C59B3CD34802FC4AC3	2015-08-15
	A5C59B3FD5625A0D85D1	A5C59B3CD34802FC4AC3	2015-08-15
n3q7l52nfpm77vnf	A0E83AA191220B240EC0	A0E83AA115098CA7FE9B	2016-11-27
	A0E83AA28382135DC839	A0E83AA115098CA7FE9B	2016-11-27
	EBF154DA21B49101ED5B	EBF154D809425D3E923E	2016-11-28
	EBF154D8BB6EECCC2921	EBF154D809425D3E923E	2016-11-28
	EBF154D9E2B10A2420E0	EBF154D809425D3E923E	2016-11-28
	6761D2BE758FA0D76822	6761D2BCF40FF34274F3	2016-11-29
	59E415D78921BFF88168	59E415D5075157CAADB7	2016-11-29
	26597E62875C498AC139	26597E6048BF7CC9D593	2016-11-30
	26597E61DDFEE78F336D	26597E6048BF7CC9D593	2016-11-30
	7CDB224FE64F2A50CC50	7CDB224DC51432C037C5	2016-11-30
	2D148D3EBF9D2B9D8CCB	2D148D3CB6C5FC4DCA14	2016-12-01
	2E25D8469331FEAE933D	2E25D842BF5DDA936BA2	2016-12-05
	2E25D847E579AED1B0EC	2E25D842BF5DDA936BA2	2016-12-05
	2E25D8454C96E20CF153	2E25D842BF5DDA936BA2	2016-12-05
	2E25D846564DCBE43CD2	2E25D842BF5DDA936BA2	2016-12-05
	2E25D8447518DA93B4FF	2E25D842BF5DDA936BA2	2016-12-05
	264EA12B47CBCC8043C5	264EA12410F7D9CD6E54	2016-12-05
	264EA1284855A596D5D6	264EA12410F7D9CD6E54	2016-12-05
	264EA12B4C46672E002C	264EA12410F7D9CD6E54	2016-12-05
silkroadvb5piz3r	BC89A92F53581C4F6169	BC89A889D3DF7F0027A5	2013-05-21
	712CA45AF4055E7AC69A	712CA3DEF4EB21C76A95	2013-05-22
	DE15299D7EE5882F0BEF	DE1529316F5172B35B8E	2013-05-23
	FF0BF54FBEEE7A003CE6	FF0BF49076AA63C97FA2	2013-05-24
	E9F25C4899F9DC81E48E	E9F25BBA0D4501FAE18B	2013-05-28
	B81B43C015DE42D05208	B81B43637F22592ECC80	2013-05-29
	59529817C6E797D78311	5952979BD9FEECE847E7	2013-05-31
	BCB332864640653892D4	BCB33236E0AD461DF585	2013-06-02
	51FC178DFF3D0B869760	51FC172F0062B623A39D	2013-06-03
thehub7gqe43miyc	F6961286D361F825A9AD	F6961286C2FEEA8DEDEB	2015-08-22
	F6961286C453F6A6381D	F6961286C2FEEA8DEDEB	2015-08-22
	F6961286D826D7D1C0F9	F6961286C2FEEA8DEDEB	2015-08-22
	816FEE16200BE1719D00	816FEE15D26F41A72039	2015-08-22