µProxy: A Hardware Relay for Anonymous and Secure Internet Access

David Cox and David Oswald

School of Computer Science
The University of Birmingham, UK
davidcoxcontact@gmail.com, d.f.oswald@bham.ac.uk

Abstract. Privacy and anonymity on the Internet have become a serious concern. Even when anonymity tools like Tor or VPNs are used, the IP and therefore the approximate geolocation from which the user connects to such a service is still visible to an adversary who controls the network. Our proposal µProxy aims to mitigate this problem by providing a relay of user-controlled hardware proxies that allows to connect to a (potentially public) network over a large physical distance. One endpoint is connected to a public Wifi hotspot, while the other end connects (over a chain of relay nodes) to the user's computer. µProxy uses a lightweight protocol to create a secure channel between two endpoint nodes, whereas the communication can be routed over an arbitrary amount of relay nodes. The employed cryptography is based on NaCl, using Curve25519 for the key exchange as well as Salsa20 and Poly1305 for authenticated payload encryption. µProxy tunnels TCP/IP connections and can therefore be used to secure and anonymize existing, unprotected protocols. We implemented µProxy on the ESP8266, a popular Wifi microcontroller, and show that µProxy incurs a latency of 20.4 ms per hop under normal operating conditions.

Keywords: Privacy, anonymity, ESP8266, secure channel, protocol implementation, wireless networks

1 Introduction

In recent years, anonymity and privacy on the Internet has gained increasing attention. In a "post-Snowden" world, many seek to avoid ever growing government observation, be they political dissidents or simply concerned citizens. An important privacy leak is that a user's geolocation can be associated with their public IP address, giving the adversary an approximate real-time location. Often, this information is enough for an adversary to begin closing in, employing traditional methods to improve this approximation and eventually locate the user.

Current anonymity technologies, such as The Onion Routing Project (Tor) [5] or Virtual Private Networks (VPNs) aim to address this by relaying traffic through one or multiple third party nodes. This prevents an adversary from

knowing both the true source and destination of a packet. This method is effective in separating a user from the sites they are browsing, however does little to guard their location—an adversary still sees Tor or VPN traffic originating from the user's Internet connection, and can still pinpoint the respective person.

Therefore, a system is required that severs this association between public IP and actual location. In this paper, we present μProxy as a low-cost solution to this problem. μProxy achieves the goal of location anonymity using an arbitrary number of interconnected Wifi nodes that form a relay. The relay spans between the user and the network to which they wish to anonymously connect, e.g., a public Wifi hotspot. Traffic appears to stem from the final device in the relay, rather than the user. Attempts to trace the user's IP address will only lead to the relay endpoint. Further tracking requires locating an arbitrary number of (potentially covert) nodes, thus yielding an exponentially expanding search radius. Such a search is beyond the capability of all but the most well equipped and dedicated adversary and, in can case, cannot be conducted quickly.

1.1 Related Work

In the research area of unlinking a user's IP from his geolocation, two notable efforts have been made: The first is ProxyHam [4], a project that attempted to do so by providing a device forwarding Wifi connections over a 900 MHz Radio Frequency (RF) link. The unit (with an approximate cost of USD 200) could operate up to a maximum range of 2.5 km, which the developers stated would be sufficient to disrupt any physical search, should the IP be traced. The project was unexpectedly discontinued shortly before its debut at Defcon 2015 [10].

ProxyGambit is a reincarnation of ProxyHam. The project seeks to improve upon its predecessor by providing greater range and potentially global availability [13]. This is achieved via the addition of a 2G Global System for Mobile Communications (GSM) connection as a low-speed alternative when the 900 MHz link is unavailable. The project itself is in the prototype stages, and is not yet a single unit. The cost for building the device totals USD 234. ProxyGambit is, by the creators own admission, "an insecure, bare bones proof of concept", that is not in active development [13].

Tor [5] and VPN services represent the current defacto anonymity mechanisms on the Internet. As described, they effectively prevent an attacker from knowing both the source and destination of a given connection. Tor or VPNs allow a user to access online material without revealing that they have done so. Such tools are critical when the remote resources being accessed by a user are sensitive or censored. However it does little to hide that the user has connected to Tor itself, as the produced traffic is identifiable. In cases where the use of such privacy tools are grounds for suspicion or even prosecution, this is a serious problem. This, combined with the lack of location privacy are weakness in the Tor project.

1.2 Contribution and Outline

µProxy attempts to improve on the existing relay solutions (ProxyHam and ProxyGambit) outlined above by prioritizing size and affordability at the expense of range per module. We use a widely available, low-cost Wifi microcontroller, the ESP8266 (ESP) [9,8], for which ready-made modules (cf. Figure 1) can be bought for less than USD 3 per unit. Both relay and endpoint connections are then established over Wifi, which removes the need for a second band support (e.g. GSM or 900 MHz RF). This vastly reduces unit cost and form factor, and hence potentially allows covert deployment. For example, an ESP module could be hidden inside a phone charger or power outlet, which also solves the problem that the module needs a power supply. Furthermore, the reduction in unit cost

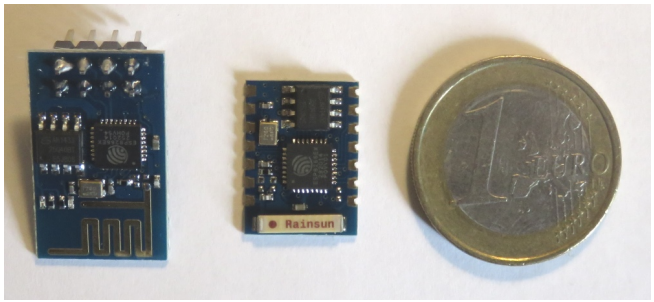

Fig. 1. Different ESP modules, Euro coin for scale.

increases the economic viability of multiple unit relays, where all previous solutions having been limited to a single pair of conspicuous devices. When used in combination with Tor or a VPN, the separation of IP address and user location prevents Tor/VPN traffic from being associated with an individual. If a Tor or VPN session, running through µProxy, is successfully reconstructed, the adversary is still unaware of the user's true location. As such, µProxy and other anonymity tools complement each other and are expected to be used in tandem. The code for µProxy is placed in the public domain.

The remainder of this paper is structured as follows: In Section 2, we describe the general design decisions behind µProxy and introduce the utilized implementation platform, the ESP. Section 3 provides details on the employed cryptographic functionality to secure the relay traffic based on the Networking and Cryptographic Library (NaCl) [1] and evaluates the system's security. In Section 4, we present the underlying protocol and evaluate the overhead of our prototypical implementation, before concluding in Section 5.

2 System Design

At its core, µProxy is formed by a Wifi *relay* (a series of nodes) with a "daisy chain" topology, as shown in Figure 2. The relay has two *endpoints*: the local endpoint to which the user connects, and the remote endpoint, which connects to the Internet, e.g., through a public Wifi hotspot. Between the endpoints, a series of *relay nodes* forwards the traffic. Note that in general, there can be N relay nodes, rather than the single one shown in Figure 2. Hidden Wifi networks are broadcast by the individual relay nodes, realized with ESP modules. These modules are placed along a physical path between the two endpoint locations. A tunnel between these two endpoints is created that seamlessly forwards all traffic. Each module connects to the Wifi of the module ahead of it in the chain, while accepting a connection from the previous module. The ability of the ESP to act both as Wifi client and access point at the same time forms the backbone of the µProxy relay.

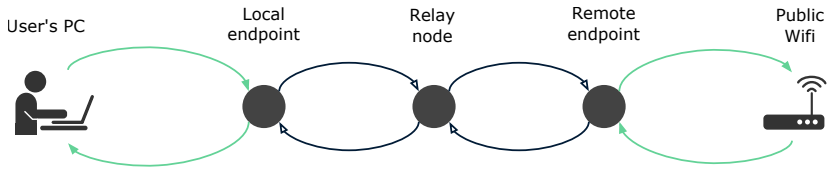

Fig. 2. µProxy topology with a single relay node. Dark arrows represent hidden Wifi connections, green arrows normal Wifi connections.

Requirements and Protocol Design The design and development of a protocol to manage the Wifi relay was required, and represented one of main undertakings of the µProxy project. The protocol has to control the set-up of the relay (with an arbitrary number of devices), provide external endpoint interfaces, support required cryptography, and perform data transmission over the relay. This protocol has to run within the restrictive embedded system environment of the ESP. As such, runtime resourse usage as well as complied code size had to be minimized. The protocol also has to be sufficiently lightweight as not not monopolize the ESPs 80 MHz processor core, as doing so would prevent the Wifi from functioning correctly.

Robustness and Reliability We require that the µProxy control protocol must be expected to run, without failure, for an indefinite period. Manually resetting relay nodes after a firmware crash would be a serious problem, as doing so might expose the existence of the relay. In order to remain covert, the individual devices must be as robust and autonomous as possible. Also, reaching devices installed in remote locations may be impractical once deployed. µProxy does not

guarantee that the end-to-end link it creates is reliable, i.e., the relay does not ensure that every packet will reach its destination. Like other hardware solutions that provide a physical networking layer link, it is assumed that transmission reliability is a responsibility of the transport layer. In the vast majority of cases this will be the TCP session underpinning the network application. This TCP session will itself ensure reliability by resending all lost packets. Reimplementing this functionality on the μProxy level would be redundant, and mandate that intra-relay communications be constructed via TCP sessions between every node. The additional complexity would increase packet processing time and thus reduce overall throughput while proving no tangible benefit to users. Therefore, the μProxy protocol does not guarantee transmission reliability.

Endpoints The endpoint nodes form a connection to the outside world; be it to the user or the remote network the relay is tunneling to. A user connects to the relay simply by connecting to the access point opened by his local endpoint. Similarly, data leaves the relay by having the remote endpoint connect to the far-off Wifi hotspot. The relay should then transparently tunnel TCP/IP connections transparently on both sides. However, this proved impossible to achieve on the ESP modules. In both cases doing so required an interface that could send or receive arbitrary packets from the device's Wifi stack. In the case of the local endpoint, all packets from the user needed to be intercepted before they were routed by the Wifi stack, and instead diverted into the relay. In the case of the remote endpoint, the ability to send arbitrary packets onto a remote network was required. These packets would be relay traffic, whose source IP had been changed to match the IP of the remote endpoint. Yet, the necessary hooks into the ESP Wifi stack do not exist in the available API. Therefore, rather than forwarding arbitrary traffic, μProxy creates TCP sockets at each end of the relay. It is the data from these connections that is then tunneled. For a user this requires to connect to a listener at the local endpoint IP. At the remote endpoint, the module connects to a preconfigured address.

This creates an end-to-end connection between the user and the server while still proving the required features of μProxy. To improve the usability of this approach, the local endpoint could provide a (automatic) proxy configuration (e.g., for HTTP(S)). An alternative solution would be to integrate the information on remote IP to connect to into the protocol and have a software component on the user's side that provides a transparent network adapter locally. Finally, the remote endpoint could also connect to a VPN server to realize transparent forwarding through the relay: the user the connects to the VPN port on the local endpoint, while the remote endpoint connects to the VPN server (with the IP pre-programmed into the endpoint's firmware).

3 Cryptography and System Security

Secure and correctly implemented cryptography is essential to μProxy, but the algorithms offered by the ESP are poorly documented and not easily accessible

to a developer. The encryption for the Wifi and TCP/IP stacks (WPA2, TLS, etc), are, to the knowledge of the author, correct implementations of standard specifications. However this functionality is private to the OS, and is not accessible through the provided Application Programming Interface (API). Therefore, it can only be used as part of a larger API call, such as connecting to a WPA2 access point. The "ESPnow" functionality [7] also provides encryption, however no documentation exists as to what scheme is employed.

NaCl Implementation Therefore, we opted to use a pure software implementation for securing the transmission channel. We selected NaCl, a cryptographic library initially presented in [1]. NaCl supports Elliptic Curve Cryptography (ECC) based key exchange using Curve25519 [2], as well as authenticated encryption using Salsa20 and Poly1305. Since it avoids any secret-dependent load addresses and branches, it is inherently protected against timing attacks.

As a starting point for porting the NaCl library to a 32-bit microcontroller such as the ESP, two variations exist: μNaCl and TweetNaCl. Rather than being optimised for desktop environments, μNaCl is a project that aims at providing optimized implementations for specific embedded microprocessors [12]. Currently, μNaCl is available for Atmel Atmega, TI MSP430 and ARM Cortex-M0. The second variation is TweetNaCl, a reimplementation of NaCl [3] that fits into 100 tweets.

We chose TweetNaCl as the foundation of the ESP NaCl implementation. Due to the code simplicity of TweetNaCl, the complexity of any modifications and implementation changes is drastically reduced. When producing an ESP compatible version of TweetNaCl, a major challenge is compiled code size. The compiled µProxy firmware is stored on the external flash of the module. The default boot behavior is to load the entire code segment of this flash image into the instruction RAM (IRAM). The unmodified TweetNaCl binaries were too large to be loaded into the ESP's limited IRAM. However, it is possible to prevent specific functions from being loading into IRAM; instead they are read from flash at runtime [14]. This allowed for the majority of the TweetNaCl binary to remain in flash, reducing IRAM usage. Rarely used functions, such as key pre-computation, remain in flash with only negligible impact on performance. However, "high traffic" areas, such as the Salsa20 stream cipher, were kept in IRAM in order to improve performance. Furthermore, we made use of the fact that the asymmetric keys of the nodes change never or infrequently. NaCl allow for pre-computation of the ECC key exchange [1], effectively reducing the computation overhead to symmetric crypto only during normal operation.

3.1 Key Distribution and Random Number Generation

Unlike typical desktop computers, the ESP does not have inbuilt support to generate cryptographically secure randomness. In fact, the device API gives no means of generating random bytes. This limits the cryptography capabilities of the device, most notably, on-device key generation. Therefore, we did not use dynamically keys entirely and opted for static key information. These key are

generated by a secure source at compile time, therefore bypassing the ESP's inherent entropy problems. Being based on public key cryptography, NaCl has the advantage that the public keys can be exchanged in the setup phase. However, as mentioned, in the implementation created for the purposes of this paper, the public key of the respective communication partner is hardcoded into the firmware of the module.

In cascade systems like Tor, re-encryption is used to hide previous header information and provide forward secrecy. Since all data takes a single path in µProxy, we did not use re-encryption: all cryptographic operations are performed on the endpoints, while the intermediary relay nodes only forward already encrypted packets. This ensures that *(i)* a compromise of a relay node does not leak any key material and *(ii)* reduces the computational load on relay nodes.

3.2 Security Considerations

For evaluating the system security of µProxy, we use the following *adversary model*: It is assumed that the adversary knows the µProxy system and can passively eavesdrop on Wifi traffic as well as actively inject packets. Furthermore, if he has physical access to an ESP module, he is able to extract the contents of the flash memory and can also replace the firmware with a modified version. Finally, we assume that the adversary cannot break the used cryptographic primitives provided by NaCl.

We do not consider the case of an adversary controlling the network beyond the remote endpoint or (parts of) the Internet. As stated, the goal of µProxy is to decouple the user's physical location from his entry point to the Internet, not to anonymize the network traffic itself. Hence, as mentioned in Section 1.2, µProxy should be combined with anonymity tools like Tor or VPN.

Relay Nodes In µProxy, packets sent across the relay are encrypted on one endpoint and decrypted on the other. All intermediary nodes do not re-encrypt data, but simply forward it to the neighboring node. Data is transmitted across the relay using local IP addresses (within the separate network between each pair of nodes), therefore, no information is leaked if an IP or MAC address is seen. When observing a packet of traffic between two relay nodes, the attacker can assume that its destination is the next relay node and its source the previous one. Besides, since hidden Wifi networks are used to connect between the relay nodes, the SSIDs do not reveal information, e.g., the location within the relay.

Successful impersonation of an intermediate node does not compromise the relay, as doing so does not yield any cryptographic keys. However, it allows an attacker to disrupt relay communications without physically locating and destroying a node. An attacker can be in the vicinity of a node and subsequently impersonate it to disrupt the relay by "black-holing" all incoming relay traffic. µProxy is not designed to protect against Denial of Service (DoS) attacks

Local Endpoint and Remote Endpoint With key information only contained within the two endpoints, they represent the main vulnerable nodes of the relay.

We assume that the local endpoint is inherently secure, as it is under direct user control, and a compromise of the local endpoint also implies that the user's location has been discovered. The connection between the user's PC and the local endpoint is secured using WPA2, hence, this connection has identical security guarantees as a standard Wifi network.

For the remote endpoint, if an adversary can overcome the obstacles of tracing and subsequently locating the external edge of the relay, then it would be possible for them to read key information from the ESP flash and decrypt the µProxy session. Alternatively, the adversary could change the ESP firmware to forward a copy of the complete traffic to his own host, or perform arbitrary modifications to the relay traffic. However, this does not substantially reduce security because once the endpoint has been located, the attacker could in any case eavesdrop or modify the decrypted traffic on the (public) hotspot network that the remote endpoint is connected to. This vulnerability is inherent to any connection across a network not in the user's control.

Furthermore, a compromise of the remote endpoint does not imply that the user's location can be instantly revealed: the adversary still only obtains information on the relay node adjacent to the remote endpoint. He then would have to discover this relay node and trace the path back to the local endpoint step-by-step.

4 Prototypical Implementation

In this section, we describe the central aspects of our µProxy implementation, and evaluate the overhead of µProxy in terms of latency added per relay node.

4.1 Relay Protocol

ESPnow As the underlying transfer protocol between relay nodes, we used ESPnow, an API functionality that provides an interface that allows ESP modules to communicate with each other below the IP layer. It allows to register a number of MAC addresses as known ESP modules. A role flag indicates the Wifi mode that the given module is currently in. Once a pair of modules have been paired, data can be sent between them in a single API call, independent of higher layers like TCP/IP. This API is used for all µProxy intra-relay communication for two reasons:

First, ESPnow allows for data transfer without the management requirements of establishing TCP sockets and subsequent sessions for each node, i.e., is simple to employ. Secondly, the payload of these messages is easily and directly accessible. This is crucial as it allows manipulation of packet data, giving the means to encrypt and decrypt traffic sent via this API using our NaCl implementation. Without ESPnow, much of the complexity and runtime computation of µProxy would be spent managing a series of TCP sessions.

A µProxy packet is composed of an 8-byte counter `ctr`, followed by an arbitrary amount of authenticated ciphertext generated by the NaCl function

`crypto_box_afternm()`. The counter is incremented for each transmitted and received packet, and taken as the lower eight byte of the 24-byte nonce used by NaCl. The upper 16-byte of the nonce are set to a fixed value, which can be specific to a μProxy instance. Having an eight byte counter allows to exchange at most 2^{64} bytes over the relay, a value that should be sufficient for practical use cases.

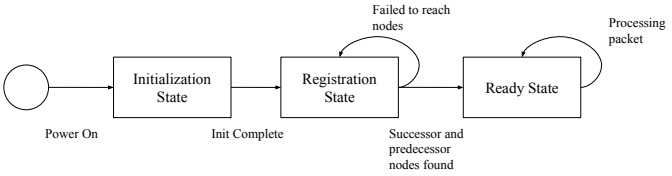

Fig. 3. Overview of the μProxy protocol.

State Machines As shown in Figure 3, the protocol begins by initializing the node. Various API calls initialize all OS functionality, and configure the access point and Wifi station. This initialization runs in a method that is automatically called after the firmware has been loaded into IRAM. Once initialization has

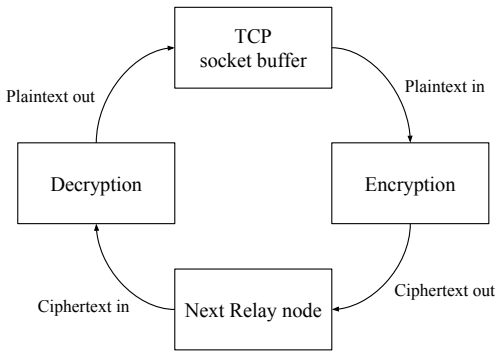

Fig. 4. Data processing loop for μProxy endpoints.

been completed, the node enters a registration phase. In this phase, the node repeatedly attempts to confirm the existence of its successor and predecessor nodes, only exiting from this state once it has found these nodes. It achieves this

by first establishing the respective Wifi connection. In the case of the predecessor, the node waits for a matching MAC address to connect to its access point, while in the case of successor, the node attempts to connect to the access point of its successor. Once these connections have been made, the MAC addresses are registered with the ESPnow API.

After being registered, the node is operational and ready to receive data. Callback functions are established for the receipt of data from another relay node, and if required, an external TCP source. The device remains in this state, processing all incoming data, until loss of power. Details on how intermediary nodes and endpoint nodes process data are shown in Figure 5 and Figure 4, respectively.

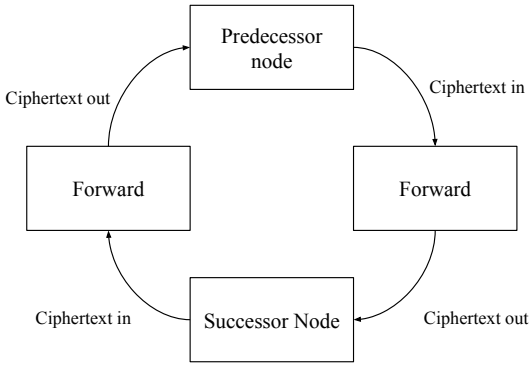

Fig. 5. Data forwarding loop for μProxy intermediary relay nodes.

Static Routing All routing between μProxy nodes is statically defined. Each module has routing information for its two neighbor nodes. This removes the need for any active routing protocol within the relay. Dynamic routing using modified variants of interior routing protocols such as Routing Information Protocol (RIP) [11] and Open Shortest Path First (OSPF) [15] were considered, allowing the relay to reconfigure after initial deployment. This would open up the possibility of adding or removing nodes within the relay and potentially allow for arbitrary network topologies. However, for reasons of security (to prevent the addition of malicious nodes) and to minimize overhead in our prototype, the decision was made to statically route the μProxy relay. The MAC address of each node's successor and predecessor are defined at compile time as part of the node's firmware.

Relay Wifi Networks Wifi networks created by relay nodes are hidden, i.e., do not broadcast their SSID. This prevents some devices from displaying the existence

of the network, while other may display the generic SSID: "Hidden Network". This measure does not hide the relay from an adversary completely, however it may help to keeping a relay unknown to nearby smartphone users. It also prevents an attacker from ascertaining a node's function from the SSID.

Robustness Care was taken to implement the relay protocol in a way that maximizes robustness. Potential failure points were identified, and fail-safes implemented that allow the relay to remain operational. Nodes will automatically attempt to reconnect to its successor if the connection was lost. All memory management exceptions, such as heap allocation failures, are caught and handled in a non-fatal way. In such situations, μProxy opts to potentially discard relay packets that cannot be allocated in order to maintain the overall operation of the relay. As discussed previously, μProxy, by design, does not provide transmission reliability, so potential packet loss is not a serious concern. Memory failures remain rare, and are only experienced when the relay is attempting to perform a transmission with throughput beyond its capacity. In these cases, the TCP session will adjust its transfer rate to match the maximum throughput of the relay, meaning that packet loss will be temporary.

Successor and predecessor nodes may be registered even if those nodes are in the active state. This allows a node to re-establish itself within the relay if it experienced power loss. This is of special importance, as continuous power availability cannot be guaranteed if devices are placed at a location outside the user's control. As a last resort, nodes are able to re-establish the relay after a kernel exception. Devices then reboot and follow the protocol steps as normal. The static design of the protocol and much of the node configuration information allows the node to reach the registration phase using only information stored in the device memory.

The result of these implementation decisions is a relay that can operate indefinitely (given power supply), allowing μProxy to provide a secure link that is available to the user without the need for post-deployment maintenance.

4.2 Performance Evaluation

Unlike other solutions, μProxy supports a relay of an arbitrary number of hops a opposed to a single pair of devices. Hence, relay scalability and the impact of length on performance become an important consideration.

To evaluate this aspect, we estimated how the number of nodes N affects the round trip time T. We define the end of the relay to be the point at which data is pushed out to the TCP socket buffer, rather than the time of receipt on an external machine. This is to remove the impact of non-relay hardware from any performance evaluation, as this hardware is outside the control of μProxy. We can define T as:

$$T = 2\,xN = 2\,\left((N-1)\,W + E + D + NK\right)$$

where x is the sum of all latency incurred by using the relay. This latency has three sources: *(i)* the series of Wifi connections that are used to transmit

data between relay nodes (W), *(ii)* the cost of cryptography at both the of the endpoints (E, D), and *(iii)* the latency incurred by other internal computations such as passing the message between internal functions (K). The term K turned out to be negligible and therefore is absorbed into W, yielding:

$$T = (2N - 2)\, W + 2E + 2D$$

From this formula, it is clear that μProxy latency scales linearly with the number of nodes. To quantify this further, data was collected on the time taken to encrypt and decrypt data as well as the overall round trip time of a packet. These values are then substituted into the formula to provide an estimate of W.

Although the cryptographic term is represented as constants E, D, this is not strictly true: The value is highly dependent of the length of the message. This relation is linear, as can be seen in the test data. Hereafter it is assumed that the relay is only transporting packets of a size of 225 bytes. This is the ESPnow Maximum Transmission Unit (MTU), and represents the worst case scenario for relay performance. Other user applications such as Telnet [16], which send exclusively single byte packets, will produce significantly better results for the cryptographic variable. At maximum packet length, average encrypt and decrypt times were $E = 1.9\,\text{ms}$ and $D = 2.3\,\text{ms}$, respectively.

It was not possible to to accurately measure the time taken to transmit over a single Wifi connection, as this would require the synchronization of separate device clocks. Therefore, the packet round trip time T was measured and interpolated to represent a single Wifi connection. Testing was conducted with a relay of $N = 3$ devices, yielding $T = 90\,\text{ms}$.

With the above data, we have $W = \frac{T}{4} - 2.1\,\text{ms} = 20.4\,\text{ms}$. This value represents the latency incurred per node beyond the two endpoints. For example, a relay of $N = 11$ nodes would have an expected round trip time of $T = 416.4\,\text{ms}$. The cost for the ESP modules to form such a relay would be USD 35 at most.

Regarding the range, the manufacturer Espressif report the module of have a maximum open-air range of 360 m [6]. This is possibly an overstatement and likely only accounts for an access point being visible on a specialized Wifi receiver. μProxy requires an access point to be visible to other ESP modules, and have a signal strength sufficient for data transmission. Our testing has shown that a value closer to 200 m is appropriate, with 100 m being an estimate with safety margin. Hence, an 11-node relay could stretch over 1 km. The latency of round trip time of $T = 416.4\,\text{ms}$ is high compared to typical connections (cables, long-range RF) over the same distance. Still, in our tests, this latency does not prevent μProxy from delivering a usable browsing experience. Furthermore, testing showed that round trip times were more than halved when sending smaller packets, e.g., representative of an SSH connection.

5 Conclusion and Future Work

In this paper we presented μProxy, a relay solution to anonymously connect to the Internet through a remote Wifi, e.g., a public hotspot. Due to the low-cost nature of the ESP, long relays can be established at minimal cost. The

measured delay per relay hop of approximately 20 ms is sufficiently low to allow for practical use of this relay in cases where geolocation privacy is crucial. To secure the relay channel, NaCl is ported to the ESP and used to encrypt and authenticate the tunneled traffic.

With respect to future work, there are several open problems in µProxy: First, it would be desirable to establish a fully transparent relay, either through potentially available functions on the side of the ESP or by providing a custom network driver for the user's OS. Secondly, a more dynamic approach of key management should be investigated, for example, exchanging long-term keys during the setup phase by placing the respective endpoints in a shielded environment and transmitting the public keys to the other partner. The necessary extensions to the protocol would be minimal and facilitate the actual use of the devices. An additional aspect would be to provide forward security by integrating rekeying into the protocol, e.g., by exchanging ephemeral keys at certain intervals, authenticated with the long-term keys.

To facilitate such modifications and improvements as well as security reviews, we placed the source code of µProxy in the public domain and published it online at: https://github.com/david-oswald/microproxy.

References

1. D. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In A. Hevia and G. Neven, editors, *Proceedings of LatinCrypt 2012*, page 159–176. Springer, Nov. 2012.
2. D. J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *International Conference on Theory and Practice in Public-Key Cryptography – PKC'06*, pages 207–228. Springer, 2006.
3. D. J. Bernstein, B. van Gastel, W. Janssen, T. Lange, P. Schwabe, and S. Smetsers. TweetNaCl: a crypto library in 100 tweets. In A. Hevia and G. Neven, editors, *Proceedings of LatinCrypt 2014*. Springer, Sept. 2014.
4. B. Caudill. Paranoia and ProxyHam: High-Stakes Anonymity on the Internet. https://www.defcon.org/html/defcon-23/dc-23-speakers.html, Oct. 2015.
5. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, SSYM'04. USENIX Association, 2004.
6. Espressif. Espressif Smart Connectivity Platform: ESP8266. available online: https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf, Oct. 2013.
7. Espressif. ESP-NOW User Guide. available online: https://espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf, July 2016.
8. Espressif. ESP8266 Datasheet. available online: https://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf, Feb. 2016.
9. Espressif. ESP8266EX. available online: http://espressif.com/products/hardware/esp8266ex/overview/, Mar. 2016.
10. A. Greenberg. Online Anonymity Project ProxyHam Mysteriously Vanishes. available online: http://www.wired.com/2015/07/online-anonymity-project-proxyham-mysteriously-vanishes/, July 2015.

11. C. Hedrick. Routing Information Protocol. RFC 1058, RFC Editor, June 1988.
12. M. Hutter and P. Schwabe. μNaCl – The Networking and Cryptography library for microcontrollers. available online: http://munacl.cryptojedi.org/index.shtml, Oct. 2015.
13. S. Kamkar. ProxyGambit. available online: http://samy.pl/proxygambit/, Oct. 2015.
14. C. Lohr. How to directly program an inexpensive ESP8266 Wifi module. available online: http://hackaday.com/2015/03/18/how-to-directly-program-an-inexpensive-esp8266-wifi-module/, Mar. 2015.
15. J. Moy. OSPF Version 2. RFC 2178, RFC Editor, July 1997. available online: https://www.rfc-editor.org/info/rfc2178.
16. T. O'Sullivan. Telnet Protocol: A Proposed Document. RFC 0495, RFC Editor, May 1971.