Hardening Tor against State-Level Traffic Correlation Attacks with K-Anonymous Circuits

(extended abstract of the MSc dissertation)

Vítor Nunes
Departamento de Engenharia Informática, Instituto Superior Técnico
Advisor: Professor Nuno Santos

***Abstract*—Tor is a popular low-latency anonymous protocol that allows users to surf the Internet anonymously. Despite its popularity Tor, like other low-latency anonymous systems, is vulnerable to traffic correlation attacks, which enables an attacker to match a source IP with a destination IP. This can be accomplished by a state-level adversary capable of observing both endpoints of a Tor circuit and correlating the corresponding outgoing and incoming flows using traffic analysis techniques. Although correlation attacks are a known weakness in Tor's design, no practical solutions to mitigate this attack without significantly degrading the performance of the system have been proposed. To tackle this problem we present TorK - a Tor pluggable transport that leverages the notion of indistinguishable flows to ensure k-anonymity amongst multiple sources of Tor connections. Thus, we aim at improving the existent Tor system to make it more robust against correlation attacks launched by a global adversary that observes all flows transiting a given circuit. In this way, we can mitigate the impact of correlation attacks without significantly damaging the system performance.**

I. Introduction

Tor [1] is a widely used overlay network which allows a user to browse the Internet anonymously, i.e. without exposing its IP address to the destination. However, despite the sophisticated defences of the Tor network, Tor users may be subject to deanonymization attacks based on *traffic correlation* [2]. Such attacks can be carried out when an adversary can observe the traffic at the circuit's entry and exit relays. By intercepting the traffic at these relay nodes, the adversary can use features such as the volume of traffic and inter-packet arrival times to perform flow correlation. If two flows are correlated, the adversary can determine with high degree of certainty the IP address of the circuit endpoints.

Today, traffic correlation attacks are realistic threats to Tor users since they can be launched at large scale by state-level adversaries. Nithyanand et al. [3] presented an empirical study where they discovered that up to 40% of all Tor traffic is vulnerable to attacks by traffic correlation from network-level attackers; if the network-level attackers are Autonomous Systems (ASes) that may collude with each other the amount of vulnerable Tor traffic increases up to 42%. If the same network-level attacker are state-level adversaries, then 85% Tor traffic is vulnerable. Users in some countries (China and Iran) are particularly affected, since 95% of all possible circuits are vulnerable to correlation.

There are several efficient correlation attacks to Tor known today that rely on the existent of malicious entities in control of the adversary, such as clients, relays or directory authorities. Several timing attacks [4], [5] allow attackers to determine endpoints of a circuit by correlating the time it takes for a flow to travel from both edges of a Tor circuit. Several solutions were proposed [6] in the literature to thwart these attacks and rely on *perfect interference* which states that all output should have the same shape.

In this paper we present TorK, a Tor pluggable transport and controller that provides additional protections to Tor against traffic correlation attacks. TorK shapes traffic directed to Tor bridges and ensures *k-anonymity* among all connected users. In particular, TorK guarantees that a given ingress client flow cannot be correlated with the corresponding egress flow exposing the identity of the sender.

Our approach is based on the idea of constructing k-*circuits*: Tor circuits with k ingress flows and a multiple egress flows, such that is not possible to link an ingress to a corresponding egress flow. To build k-circuits in the Tor network we need to overcome several challenges. Currently, Tor circuits offer point-to-point channels. However, in order to enable k cooperating users to participate in the creation of a k-circuit (to help a given user to get in touch with a certain destination) adaptations in the way Tor circuits are currently built need to be performed. In particular, we need to have a form to generate k ingress flows that can be coalesced into a single one. Ideally we want to provide this service without changing the underlying Tor protocols or the existing software implementations. Second, we need to ensure that the k ingress flows cannot be distinguished by the adversary, even when employing advanced traffic classifiers; i.e., the k ingress flows must be *indistinguishable*. Third, the performance overheads introduced by our components should be tolerable to the end users.

We show that TorK can successfully setup an indistinguishable channel by conducting an experiment with a state-of-the-art classifier for conducting traffic analysis. We also show that TorK managed to withstand against active attacks, not disclosing the identity of the real sender.

II. Threat Model

The adversary has complete access to the global network infrastructure of the Tor network, having access to all the network links and middleboxes interconnecting Tor ecosystem hosts: Tor users' computers, TorK bridges, Tor relays, and any remote host contacted by Tor users. Thus, the adversary is able to eavesdrop and intercept any massage that has been exchanged between the communicating parties. The same attacker can launch active attacks aimed at flustering the network such as delaying or dropping packages to study how the system responds, and deploy malicious entities (sybil attacks).

III. Related Work

Traffic correlation in Tor remains an open research problem since there are no practical solutions. Evidence suggests

that powerful adversaries exist today in the form of actors being able to control entire networks within national boundaries (oppressive regimes) [7], or to able tap into the Internet backbone across countries (intelligence agencies) [8]. We now describe two main approaches that actively avoid unsafe i) relay nodes or ii) autonomous systems. Then, we discuss the concept of K-anonymity.

Avoiding Unsafe Relay Nodes: To mitigate adversary's attempts to launch correlation attacks, clients may attempt to avoid unsafe relay nodes by employing several strategies:

Clients may run a co-located trusted relay node alongside the Tor client, and use that relay as entry node to the Tor circuits created by the user. As a result, the downstream relay nodes will not be able to determine whether the traffic forwarded by the trusted relay node was originally produced by the local user or by another user that may be using that same relay node for building its own circuits. A second strategy, currently used by Tor, is to scan and identify bad relay nodes to decrease the possibility of clients selecting malicious relays, by detecting and reporting bad relay nodes. Generally, a relay is considered to be bad if it is malicious, misconfigured, or unreliable. To mark relays, Tor uses a set of labels designated as *flags*.Tor also restricts the set of entry nodes used by a client in an attempt to mitigate the Guard Rotation Weakness [9]. If a client was unlucky and selected a malicious guard, he had the chance of regaining anonymity when its current guard changed.

Avoiding Unsafe Autonomous Systems: An adversary may not even need to control any specific relay node to deanonymize Tor traffic, but only to possess the ability to eavesdrop on the inter-relay traffic that crosses the network controlled by the adversary [2], [10]. Typically, such approaches attempt to help Tor clients' to choose paths away from the prying eyes of ASes by leveraging the analysis of the Internet topology boundaries and inter-relay latencies [11]. As a way to prevent attacks based on traffic interception through BGP hijacking, Sun et al. [12] proposed a monitoring framework for detecting BGP changes, allowing Tor to inform vulnerable clients, which in turn can opt to suspend Tor communications or use another relay. A number of authors have proposed AS-aware path selection algorithms for decreasing the chance of an AS-level attacker to observe traffic flowing between both endpoints of a Tor circuit [13], [14], [12], [3]. These solutions even though decrease the probability that an AS is able to eavesdrop in both ends of a connection, they did not sufficiently mitigate the possibility for an AS to perform traffic correlation.

K-anonymity: Samarati and Sweeney [15] proposed in 1998 K-anonymity to solve the problem of disclosing privacy-sensitive database records that needed to be anonymized (e.g. medical records). K-anonymity is then a property of released data such that the information about any given individual cannot be distinguished from at least $K-1$ individuals. For a larger value of k, the anonymity set is larger, thus anonymity is stronger. K-anonymity has been extensively studied and applied in a wide range of areas [16], [17].

IV. Design

This section presents a system model for TorK which introduces the notion of *k-circuits*, a new anonymity-preserving primitive leveraging k-anonymity that aims to enhance the Tor network against adversaries capable of performing arbitrary traffic correlation attacks. This is achieved by providing K-anonymous sender communication properties on top of Tor's preexisting infrastructure. In the following subsections, we present its architecture and the most relevant technical details as well as strategies to withstand against a range of attacks.

TorK employs a client-server architecture whose main components are the bridge (server) and the client gateway as portrayed in Figure 1. Both endpoints communication is done through TorK Protocol over a secure channel. The channel along with the protocol allows k users to send arbitrary data, in a way that an attacker cannot link a given egress flow to a specific user.

A. System Model

TorK prevents correlation attacks by allowing $k-1$ additional users – in the Figure 2, just Alice and Charlie – to collaborate with Bob such that an adversary will not be able to distinguish who amongst them is the real originator of traffic associated with this circuit. To achieve this, prior to the circuit establishment phase, all the k users initiate a communication channel between each local client and the bridge. Each of these channels is a *segment*. Each segment acts as a tunnel between the local client and the bridge such that the local client can send arbitrary traffic through it.

TorK aims to provide K-anonymity by allocating groups of size K. We use the term *k-circuit* to refer to a coordinated setup of K segments for tunneling Tor circuits owned by a group of K users. Although the k-circuit in Figure 2 tunnels a single Tor circuit owned by Bob, multiple circuits should be allowed to be tunneled through. Thus, Alice, Bob and Charlie form one k-circuit with K = 3.

To prevent the adversary from extracting useful information from the segments (this could be achieved by observing differences in the volume and timing properties of the traffic), the traffic of all participating segments is encrypted and modulated according to a common traffic shaping function. Thus, even though the attacker is able to capture and inspect the traffic from each of the three users (red ellipse) it cannot correlate the message to the original sender, i.e. from the three users discover the one accountable for the unique egress Tor circuit due to indistinguishability between segments. The attacker can inspect all hops in the network but he would have to randomly guess the sender having a $1/k$ probability of succeeding.

Achieving the desired properties while delivering good performance can be accomplished at the expense of increase, yet tolerable, bandwidth consumption. It is realistic to allocate some of the spare network resources currently present in the Tor network. Besides, according to a recent study [18] more than half of Tor bandwidth is currently unused, since

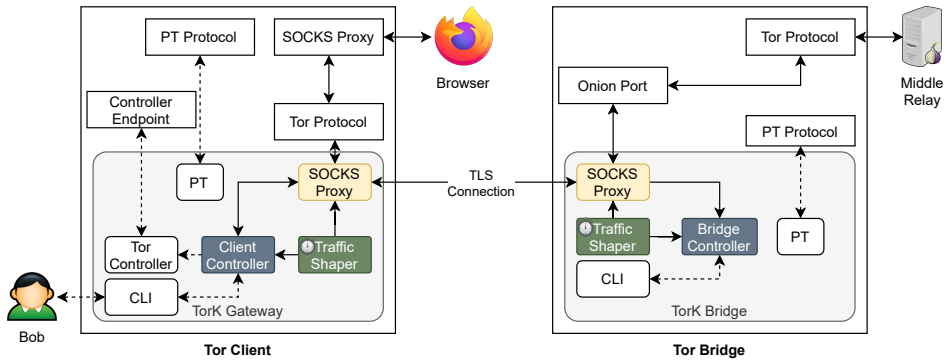


Figure 1: TorK internal components

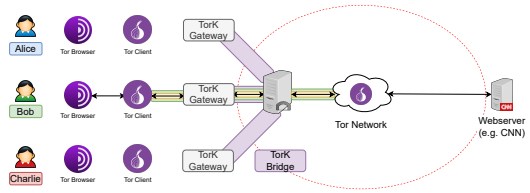


Figure 2: TorK system model depicting a k-circuit. All hops can be observed by an attacker. Onion layers are colored.

only 45.5% of the advertise bandwidth is actually used. This means that there is currently room for leveraging existing spare bandwidth in Tor without affecting the operations of the system or its ability to grow.

B. K-circuits Specified

The specific k value to be used for a given k-circuit will be determined by the TorK bridge whenever a set of restrictions imposed by the group of potential participants is met. Specifically, each client interested in participating in a k-circuit will be able to specify a lower bound *k-min* to the minimum number of participants that must be present in the k-circuit. This value allows each user to indicate the smallest acceptable anonymity set for his communications. For instance, Alice may indicate that she is only interested in joining k-circuits with no less than *k-min*$= 3$ users, i.e., $k \geq$ *k-min*. The TorK bridge will then maintain a pool of participants in a waiting line until their respective *k-min* value restrictions are met. As soon as a group of k of participants is available such that k is smaller or equal than the *k-min* value restriction of each participant, then a new k-circuit can be formed for that group.

Public and private membership: TorK also offers two k-circuit membership types: public and private. A public k-circuit admits all sorts of member and are managed by bridges. By contrast, a private k-circuit can be requested by users and are solely used by members who know their existence. Private k-circuits are important to prevent Sybil attacks and establish some ground of trust for all the participating members in a given k-circuit.

C. Thwarting Protocol Level Attacks

To assure that K-anonymity is preserved, TorK needs to coordinate the action of several users joining a k-circuit. Without proper coordination, it would be trivial for an attacker to exploit different users' behaviors. For instance, observing the Tor network allows an adversary to determine who among the k users are actually sending traffic, even though the traffic is encrypted. To this end, TorK constructs groups of k users designated as k-circuits and uses a custom protocol to coordinate the actions of its members.

K-circuit establishment request: When a client connects to a bridge, the TorK protocol begin (Figure 4). Initially, each client gateway, upon connection to the bridge, sends a HELLO message containing the specification of its k-min restriction value. In other words, gateways define the minimum number of users that should be connected at the bridge in order to establish a circuit. The bridge validates this number and reply with HELLO_OK. In the case of Figure 4, Alice requests a minimum of two users that must be present within the same k-circuit.

Tor circuit creation request: Later, when a client intends to open a Tor circuit it will ask the bridge by sending an ACTIVE control frame. Bridges will reply either with ACTIVE when all k-circuit members' k-restrictions are hold, and consequently, the client is authorized to open the Tor circuit, or with WAIT otherwise. Since Alice requested two clients and only one, herself, is connected at this moment, she will wait until another client is placed in the same k-circuit.

K-circuit ready: Bob joins requesting also a k-min of 2. At this instant, bridge has two connected clients and therefore Alice has authorization to create a Tor circuit since their restriction is now fulfilled. From this point onward, Bob is authorized to send data frames towards the bridge, containing Tor circuit cells, and consequently, he is also allowed to open a Tor circuit, which is going to be forwarded to the Tor network.

Reaction to changes in k-circuit conditions: At any time, an active client can receive a WAIT frame. Such indication tells clients that their restrictions, despite having

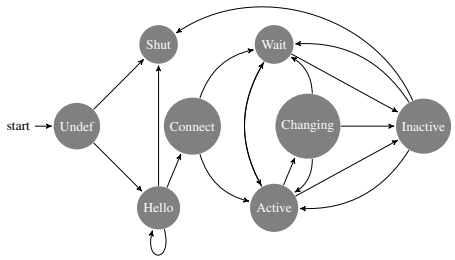


Figure 3: TorK Protocol: Client State Machine

been satisfied in the past, they do not hold currently anymore (e.g., due to a participant that has left). Clients react by destructing the circuit immediately and wait further to create a new circuit. Apart from WAIT frame, clients can receive a CHANGE frame. Such frame requires that client replace their current circuit. In other words, tear-down the existing circuit and open another one.

Tearing-down circuits: However, client gateways can tear-down the circuit by their will or give up of waiting for bridge authorization. Active or waiting clients can send an INACTIVE to indicate that they will tear-down their circuit or they no longer intended to open one, respectively.

K-Circuit State Machines: A state machine denoting the clients synchronization process is depicted in Figure 3. The first phase consists on a handshake between the new client and the bridge where clients define their *k-min* (Hello). After the handshake phase, the bridge will place the new client in an already existent k-circuit or constructing new one (Connect).

D. Thwarting Traffic Pattern Attacks

TorK is also designed to hold out against potential network-level active and passive attacks. For instance, consider the example in Figure 2. Assume also that the bridge is trusted and all cooperating clients – say, Alice and Charlie – are benign users. In this circumstances, notwithstanding, an adversary with the ability to drop, inject, or modify the sequence of network packets could try to deanonymize Bob's communication by instrumentally dropping packets of each segment of the k-circuit, and observe for which of the segments there is a visible reduction in the throughput of the bridge's outgoing connection to the middle node. In this case, dropping packets in Bob's segments will interrupt the transmission of Tor circuit cells which will be reflected in a smaller amount of packets exiting the bridge and allow the adversary to identify the real sender: Bob. TorK uses two strategies to resist against active attacks:

Traffic shaping: In order to shape traffic, we used a *perfect interference* strategy [6], ensuring that all flows have the same shape regarding timing and size properties. Clients and bridges implement a module aim at draining traffic at a constant rate using fixed-size packets, called *chunks*. This module receives the client's Tor circuit cells, split them into several chunks and send them to the bridge. Upon reception, the bridge receives all chunks passing them to Tor, whose next destination is the client's middle relay. The chunks have fixed-size and thus, padding is used to ensure that the chunk size remains always the same, apart from the real data length.

Even though the clients traffic share the same properties, attacks are still possible based on clients' sending behavior. In a k-circuit with two users, one attacker can observe an ingress flow, followed up by an egress flow. Such observation uncloaks the responsible client of the egress flow, because only one client sent chunks. Taking that into account, this module also generates chaff chunks. Such chunks do not carry useful data and their goal is to create a common sending behavior among the same clients under the same circuit outwitting the attacker.

Flow control: To thwart attacks where an attacker can deliberately delay certain clients segments, we propose to augment the synchronization protocols introduced in Section IV-C so that all segments of a k-circuit are proactively throttled by the bridge to ensure they all deliver the same throughput. This means that the effects of dropping packets in any single member's segment will cause a corresponding leveling down in the throughput of all other members' segments. As a result, the adversary will not be able to distinguish which segment carries Tor traffic.

E. Thwarting Bridge Impersonation Attacks

The anonymity properties of k-circuits depend upon the fact that TorK bridges are trusted *not to reveal* the identity of the participating clients. In fact, a bridge knows which of the participants use their respective segments for forwarding real traffic. Therefore, it is necessary to ensure that TorK bridges do not reveal this information to an adversary. However, given that the Tor network is open, anyone can deploy TorK bridges, including malicious ones that can be leveraged to deanonymize end users.

To detect potentially malicious TorK bridges, TorK can run on top of trusted computing hardware. Inspired by the approach of Kim et al. [19], our solution requires the deployment of trusted hardware on the bridge host and consists in deploying the TorK bridge software inside a trusted execution environment (TEE). The goal is to protect TorK memory space even when the bridge's owner has hardware access and high privileges – root access. Ideally, clients and bridges can perform remote attestation, i.e. to attest the integrity of that program by checking a signature of the program's hash signed with a hardware key. Thus, clients can verify the bridges' trustworthiness and discard them if attestation fail.

F. Thwarting Client-side Sybil Attacks

So far we have assumed that all participants in a k-circuit are benign. However, if one or more participants are malicious and collude with an adversary that can monitor the network traffic, it may be possible to deanonimyze a legitimate client sending Tor traffic through that k-circuit. Consider for instance the example shown in Figure 2. If

Alice and Charlie are malicious, they can inform the network eavesdropper that they are not sending any traffic through their specific segments. Given that the network eavesdropper can observe that the number of participants in that k-circuit is three and he can rule out Alice and Charlie, then the only possible source of the traffic exiting the bridge is Bob.

To mitigate such attacks, we propose to investigate several different strategies, which can possibly be deployed and operate in combination. One idea is to implement more clever *participant selection policies* at the bridge. Essentially, in addition to satisfying the *k-min* value restrictions of participants, the bridge can be more selective of which participants to include in a given k-circuit so as to increase their geographical and / or AS diversity. This is based on the intuition that it may be more difficult for an adversary to control a large number of clients deployed across different ASes or geographic regions. Another idea is to run an *alibi agent* at the bridge itself. Essentially, for each k-circuit, the bridge would run a piece of software that mimics the behavior of a real client by establishing Tor circuits and tunneling traffic through these circuits. As a result, even if the attacker could control all of $k-1$ participants, it could not know whether the true originator of the traffic exiting the bridge is the remaining participant or the alibi agent.

V. Implementation

We have implemented TorK prototype in C++ using the OpenSSL and Boost libraries. One main goal is to implement TorK without changing the underlying Tor infrastructure, decoupling TorK from Tor's implementation and improve code maintainability of both tools.

TorK is structured in six main components: i) Client Pluggable Transport, ii) SOCKS proxy, iii) Tor Controller, iv) TorK Controller Interface, v) Traffic Shaper and vi) Bridge/Client Controller. The Pluggable Transport Client implements the Tor PT API instructing Tor how to open and control TorK process. The SOCKS proxy is responsible to receive Tor data which is then shaped and sent according to the Traffic Shaper method. The Tor controller implements the Tor Control Spec empowering TorK to control Tor circuits (e.g. creation, destruction). Finally, the TorK Controller Interface allows clients to interact with TorK (e.g. specifying k-min value, joining private k-circuits).

TorK's indistinguishable channel encloses a Traffic Shaping function. Such function is responsible for sending fixed sized TorK packets – called *frames*. Each frame is divided into smaller slices called *chunks*. Each chunk share the same size and properties, apart from the first one, which contains a header. Therefore, all chunks except the first one can carry data entirely. This design choice prevents Tor cells' fragmentation over multiple chunks and consequently over multiple frames. In other words, it allows us to have chunks large enough to contain one or several Tor cells without fragmentation while sending only the necessary data (chunks) minimizing the sending of unnecessary bytes. Therefore, the size of the chunks should be set to a multiple of the Tor cell size. The chunks' transmission rate is also configured to provide a target throughput. TorK frames can be labeled as data, chaff or control frames.

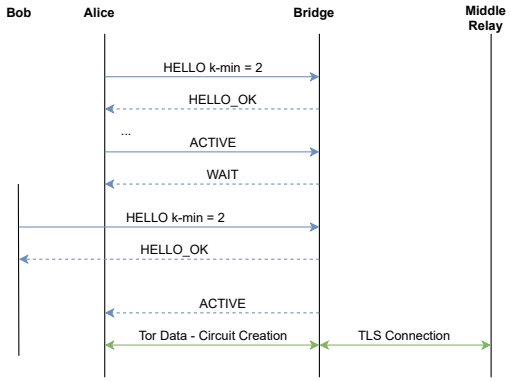


Figure 4: TorK protocol for a two k-circuit users (k-min=2).

Traffic shaper (TS) rate varies according to the number of connected users, within a defined rate interval window (minimum, maximum). Having a higher number of connected users requires more frame processing carry out by bridges. Thus, we decided to decrease the rate proportionally to the number of connected users. To select minimum and maximum values an experiment (Section VI-B) was rolled out to measure the time taken by the traffic shaper to prepare a single data and chaff frame.

Apart of the indistinguishable channel it is necessary to coordinate the actions of k members acting in group to cause plausible doubt using a synchronization protocol – TorK Protocol. The protocol allows members to share their intentions (e.g. open or close Tor circuits) to the bridge and wait for authorization. Figure 4 sketched the protocol for two users (Alice and Bob) depicting the messages exchanged between them. Some TorK protocol commands required Tor circuits to be tear down abruptly. For instance, when bridges perceive that some client should change their circuit (e.g. a new client joined the same k-circuit), they notify the old living client and block all outgoing data traffic from that client, preventing it from entering the Tor network and therefore, being visible to a network attacker. The blockade imposed by the bridge will drop every data frame and it will remain active until the client acknowledgement. By dropping Tor data cells we are inadvertently disrupting the counter values used in Tor cells TLS encryption. As a consequence, the connection is shattered upon restoring, i.e. upon lifting the blocking, mainly due to the reception of TLS Tor cells containing wrong/unexpected counter value. To prevent such scenario bridges and clients abruptly close their connection to Tor. However, Tor clients mark bridges as faulty when the local connection closes abruptly. Tor will not use the faulty bridge in the future. To overcome the issue, we needed to change a single line in Tor source code which prevents Tor from marking faulty bridges. We stress that such requirement do not introduce any change in Tor protocols, and consisted in disabling the call to the function `entry_guards_note_guard_failure` within the

`entry_guard_failed` in `src/feature/client/entrynodes.c`.

VI. Evaluation

A. Evaluation Methodology

This section describes our methodology for assessing TorK's prototype performance and resistance against multiple traffic analysis attacks.

Experimental testbed: Our testbed is composed of a TorK bridge and 50 TorK clients (leveraging Tor v0.4.2.8). Our nodes are executed within Docker containers, provisioned either with 2 vCPU and 2GB of RAM in the case of the bridge node, or 1 vCPU and 1GB RAM for client nodes. The bridge and client nodes run in separate physical hosts equipped with 2 Intel Xeon CPU E5506 vCPUs. Each client was configured to use a specific Tor circuit throughout our experiments. We manually selected all relays comprising each Tor circuit based on Tor's *TopRelays* list, and choose different nodes within Europe.

Datasets and Classifier: For assessing the resistance of TorK against traffic analysis, we aim to understand whether it is possible a) to identify which flows between clients and bridges transport useful data instead of chaff traffic, and b) to correlate the useful data flows produced by clients with the outbound (egress) flows from the bridge towards Tor.

For conducting the above experiment, we build a dataset of connections between clients and a bridge, and, respectively, between the bridge and the Tor network. For case a), we select half of our clients to receive chaff traffic from the bridge, while the remaining 25 clients will fetch a file hosted in a private server, over TorK, repeating this process 100 times (for a total of 5000 samples). For case b), all clients fetch continuously a file, repeating this process 100 times (for a total of 5000 samples). We capture traffic in/outbound the bridge during 15 seconds.

To assess TorK's traffic analysis resistance properties, we leverage the XGBoost [20] classifier using two sets of features based on summary statistics over the packet length and inter arrival times, such as burst behaviors and high-order statistics (e.g., kurtosis), and quantized packet length distributions. We train and test the classifier through 10-fold stratified cross validation. The rationale for our choice is based on the fact that multiple state-of-the-art techniques which are able to efficiently differentiate classes of traffic over encrypted network streams employ classifiers based in decision-trees (e.g., Random Forests or XGBoost [20]).

Security metrics: We leverage the following metrics to assess TorK's k-circuit indistinguishability: true positive rate (TPR), false positive rate (FPR), and the area under the ROC curve (AUC). The TPR measures the fraction of TorK flows that are correctly identified as carrying useful data, while the FPR measures the fraction of chaff traffic flows that are erroneously classified as useful traffic. For case b), the TPR measures the fraction of client-bridge and bridge-Tor pairs that are correctly correlated, whereas the FPR measures the fraction of uncorrelated pairs that are deemed as correlated. The AUC summarizes the trade-off between the TPR and the FPR for the different possible cutout points for classifiers possessing adjustable internal thresholds. An AUC of 0.5 is equivalent to random guessing.

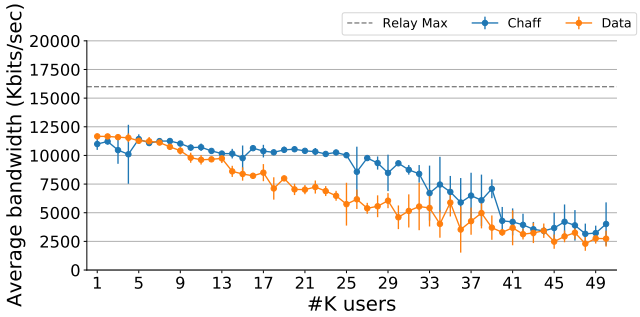


Figure 5: Throughput observed by number of clients.

To select a reasonable number of users to test TorK we estimated the amount of daily connected users in Tor bridges based on monthly statistics [21]. Thus, 50 clients represent an average of daily users in Tor bridges.

Performance metrics: To assess TorK's performance, we measure three different properties of connections established through TorK circuits. First, we measure *throughput* by instructing a client to connect to a private server via TorK using `iperf3`. Second, we measure *latency* by estimating the round-trip-time overt the channel using `httping`. This tool performs a HTTPS request and measures the time it takes to receive the first byte of the header – header time to first byte. We choose it since Tor does not support the forwarding of typical ICMP packets using `ping`.

B. TorK Baseline Deployment

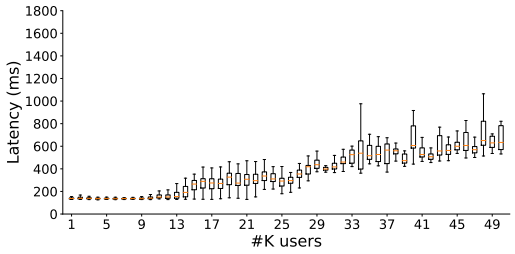
Frame size: The current Tor specification multiplexes cells containing Tor payload data over the same TCP stream along a given circuit [1]. Tor currently uses 514 B cells. In the wire, the TCP data length of a single Tor cell occupies 536 B, considering the TCP and TLS headers. For TorK's baseline configuration, we select the TorK frame's chunk size to 536 B each frame containing up to two chunks, so that each chunk contains at most one Tor cell.

Send rate: Processing TorK frames takes an average $9.32\,\mu s$ and $10\,\mu s$ at the 90th percentile. To avoiding send empty TorK cells, while allowing the bridges to fully perform a communication round, we define the function: $Rate(n) = n \times 10\,\mu s$, where n is the number of connected clients.

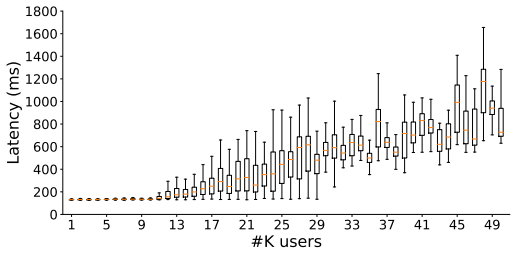
Next, we assess the performance of a baseline TorK deployment configured with the above parameters.

C. TorK Baseline Performance

Network performance: Our baseline configuration was set to use 536 B chunk frames under a dynamic $10\,\mu s$ to $500\,\mu s$ rate, which varies according to the number of connected users, leads to a maximum theoretical throughput of 8.5 Mbps with 50 simultaneous clients connected. This is also the amount of chaff we are introducing in the network.



(a) Clients receiving chaff.



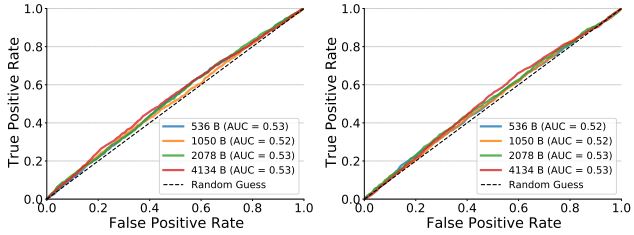
(b) Clients receiving data.

Figure 6: Latency observed by number of bridge clients.

Figure 5 depicts the achieved throughput based on the average bandwidth of 5 runs according to the increasing number of connected users K receiving chaff. It also depicts the achieved throughput according to the increasing number of connected users K receiving actual data through Tor. The sender and receiver values are the throughput values observed by the iperf running on TorK client gateways (sender) and on the private server (receiver).

For testing purpose all 5 runs used the same relay configuration. It is important to notice that even though relays advertise bandwidth as part of the Tor consensus, relays may also impose bandwidth restrictions per user/relay connection using the options: PERCONNBWRATE, PERCONNBWBURST, RELAYBANDWIDTHRATE and RELAYBANDWIDTHBURST which use token buckets to rate the bandwidth of client or relay data evenly to every connected client/relay discouraging greedy users. Unlike the advertised bandwidth, per connection rates are not public, hence we conducted an throughput experiment (in 5 runs) without TorK, i.e. using only Tor, under the same relays configuration to measure the actual per client throughput of the testing relays. On average, under this testing setup, we observe a 16 Mbps throughput (denoted as a gray line).

The results show that one TorK client can achieve up to 12 Mbps out of the total 16 Mbps achieved by these relays when not using TorK. With a higher number of connected clients, TorK decreases the rate and thus reducing the throughput. This throughput reduction accounts that processing chaff frames is less burdensome than data frames since the first are discarded upon reception and the second requires receiving all chunks and deliver the content to the Tor network.



(a) Summary Statistics (b) Packet Length

Figure 7: Performance of the classifier when distinguishing between k-circuits carrying real Tor data and chaff.

An increase in the number of connected users causes the latency to increase. New data frames causes slightly higher increases in latency than chaff frames. It is important to notice that the bandwidth control measure is also in place (Section VI-E). Therefore, the delivering process of data frames' content (to the Tor Network) may be intentionally delayed as reactive measure to active attacks.

Indistinguishability of k-circuits: To ensure the k-anonymity of clients, each individual k-circuit must not be distinguishable from another by a network adversary. Thus, the adversary must not be able to tell apart whose k-circuit carries actual Tor data from those carrying chaff data only.

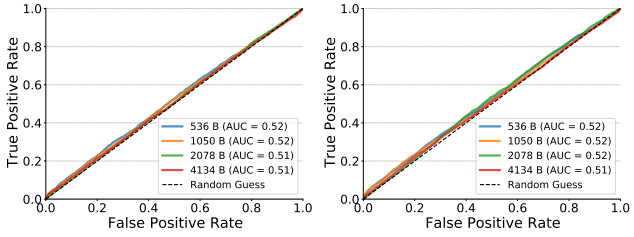
To show that TorK provides indistinguishability for k-circuits, we capture network traces at the bridge following case a) in order to generating a dataset and training an XGBoost classifier aimed at distinguishing whether a given flow pertaining to a TorK circuit carries useful data or chaff.

Figure 7a depicts the XGBoost ROC curves using summary statistics under differ chunk sizes and traffic shaping rates configuration. The summary statistics include the minimum, maximum, mean, standard deviation, percentiles, kurtosis and skew. Kurtosis measures the distribution of the data relative to a normal distribution and skew measures the symmetry of the packet length. Figure 7b depicts the XGBoost ROC curve using only the packet length (PL) statistics. The PL feature encloses a frequency distribution table, represented in buckets of packet lengths. We can observe that the maximum AUC achieved by the classifier is 0.53, suggesting that the classifier brings no advantage, regardless the chunk size, to an adversary aimed at revealing whether a client is sending chaff or real Tor traffic.

Unlinkability of k-circuits and Tor traffic: To ensure k-anonymity TorK must also prevent an adversary from being able to link a real Tor traffic flow departing from the bridge with the particular k-circuit that originated such traffic.

To show that TorK also prevents this attack, we captured network traces at the bridge following case b) in order to generate a dataset of flow pairs corresponding to the TorK traffic observed between clients and the bridge, and between the bridge and Tor middle nodes.

Figure 8a and Figure 8b depict the performance of the classifier when matching a given k-circuit with an outgoing



(a) Summary Statistics (b) Packet Length

Figure 8: Performance of the XGBoost classifier when matching TorK flows with the corresponding Tor circuit.

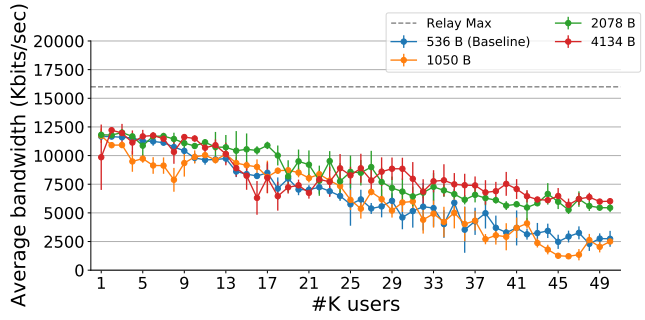
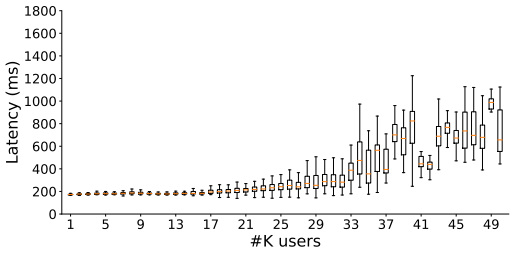
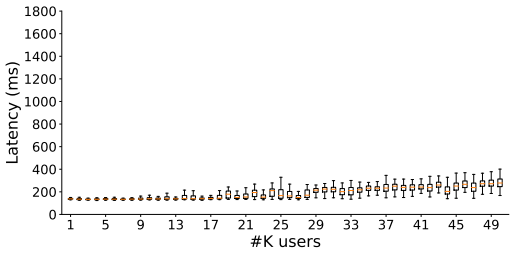


Figure 9: Throughput achieved with different chunk sizes. All clients are receiving data.



(a) 1050 B chunk (up to 2 Tor cells per chunk).



(b) 2078 B chunk (up to 4 Tor cells per chunk).

Figure 10: Latency by number of bridge clients when using different chunk sizes. Clients are receiving data.

flow from the bridge towards the Tor network. The figures show that the classifier is only able to obtain a maximum AUC of 0.52, suggesting that a passive network adversary is unable to link k-circuits carrying Tor traffic with the actual Tor flows departing from the bridge.

D. Varying Traffic Shaping Parameters

The baseline experiment uses 536 B chunk size which can carry a single Tor cell within each chunk. To study how the chunk size affects the throughput and latency, those experiments were repeated under different chunk sizes: 1050 B, 2078 B and 4134 B. These values allow chunks to contain up to 2, 4 and 8 Tor cells, respectively. The traffic shaper rate function was modified to maintain the same minimum and maximum throughput. In other words, the TS rate minimum and maximum values for each chunk size are: (20 µs,1000 µs), (40 µs,2000 µs), (80 µs,4000 µs).

Figure 9 depicts the throughput results with different chunk sizes. By theory the throughput should remain, approximately the same for each configuration, which is 8.5 Mbps with 50 connected clients. However it is possible to observe that for the same number of 50 connected clients, having a higher chunk sizes allow TorK to approximate throughput values to the maximum theoretical values. Since the maximum theoretical throughput remains the same across all chunk sizes, 2078 B and 4134 B turns to be more efficient than the baseline chunk size (536 B).

Baseline latency results shows increased latency values for higher number of participants as expected, having higher incidence when $\#K > 13$. Comparing baseline latency results (Figure 6) against the results depicted in Figure 10 (a), the same behavior occurs only with $\#K > 25$. Such behavior may occur due to fragmentation of several multiplexed Tor cells inside a single TorK frame chunk in combination with higher CPU load at the bridge. If one or several Tor cells are scattered over several chunks, it will require multiple traffic shaping iterations to receive a single Tor cell. Additionally, a high number of participants causes the latency to increase due to additional overhead at bridges. However, when the chunk size comprises more than 2 Tor cells (Figure 10 (b)), there is almost no large latency fluctuation even with 50 connected clients. Such results show that higher the chunk is, the less prone TorK is to Tor cell fragmentation.

E. Resisting Active Network Events

So far, the results of our experiments have shown that TorK is able to ensure the indistinguishability and unlinkability of traffic produced by k-circuits when the k-anonymity set is preserved. In this section, we assess TorK's ability to adhere to the k-anonymity settings of clients in the event of possible client churn and malicious active network manipulations aimed at disclosing the identity of clients.

Resisting deanonymization due to circuit reuse: A strawman k-anonymous circuitry established by TorK may be prone to deanonymization attacks in some specific circumstances. For instance, consider the case of a k-group of two clients where both send traffic towards the Tor network. Consider now that one of such clients leaves the group and, later, another one joins. If the client which remained

in the k-group leverages the same Tor circuit that it was using before, the adversary can trivially link that client with the Tor traffic it produced earlier. In order to avoid this potential vulnerability, TorK requires clients to establish new Tor circuits everytime a k-group is back in action.

Resisting deanonymization due to bandwidth manipulation: To test a scenario where an attacker deliberately drops client's packets in order to disclose the respective Tor circuit an experiment was conducted. Two clients, forming a k-circuit with $k = 2$, will download the same file over the Tor network. During the process, at a specific instance, all packets to and from client 2 will be dropped during a limited period, simulating a packet drop attack. It was used the Traffic Control – TC – Linux command to instruct the kernel queue discipline and emulate a total packet loss.

Figure 11 depicts the bytes statistics without (above) and with (below) bandwidth control protection. At the instance 51, in the above figure, all packets from client 2 are dropped. No bandwidth protection is employed, thus bridge continues to receive and sent traffic to client 1. After applying the bandwidth protection measure (below), the same dropping behavior on client 2 causes the bridge to stall all traffic to client 1. Around instance 80, client 2 packet's drop is lifted, causing client 2 to continuously receive traffic.

Exploiting side effects of packet drops become a complete failure to an attacker. The bandwidth protection mechanism shows to be very useful even in cases where a malicious client deliberately change the software. For instance, the attacker can recompile from source, misrepresenting the traffic shaper by omitting the send of chaff frames. Thus, bridges will only deliver Tor traffic (transported in data frames) upon receiving at least one frame, regardless its type, from all other k-circuit members.

F. Network Utilization

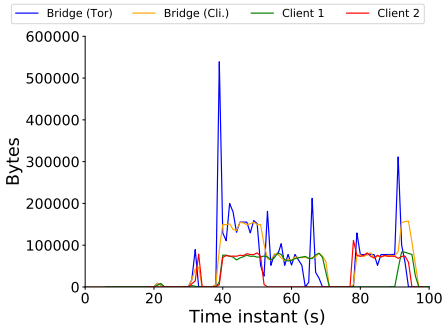
We choose to design TorK to offer protection against correlation attacks without significantly harming performance at the expense of a slight increase in network utilization.

Figure 12 depicts the bridge outgoing network utilization for the baseline configuration and additional chunk sizes and traffic shaping rates. The stats were collected during approximately, one minute and half, since was time taken to run each performance and latency tests individually. Usually, the higher the chunk size is, the higher the network usage will be, even though the traffic shaper function is adjusted to guarantee the same throughput across all configurations.

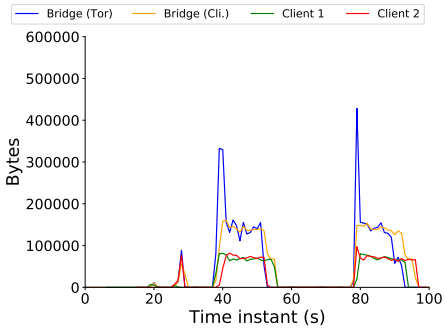
G. Hardening TorK Nodes against Runtime Compromise

Hardening TorK using SGX technologies translates in protecting the TorK memory space by adding a layer of isolation with hardware support. In this subsection we will study the level of additional security provided by SGX enclaves and a comparison regarding performance.

To evaluate the level of security provided by SGX enclaves, we setup a bridge environment with and without SGX, as we denote for the request of the section as *TorK-SGX* and *regular* TorK respectively. A network attacker

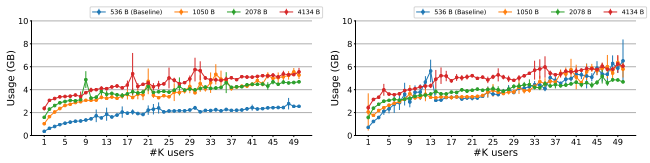


(a) 1050 B chunk (up to 2 Tor cells per chunk).



(b) 2078 B chunk (up to 4 Tor cells per chunk).

Figure 11: Bytes received by clients when dropping all packets from client 2, without and with bandwidth protection.



(a) Clients receiving chaff. (b) Clients receiving data.

Figure 12: Bridge outward (towards clients and Tor Network) network usage according chunk size/TS rate.

cannot infer the number of real clients, i.e. those that are connected to the Tor Network, from those that are chaff-only based on network traces. SGX ensures that the TorK memory space is encrypted preventing an attacker from analyze it and find whether a given frame contain chaff or real data traffic.

One of the main possible drawbacks regarding SGX is it performance impact, which is predictable to decrease due to encryption and decryption operations from and to the main memory. The baseline experiment from Section VI-C was repeated using a capable SGX machine and the throughput results are depicted in Figure 13. Our nodes are again executed within Docker containers provisioned with the same virtual CPU and RAM. Bridges and clients run in separated hosts where the bridge (SGX machine) is equipped

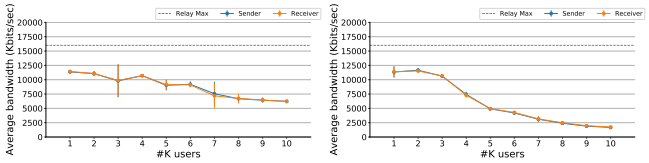


Figure 13: Throughput achieved by TorK-SGX baseline configuration (536 B chunk) with an increasing number of chaff and data clients, respectively.

with an Intel Core i9-9900K CPU. However due to hardware limitations, the SGX experiment could only be tested with 10 clients. Thus, the TS rate window used was $50\,\mu s$ to $500\,\mu s$ to accommodate the same theoretical throughput.

Notwithstanding the memory protection offered by SGX, we study the possibility to perform remote attestation of both clients and bridges. Currently, the lack of remote attestation in our prototype remains an important limitation.

VII. Conclusions

Similar to other low-latency anonymity approaches, the current Tor specification is known to be vulnerable to correlation attacks launched by global adversaries. Inspired by the concept of k-anonymity, this paper proposes the establishment of indistinguishable k-circuits over Tor. As shown in our experiments, this primitive prevents an adversary with flow correlation capabilities from identifying a Tor user amongst a group of k other users even in the presence of active attacks launched by adversaries. By tuning the configuration parameters it is possible to achieve a reasonable throughput and latency without substantially increasing network usage.

Acknowledgments

We would like to thank professor Luís Rodrigues for the initial ideas and discussion regarding the system design and attack defenses. We are also grateful to Diogo Barradas for feedback and fruitful discussions provided during the preparation of this work.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, Aug 2004.

[2] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *ACM SIGSAC CCS*, Nov 2013.

[3] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and mitigating as-level adversaries against tor," in *NDSS*, Feb 2016.

[4] S. Chakravarty, A. Stavrou, and A. D. Keromytis, "Traffic analysis against low-latency anonymity networks using available bandwidth estimation," in *European Symposium on Research in Computer Security*, Sep 2010.

[5] R. Pries, W. Yu, X. Fu, and W. Zhao, "A new replay attack against anonymous communication networks," in *IEEE International Conference on Communications*, May 2008.

[6] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *IEEE Symposium on Security and Privacy*, May 2005.

[7] K. Bock, G. Hughey, X. Qiang, and D. Levin, "Geneva: Evolving censorship evasion strategies," in *ACM SIGSAC CCS*, Nov 2019.

[8] B. Schneier, "How the nsa attacks tor/firefox users with quantum and foxacid," https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att, Oct 2013, accessed: 2020-01-05.

[9] R. Dingledine and G. Kadianakis, "One fast guard for life (or 9 months)," in *Privacy Enhancing Technologies*, Jul 2014.

[10] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar, "Defending tor from network adversaries: A case study of network path prediction," *Privacy Enhancing Technologies*, Jun 2015.

[11] C. Wacek, H. Tan, K. S. Bauer, and M. Sherr, "An empirical evaluation of relay selection in tor," in *NDSS*, Feb 2013.

[12] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in tor," in *USENIX Security Symposium*, Washington, D.C., Aug 2015.

[13] M. Edman and P. Syverson, "As-awareness in tor path selection," in *ACM SIGSAC CCS*, Nov 2009.

[14] M. Akhoondi, C. Yu, and H. V. Madhyastha, "Lastor: A low-latency as-aware tor client," in *IEEE Symposium on Security and Privacy*, May 2012.

[15] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information," in *Principles of Database Systems Symposium (PODS)*, Jun 1998.

[16] A. Campan and T. M. Truta, "Data and structural k-anonymity in social networks," in *Privacy, Security, and Trust in KDD: Second ACM SIGKDD International Workshop*, Aug 2009.

[17] A. Gupta and N. Shukla, "Privacy preservation in big data using k-anonymity algorithm with privacy key," *International Journal of Computer Applications*, Nov 2016.

[18] R. Wails, A. Johnson, D. Starin, A. Yerukhimovich, and S. D. Gordon, "Stormy: Statistics in tor by measuring securely," in *ACM SIGSAC CCS*, Nov 2019. [Online]. Available: http://doi.acm.org/10.1145/3319535.3345650

[19] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Enhancing security and privacy of tor's ecosystem by using trusted execution environments," in *USENIX on Networked Systems Design and Implementation*, Boston, MA, Mar 2017.

[20] D. Barradas, N. Santos, and L. Rodrigues, "Effective detection of multimedia protocol tunneling using machine learning," in *USENIX Security Symposium*, Aug 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=3277203.3277217

[21] S. Matic, C. Troncoso, and J. Caballero, "Dissecting tor bridges: A security evaluation of their private and public infrastructures," in *Network and Distributed System Security Symposium (NDSS)*, Feb 2017.