Free and Open
COMMUNICATIONS
https://foci.community on the Internet

Security Notions for Fully Encrypted Protocols

Ellis Fenske
U.S. Naval Academy

Aaron Johnson
U.S. Naval Research Laboratory

Free and Open Communications on the Internet 2023(1), 24-29

Security Notions for Fully Encrypted Protocols

Ellis Fenske
U.S. Naval Academy
fenske@usna.edu

Aaron Johnson
U.S. Naval Research Laboratory
aaron.m.johnson@nrl.navy.mil

Abstract

A common strategy to avoid network censorship is to use a *fully encrypted protocol* (FEP). Such protocols are designed to make all bytes appear uniformly random and make packet sizes unpredictable. These goals depend on cryptographic assumptions, but no mathematical formalization of them has been presented. We give the first definitions for FEPs, examples of how existing protocols fail to satisfy them, and a novel protocol that does satisfy them.

1 Introduction

A common strategy to avoid network censorship is to use *fully encrypted protocols* (FEPs) [16, 30, 31], also called "look-like-nothing", "obfuscated", or "randomized" protocols [14]. FEPs attempt to eliminate message features that would identify the protocol being used. Examples of this strategy include Obfs4 [19], Shadowsocks [2], Obfuscated OpenSSH [1], and VMess [28]. Such protocols are typically used to connect to a proxy to relay data to and from the intended (and censored) destination. This strategy has proven effective in practice, even though it requires a proxy unknown to the adversary, an adversary unwilling to block connections using protocols it does not recognize [25], and an adversary that does not employ sophisticated detection techniques that can recognize fully encrypted protocols [29]. Popular systems employing FEPs include Tor [10], V2Ray [26], Outline VPN [20], and OpenVPN (via the XOR patch [32]).

In addition to censorship circumvention, fully encrypting *all* Internet traffic would significantly enhance privacy. Recent encrypted transports, such as TLS 1.3 and QUIC, have already applied encryption to some sensitive meta-data (e.g. SNI encryption). FEPs represent a natural endpoint of such development — all protocol meta-data is hidden, preventing an adversary from learning which cryptographic techniques are used, how much genuine data is transmitted, and even the identity of the FEP itself. Hiding such data can prevent clients from being fingerprinted [14], reduce the effectiveness of traffic analysis based on packet lengths [24, 27], and prevent blocking application-specific encrypted protocols [9, 18, 23].

The common goal of FEPs is to eliminate identifying patterns by randomizing the protocol messages. FEPs attempt to make each byte in a network packet to appear uniformly and independently random to an observer, and they frequently attempt to eliminate identifiable packet sizes by modifying them as well. These goals are intertwined with cryptographic security notions, as encryption is used to make data appear random, and message sizes are affected by length requirements of the cryptosystems employed. However, no cryptographic definition of FEPs has yet been given. Closely-related definitions include the IND\$-CPA notion of Rogaway [22] and "boundary hiding" from Boldyreva et al. [8], but they only apply to the encryption of individual messages and to the existing boundaries within a message sequence, respectively.

In this work, we (1) provide mathematical definitions for fully encrypted protocols, (2) show how existing protocols fail to satisfy them, and (3) present a new protocol that is provably fully encrypted. We focus on the task of designing an encrypted protocol to operate over an existing transport protocol, as solutions lower in the network stack cannot be used without raw-socket permissions or a modified network stack. Our results represent work in progress and are currently limited to the *datastream* setting [12], where a reliable-delivery protocol like TCP is assumed to be used, and to the *data-transport* phase, where a symmetric key is assumed to already have been shared. Moreover, we do not consider the related problem of proxy-detection via active probing [7, 13, 15].

2 Security Definitions

We give our model and notation before introducing novel security notions for fully encrypted protocols.

2.1 Model and Notation

We assume an AEAD encryption scheme $(\textsc{Init}(), \textsc{Enc}_k(\texttt{ad}, m), \textsc{Dec}_k(c))$, and we assume the encryption is randomized in that a random IV is produced implicitly whenever $\textsc{Enc}$ is called and included in the ciphertext output. We assume that when an input fails to decrypt, $\textsc{Dec}$ outputs $\bot_{\textsc{Dec}}$. We require and assume the AEAD scheme is *length-regular*, meaning that, for any two messages m_1 and m_2 with $|m_1| = |m_2|$, associated data values d_1 and d_2, and keys k_1 and k_2, $|\textsc{Enc}_{k_1}(d_1, m_1)| = |\textsc{Enc}_{k_2}(d_2, m_2)|$. Finally, we require the AEAD scheme to have ciphertexts indistinguishable from random (*i.e.* IND\$-CPA [22]). Typical AEAD schemes, such as AES-GCM or ChaCha20/Poly1305, satisfy all of these requirements.

We express protocol functionality using the channel model of Fischlin *et al.* [12]. It consists of the algorithms (1) $\textsc{Init}()$, which sets up state; (2) $\textsc{Send}(\texttt{st}_S, m, p, f)$, which takes sender state $\texttt{st}_S$, plaintext message m, desired output length p, and flush flag f, and outputs a ciphertext; and (3) $\textsc{Recv}(\texttt{st}_R, c)$, which takes receiver state $\texttt{st}_R$ and ciphertext c and outputs a plaintext (possibly empty or an error symbol) and a channel closure flag. These provide a unidirectional channel permitting arbitrary plaintext fragmentation, where inputs may be partially buffered by $\textsc{Send}$, and ciphertext fragmentation, where partial ciphertexts may be delivered to $\textsc{Recv}$. A protocol in this model is *correct* if (1) after a series of $\textsc{Send}$ calls with the last one given $f = 1$, passing the ciphertexts from SEND to RECV (possibly fragmented) yields the plaintext messages submitted to SEND (possibly fragmented); (2) passing any prefix of the ciphertexts from any sequence of SEND calls to RECV (possibly fragmented) produces some valid prefix of the plaintext inputs (possibly fragmented) and no channel closures.

This model includes modifications to Fischlin *et al.* [12] to express important characteristics of FEPs. First, since the termination of a TCP connection is observable to a passive adversary and is a typical protocol response to malicious behavior, we include it as a potential output of RECV (termination by sender is not modeled). Second, we include the desired output length p as an additional SEND parameter to allow for our length shaping security definition (Section 2.4).

We operate primarily on strings of bytes. ε denotes the empty string. $s\|t$ denotes concatenation of s and t. $x[i..j]$ indicates the subsequence of x from the ith to jth bytes, with indexing beginning at 1. We use % to mean "without the prefix", i.e., $\mathtt{xyxz}\%\mathtt{xy} = \mathtt{xz}$. $[\![A,B]\!]$ denotes the longest common prefix between A and B. $\preceq$ ($\prec$) denotes (strict) string prefixes.

We adopt the stream-based security notions for integrity and confidentiality without modification [12]: IND-CCFA (Indistinguishability under Chosen Ciphertext-Fragment Attacks) and INT-CST (Integrity of Ciphertext Streams). Our new definitions for FEPs do not necessarily imply these security notions, and so a FEP should satisfy them as well.

Our passive and active security notions will both be quantified using the security experiment in Algorithm 1. Following Fischlin *et al.*, it gives the adversary a send oracle in the passive attack and adds a receive oracle for the active case. Note that the experiment defines the following global variables: C_S, which will store the ciphertext outputs of the channel SEND; C_R, which will store the ciphertext inputs to the channel RECV; $\mathtt{st}_S$, the state of the channel SEND; and $\mathtt{st}_R$, the state of the channel RECV. Definition 1 then requires that the adversary's success in the security experiment be negligible.

Algorithm 1 $\mathsf{Exp}^{\text{FEP-ATK},b}_{\mathcal{A}}(1^\lambda)$

```
1: $(\mathtt{st}_S, \mathtt{st}_R) \leftarrow \textsc{Init}(1^\lambda)$
2: $C_S \leftarrow \varepsilon$
3: $C_R \leftarrow \varepsilon$
4: $b' \leftarrow \begin{cases} \mathcal{A}^{O^b_{\textsc{Send}}()}(1^\lambda) & \text{if ATK = CPFA} \\ \mathcal{A}^{O^b_{\textsc{Send}}(),O^b_{\textsc{Recv}}()}(1^\lambda) & \text{if ATK = CCFA} \end{cases}$
5: return $b' = b$
```

Definition 1. *A channel satisfies FEP-ATK if, for a security parameter λ and PPT adversary $\mathcal{A}$, $\left|P\left[\mathsf{Exp}^{\text{FEP-ATK},b}_{\mathcal{A}}(1^\lambda) = 1\right] - 1/2\right|$ is negligible in λ.*

We motivate each of the following definitions with a brief example of an existing fully encrypted protocol that fails to satisfy the corresponding security notion.

2.2 Passive Security

Fully encrypted protocols have adopted the informal security goal of attempting to appear like random bytes to passive adversaries. For example, Obfuscated OpenSSH [1] is a plugin for SSH that requires the client send a random value to initiate a connection, which is then used to derive a symmetric key. The key is used to encrypt the messages that follow. While the cipher suite in use may satisfy IND\$-CPA, if the plugin is configured without an optional pre-shared key, all key material is observable so protocol is easily distinguishable from random and does not satisfy our notion for passive security, FEP-CPFA (Fully Encrypted Protocol under Chosen Plaintext-Fragment Attacks).

The security-experiment oracle $O_{\textsc{Recv}}$ is defined in Algorithm 2. The oracle allows an adversary to input arbitrary plaintexts, and if the challenge bit is set (i.e., $b = 1$) it uses a RAND function to replace the ciphertext output with a random string of the same length. The experiment returns 1 if adversary guesses the challenge bit and returns 0 otherwise.

Algorithm 2 $O^b_{\textsc{Send}}(m, p, f)$

```
1: $(\mathtt{st}_S, c_0) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p, f)$
2: $c_1 \leftarrow \textsc{Rand}(|c_0|)$
3: $C_S \leftarrow C_S \| c_b$
4: return $c_b$ to $\mathcal{A}$
```

FEP-CPFA is not hard to satisfy, but it does preclude the use of plaintext metadata fields. In particular, plaintext length headers are commonly used to identify ciphertext boundaries within the stream of incoming data, while FEPs must use a different method.

2.3 Active Security

Even if protocol outputs are indistinguishable from random to a passive adversary, protocols may nonetheless leak identifying information when confronted with modified or forged messages. For example, the data transport phase of the Obfs4 protocol [19] is structured as a sequence of AEAD encrypted ciphertexts prepended by their lengths in the form of two-byte unsigned integers. The length field is encrypted by an unauthenticated stream cipher. When any Obfs4 ciphertext fails to decrypt, the TCP connection is immediately terminated on the receiver side. Thus, an active adversary can modify this length field in transit so that it is k, causing the receiver to interpret the following k bytes as a ciphertext, which causes a predictable channel closure exactly when Obfs4 is in use. We have experimentally confirmed this behavior, and since the messages sent by the client to initiate a standard Tor connection have consistent lengths, we were able to use it to reliably distinguish Tor-over-Obfs4 connections from other FEPs.

This attack and others like it motivate our second security notion, FEP-CCFA (Fully Encrypted Protocol under Chosen Ciphertext-Fragment Attacks). Algorithm 3 presents the receive oracle $O_{\textsc{Recv}}$ defining this notion, which is similar to the analogous oracle used in the definitions of IND-CCFA and INT-CST [12]. The stream of ciphertext inputs given to $\mathcal{O}_{\textsc{Recv}}$ is considered "in sync" until ciphertext fragments are given to $\mathcal{O}_{\textsc{Recv}}$ that differ from or exceed the genuine stream produced by $\mathcal{O}_{\textsc{Send}}$. While the stream is in sync, outputs are suppressed, mimicking the IND-CCA definition in that the adversary is not allowed to decrypt the challenge ciphertext. After the stream is out of sync, the adversary is given the plaintext outputs. The adversary *always* obtains channel closures, if any, regardless of the sync state of the oracle.

In response to ciphertext manipulation, our definition requires that a fully encrypted protocol simply holds the channel open and drops incoming messages, in line with the recommendation from Frolov *et al.* [13]. Doing so avoids leaking any observable information about the point at which the error was detected, which could leak information about ciphertext boundaries in the stream, and it is achievable by a variety of transport protocol designs.

Observe that achieving FEP-CCFA requires that the ciphertext stream be authenticated. In the $b = 1$ case no messages or channel closures are ever sent to the adversary, and so, to satisfy the definition, in the $b = 0$ case the protocol must detect the de-sync and stop delivering messages at that point.

Algorithm 3 $\mathcal{O}^b_{\textsc{Recv}}(c)$

```
if $b = 0$ then
    if $\mathtt{sync} = 0$ then  // already out of sync with Send
        $(\mathtt{st}_R, m, \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
        return $(m, \mathtt{cl})$ to $\mathcal{A}$
    else if $C_R \| c \preceq C_S$ then  // in sync with Send
        $(\mathtt{st}_R, m, \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
        $C_R \leftarrow C_R \| c$
        return $(\varepsilon, \mathtt{cl})$ to $\mathcal{A}$
    else // either deviating or exceeding Send outputs
        if $C_R \prec [\![C_R \| c, C_S]\!]$ then  // $c$ partially in sync
            $\widetilde{c} \leftarrow [\![C_R \| c, C_S]\!] \,\%\, C_R$  // in-sync part
            $(\widetilde{\mathtt{st}_R}, \widetilde{m}, \widetilde{\mathtt{cl}}) \leftarrow \textsc{Recv}(\mathtt{st}_R, \widetilde{c})$
            $(\mathtt{st}_R, m, \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
            $m' \leftarrow m \,\%\, [\![m, \widetilde{m}]\!]$  // out-of-sync output
        else // none of $c$ in sync with Send outputs
            $(\mathtt{st}_R, m', \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
        if $C_S \not\preceq C_R \| c$ or $m' \neq \varepsilon$ then
            $\mathtt{sync} \leftarrow 0$
        $C_R \leftarrow C_R \| c$
        return $(m', \mathtt{cl})$ to $\mathcal{A}$
else // $b = 1$, return empty string and keep channel open
    return $(\varepsilon, 0)$ to $\mathcal{A}$
```

2.4 Length Shaping

Traffic analysis is a serious concern for FEPs in practice [32], but their approaches to modifying message lengths are incomplete or non-existent. For example, while the widely-used Shadowsocks proxy protocol [2] does satisfy FEP-CCFA, it has no support for modifying message lengths. Shadowsocks simply sends an AEAD-encrypted fixed-size length field followed by the AEAD encryption of the message. Obfs4 has a similar design, and while it does enable a specified amount of padding to be added to the message, it does not enable ciphertext lengths to be *reduced* (e.g. by fragmenting a ciphertext). A result of these choices is that the protocols have minimum output message lengths, and their slightly different designs make these values distinct. Thus, when when they are used to transport short messages, these transports can be distinguished based on the minimum observed message length alone. We note that the wire formats of these protocols need not necessarily change to enable arbitrary message lengths, but the implementations would need to include some kind of ciphertext fragmentation and buffering.

To avoid traffic analysis based on message length, we give a novel security notion for FEPs called *length shaping*, in part inspired by real-world concerns [11]. It requires that the protocol be capable of producing any given number p of bytes of valid ciphertext data on command. While protocols like Obfs4 will add specified padding to the *input*, we require length shaping to apply to the *output* to provide greater control over the lengths of network messages. Length shaping precludes the existence of a minimum message length, and, more generally, the output lengths can be shaped arbitrarily, such as into a data-independent pattern or that of a different FEP. Definition 2 formalizes this new notion.

Definition 2. *A channel satisfies length shaping if, for any state st_S, message m, and integer $p \geq 0$, with $(\_, c_f) \leftarrow$ $\textsc{Send}(st_S, m, p, f)$, $f \in \{0, 1\}$, $|c_0| = p$ and $|c_1| \geq p$.*

Note that the definition allows the protocol to produce more bytes than requested when the flush flag is set. Such an exception is necessary to satisfy the channel correctness requirement because Send may be called with the flush flag set and insufficient output bytes to transport the required data. Also observe that the definition permits Send to be called with length shaping "turned off" by setting $p < 0$, in which case it imposes no requirements on protocol behavior. Furthermore, length shaping requires arbitrary buffering because Send can be called with $p = f = 0$ and $m \neq \varepsilon$, and it also requires a padding mechanism because Send can be called with $m = \varepsilon$ and $p > 0$, $f = 0$.

3 A Stream-based Construction

Since no existing FEP satisfies all our security notions simultaneously, we present a novel construction that does. At a high level, the construction keeps an output buffer, which is extended as needed to accommodate the output length parameter; either by encrypting some plaintext in the input buffer, building a length-payload block pair where some or all of the payload block consists of padding bytes, or creating a length

Figure 1: A Stream-based Fully Encrypted Protocol

```
INIT($1^\lambda$):
    $K \leftarrow_\$ \mathcal{K}$
    $\mathtt{st}_S = (K, 0, \varepsilon, \varepsilon)$
    $\mathtt{st}_R = (K, 0, \varepsilon, 0)$
    return $(\mathtt{st}_S, \mathtt{st}_R)$
```

```
SEND($\mathtt{st}_S, m, p, f$):
    $(K, \mathtt{seqno}, \mathtt{buf}, \mathtt{obuf}) \leftarrow \mathtt{st}_S$
    $\mathtt{buf} \leftarrow \mathtt{buf} \| m$
    if $(|\mathtt{obuf}| \geq p) \wedge ((f = 0) \vee (\mathtt{buf} = \varepsilon))$ then
        $c \leftarrow \mathtt{obuf}[1..\mathrm{MAX}(p, f * |\mathtt{obuf}|)]$
        $\mathtt{obuf} \leftarrow \mathtt{obuf} \% c$
        return $(\mathtt{st}_S, c)$
    $o \leftarrow \mathrm{MIN}(|\mathtt{buf}|, \mathtt{il})$
    $\ell_p \leftarrow 0$ // Padding length
    $\ell_c \leftarrow |\mathrm{ENC}(\mathtt{seqno}, 0^{(2+o)})|$ // Ciphertext length
    while $(|\mathtt{obuf}| + \ell_c + \ell_{\mathtt{len}} < p) \wedge (\ell_c \neq \mathtt{ol})$ do
        $\ell_p \leftarrow \ell_p + 1$
        $\ell_c \leftarrow |\mathrm{ENC}(\mathtt{seqno}, 0^{(2+o+\ell_p)})|$
    $c \leftarrow \mathrm{ENC}(\mathtt{seqno}, \ell_c)$
    $\mathtt{seqno} \leftarrow \mathtt{seqno} + 1$
    if $\ell_c > 0$ then
        $m' \leftarrow \ell_p \| (0^{\ell_p}) \| \mathtt{buf}[1..o]$
        $\mathtt{buf} \leftarrow \mathtt{buf} \% \mathtt{buf}[1..o]$
        $c \leftarrow c \| \mathrm{ENC}(\mathtt{seqno}, m')$
        $\mathtt{seqno} \leftarrow \mathtt{seqno} + 1$
    $\mathtt{obuf} \leftarrow \mathtt{obuf} \| c$
    return SEND($\mathtt{st}_S, \varepsilon, p, f$)
```

```
RECV($\mathtt{st}_R, c$):
    $(K, \mathtt{seqno}, \mathtt{buf}, \mathtt{fail}) \leftarrow \mathtt{st}_R$
    if $\mathtt{fail} = 1$ then
        return $(\mathtt{st}_R, \varepsilon, 0)$
    $\mathtt{buf} \leftarrow \mathtt{buf} \| c$
    $m \leftarrow \varepsilon$
    while $|\mathtt{buf}| \geq \ell_{\mathtt{len}}$ do
        $\ell_c \leftarrow \mathrm{DEC}(\mathtt{seqno}, \mathtt{buf}[1..\ell_{\mathtt{len}}])$
        if $\ell_c = \perp_{\mathrm{DEC}}$ then
            $\mathtt{fail} \leftarrow 1$
            return $(\mathtt{st}_R, \varepsilon, 0)$
        else if $\ell_c = 0$ then
            $\mathtt{buf} \leftarrow \mathtt{buf} \% \mathtt{buf}[1..\ell_{\mathtt{len}}]$
            $\mathtt{seqno} \leftarrow \mathtt{seqno} + 1$
        else if $|\mathtt{buf}| \geq \ell_{\mathtt{len}} + \ell_c$ then
            $c' \leftarrow \mathtt{buf}[\ell_{\mathtt{len}} + 1..\ell_{\mathtt{len}} + \ell_c]$
            $\mathtt{buf} \leftarrow \mathtt{buf} \% \mathtt{buf}[1..1 + \ell_{\mathtt{len}} + \ell_c]$
            $m' \leftarrow \mathrm{DEC}(\mathtt{seqno} + 1, c')$
            $\mathtt{seqno} \leftarrow \mathtt{seqno} + 2$
            if $m' = \perp_{\mathrm{DEC}}$ then
                $\mathtt{fail} \leftarrow 1$
                return $(\mathtt{st}_R, \varepsilon, 0)$
            $\ell_p \leftarrow \mathrm{MIN}(m'[1,2], |m'| - 2)$
            $m \leftarrow m \| m'[3 + \ell_p..]$
        else
            break
    return $(\mathtt{st}_R, m, 0)$
```

block with length field 0 (when a small number of padding bytes are required).

Our construction is inspired by Shadowsocks, but with our payload blocks beginning with a fixed two-byte padding field indicating the actual plaintext length. We let $\ell_{\mathtt{len}}$ be the length of a the ciphertext produced by encrypted a two-byte message. Length shaping is achieved by outputting arbitrary-length fragments of a stream of ciphertext blocks. We set a maximum ciphertext length for our scheme as $\mathtt{ol} = 2^{16}$ and use $\mathtt{il}$ to refer to the input length such that, when a two-byte padding length prefix is added to $\mathtt{il}$ bytes of plaintext and the result is encrypted, the output length is $\mathtt{ol}$. The construction is presented in Figure 1.

The SEND function first determines if it can immediately return p bytes. If not, it extends the output buffer by at least one block, and then calls itself with an empty plaintext input.

RECV waits for a full length block, and if the resulting length value is nonzero it waits for the following payload block, consuming its buffer as the outputs are produced. Any decryption errors result in the protocol returning ε indefinitely without producing any further output or channel closure.

4 Related and Future Work

We build on work around secure symmetric encryption in the presence of ciphertext and plaintext fragmentation in the stream setting [3–5, 8, 12], which is related to the problem of expressing ciphertext lengths without plaintext metadata.

Fully encrypted protocols have been analyzed against active probing [7, 13] and redirection [17, 21] attacks. Some work has also developed methods to identify FEPs [6, 14, 29, 32].

Obfs4 is based on Scramblesuit [31] and is the fourth iteration of Tor's fully encrypted protocol. Shadowsocks [2], Obfuscated OpenSSH [1], and VMess [28] come from the open-source community, and they are among the more sophisticated and popular fully encrypted protocols, which also have become useful to avoid VPN blocking [32].

This work is ongoing. Future work includes presenting proofs that our construction satisfies the required security notions, deriving relations between the security definitions, addressing forward secrecy via key exchange in the protocol, and extending our definitions to the datagram setting.

Acknowledgements

This work was supported by the Office of Naval Research (ONR) and the Defense Advanced Research Projects Agency (DARPA). We thank the anonymous reviewers for their feedback.

References

[1] Obfuscated openssh. https://github.com/brl/obfuscated-openssh. Accessed September 29, 2022.

[2] Shadowsocks. https://github.com/shadowsocks. Accessed September 24, 2022.

[3] Martin R Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G Paterson. A surfeit of ssh cipher suites. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1480–1491, 2016.

[4] Martin R Albrecht, Torben Brandt Hansen, and Kenneth G Paterson. libintermac: Beyond confidentiality and integrity in practice. *IACR Transactions on Symmetric Cryptology*, pages 46–83, 2019.

[5] Martin R Albrecht, Kenneth G Paterson, and Gaven J Watson. Plaintext recovery attacks against ssh. In *2009 30th IEEE Symposium on Security and Privacy*, pages 16–26. IEEE, 2009.

[6] Anonymous, Kevin Bock, Jackson Sippe, Shelikhoo, David Fifield, Eric Wustrow, Dave Levin, and Amir Houmansadr. Exposing the great firewall's dynamic blocking of fully encrypted traffic. Technical report, Open Technology Fund, 2022.

[7] Jan Beznazwy and Amir Houmansadr. How china detects and blocks shadowsocks. In *Proceedings of the ACM Internet Measurement Conference*, pages 111–124, 2020.

[8] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 682–699. Springer, 2012.

[9] Bolt #8: Encrypted and authenticated transport. https://github.com/lightning/bolts/blob/master/08-transport.md. Accessed October 14, 2022.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[11] David Fifield. The limits of timing obfuscation in obfs4, 2017. https://lists.torproject.org/pipermail/tor-dev/2017-June/012310.html. Accessed October 7, 2022.

[12] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G Paterson. Data is a stream: Security of stream-based channels. In *Annual Cryptology Conference*, 2015.

[13] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting probe-resistant proxies. In *NDSS*, 2020.

[14] Sergey Frolov and Eric Wustrow. The use of tls in censorship circumvention. In *NDSS*, 2019.

[15] Sergey Frolov and Eric Wustrow. HTTPT: A probe-resistant proxy. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[16] Erik Hjelmvik and Wolfgang John. Breaking and improving protocol obfuscation. Technical report, Chalmers University of Technology, 2010.

[17] Qingbing Ji, Zhihong Rao, Man Chen, and Jie Luo. Security analysis of shadowsocks (r) protocol. *Security and Communication Networks*, 2022, 2022.

[18] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargavan. Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[19] obfs4 (the obfourscator). https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt. Accessed October 14, 2022.

[20] Outline client. https://github.com/Jigsaw-Code/outline-client. Accessed October 14, 2022.

[21] Zhiniang Peng. Redirect attack on shadowsocks stream ciphers. https://github.com/edwardz246003/shadowsocks.

[22] Phillip Rogaway. Nonce-based symmetric encryption. In *International workshop on fast software encryption*, pages 348–358. Springer, 2004.

[23] The rlpx transport protocol. https://github.com/ethereum/devp2p/blob/master/rlpx.md. Accessed October 14, 2022.

[24] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[25] Michael Carl Tschantz, Sadia Afroz, Vern Paxson, et al. Sok: Towards grounding censorship circumvention in empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[26] Project v. https://www.v2ray.com/en/. Accessed October 14, 2022.

[27] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[28] Vmess. https://www.v2ray.com/chapter_02/protocols/vmess.html. Accessed October 14, 2022.

[29] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 57–69, 2015.

[30] Brandon Wiley. Dust: A blocking-resistant internet transport protocol. Technical report, University of Texas at Austin, 2011.

[31] Philipp Winter, Tobias Pulls, and Juergen Fuss. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224, 2013.

[32] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. OpenVPN is open to VPN fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.