TorKameleon: Improving Tor's Censorship Resistance With K-anonimization and Media-based Covert Channels

João Afonso Vilalonga[1], João S. Resende[1], and Henrique Domingos[1]

NOVA LINCS & DI, FCT, Universidade NOVA de Lisboa
{j.vilalonga}@campus.fct.unl.pt {jresende,hj}@fct.unl.pt

Abstract. The use of anonymity networks such as Tor and similar tools can greatly enhance the privacy and anonymity of online communications. Tor, in particular, is currently the most widely used system for ensuring anonymity on the Internet. However, recent research has shown that Tor is vulnerable to correlation attacks carried out by state-level adversaries or colluding Internet censors. Therefore, new and more effective solutions emerged to protect online anonymity. Promising results have been achieved by implementing covert channels based on media traffic in modern anonymization systems, which have proven to be a reliable and practical approach to defend against powerful traffic correlation attacks. In this paper, we present TorKameleon, a censorship evasion solution that better protects Tor users from powerful traffic correlation attacks carried out by state-level adversaries. TorKameleon can be used either as a fully integrated Tor pluggable transport or as a standalone anonymization system that uses K-anonymization and encapsulation of user traffic in covert media channels. Our main goal is to protect users from machine and deep learning correlation attacks on anonymization networks like Tor. We have developed the TorKameleon prototype and performed extensive validations to verify the accuracy and experimental performance of the proposed solution in the Tor environment, including state-of-the-art active correlation attacks. As far as we know, we are the first to develop and study a system that uses both anonymization mechanisms described above against active correlation attacks.

Keywords: Censorship Circunvention · Tor Network · Traffic Correlation Attacks · WebRTC-based Traffic Encapsulation · K-anonymization

1 Introduction

Tor [1] is a low-latency anonymous network based on the Onion Routing protocol. It provides anonymity to its users by using network paths (i.e., Tor circuits) consisting of multiple nodes or proxies (i.e., Tor relays) to route traffic from the user to its destination. In theory, this ensures unlikability between the incoming flow and the corresponding outgoing flow leaving the Tor network.

Anonymization systems like Tor aim to provide unobservability and prevent detection while being unblockable for users [2]. However, Tor makes a tradeoff between usability and privacy. Tor alone does not obfuscate traffic characteristics, which allows attackers to use statistical analysis and machine learning models to identify pairs of input and output network flows that share similar characteristics. Therefore, Tor is vulnerable to deanonymization attacks [3,4,5,6,7,8,9], with a large percentage of circuits vulnerable to correlation attacks by network-level and state-level adversaries. Specifically, 40% are vulnerable to network-level adversaries, 42% to colluding network-level adversaries, and 85% to state-level adversaries, with up to 95% in some countries [10].

The Tor project has developed pluggable transports to prevent deanonymization attacks against Tor and relay blocking. Pluggable transports use client-side software to obfuscate Tor traffic on the user's device, along with server-side software on the entry Tor relay, to receive and deobfuscate traffic. This approach randomizes and conceals the metadata and characteristics of incoming traffic and makes it difficult to perform traffic correlation attacks. Pluggable transports improve the privacy of the Tor network and make it more resilient to blocking and censorship efforts.

However, most well-known pluggable transport systems [11,12,13] are vulnerable to deanonymization attacks and therefore may be ineffective [14,15,16,17] against a state-level adversary performing statistical analysis of traffic. To counter this, new standalone anonymization systems [18,19] have been developed that encapsulate traffic in media protocols and can resist passive correlation attacks. However, these systems are independent of Tor and have not yet been tested against active correlation attacks, in which an attacker disrupts the data stream by altering its behavior. The effectiveness of these new systems against these types of attacks has not yet been demonstrated.

To achieve the goal of defending against such attacks, we developed TorKameleon, an Internet censorship evasion tool that can be used as a fully integrated Tor pluggable transport, as a standalone system, or by combining both modes. It uses traffic encapsulation to hide Tor or user traffic in WebRTC [20] video conferencing streams or TLS tunnels and also K-anonymization user input circuits to create networks of TorKameleon proxies and users where user traffic can be fragmented and routed. We show that by encapsulating traffic in WebRTC video conferences alone, TorKameleon can withstand deep-learning-based active correlation attacks from state-level adversaries while maintaining reasonable throughput for low-throughput Internet tasks.

The contributions of this work can be summarised as follows: (1) A full specification of the TorKameleon system based on K-anonymization and WebRTC-based covert channels or secure TLS tunneling; (2) An implementation of the designed solution available as an open-source prototype; (3) A comprehensive experimental evaluation of the system in terms of performance and unobservability against active correlation attacks.

The remainder of the article is organized as follows. Section 2 gives a background and related work on Tor pluggable transport, active correlation attacks, K-anonymization, and encapsulation in media traffic. Section 3 describes the TorKameleon system model and its main features, and Section 4 describes the prototype implementation. Section 5 presents the experimental evaluation and Section 6 concludes the paper.

2 Related Work

Active and passive correlation attacks have become an increasingly pressing problem for the Tor network and other anonymizing systems, especially given the ability of large state organizations, such as intelligence agencies, to access the Internet backbone and the ability of repressive regimes to control large portions of the Internet. In this context, it is important to describe both passive and active correlation attacks, as well as some of the mechanisms and approaches we use that aim to defend against these types of attacks.

Passive and active correlation attacks Correlation attacks [4,6,8,9,7] refer to techniques used to extract information and create user profiles of a specific target or deanonymize communicating endpoints in a network. These attacks can be carried out by state-level adversaries who control multiple autonomous systems (AS) regions and collude with organizations such as ISPs. When it comes to Tor, an attacker controlling both inbound and outbound Tor relays in a circuit will attempt to correlate inbound and outbound traffic to determine which pairs of flows are part of the same overall flow. By analyzing metadata such as inter-packet arrival times, packet lengths, and volumes, the censor can confirm with a high degree of probability that a particular user is using a particular Web service. Passive correlation attacks [4,5,7] intercept traffic to obtain information, while active correlation attacks [6,9] inject a watermark into packets to uniquely identify traffic. This involves inserting a recognizable pattern into traffic passing through a specific point in the network in the hope that the manipulated traffic can be identified by the watermark at any network segment the attacker wishes to observe.

Tor pluggable transports Over the years, there has been significant development of Tor's pluggable transports to mitigate correlation attacks [11,13,12]. Currently, the Tor project supports three pluggable transports [11], namely Snowflake, Meek, and Obfs4. Although Meek and Obfs4 use different obfuscation methods, they both suffer from the same problem of being observable or easily detected and blocked [16,17,21]. Snowflake also uses WebRTC to encapsulate data but uses data channels (which are normally used to transmit arbitrary data) instead of media channels (which are used by our system). It has been shown that Snowflake is not unobservable [15].

K-anonymization Samarati and Sweeney [22] proposed K-anonymization in 1998 as a method for anonymizing privacy-sensitive database records that must be disclosed. To ensure anonymity, the group record set must encompass a minimum of K individuals. The same principle has been applied to different domains so that the probability of an attacker correctly identifying the target is at most 1/K. Efforts have been made to develop K-anonymization systems for Tor traffic, such as TorK [23] and Tir [24]. However, these systems have only been tested against passive correlation attacks and are not currently deployed in the wild.

Media tunneling solutions Media tunneling encodes data into popular streaming applications' audio or video streams, allowing for covert data transmission, and is commonly used to bypass Internet censorship. As media streaming protocols make up a significant portion of online traffic [25,26], they are ideal for covert data tunneling. However, many standalone systems [27,28,29,30] have not been proven to withstand statistical analysis and deep packet inspection with machine learning models [2,21]. Protozoa [18] and Stegozoa [19] are WebRTC-based solutions that are resilient to current machine traffic analysis techniques, but they have not been tested against active correlation attacks, are not integrated into the Tor ecosystem, and are not easily deployable.

3 System Model

TorKameleon is a Tor pluggable transport that integrates WebRTC traffic encapsulation and TLS tunneling to securely transmit Tor traffic between the user and TorKameleon Tor Bridges (Tor entry relays). In addition, it can be combined with TorKameleon proxies to form a pre-Tor network of TorKameleon proxies to route traffic between them and decouple user traffic from the user itself before it is forwarded to the Tor network via a TorKameleon Tor Bridge.

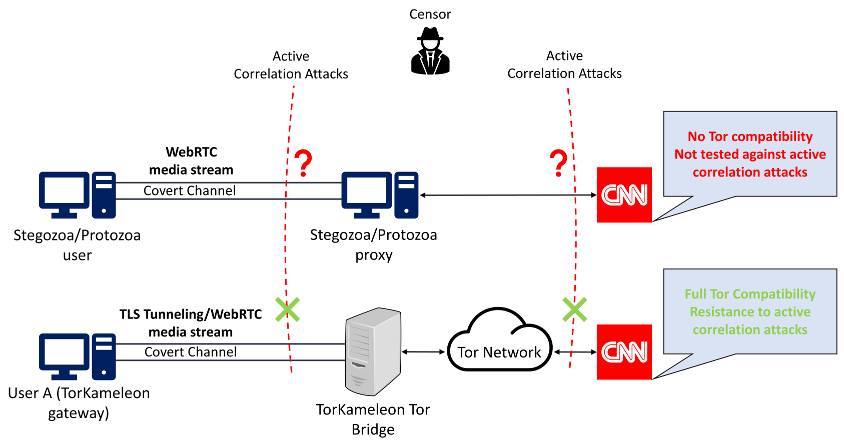

Censor
Active Correlation Attacks
Active Correlation Attacks
WebRTC media stream
Covert Channel
?
?
Stegozoa/Protozoa user
Stegozoa/Protozoa proxy
No Tor compatibility Not tested against active correlation attacks
TLS Tunneling/WebRTC media stream
Covert Channel
User A (TorKameleon gateway)
TorKameleon Tor Bridge
Tor Network
Full Tor Compatibility Resistance to active correlation attacks

Fig. 1. System model of the TorKameleon transport and the limitations of the state-of-the-art. State-of-the-art on top and our system at the bottom.

The software package consists of three main components, namely the TorKameleon Tor Bridge, the TorKameleon Proxy and the TorKameleon Gateway. The

TorKameleon Gateway runs on the user's device and serves as the primary interface for accessing the TorKameleon proxies and bridges. The TorKameleon proxy and TorKameleon bridge are used to set up proxies and bridges, respectively.

We can see in Figure 1 that one of the main advantages of the TorKameleon pluggable transport over the state-of-the-art is the Tor integration and the improved resilience of the Tor network to correlation attacks. A censor attempting to correlate incoming and outgoing Tor network flows can render its efforts ineffective through the WebRTC encapsulation mechanism. Attempting to correlate inbound WebRTC encapsulated traffic with outbound TLS traffic is a difficult challenge against modern deep learning-based correlation attacks, even more so if the traffic was routed through the Tor network.

In Figure 2, we see the normal workflow of the TorKameleon solution when used both as a pluggable transport and as a pre-Tor network of TorKameleon proxies (what we call the TorKameleon environment). First (1)) user A sets the network path to be used by the TorKameleon gateway. Then (2)) a connection is established to the first proxy through a TLS tunnel or a covert WebRTC channel embedded in a video conference between the TorKameleon gateway and the TorKameleon proxy. Then (3)) a second connection is established between the first proxy and the second proxy, also over a covert WebRTC channel embedded in a video conference or TLS tunnel. This process is repeated as many times as there are proxies in the network path established by the TorKameleon gateway. Then (4)) the last TorKameleon proxy sends the user traffic, now as Tor traffic, to the TorKameleon Tor Bridge using one of the encapsulation methods described above. Finally (5), 6)), the TorKameleon Tor Bridge forwards the Tor traffic until the final destination is reached.

The pre-Tor network of K proxies allows users to coordinate with K-1 other users to deploy their own proxies while generating covert traffic, creating a larger traffic pool that can mask individual users' traffic and allow them to access desired content while "hiding" in the crowd.

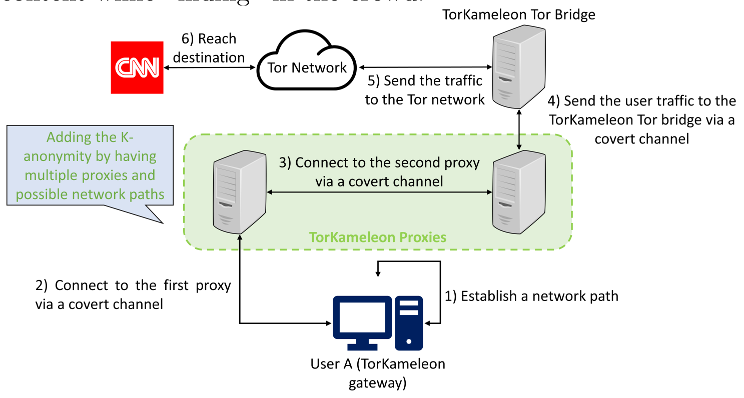

TorKameleon Tor Bridge
6) Reach destination
CNN
Tor Network
5) Send the traffic to the Tor network
4) Send the user traffic to the TorKameleon Tor bridge via a covert channel
Adding the K-anonymity by having multiple proxies and possible network paths
3) Connect to the second proxy via a covert channel
TorKameleon Proxies
2) Connect to the first proxy via a covert channel
1) Establish a network path
User A (TorKameleon gateway)

Fig. 2. System model and workflow of the TorKameleon ecosystem. When using the pluggable transport without proxies, the user connects directly to the TorKameleon Tor bridge through the TorKameleon gateway.

3.1 Threat Model

We assume a state-level adversary that can cooperate with organizations such as ISPs and other governments. The main goal of the censor is to detect and block TorKameleon usage without affecting legitimate WebRTC and TLS connections. The censor can observe, collect, analyze, and interfere with all network traffic originating from the user, TorKameleon proxies, TorKameleon bridges, and the Tor network provided that all network segments accessed are under its jurisdiction or that of the adversary parties involved.

However, we assume that the software installed on users' devices and TorKameleon Tor bridges and proxies is not tampered with and that the censor does not block or arbitrarily disrupt all WebRTC video conferences or TLS communications due to the potential collateral damage it could cause, as these protocols are widely used by organizations and services that strongly support the regimes financially.

3.2 System Architecture

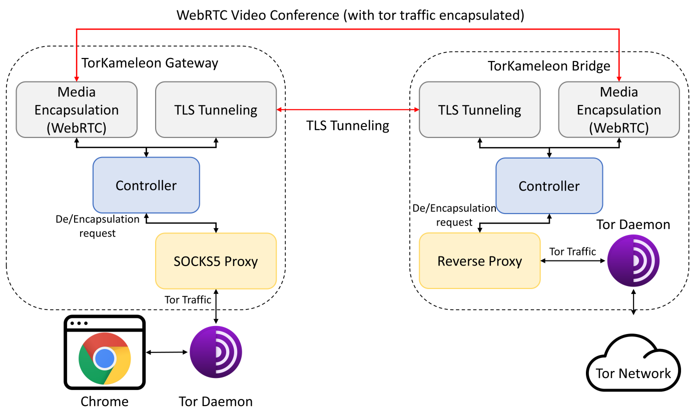


Fig. 3. System architecture of the TorKameleon pluggable transport.

In Figure 3 we can see how the system architecture of TorKameleon is built. We focus mainly on the architecture of TorKameleon's pluggable transport system since it is the most difficult to understand, but we also explain how the architecture of the TorKameleon proxy works at the end of the section. When a client wants to use the TorKameleon pluggable transport service, it first starts the TorKameleon gateway and the Tor daemon (the Tor software) on its local machine. The Tor daemon connects to the TorKameleon gateway via a SOCKS5 proxy. Traffic can now be routed through the Tor daemon to the TorKameleon gateway. This traffic can be any user traffic that is supported by Tor (the figure uses the Chrome browser as an example, but it can also be any other user application that can use the Tor daemon as a proxy). Tor traffic is now managed by the controller, which is responsible for determining how traffic is encapsulated, the size of packets to be encapsulated, and where traffic should be sent (in the figure, De/Encapsulation request). Once the traffic is encapsulated, it is sent to the TorKameleon Tor bridge (on the right of the figure). The traffic is now decapsulated depending on the type of encapsulation used, and the decapsulated Tor traffic is sent to the Tor network via the Tor daemon running on the bridge through a reverse proxy. The responses sent by the bridge follow the same flow as described above.

When you use the TorKameleon proxies, the workflow is the same, with a few differences. The traffic that goes through the TorKameleon gateway is sent directly from the user application and not from the Tor daemon, so no SOCKS5 connection is needed (a simple connection can be established from the user application to the TorKameleon gateway). The TorKameleon gateway sends traffic to other proxies via WebRTC encapsulation or TLS tunneling through a network path configured by the controller. The last proxy in the path sends traffic locally to the Tor daemon, which also runs on the proxy device, and the workflow in Figure 3 is executed to send traffic to the Tor network.

3.3 Media Traffic Encapsulation

In this section, we turn our attention to the WebRTC encapsulation component (in figure 3). A browser-based videoconferencing web application was developed using the WebRTC technology stack to enable video conferencing between two participants. The application was designed to encapsulate user traffic in WebRTC video frames, with the video serving as the carrier for the traffic.

We now explain how the WebRTC-based application is initialized and how is the data encapsulation process.

WebRTC Initialization To launch the WebRTC-based web application, a browser must be launched with the web application's code. For this purpose, when TorKameleon is launched, it automatically starts a local server that provides the web page with the scripts and files needed to run the web application. In our system, we have two participant roles for the initialization process, the initiator and the receiver of the connection, and both have different roles in establishing the connection. Therefore, we use two local servers, one that serves the web application in receiver mode and one that serves the web application in initiator mode. To automate browser functions and access the WebRTC web application, the Selenium [31] framework was used. WebSockets were necessary to connect the TorKameleon system to the WebRTC-based web application, as regular socket connections are prohibited in browsers for security reasons. The data received by the TorKameleon system is converted to a Base64 string and sent to the web application via a WebSocket connection. The web application encapsulates the data in video frames and sends the video stream to the other remote peer (e.g., a TorKameleon Tor Bridge). The data received from the remote peer is decapsulated and converted back into bytes from base64 and sent to the TorKameleon system. Finally, a signaling process [32] ensures that the connection between peers is established and is mandatory for any WebRTC connection. For this purpose, we have also designed a signaling process and a server for communication establishment.

WebRTC Application Initialization Figure 4 shows the workflow of WebRTC encapsulation. First, we split the video stream to be transmitted into its audio and video tracks so that the video track can be isolated and processed. Next, we extract the video frames from the video track and use one of the developed data encoding mechanisms (see Section 3.4) to insert the data to be encapsulated into the frames (this data is sent from the user application or Tor to the TorKameleon system and transferred to the WebRTC application via Websockets, as mentioned earlier). Finally, we combine the audio track with the new video track containing the encapsulated data to create a new media stream. This stream is then transmitted over the network to the other participant in the video conference, which can be a TorKameleon Tor bridge, a proxy, or a user. The traffic resulting from this process is no different from normal WebRTC traffic to a censor or attacker. Thus, the covert content remains unobservable. It is important to note that the read frames have already been encoded by the video codec, and therefore there is no need to implement a robust method to protect against lossy video codecs.

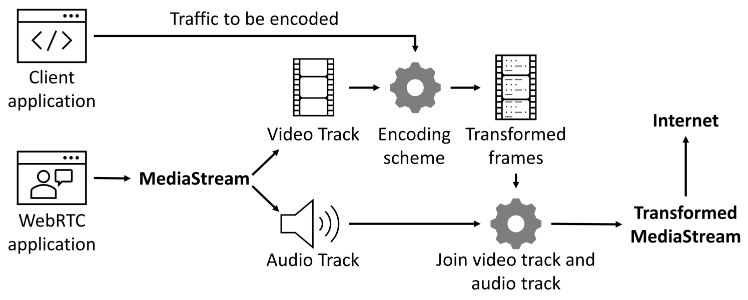


Fig. 4. Encapsulation of user traffic in WebRTC video frames workflow.

3.4 Data Encoding

With TLS encapsulation, no additional implementation effort is required other than setting up the secure socket between the communication participants (i.e., an SSL tunnel). However, this is not the case with WebRTC data encapsulation.

Once the web application receives an array of bytes from TorKameleon with a specific size of X bytes (user configurable and managed by the controller), it stores the array and waits for a video frame to be transmitted so it can encapsulate the data. This array of bytes is called a data block.

WebRTC encapsulation mode has two different modes for embedding data blocks in frames: ADD (the data block is added to the end of the frame without removing or replacing any content) and REPLACE (the content of the video frame is replaced with the data block to be encoded, leaving the frame header untouched). A brief description of each mode follows.

The ADD mode integrates the entire data block into an ADD mode packet without fragmentation, and the packet is attached to a single video frame. To ensure that the packets are sorted at the receiver, the packet contains three header fields: the packet sequence number, the length of the data block, and a special code that indicates the beginning of the packet and that the frame contains encapsulated data. The data is transmitted in the remaining part of the packet. This mode allows for higher throughput but compromises unobservability due to the increase in frame size.

The REPLACE mode is more complex than the ADD mode because it may require fragmentation of the data block into smaller blocks based on the available size of the video frame used for encapsulation. Due to the possible fragmentation of the data block and the required reassembly at the receiver side, it was necessary to keep the fields used in the ADD packet and add additional new fields. In particular, we included the following fields in the header: the LC flag field, which indicates whether this packet contains the last chunk of a particular data block (if the entire data block can be encapsulated in a single packet, or if the chunk sent is the last of a particular data block, this field is used); and the seg num field, which indicates the sequence number of the chunk so that they can be reordered. Unlike the ADD mode, the REPLACE mode may have lower throughput (depending on the frame size) because we don't increase the frame size. However, it provides better guarantees of unobservability, since the image size remains unchanged.

4 Implementation

We developed the TorKameleon prototype using Java and JavaScript. The prototype consists of two main components: the WebRTC-based web application and the TorKameleon core system. It includes about 4,000 lines of code and is publicly available on a GitHub repository for research purposes (`https://github.com/AfonsoVi/TorKameleon`). It can be used to study censorship circumvention techniques and possible extensions to the core solution by practitioners and researchers alike. The TorKameleon system is dockerized and has been developed and tested in Ubuntu 20.04.

We used JavaScript to develop the WebRTC-based web application. The web application is served by a local nodeJS server that uses the EJS framework (version 3.1.8) for HTML templating. We also used the WebRTC JavaScript APIs [33] to develop the base of the web application and the Insertable Streams API [34] to embed the covert data into the video frames. The signaling server was also developed in nodeJS and is a simple backend server that accepts secure WebSocket connections from the web application using the library Socket.IO (version 4.0).

The TorKameleon system core was developed using Java 11 and uses the Selenium framework (version 4.4.0 for Java) to programmatically launch the web application using the Chrome browser without a GUI to access local nodeJS servers. For the Selenium framework to work correctly, a special browser driver for Chrome must be used. The Chrome driver version 104.0.5112.79 was used, and a WebSockets Java library [35] implementation was utilized for communication and data exchange between the web application and video frames.

5 Evaluation

Our experimental evaluation focused on two main goals: first, to determine the resilience of the WebRTC encapsulation mechanism to active correlation attacks; and second, to evaluate the performance impact of using such mechanisms compared to normal Tor usage. In this chapter, we describe the experimental evaluation results in terms of performance, resource utilization, and unobservability.

5.1 Setup

Our experimental setup consisted of four machines, three of which were virtual private servers provided by the OVH service. These servers had the following hardware specifications: an Intel 8-core processor running at 2.4 GHz CPU, 32 GB RAM, and 2 Gbps bandwidth. The fourth machine was a local machine with an Intel i5-9300H CPU (4 cores at 2.4 GHz), and 16 GB RAM. All these machines had Ubuntu 20.04 installed, which served as the operating system for our tests.

The local machine acted as the user/client machine, while one of the VPS servers acted as a TorKameleon Tor bridge deployed in the UK. The second VPS server acted as a TURN /STUN server for the WebRTC connection and was deployed in France. Finally, the last VPS server acted as an HTTP server and was deployed in Canada. To reduce latency and reduce throughput variation between experiments, we fixed the Tor network circuit relays (three relays) from which we made observations. The middle relay was deployed in Germany, while the exit relay was in the Netherlands.

5.2 Metrics and Methodology

In our evaluation, performance was measured using two parameters: Throughput and Latency. Throughput was calculated by downloading a 250 KB file from the HTTP server, while latency was measured using the httping tool to measure the time it took to get the first byte of the response to an HTTP or HTTPS request to the server. We compared our results with those of Tor vanilla (Tor without our solution) to assess the performance impact of our system. To ensure the accuracy of our measurements, we took five samples of download time for the throughput measurement and ten samples of latency for each experiment. We then calculated the average of these samples to determine the final throughput or latency values. We repeated each experiment twice to ensure the consistency and reliability of our results.

The number of daily users of bridges is difficult to estimate because of their nature. Many bridges appear to have a small number of users, while a small number of bridges are used by most users. Based on monthly statistics [36], we decided on a maximum of 50 parallel clients, although we expect TorKameleon Tor Bridges to have fewer users in parallel.

We used TorMarker [9] for unobservability tests and measured FPR, TPR, and accuracy. FPR evaluates the reliability of the model by indicating the percentage of regular data flows that are misidentified as watermarked streams, while TPR measures the ability of the model to detect watermarks in traffic. Accuracy evaluates the model's precision in correctly classifying regular and watermarked traffic.

5.3 Performance

To evaluate the performance of the system, we conducted tests using TorKameleon as a pluggable transport.

Throughput Graph 5 shows the throughput of TorKameleon when used as a pluggable transport with TLS encapsulation. As the graph shows, there is little difference in throughput when different data block sizes are used.

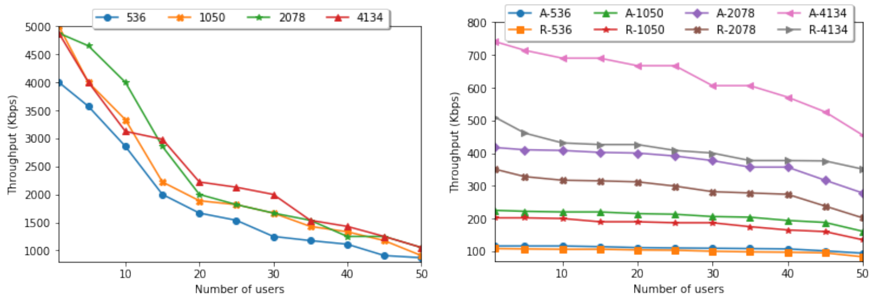


Fig. 5. Throughput graphs for different modes of encapsulation and data block sizes. Left: Throughput graph for TLS encapsulation; Right: Throughput graph for WebRTC encapsulation. A-ADD mode; R-REPLACE MODE.

Graph 5 also illustrates the throughput of a client using WebRTC-based encapsulation mode when 49 clients are added incrementally to download the same file. Of the 49 clients, four use WebRTC-based encapsulation mode, while the remaining 45 use TLS encapsulation mode. For every nine clients using TLS tunneling, the tenth client uses WebRTC-based encapsulation mode. We did this to avoid overloading the TorKameleon Tor bridge (see Section 5.4). The graph depicts the throughput values for the ADD and REPLACE modes for different data block sizes. Since TorKameleon is intended to be used and deployed voluntarily by users, a maximum of 5 WebRTC encapsulation users in a Tor bridge seems plausible.

The results we obtain for TLS encapsulation are similar to those obtained with Tor vanilla (5128 Kbps). The results for WebRTC encapsulation, for both ADD and REPLACE, show reasonable and expected reductions that do not make TorKameleon unusable for low-throughput Internet tasks (especially at higher data block sizes), compared to Tor vanilla.

Latency The graph 6 shows the latency of TLS encapsulation, WebRTC-based encapsulation in ADD mode, and WebRTC-based encapsulation in REPLACE mode, respectively, for the different data block sizes tested. It is worth noting that all encapsulation modes show similar latency values for the different data block sizes.

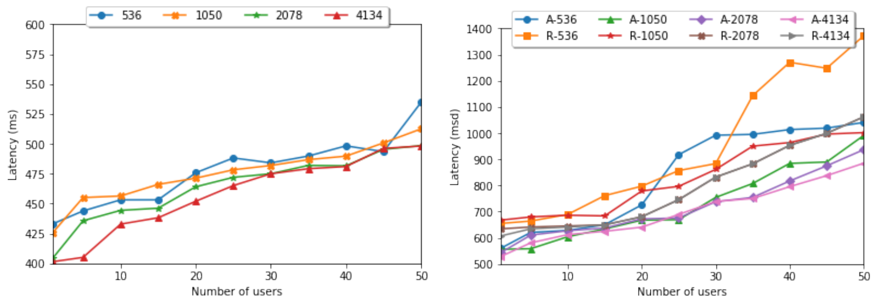


Fig. 6. Latency graphs for different modes of encapsulation and data block sizes. Left: Latency graph for TLS encapsulation; Right: Latency graph for WebRTC encapsulation. A-ADD mode; R-REPLACE MODE.

Our latency results with TLS encapsulation were comparable to those of Tor vanilla (with a measured latency of 398.2 ms for the Tor vanilla baseline). Additionally, the latency values for WebRTC encapsulation were reasonably and predictably higher than those of Tor Vanilla, in both ADD and REPLACE modes. Nonetheless, these latency increases did not render TorKameleon unusable with either encapsulation mode compared to Tor vanilla.

5.4 Resource Utilization

Relevant metrics, including CPU and RAM usage percentage, can be retrieved using the Linux top command. Table 1 shows the CPU usage of the Chrome browser (running the WebRTC web application) and the TorKameleon Java core for the TorKameleon client-side pluggable transport and the TorKameleon server-side pluggable transport on the TorKameleon Tor Bridge. Our analysis shows that the Chrome browser requires about 40% of a CPU core, while the TorKameleon Java core requires only about 1% for a WebRTC connection. The primary conclusion from the data is that the WebRTC-based web application imposes a higher CPU workload compared to other components of TorKameleon. We can also conclude that a bridge with multiple WebRTC-based encapsulation clients must have high CPU processing power to avoid throttling.

	Java Core (Client)	Java Core (Server)	Google Chrome (Client)	Google Chrome (Server)
CPU Usage (%)	1	0.3	42.6	40.2
Memory Usage (%)	2.1	0.9	1	0.5
Share Memory Size (Kb)	28923	29148	100444	1062245
Physical RAM Usage (Kb)	339088	276604	167404	169976

Table 1. Resource usage table for TorKameleon pluggable transport on the client side (client) and the server side (server).

5.5 Unobservability

To test for unobservability, we used TorMarker [9], a tool that allows small delays to be inserted into traffic to create an observable watermark in another segment of the network. To detect watermarked traffic in the network, TorMarker uses models based on deep learning.

TorMarker was trained with data sets of 60,000 packets, the same size used in its experimental evaluation. An amplitude of 120 ms (the delays induced) was chosen for the ingress flows because it provided the best results in terms of accuracy, FPR, and TPR among the tested amplitudes. We also used flow sizes of 150 packets for the same reasons as described above.

To train TorMarker, we first forwarded 30,000 packets to the HTTP server and sent them through the TorKameleon Tor bridge using TorKameleon as a pluggable transport with the WebRTC encapsulation mechanism. These 30,000 packets are referred to as regular traffic. Next, we forwarded the same 30,000 packets and embedded the watermarks into them, resulting in 30,000 watermarked packets. Both the regular traffic and the watermarked traffic, consisting of 30,000 packets each, were collected and used to train the TorMarker detection component.

Two main conclusions can be drawn from the graph 7. First, the size of the data blocks affects the unobservability of TorKameleon, with unobservability being greater for smaller data blocks; smaller blocks increase FPR and decrease TPR rate and accuracy. Second, discrepancies between REPLACE and ADD modes also increase with block size, with REPLACE mode being more resistant to watermarking attacks.

We claim that TorKameleon is unobservable for values of 536 and 1050 bytes for the ADD and REPLACE modes, based on our FPR and accuracy thresholds (false positive rate (FPR) not less than or equal to 10%, and accuracy value not greater than or equal to 85%). These thresholds were derived from the results of the experimental evaluation of TorMarker [9], deepcorr [8], and DeepCoFFEA [7], and we argue that they represent the minimum values required to classify a model as reliable, accurate, and precise. We also propose

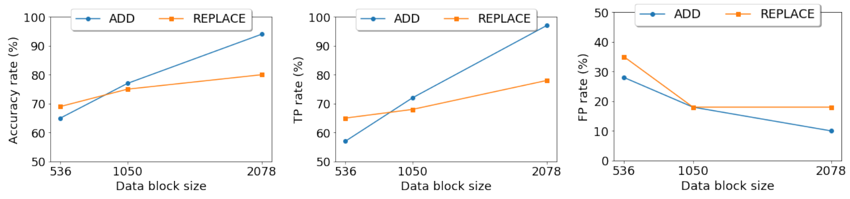


Fig. 7. Unboservability graphs for ADD and REPLACE MODE with different data block sizes. Left graph: accuracy rate graph; middle graph: TPR graph; right graph: FPR graph.

that TorKameleon can be considered unobservable for data blocks of 2078 bytes in REPLACE mode since blocking TorKameleon traffic with a data block size of 2078 bytes would cause significant collateral damage to regular WebRTC traffic, although the results are closer to our thresholds.

6 Conclusion

We have developed TorKameleon, an Internet censorship evasion tool that uses K-anonymization and encapsulation mechanisms for WebRTC and TLS traffic and can resist modern correlation attacks. We have also conducted an extensive performance and resource consumption evaluation and unobservability testing to measure TorKameleon's performance and resilience against active correlation attacks. As far as we know, TorKameleon is the first tool to integrate these two mechanisms.

TorKameleon provides a state-of-the-art, fully integrated, Tor pluggable transport with WebRTC-based covert channels that can withstand active correlation attacks. Current WebRTC-based encapsulation systems do not have Tor integration and have not been tested against active correlation attacks. We also enable the creation of a pre-Tor network consisting of K-proxies from which user traffic can be routed. This represents two main ideas that have not been combined before.

In the future, we plan to further analyze the code to identify potential optimization areas, as well as investigate different browser options for the prototype web application. We also plan to expand our experimental evaluation to include additional performance and unobservability tests to gain a more comprehensive understanding of the tool's functionality and its ability to withstand correlation attacks and traffic analysis.

References

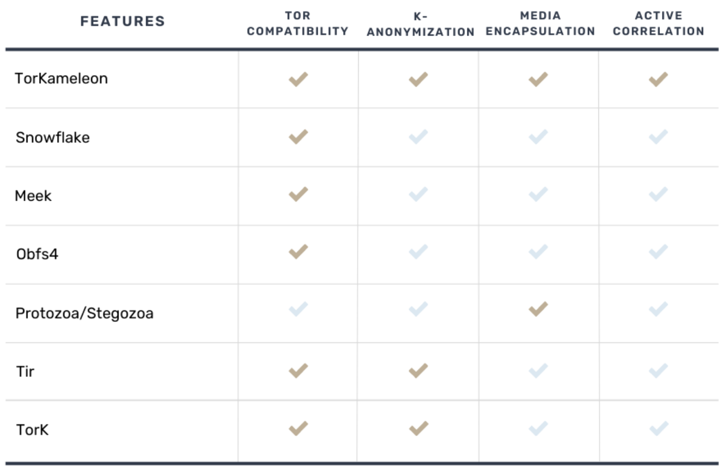
1. R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation onion router," in *13th USENIX Security Symposium (USENIX Security 04)*, (San Diego, CA), USENIX Association, Aug. 2004.

2. D. Barradas, N. Santos, and L. Rodrigues, "Effective detection of multimedia protocol tunneling using machine learning," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 169–185, USENIX Association, Aug. 2018.
3. I. Karunanayake, N. Ahmed, R. Malaney, R. Islam, and S. K. Jha, "De-anonymisation attacks on tor: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2324–2350, 2021.
4. Q. Tan, X. Wang, W. Shi, J. Tang, and Z. Tian, "An anonymity vulnerability in tor," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2574–2587, 2022.
5. Z. Guan, C. Liu, G. Xiong, Z. Li, and G. Gou, "Flowtracker: Improved flow correlation attacks with denoising and contrastive learning," *Computers & Security*, vol. 125, p. 103018, 2023.
6. F. Rezaei and A. Houmansadr, *FINN: Fingerprinting Network Flows Using Neural Networks*, p. 1011–1024. New York, NY, USA: Association for Computing Machinery, 2021.
7. S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification," in *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1915–1932, 2022.
8. M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 1 2018.
9. M. Horta, "Tor k-anonymity against deep learning watermarking attacks: validating a tor k-anonimity input circuit enforcement against a deep learning watermarking attack," Master's thesis, NOVA School of Science and Technology, 10 2022.
10. R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and mitigating as-level adversaries against tor," *CoRR*, vol. abs/1505.05173, 2015.
11. T. Project, "Circumvention." `https://tb-manual.torproject.org/circumvention/`. Accessed: 2023-03-24.
12. H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for tor bridges," CCS '12, (New York, NY, USA), p. 97–108, Association for Computing Machinery, 2012.
13. Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "Stegotorus: A camouflage proxy for the tor anonymity system," pp. 109–120, 10 2012.
14. J. Chen, G. Cheng, and H. Mei, "F-accumul: A protocol fingerprint and accumulative payload length sample-based tor-snowflake traffic-identifying framework," *Applied Sciences*, vol. 13, no. 1, 2023.
15. K. MacMillan, J. Holland, and P. Mittal, "Evaluating snowflake as an indistinguishable censorship circumvention tool," *CoRR*, vol. abs/2008.03254, 2020.
16. L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), p. 57–69, Association for Computing Machinery, 2015.
17. A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *2013 IEEE Symposium on Security and Privacy*, pp. 65–79, 2013.
18. D. Barradas, N. Santos, L. Rodrigues, and V. Nunes, "Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on webrtc," CCS '20, (New York, NY, USA), p. 35–48, Association for Computing Machinery, 2020.

19. G. Figueira, D. Barradas, and N. Santos, "Stegozoa: Enhancing webrtc covert channels with video steganography for internet censorship circumvention," 05 2022.
20. S. Loreto and S. P. Romano, "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts," *Internet Computing, IEEE*, vol. 16, pp. 68–73, 09 2012.
21. J. Geddes, M. Schuchard, and N. Hopper, "Cover your acks: Pitfalls of covert channel censorship circumvention," pp. 361–372, 11 2013.
22. P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information (abstract)," in *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, (New York, NY, USA), p. 188, Association for Computing Machinery, 1998.
23. V. M. S. Nunes and N. M. C. dos Santos, "Hardening tor against state-level traffic correlation attacks with k-anonymous circuits," Master's thesis, IST - University of Lisbon, 2021.
24. J. A. F. Teixeira and H. Domingos, "Strengthening of tor against traffic correlation with k-anonymity input circuits," Master's thesis, Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa, 2021.
25. O. Wallach, "The world's most used apps, by downstream traffic." `https://www.visualcapitalist.com/the-worlds-most-used-apps-by-downstream-traffic/`. Accessed: 2023-03-24.
26. A. Creative, "By 2021, 82% of consumer internet traffic will be video, are we ready?." `https://www.adgcreative.net/resources/by-2021-82-of-consumer-internet-traffic-will-be-video-are-we-ready/`. Accessed: 2023-03-24.
27. S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014.
28. R. McPherson, A. Houmansadr, and V. Shmatikov, "Covertcast: Using live streaming to evade internet censorship," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, 07 2016.
29. A. Houmansadr, T. Riedl, N. Borisov, and A. Singer, "Ip over voice-over-ip for censorship circumvention," 2012.
30. D. Barradas, N. Santos, and L. Rodrigues, "Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, 10 2017.
31. Selenium, "Selenium." `https://www.selenium.dev/`. Accessed: 2023-03-24.
32. M. W. Docs, "Signaling and video calling." `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Signaling_and_video_calling`. Accessed: 2023-03-24.
33. M. W. Docs, "Webrtc api." `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API`. Accessed: 2023-03-24.
34. M. W. Docs, "Insertable streams for mediastreamtrack api." `https://developer.mozilla.org/en-US/docs/Web/API/Insertable_Streams_for_MediaStreamTrack_API`. Accessed: 2023-03-24.
35. N. Rajlich, "Java-websocket." `https://github.com/TooTallNate/Java-WebSocket`. Accessed: 2023-03-24.
36. S. Matic, C. Troncoso, and J. Caballero, "Dissecting tor bridges: A security evaluation of their private and public infrastructures," in *NDSS*, 2017.

Appendix

7 Related Work - Table



FEATURES	TOR COMPATIBILITY	K-ANONYMIZATION	MEDIA ENCAPSULATION	ACTIVE CORRELATION
TorKameleon	✔	✔	✔	✔
Snowflake	✔	✓	✓	✓
Meek	✔	✓	✓	✓
Obfs4	✔	✓	✓	✓
Protozoa/Stegozoa	✓	✓	✔	✓
Tir	✔	✔	✓	✓
TorK	✔	✔	✓	✓

Fig. 8. Comparison between TorKameleon and other relevant systems presented in related work, in terms of features. *Snowflake allows traffic encapsulation in WebRTC data channels, but not in media channels.