Bytes to Schlep? Use a FEP: Hiding Protocol Metadata with Fully Encrypted Protocols

Ellis Fenske
U.S. Naval Academy
Annapolis, Maryland, U.S.A.
fenske@usna.edu

Aaron Johnson
U.S. Naval Research Laboratory
Washington, D.C., U.S.A.
aaron.m.johnson213.civ@us.navy.mil

Abstract

Fully Encrypted Protocols (FEPs) have arisen in practice as a technique to avoid network censorship. Such protocols are designed to produce messages that appear completely random. This design hides communications metadata, such as version and length fields, and makes it difficult to even determine what protocol is being used. Moreover, these protocols frequently support padding to hide the length of protocol fields and the contained message. These techniques have relevance well beyond censorship circumvention, as protecting protocol metadata has security and privacy benefits for all Internet communications. The security of FEP designs depends on cryptographic assumptions, but neither security definitions nor proofs exist for them. We provide novel security definitions that capture the metadata-protection goals of FEPs. Our definitions are given in both the datastream and datagram settings, which model the ubiquitous TCP and UDP interfaces available to protocol designers. We prove relations among these new notions and existing security definitions. We further present new FEP constructions and prove their security. Finally, we survey existing FEP candidates and characterize the extent to which they satisfy FEP security. We identify novel ways in which these protocols are identifiable, including their responses to the introduction of data errors and the sizes of their smallest protocol messages.

CCS Concepts

• **Security and privacy** → *Symmetric cryptography and hash functions*; **Privacy-preserving protocols**; *Cryptanalysis and other attacks*; • **Networks** → **Network protocol design**; *Presentation protocols*; **Network privacy and anonymity**; **Security protocols**.

Keywords

Cryptography; Network Protocols; Censorship Circumvention; Fully Encrypted Protocols

ACM Reference Format:
Ellis Fenske and Aaron Johnson. 2024. Bytes to Schlep? Use a FEP: Hiding Protocol Metadata with Fully Encrypted Protocols. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690198

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA
2024. ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690198

1 Introduction

One approach to avoid network censorship is to use a Fully Encrypted Protocol (FEP). FEPs are designed to hide communications metadata, such as the precise length of a plaintext message, the encryption algorithms being used, and the exact protocol being run. A goal of FEPs is that each byte appears uniformly random (a design sometimes referred to as "obfuscated", "look-like-nothing", "randomized", or "unfingerprintable" [17, 39, 52, 56]). Censorship circumvention tools using FEPs include the popular Shadowsocks [47], obfs4 [34], and VMess [50] systems, each of which provides its own unique FEP. The Noise protocol framework [39] requires ciphertexts be indistinguishable from random to be "censorship-resistant" (although it fails to specify all protocol aspects needed to satisfy that goal). While methods to identify and block FEPs exist [51, 52], FEPs nonetheless continue to be effective and popular [34, 40, 47].

FEPs are also helpful outside of a censorship setting. Hiding protocol metadata can improve security (*e.g.*, preventing attacks targeted at a specific protocol or implementation) and privacy (*e.g.*, preventing traffic analysis of message lengths). Indeed, standard encrypted transport protocols like TLS and QUIC have moved over time towards encrypting more protocol metadata, as it is repeatedly observed that seemingly innocuous metadata is unexpectedly sensitive. The Pseudorandom cTLS extension [46] is a FEP that has recently been proposed to the IETF, citing security and privacy as its main motivations. In addition, the use of FEPs can prevent Internet ossification because they provide little metadata for middleboxes to operate on, a strategy similar to the use of random extensions in GREASE [9]. Thus, we consider FEPs to be a natural endpoint of encrypted transport protocol development.

Despite these motivations, FEPs have primarily been developed by the open-source community. Consequently, real-world protocols have been developed without mathematical security goals, even though their security depends on cryptographic assumptions[1]. Also, in practice, different developers make a variety of design errors [23], as they lack a common set of FEP techniques and pitfalls.

Therefore, we propose novel, precise security definitions for FEPs. We formalize the goal of producing apparently random protocol messages with a passive adversary, and then we extend that to the goal of appearing otherwise predictable (and thus not leaking information) with respect to an active adversary. The predictability applies to the protocol's behavior when ciphertexts are modified and to the protocol's use of *channel closures*, an observable feature of connection-based transport protocols such as QUIC and TCP.

[1]The Pseudorandom cTLS proposal [46] cites the need for such formalization: "TODO: More precise security properties and security proof. The goal we're after hasn't been widely considered in the literature so far, at least as far as we can tell."

Research has shown that channel closures can be used to identify specific FEPs [26], and real-world measurements indicate that censors are already doing so [4].

In addition, most FEPs (and many other encrypted protocols) include message padding as a way to prevent traffic analysis from making inferences on traffic contents based on ciphertext lengths. For FEPs, an effective padding strategy is particularly powerful because each byte appears random, and so the number of bytes is the main remaining source of communications metadata. However, existing padding mechanisms are ad hoc and have no precisely stated goals. As a consequence, they fail to provide full control over ciphertext lengths, a problem that has been observed by FEP developers themselves [22], which serves as the basis for state-of-the-art FEP detection using packet lengths [51], and has been exploited in practice by censors [4]. We thus propose a powerful property called *Traffic Shaping*, which requires that protocols be capable of producing ciphertexts of arbitrary lengths on command.

We prove relations among our novel FEP security definitions and existing notions of confidentiality and integrity. Our analysis of existing FEPs indicates that none of them satisfies our notions, and so we construct a protocol and prove that it does. These results are given in the *datastream* setting, which models a protocol that uses an underlying reliable, byte-oriented transport such as TCP. We subsequently give definitions and related results in the *datagram* setting, which models the use of an unreliable transport such as UDP, and we present a provably secure FEP in it as well.

Our theoretical results are inspired by observing persistent problems in real-world FEP deployments and our work provides concrete guidance for future FEP development. Our security definitions provide explicit security goals for FEPs to satisfy, and allow FEP maintainers to prove their protocols secure. Our Traffic Shaping notions give FEP designers a length obfuscation capability that is maximally flexible and practical to implement. In addition our definitions of secure close functions highlight the importance of carefully implementing connection closures, and explicitly provides a narrow set of possible close behaviors which do not reveal metadata unnecessarily.

Finally, we analyze a wide variety of existing FEPs under our proposed security definitions, including the previously mentioned Shadowsocks [47], obfs4 [34], and VMess [50] protocols. We examine their source code and documentation, and we run experiments to measure their error responses and message sizes. Our results indicate that nearly all of them do produce outputs that are indistinguishable from randomness, satisfying our passive FEP security definition. However, among the datastream protocols, their channel closures in response to errors make them identifiable, and they consequently fail our active FEP security definition. Our experiments further uncover integrity failures in V2Ray (the system providing the VMess protocol [40]). Moreover, all of the FEPs exhibit *unique minimum message sizes*, which we both predict via source-code analysis and verify experimentally. The catalog we produce of channel closures and minimum message sizes provides a new method by which existing FEPs can be individually identified and thus blocked. It also supports our proposed security definitions, which normalize channel closures and preclude minimum message sizes.

The main contributions of our paper are thus as follows. First, we present security definitions for datastream FEPs, compare them to existing security definitions, and provide a provably secure datastream FEP construction. Second, we similarly provide and analyze datagram FEP definitions, and we provide a provably secure datagram FEP. Third, we evaluate many existing FEPs against our security definitions, uncovering novel identifying features in both error-induced channel closures and minimum message sizes.

This paper improves and expands on an early version of this work [20] by including a more developed notion of close functions, security definitions and a construction in the datagram setting, analyses of relations between the new security notions and existing ones, and an analysis of and experiments on existing FEPs. The full version of this work [21] also contains appendices with proofs and additional details.

2 Background and Preliminaries

We present the notation, primitives, and concepts required to describe our constructions, definitions, and counterexamples.

We use ϵ to denote the empty byte string and $s \| t$ to denote the concatenation of two byte strings s and t. We use uppercase variables for lists, and L_j denotes the jth item of list L. Given a list of byte strings L we denote its in-order concatenation by $\|L$. We use $[]$ to denote the empty list. $\textsc{Rand}(n)$ denotes a function that produces a uniformly random string of n bytes, and $\textsc{Append}(L, x)$ returns a new list constructed by appending the element x to a list L. Given a byte string x, we use $x[i..j]$ to denote the substring from byte i to byte j, inclusive, with indexing beginning at 1. We use the binary % operator to mean "without the prefix"; for example, abcd%ab = cd. $[\![a, b]\!]$ denotes the longest common prefix of byte strings a and b. $\leq$ ($<$) indicates (strict) string prefixes. $|x|$ denotes the length of x in bytes. $x \xleftarrow{R} S$ denotes sampling an element x uniformly at random from finite set S.

We use a generic AEAD encryption scheme for our channel constructions in Sections 5.1 and 8.1, as well as certain counterexamples. The scheme consists of a triple of algorithms $\textsc{Gen}(\lambda)$, $\textsc{Enc}_k(\mathsf{nonce}, m, \mathsf{ad})$, $\textsc{Dec}_k(\mathsf{nonce}, c, \mathsf{ad})$, as defined by Rogaway [44], with both the AUTH and PRIV properties for integrity and confidentiality, respectively. In the remainder, we drop the associated data argument since it is not necessary for our work. We denote decryption errors with the distinguished error symbol $\perp_{\textsc{Dec}}$. We assume that the scheme satisfies IND\$-CPA [45], meaning that its ciphertext outputs are indistinguishable from random bytes. We further assume that the scheme has a fixed nonce size, ℓ_{Nonce}, a fixed authentication tag size ℓ_{Tag} and assume that the scheme has a fixed overhead associated with encryption in the form of an additive constant, which we refer to as ℓ_{Overhead} meaning that $|\textsc{Enc}_k(\mathsf{nonce}, m)| = |m| + \ell_{\text{Overhead}}$ for any valid key k, nonce nonce, and message m. We call schemes with this property *length additive*. We note that this property implies *length regularity*; that is, input plaintexts of the same length produce output ciphertexts of the same length. Standard AEAD schemes, such as AES-GCM, are believed to satisfy all of the properties we assume.

Datastream channels are intended to model the interface provided by TCP, where correctness requires, and only requires, that the sent *bytes* are all received and in the same order that they were sent. Datastream channels are connection-based and can be *closed* explicitly (*e.g.*, with a TCP FIN packet), observably terminating the connection. Messages sent in the datastream model may be fragmented, with internal buffering behavior, network conditions, and adversarial manipulation all affecting the size of the messages that are sent and received, which can be distinct from the sizes of the messages explicitly passed to the channel interface at the level of an application. *Datagram* channels are intended to model the interface provided by the UDP and IP protocols, where messages can arrive out of order, fail to arrive at all, or arrive multiple times, and all messages have an explicit length.

Let $\mathcal{M} = \{0,1\}^*$ be the plaintext message space for a channel. We use $\bot$ as another distinguished error symbol (*i.e.*, $\bot \notin \mathcal{M}$), which will indicate an error in the channel operation.

Traffic analysis [42] is used by a network adversary to infer sensitive information about the metadata (*e.g.*, the sender or receiver) or content of encrypted traffic. Modern traffic analysis techniques rely on many features to make these inferences, including timing data, plaintext message headers, protocol control messages, and message lengths. *Traffic shaping* [6, 11, 22, 55], used in the fields of anonymous communication, network privacy, and censorship circumvention, frustrates traffic analysis by changing the timing, number, and lengths of the messages.

3 Data Stream Channels

We present security notions for Fully Encrypted Protocols in the datastream setting using the model of Fischlin et al. [24]. We make two major additions to this model that are important in FEP context. First, we allow the sender to indicate the desired length of the output ciphertext, which enables Traffic Shaping for metadata protection. Second, we allow the receiver to close the channel in response to an input ciphertext, which models this potential information leak.

3.1 Channel Model

A *datastream channel* consists of three algorithms:

(1) $(\mathtt{st}_S, \mathtt{st}_R) \leftarrow \textsc{Init}(1^\lambda)$, which takes a security parameter λ and returns an initial sender state $\mathtt{st}_S$ and receiver state $\mathtt{st}_R$.
(2) $(\mathtt{st}'_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p, f)$ which takes a sender state $\mathtt{st}_S$, a plaintext message m, an output length p, and a flush flag f, and returns an updated sender state $\mathtt{st}'_S$ and a (possibly empty) ciphertext fragment c.
(3) $(\mathtt{st}'_R, m, \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$ which takes a receiver state $\mathtt{st}_R$ and a ciphertext fragment c, and produces an updated receiver state $\mathtt{st}'_R$, a plaintext fragment m, and a flag $\mathtt{cl}$ indicating whether the channel is to be closed.

These algorithms provide a unidirectional communication channel from the sender to the receiver. To use a channel, Init is called to produce the initial states. States $\mathtt{st}_S$ and $\mathtt{st}_R$ should be shared with the sender and receiver, respectively, using an out-of-band process. Such states may, for example, include a shared symmetric key or public/private keypairs for each party.

The sender calls Send to send a message $m \in \mathcal{M}$ to the receiver. Note that the correctness requirement will not guarantee that the output ciphertext c contains all of m (*i.e.*, plaintext fragmentation is allowed), nor that it is even a full ciphertext (*i.e.*, ciphertext fragmentation is allowed). Thus the sender will be allowed to buffer inputs, such as for performance reasons. Moreover, c might in fact contain multiple ciphertexts, in the sense that Recv may need to perform multiple decryption operations to obtain all the contained plaintext. The output length p will be used to support traffic shaping. The flush flag f, if set, forces the output of all messages provided as input to a Send call up to the given call.

The receiver calls Recv on a received ciphertext to obtain the sent plaintext. Correctness will require that Recv can take as its inputs a fragmentation or merging of previous outputs of Send. Thus the receiver will also be allowed to buffer inputs so that it can produce the sent plaintext once sufficient ciphertext fragments are received. The output message m may contain errors (*i.e.*, $m \in \{0,1,\bot\}^*$). If the close flag $\mathtt{cl}$ is set, that indicates that the receiver actively closes the channel after the given Recv call (*e.g.*, by sending a TCP FIN).

3.2 Correctness

A channel should be considered correct if it delivers the data from the sender to the receiver in the absence of malicious interference. As given by Fischlin et al. [24], datastream correctness requires that the plaintext message bytes produced by Recv should match the message bytes given to Send if the ciphertext bytes are correctly delivered from Send to Recv. This requirement tolerates arbitrary fragmentation of the plaintexts and ciphertexts. However, we add to this correctness notion a requirement related to channel closures.

The inclusion of closures raises the possibility of a trivial correct channel that performs no useful data transfer and instead just immediately closes the channel. However, we also want to allow the possibility that the receiver does close the channel for a reason such as some plaintext command being received or a limit on received data being reached. To rule out the former while allowing the latter, we introduce the notion of a close function and use it to parameterize correctness. A close function $\mathscr{C}$ is a randomized function that takes as input a sequence of channel operations (*i.e.*, function inputs and outputs starting with Init and then including Send and Recv calls) ending in a final Recv call. It outputs a bit indicating if that last call should set its close output. We use $\mathscr{C} \equiv 0$ as a default close behavior, *i.e.*, the channel is never closed.

Our correctness requirement is as follows:

Definition 1. *Let S be a sequence of channel operations, starting with Init and followed by (possibly interleaved) calls to Send and Recv. Assume each Send or Recv call receives as input the state produced by its previous invocation (or the output of Init on the first such call). Let n_s be the number of Send calls and n_r be the number of Recv calls. Let M, P, and F be lists containing the inputs to the Send calls, C be a list of the Send outputs, C' be a list of the Recv inputs, and M' and $\mathtt{CL}$ be lists of the Recv outputs. A channel satisfies* correctness *with respect to close function $\mathscr{C}$ if, for all such sequences S where $\| C' \preceq \| C$, the following properties hold:*

(1) ***Stream Preservation****: $\| M' \preceq \| M$*
(2) ***Flushing****: If $F_{n_s} = 1$ and $\| C' = \| C$, then $\| M' = \| M$.*
(3) ***Close Coherence****: For any $i \in [1..n_r]$, if $\mathtt{CL}_i = 1$, then for any j with $n_r \geq j > i$, $(M'_j, \mathtt{CL}_j) = (\epsilon, 0)$.*
(4) ***Close Correctness****: If the last channel operation is Recv, the distribution of its close output, given its inputs, is identically distributed to that of $\mathscr{C}$ on S.*

The Stream Preservation property ensures that delivered output bytes are always some prefix of the plaintext input bytes. The Flushing property enforces that if the flush flag is given, then all the plaintext input bytes are output by Send. Close Coherence ensures that a channel is closed only once and that Recv outputs no messages afterwards. Close Correctness enforces that the channel implements the externally supplied close behavior $\mathscr{C}$. The first two requirements of our correctness definition coincide with the correctness definition of Fischlin *et al.* [24].

3.3 Secure Close Functions

While for correctness we make no assumptions about the desired channel-close behavior, for security we will want to limit it in some ways. Arbitrary close behavior may leak secret information. For example, a close function might close the channel if a certain plaintext byte sequence is received, which would violate confidentiality.

We therefore define a *secure close function*, which will prevent closures from leaking any information beyond what is implied by the stream of sent bytes. We require that a secure close function be able to be expressed as a function taking the following inputs: (1) C, a concatenation of the list of the ciphertexts produced by Send calls; (2) C', the list of the ciphertexts given as inputs to Recv calls; (3) $\mathtt{CL}$, the list of the channel-close outputs of Recv calls; and (4) c, the final ciphertext input to Recv. As with all close functions, a secure close function returns a bit indicating if the channel should close. A secure close function must also be a probabilistic polynomial-time function. With its limited inputs and computational complexity, a secure close function not only prevents closures from leaking additional information beyond what is implied by the Send outputs, it further restricts that leakage to that implied by their *concatenation*. This choice prevents closures from revealing boundaries between Send outputs, which can be hidden from real-world network adversaries due to fragmentation below the application layer. Note that, although a passive adversary can observe the ordering of Send and Recv calls, we need not give that information to a secure close function, as the receiver itself cannot observe that ordering and thus could never realize any close function that depends on it.

In addition to preventing the leakage of plaintext data, secure close functions also prevent leaking certain metadata. For example, they preclude closing as soon as an error is detected in a protocol with variable-length ciphertexts, which would reveal some of the internal structure of the protocol messages, as two Send outputs are indistinguishable to a secure close function from a single Send output of the same total length due to their concatenation in the input. Such a behavior is frequently observed in real-world FEPs, such as obfs4 and one direction of Shadowsocks (see § 9).

Despite their restrictions, secure close functions can still express interesting and useful behavior. For example, they include closures that occur after receiving some maximum amount of data. They also allow for closures at some point after an error is introduced, for example, after a modified byte is received and the total bytes received is a multiple of 1000 (*i.e.*, the strategy employed by the interMAC protocol [12]).

Finally, secure close functions are realistic. Both academic and real-world protocols have close behaviors which can, in whole or in part, be realized by a secure close function, such as the fixed-boundary closures of InterMAC [2] and the never-close behavior of Shadowsocks in one direction [47]. We also observe a real-world attempt, in the VMess protocol [50], to obscure when ciphertext errors are detected by randomizing the timing of a subsequent closure, which doesn't quite satisfy secure closures and consequently leaks metadata (see § 9).

3.4 Confidentiality and Integrity Definitions

We adopt several datastream confidentiality and integrity definitions from Fischlin *et al.* [24]. We use IND-CPFA/IND-CCFA for passive/active indistinguishability (*i.e.*, against chosen plaintext/ciphertext fragment attacks). We use INT-PST/INT-CST for integrity against plaintext/ciphertext stream manipulation. We use these notions without alteration beyond the new function signatures associated with our channel model (*i.e.*, adding the Traffic Shaping parameter p and close-flag f to the inputs of Send and outputs of Recv, respectively).

We also introduce a new passive confidentiality notion that includes adversarial observation of closures. The modified notion, IND-CPFA-CL, gives a receiving oracle to the adversary that only returns channel closures. Moreover, as the adversary is passive, the definition requires that the adversary correctly deliver to the receiving oracle the ciphertext bytestream output produced by the sending oracle. IND-CPFA-CL is a confidentiality goal for FEPs, but we do use standard IND-CPFA as a means to prove other properties. See the full version [21] for a precise definition of IND-CPFA-CL.

3.5 Fully Encrypted Datastream Protocols

We introduce passive and active security notions for Fully Encrypted Protocols in the datastream setting. The goal of these definitions is to ensure that protocols satisfying them do not reveal protocol metadata through the observable bytes and channels closures. The passive definition is FEP-CPFA, or FEP security against a chosen plaintext fragment attack, and the active definition is FEP-CCFA , or FEP security against a chosen ciphertext fragment attack. The active notion is implicitly parameterized by a secure close function, and we use FEP-CCFA-$\mathscr{C}$ to explicitly indicate security with respect to the close function $\mathscr{C}$. The definition for both notions is as follows:

Definition 2. *A channel satisfies FEP-x, for $x \in \{CPFA, CCFA\}$ if, for any PPT adversary $\mathcal{A}$, $\left|P\left[\mathsf{Exp}_{\mathcal{A}}^{FEP\text{-}x,b}(1^\lambda) = 1 \middle| b \xleftarrow{R} \{0,1\}\right] - 1/2\right|$ is negligible in the security parameter λ.*

The security experiment used in this definition (Algorithm 1) gives adaptive access to a sending oracle (Algorithm 2). That oracle calls the Send function of the underlying channel and then returns to the adversary either the output or the same number of genuinely random bytes, depending on a secret random bit b. The adversary is challenged to distinguish between observing the outputs of Send and random byte strings of the same lengths. This experiment captures a key goal of a passive FEP—that every byte sent should appear random to the adversary.

In the active setting, the adversary is also given adaptive access to a receiving oracle (Algorithm 3). The behavior of the oracle depends on the secret bit b. If $b = 0$, then the oracle calls the channel Recv.

If the sending and receiving byte streams are still in sync (*i.e.*, no received byte differs from the byte in the same position in the sent byte stream), then only the close flag from Recv is returned to the adversary. If those byte streams are out of synchrony, then both the output message and the close flag are returned. If the streams just become out of sync, then the in-sync and out-of-sync components are separated before returning any message produced out-of-sync to the adversary (the logic largely follows the similar oracle in the IND-CCFA definition of Fischlin et al. [24]). If $b = 1$, the oracle simply returns the empty string and the close flag prescribed by the close function $\mathscr{C}$.

For active security, the receiving oracle should not yield outputs distinguishing the $b = 0$ and $b = 1$ cases. The $b = 1$ case yields a simple behavior, which the $b = 0$ case only differs from if a non-empty message is ever output or the prescribed close behavior is not followed. The relevance of non-empty outputs to the FEP goal is that they occur when modified messages yield valid outputs, which may influence observable behavior of the receiver and thus reveal metadata about the integrity features of the protocol. Because the prescribed close behavior is a secure close function, conforming to it ensures that the closures leak no information beyond what is already revealed by the ciphertext byte sequence. Thus, if passive security is already satisfied, active security ensures that only the ciphertext lengths could potentially reveal protocol metadata.

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

$(\mathtt{st}_S, \mathtt{st}_R) \leftarrow \textsc{Init}(1^\lambda)$

$C_S, C_R, C_{\mathsf{CL}}, \mathsf{sync} \leftarrow [], [], [], 1$

$b' \leftarrow \begin{cases} \mathcal{A}^{O^b_{\textsc{Send}}()}(1^\lambda) & \text{if } x = \text{CPFA} \\ \mathcal{A}^{O^b_{\textsc{Send}}(), O^b_{\textsc{Recv}}()}(1^\lambda) & \text{if } x = \text{CCFA} \end{cases}$

return $b' = b$

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

$(\mathtt{st}_S, c_0) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p, f)$

$c_1 \leftarrow \textsc{Rand}(|c_0|)$

$C_S \leftarrow \textsc{Append}(C_S, c_b)$

return c_b to $\mathcal{A}$

Algorithm 3 $O^b_{\textsc{Recv}}(c)$: FEP receiving oracle

if $b = 0$ **then**
  if $\mathsf{sync} = 0$ **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
    $(\mathtt{st}_R, m, \mathtt{cl}) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
    $C_R \leftarrow \textsc{Append}(C_R, c)$
    return $(\epsilon, \mathtt{cl})$ to $\mathcal{A}$
  else // either deviating or exceeding Send outputs
    if $(\| C_R) \prec [\![(\| C_R) \| c, (\| C_S)]\!]$ **then** // partial sync
      $\widetilde{c} \leftarrow [\![(\| C_R) \| c, (\| C_S)]\!] \,\%\, (\| C_R)$ // 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 $(\| C_S) \npreceq (\| C_R) \| c$ **or** $m' \neq \epsilon$ **then**
      $\mathsf{sync} \leftarrow 0$
    $C_R \leftarrow \textsc{Append}(C_R, c)$
    return $(m', \mathtt{cl})$ to $\mathcal{A}$
else // $b = 1$, return empty string and desired closure
  $\mathtt{cl} \leftarrow \mathscr{C}(\| C_S, C_R, C_{\mathsf{CL}}, c)$ // produce close output
  $C_R \leftarrow \textsc{Append}(C_R, c)$
  $C_{\mathsf{CL}} \leftarrow \textsc{Append}(C_{\mathsf{CL}}, \mathtt{cl})$
  return $(\epsilon, \mathtt{cl})$ to $\mathcal{A}$

3.6 Traffic Shaping

The passive and active FEP security definitions essentially ensure that only the ciphertext lengths can leak protocol metadata. Those lengths can also reveal message data, as the length of a message may in some settings be related to its contents. To enable full metadata protection, our channel definition allows that a requested length p can be provided to each call to Send. We say that channels that provide the requested lengths satisfy Traffic Shaping:

Definition 3. *A channel satisfies Traffic Shaping if, for any state* $\mathtt{st}_S$*, message* m*, and integer* $p \geq 0$*, with* $(\mathtt{st}_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p, f)$*, if* $f = 0$ *then* $|c| = p$*, otherwise* $|c| \geq p$*.*

Note that, in this definition, if the flush flag is set then the desired length may be exceeded, which allows for the case that there are a large number of buffered message bytes that must be flushed. The definition also only imposes a requirement for $p \geq 0$, which allows channels freedom to implement alternate behaviors for negative p values.

Traffic shaping enables protection of protocol metadata by setting the p inputs to values that are independent of the protocol being used. For example, the p values could all be set to a constant, which would hide the number and sizes of metadata fields as well as the length of the plaintext messages. This is similar to the strategy used by the Tor protocol of putting all data into fixed-size cells to prevent traffic analysis [15].

4 Relations Between Datastream Notions

In this section we present relations among our novel security notions and previous security notions for datastream channels.

While our novel FEP definitions are designed primarily to enforce that all output bytes appear indistinguishable from random, we observe in this section that these properties actually imply standard cryptographic datastream security properties. In particular, FEP-CCFA is a strong property that directly implies ciphertext stream integrity, and FEP-CPFA alongside channel length regularity (a minor property discussed below) imply passive confidentiality. Finally, we highlight as a primary conclusion from this section the observation from Figure 1 that a channel satisfying FEP-CCFA for a secure close function and channel length regularity satisfies all other security properties we identify.

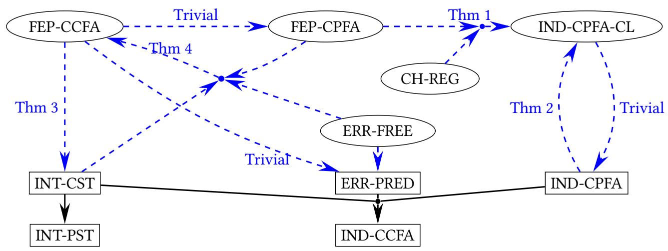

Figure 1: Relations between notions for correct datastream channels that realize a secure close function C. Rectangles and solid arrows are results from Fischlin *et al.* [24]; ellipses and dashed arrows are novel notions and relations

Figure 1 summarizes our results relating datastream security notions. First we include relevant notions and results from Fischlin *et al.* [24] with solid lines and rectangles, which establishes the result that the passive confidentiality notion (IND-CPFA) and the integrity of ciphertexts (INT-CST), together with a new notion ERR-PRED, which establishes the predictability of error symbols that will be produced by Recv, imply their active confidentiality notion IND-CCFA. CH-REG refers to Datastream Channel Length Regularity (Definition 4). Since in our construction we do not use error symbols, we for convenience include the notion ERR-FREE, which refers to the property that Recv never produces in-band error symbols and trivially implies ERR-PRED.

Below we give theorem statements for each relation in Figure 1, with the proofs in the full version [21].

While our notions do not imply confidentiality, this is only because of the issue of ciphertext lengths, and we formalize this intuition below. First, we define a length-regularity property for a channel, where we require that the length of the outputs of Send do not depend on the content of the messages.

Definition 4. *Let M^0 and M^1 be n-length lists of messages such that, for all i, $|M_i^0| = |M_i^1|$. Let P and F be an n-length integer sequence and an n-length boolean sequence. Let $(\mathtt{st}_S^0, C_i^0) \leftarrow$ Send$(\mathtt{st}_S^0, M_i^0, F_i, P_i)$ and $(\mathtt{st}_S^1, C_i^1) \leftarrow$ Send$(\mathtt{st}_S^1, M_i^1, F_i, P_i)$, where in both cases Send is initialized with Init and is then called sequentially as $i = 1..n$, updating its state with each call. A datastream channel is* length regular *if, for any such M^0, M^1, F and P, and for all i, $|C_i^0| = |C_i^1|$.*

Next, Theorem 1 shows that a length-regular channel satisfying FEP-CPFA provides confidentiality.

Theorem 1. *Suppose that a channel satisfies FEP-CPFA and further that the Send function is length regular in the sense of Definition 4. Then that channel satisfies IND-CPFA.*

Theorem 2 shows that if a channel is correct and provides IND-CPFA (the standard confidentiality notion), then it provides the similar confidentiality notion with closures, IND-CPFA-CL.

Theorem 2. *If a channel satisfies correctness for a given secure close function C, and IND-CPFA, then it satisfies IND-CPFA-CL.*

We show in Theorem 3 that FEP-CCFA by itself implies the strong integrity notion INT-CST.

Theorem 3. *If a channel satisfies FEP-CCFA , then it satisfies INT-CST.*

We give in Theorem 4 a general result showing sufficient conditions for passive FEP security (FEP-CPFA) to imply active FEP security (FEP-CCFA). Note that the ERR-FREE condition means that recv does not produce in-band errors.

Theorem 4. *If a channel satisfies correctness for a given secure close function $\mathscr{C}$, FEP-CPFA, ERR-FREE, INT-CST, then it satisfies FEP-CCFA-$\mathscr{C}$.*

Finally, we give some negative results showing that, similar to IND\$-CPA in the atomic setting, our datastream FEP security notions do not imply and are not implied by confidentiality notions.

We observe that FEP-CPFA does not imply IND-CPFA. Even if a channel satisfies FEP-CPFA, it may still embed plaintext information in the length of ciphertext fragments it produces, violating confidentiality. We give an explicit counterexample in the full version [21].

We similarly observe that IND-CCFA and IND-CPFA do not imply FEP-CPFA. To show that neither of these datastream confidentiality notions implies FEP-CPFA, we simply observe that the AEAD-based construction of Fischlin *et al.* [24] satisfies both of them but includes unencrypted length fields to delimit ciphertext block boundaries, clearly failing to satisfy FEP-CPFA.

5 A Datastream Fully Encrypted Protocol

In this section we give a construction for a Fully Encrypted Protocol in the datastream setting, and we prove that it satisfies all of the desired security properties. The key challenge in designing the construction is to simultaneously achieve Traffic Shaping, correctness, and security.

5.1 The Construction

We give a construction that satisfies all our security notions for the secure close function $C \equiv 0$, presented in Figure 2. Our approach is inspired by Shadowsocks, and is designed around producing pairs of ciphertexts, the first of which is a "length block" which has a fixed length, denoted $\ell_{\mathtt{len}} = 2 + \ell_{\mathrm{Tag}}$, and contains the length of the subsequent "payload block", which is limited to a length that can be represented within two bytes. The payload block also contains a two byte padding length field to denote internal (plaintext) padding. We denote the largest payload block length as $\mathtt{ol} = 2^{16} - 1$ and set $\mathtt{il}$ as the largest plaintext length that can fit within a payload block $(2^{16} - 3 - \ell_{\mathrm{Tag}})$.

The Send function contains an input buffer `buf` for plaintext and produces an output buffer `obuf` of ciphertext block pairs, outputting fragments from this buffer as requested by the caller. When Send is called, it first checks to determine whether it can return the required number of bytes (determined by the conditional in Line 3 of Send, which checks if p output bytes are available and $f = 0$, or if $f = 1$ and the plaintext buffer is empty). If possible, Send outputs the appropriate number of bytes from the output buffer (p or $|\mathtt{obuf}|$, depending on f). Otherwise, Send constructs a pair of ciphertext blocks according to our scheme in Lines 7–18 and calls itself which will result in termination or a further extension of `obuf` and another recursive call. The ciphertexts are constructed by

Figure 2: A Datastream Fully Encrypted Protocol

$\textsc{Init}(1^\lambda)$:

1: $k \leftarrow_\$ \mathcal{K}$
2: $\mathtt{st}_S = (k, 0, \epsilon, \epsilon)$
3: $\mathtt{st}_R = (k, 0, \epsilon, 0)$
4: **return** $(\mathtt{st}_S, \mathtt{st}_R)$

$\textsc{Send}(\mathtt{st}_S, m, p, f)$:

1: $(k, \mathsf{seqno}, \mathsf{buf}, \mathsf{obuf}) \leftarrow \mathtt{st}_S$
2: $\mathsf{buf} \leftarrow \mathsf{buf} \| m$
3: **if** $(|\mathsf{obuf}| \geq p) \wedge ((f = 0) \vee (\mathsf{buf} = \epsilon))$ **then**
4: $\quad c \leftarrow \mathsf{obuf}[1..\textsc{Max}(p, f * |\mathsf{obuf}|)]$
5: $\quad \mathsf{obuf} \leftarrow \mathsf{obuf} \% c$
6: $\quad$ **return** $(\mathtt{st}_S, c)$
7: $o \leftarrow \textsc{Min}(|\mathsf{buf}|, \mathtt{il})$ ▷ Plaintext Length
8: $\ell_p \leftarrow 0$ ▷ L8-12 calculates ℓ_p, ℓ_c
9: $\ell_c \leftarrow |\textsc{Enc}_k(\mathsf{seqno}, 0^{(2+o)})|$
10: **while** $(|\mathsf{obuf}| + \ell_c + \ell_{\mathtt{len}} < p) \wedge (\ell_c \neq \mathtt{ol})$ **do**
11: $\quad \ell_p \leftarrow \ell_p + 1$
12: $\quad \ell_c \leftarrow |\textsc{Enc}_k(\mathsf{seqno}, 0^{(2+o+\ell_p)})|$
13: $c \leftarrow \textsc{Enc}_k(\mathsf{seqno}, \ell_c)$ ▷ Length Block
14: $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
15: $m' \leftarrow \ell_p \| (0^{\ell_p}) \| \mathsf{buf}[1..o]$
16: $\mathsf{buf} \leftarrow \mathsf{buf} \% \mathsf{buf}[1..o]$
17: $c \leftarrow c \| \textsc{Enc}_k(\mathsf{seqno}, m')$ ▷ Payload Block
18: $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
19: $\mathsf{obuf} \leftarrow \mathsf{obuf} \| c$
20: **return** $\textsc{Send}(\mathtt{st}_S, \epsilon, p, f)$

$\textsc{Recv}(\mathtt{st}_R, c)$:

1: $(k, \mathsf{seqno}, \mathsf{buf}, \mathtt{fail}) \leftarrow \mathtt{st}_R$
2: **if** $\mathtt{fail} = 1$ **then**
3: $\quad$ **return** $(\mathtt{st}_R, \epsilon, 0)$
4: $\mathsf{buf} \leftarrow \mathsf{buf} \| c$
5: $m \leftarrow \epsilon$
6: **while** $|\mathsf{buf}| \geq \ell_{\mathtt{len}}$ **do**
7: $\quad \ell_c \leftarrow \textsc{Dec}_k(\mathsf{seqno}, \mathsf{buf}[1..\ell_{\mathtt{len}}])$ ▷ Decrypt Length Block
8: $\quad$ **if** $\ell_c = \perp_{\textsc{Dec}}$ **then**
9: $\quad\quad \mathtt{fail} \leftarrow 1$
10: $\quad\quad$ **return** $(\mathtt{st}_R, \epsilon, 0)$
11: $\quad$ **else if** $|\mathsf{buf}| \geq \ell_{\mathtt{len}} + \ell_c$ **then**
12: $\quad\quad c' \leftarrow \mathsf{buf}[\ell_{\mathtt{len}} + 1..\ell_{\mathtt{len}} + \ell_c]$
13: $\quad\quad \mathsf{buf} \leftarrow \mathsf{buf} \% \mathsf{buf}[1..1 + \ell_{\mathtt{len}} + \ell_c]$
14: $\quad\quad m' \leftarrow \textsc{Dec}_k(\mathsf{seqno} + 1, c')$ ▷ Decrypt Payload Block
15: $\quad\quad \mathsf{seqno} \leftarrow \mathsf{seqno} + 2$
16: $\quad\quad$ **if** $m' = \perp_{\textsc{Dec}}$ **then**
17: $\quad\quad\quad \mathtt{fail} \leftarrow 1$
18: $\quad\quad\quad$ **return** $(\mathtt{st}_R, \epsilon, 0)$
19: $\quad\quad \ell_p \leftarrow \textsc{min}(m'[1, 2], |m'| - 2)$ ▷ Calculate Padding Length
20: $\quad\quad m \leftarrow m \| m'[3 + \ell_p..]$
21: $\quad$ **else**
22: $\quad\quad$ **break**
23: **return** $(\mathtt{st}_R, m, 0)$

encrypting as much of buf as possible, represented by the variable o, and then filling in padding bytes if needed so that the block pair has size p or is the maximum size of $\ell_{\mathtt{len}} + \mathtt{ol}$.

The Recv function keeps a buffer of received ciphertext bytes and waits to receive $\ell_{\mathtt{len}}$ bytes, then decrypts and parses the length block to recover the length of the subsequent payload block. It then waits until the full payload block has been received before decrypting and returning the plaintext, stripping off the padding length and any padding bytes. Recv checks for decryption errors and enters a fail state if either block type fails to decrypt. Our construction does not produce channel closures or error symbols.

We note that our construction generalizes in some ways. Our buffering and padding approach can be easily adapted for other FEP designs, such as obfs4 [34] and InterMAC [12], so that they satisfy Traffic Shaping. Further, although timing is outside of our model, an implementation could close the connection based on a timeout, where the timeout must depend only on how much time has elapsed since the last ciphertext fragment was received.

5.2 Properties

We show that our construction satisfies all of the desired properties in Theorem 5, with the proofs contained in the full version [21]. Our general approach is to first show the channel satisfies the three non-cryptographic properties: correctness, Traffic Shaping, and channel length regularity. We then establish individually each property required to apply Theorem 4: FEP-CPFA, INT-CST, and ERR-FREE (which is trivial). Theorem 4 then implies the channel satisfies FEP-CCFA, which (along with channel length regularity) implies IND-CCFA, so we arrive at all of the desired datastream channel properties established in the previous sections.

Theorem 5. *If the AEAD scheme (Gen, Enc, Dec) in the construction in Figure 2 satisfies IND\$-CPA, INT-CTXT, and is length additive (see Sec 2) then the channel construction satisfies correctness for the secure close function $\mathscr{C} \equiv 0$, Traffic Shaping, FEP-CCFA, INT-CST, and IND-CCFA.*

6 Fully Encrypted Datagram Transport Protocols

In the datagram setting, messages are transmitted atomically (i.e. without fragmentation). Implied in this property is that the length of each message can be determined by the receiver, unlike in the datastream model where messages can be arbitrarily fragmented or merged. Even in benign circumstances, messages in the datagram model may be delivered out of order or dropped. This setting models the UDP transport protocol.

Let ℓ_{in} be the maximum number of bytes in an input message. This number may vary with the channel (i.e. with the protocol), but it is assumed that $\ell_{\text{in}} > 0$. Let $\mathcal{M}_{\ell_{\text{in}}} = \cup_{0 \leq i \leq \ell_{\text{in}}} \{0, 1\}^{8i}$ be the space of valid input messages. Thus, any message m with $|m| \leq \ell_{\text{in}}$ is a valid input while sending. Let ϵ be the "empty" message, that is, the message of length zero. Note that $\epsilon \in \mathcal{M}_{\ell_{\text{in}}}$. We assume that there is a maximum number of bytes ℓ_{out} in deliverable outputs, which is typically larger than ℓ_{in}. ℓ_{out} does not vary with the channel and models the limitations of the underlying delivery mechanism (*e.g.*, the maximum size of a UDP datagram). Let $\perp$ be a distinguished error symbol (i.e., an item distinct from other outputs), which can be produced when sending or receiving a message, such as when the message to send is too long or the received message is malformed.

We also introduce the distinguished symbol $\top$ to indicate a "null" message. This symbol is different from the empty message ϵ in that it indicates no message rather than an empty message. We introduce this symbol to distinguish "chaff" messages, which are those sent purely to fool a network observer, from messages that are intended to be delivered to the receiver. A special case of this is when the sender requests an output length that is too small to contain an authentication tag, which we want to support to provide Traffic Shaping. $\top$ can be used as both an input while sending and an output while receiving, and so to avoid ambiguity it is not in $\mathcal{M}_{\ell_{\text{in}}}$. $\top$ can be used as an input to produce a chaff message. If it is produced as an output while receiving, the receiver should react as if no message was received. This design choice is driven by our desire to ensure that if the channel operates normally, the receiver does not produce an error symbol. This combined with the requirement that messages can be produced of arbitrary size by an honest sender (because of Traffic Shaping), and the requirement that messages may be dropped or re-arranged without error means that in certain cases, malicious messages or errors introduced by an active adversary will be indistinguishable from valid chaff messages, which required changes to the Recv oracle in our security definitions, presented in Algorithm 6.

The datagram model offers different challenges to defining and achieving Fully Encrypted Protocols. Atomic messaging avoids the necessity of communicating message length. However, the lack of delivery guarantees means that each message must be individually decryptable, which, for example, rules out using certain block-cipher modes across messages. At the same time, however, the Traffic Shaping requirement will require sending messages so small they cannot even contain authentication tags, as mentioned.

6.1 Channel Model

The datagram channel model consists of the following algorithms, which provide a unidirectional channel:

(1) $(\mathtt{st}_R, \mathtt{st}_S) \leftarrow \textsc{Init}(1^\lambda)$, which takes a security parameter λ and generates the sender and receiver state.
(2) $(\mathtt{st}'_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p)$, which takes a sender state and a plaintext message m, and a desired output length p. Its output $\mathtt{st}'_S$ is an updated sender state, and its output c is either a ciphertext or an error.
(3) $(\mathtt{st}'_R, m) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$, which takes a receiver state and a ciphertext c. Its output $\mathtt{st}'_R$ is an updated receiver state, and its output m is either a plaintext message or an error.

6.2 Correctness

The correctness requirement differs significantly from the datastream setting due to the lack of reliable in-order delivery. Instead, the requirement should simply enforce that a message m sent through Send and then Recv produces m again. This basic idea is complicated by the possibility that Send may yield an error. We allow for such errors in the following cases: (1) m is too large (i.e., $|m| > \ell_{\text{in}}$), (2) p is too small for m, and (3) p is too large (i.e., $p > \ell_{\text{out}}$). Definition 5 gives the correctness requirement for a datagram channel.

Definition 5. *Fix any sequence of datagram-channel operations where the first call is to Init, the remaining are (possibly interleaved) calls to Send and Recv, and the sender (receiver) state outputs are correctly provided as inputs to the subsequent Send (Recv) call.*

The datagram channel satisfies correctness *if, for all such sequences, the following properties hold:*

(1) ***Message Acceptance****: For every $m \in \mathcal{M}_{\ell_{in}} \cup \{\top\}$, there exists some $p_m \leq \ell_{out}$ such that with $(\mathtt{st}_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p)$, if $p \geq p_m$ or $p < 0$, $c \neq \perp$.*
(2) ***Message Length****: For every $m \in \mathcal{M}_{\ell_{in}} \cup \{\top\}$, with $(\mathtt{st}_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p)$, if $c \neq \perp$, $|c| \leq \ell_{out}$.*
(3) ***Message Delivery****: For any Send call with input $m \in \mathcal{M}_{\ell_{in}} \cup \{\top\}$ and output $(\mathtt{st}_S, c)$, where $c \neq \perp$, any subsequent Recv call with input c must have output either m or $\perp$, and the first such call must have output m.*

6.3 Traffic Shaping

As with datastream channels, we would like our datagram-based protocol to support Traffic Shaping by changing the length of its output messages. The desired output length is specified by the Send parameter p. We require Send to output p bytes as long as p is large enough to accommodate the desired message and does not exceed the maximum datagram size ℓ_{out}. In particular, for the null message $\top$, values $0 \leq p \leq \ell_{\text{out}}$ should yield an output ciphertext of length p (c might be meaningless randomness if p is too small to accommodate ciphertext metadata such as a tag or nonce). Definition 6 gives our precise *Traffic Shaping* notion in the datagram context.

Definition 6. *A datagram channel satisfies* Traffic Shaping *if, for any state $\mathtt{st}_S$ produced by Init or a subsequent Send call, any message $m \in \mathcal{M}_{\ell_{in}} \cup \top$, and any integer $p \geq 0$, the following hold for $(\mathtt{st}'_S, c) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p)$:*

(1) If $c \neq \perp$ and $p \geq 0$, then $|c| = p$, and
(2) If $m = \top$ and $p \leq \ell_{out}$, then $c \neq \perp$.

Note that this definition places no constraints on the channel output when p is negative. Therefore, a client may turn off Traffic Shaping by using $p < 0$.

6.4 Security Definitions

Providing security definitions for fully encrypted protocols is simpler for datagram protocols than for datastream protocols, largely because the correctness requirement does not require tolerating plaintext and ciphertext fragmentation. As with datastream protocols, our new definitions for FEPs are distinct from existing security notions for confidentiality and integrity, and we therefore present and discuss those existing notions as well. All our security notions are with respect to a probabilistic polynomial-time (PPT) adversary.

6.4.1 *Confidentiality and Integrity definitions.*

We adopt confidentiality and integrity definitions from Bellare and Namprempre [8]. We require only slight modifications to adapt them to our stateful communication channel. Details appear in the full version [21].

For confidentiality, we use the IND-CPA and IND-CCA definitions, which provide security with respect to a passive and active adversary, respectively. In both definitions, the adversary is given adaptive access to a sending oracle, which makes use of the Send function of the underlying channel. In IND-CCA security, the adversary also has adaptive access to a receiving oracle, which uses the channel's Recv function. The sending oracle requires the adversary to submit pairs of equal-length messages, and it outputs the encryption of one of them, depending on a secret random bit b. The receiving oracle only returns a decryption of the given ciphertext if it was not an output of the sending oracle and if the decryption is neither $\perp$ nor $\top$. The channel provides confidentiality if the adversary cannot guess the secret bit b with probability non-negligibly different than random guessing.

For integrity, we use INT-CTXT, which challenges the adversary to produce an unseen ciphertext that successfully decrypts. The adversary is given adaptive access to a sending oracle and a receiving oracle, which make use of the channel's Send and Recv functions, respectively. The channel provides integrity if the adversary cannot send the receiving oracle a ciphertext that was not produced by the sending oracle but does successfully decrypt to a non-null output, except with negligible probability.

6.4.2 *Fully Encrypted Datagram Protocols.*

Our novel security definitions in the datagram setting are FEP-CPA and FEP-CCA, which define security for Fully Encrypted Protocols against a chosen plaintext attack (i.e. a passive adversary) and against a chosen ciphertext attack (i.e. an active adversary), respectively:

Definition 7. *A channel satisfies FEP-x, for $x \in \{CPA, CCA\}$ if, for a security parameter λ and PPT adversary $\mathcal{A}$, $\left|P\left[\mathsf{Exp}_{\mathcal{A}}^{FEP\text{-}x,b}(1^\lambda) = 1 \middle| b \xleftarrow{R} \{0,1\}\right] - 1/2\right|$ is negligible in λ.*

In the related security experiment (Algorithm 4), the adversary is given adaptive access to a sending oracle (Algorithm 5). That oracle either faithfully returns the output of the channel Send operation or replaces that output with the same number of uniformly random bytes, depending on a randomly selected bit b. The adversary is thus challenged to distinguish between the outputs of Send and random messages of the same length.

In the active security definition, FEP-CCA, adaptive access to a receiving oracle is added (Algorithm 6). That oracle also depends on the secret bit b. If $b = 0$, it returns the output of Recv called on the given ciphertext, except if the ciphertext was an output of Send or if it fails to decrypt to a non-null plaintext. If $b = 1$, the oracle always returns $\perp$. The active adversary can thus also attempt to get Recv to produce anything but $\perp$, which can only happen if $b = 0$ and if the adversary submits a novel, valid, and non-null ciphertext. This definition models potential information leaks through observable behavior of the recipient that would receive those outputs. Note that null outputs are excepted because, in general, the output of Send may be required to be of a size too small to contain an authentication tag, making small outputs forgeable.

Algorithm 4 $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{FEP}\text{-}x,b}(1^\lambda)$

$(\mathtt{st}_S, \mathtt{st}_R) \leftarrow \textsc{Init}(1^\lambda)$
$C \leftarrow \emptyset$
$b' \leftarrow \begin{cases} \mathcal{A}^{O^b_{\textsc{Send-DG}}()}(1^\lambda) & \text{if } x = \text{CPA} \\ \mathcal{A}^{O^b_{\textsc{Send-DG}}(), O^b_{\textsc{Recv-DG}}()}(1^\lambda) & \text{if } x = \text{CCA} \end{cases}$
return $b' = b$

Algorithm 5 $O^b_{\textsc{Send-DG}}(m, p)$

$(\mathtt{st}_S, c_0) \leftarrow \textsc{Send}(\mathtt{st}_S, m, p)$
if $c_0 = \perp$ **then**
  return $\perp$
$c_1 \leftarrow \textsc{Rand}(|c_0|)$
$C \leftarrow C \cup \{c_b\}$
return c_b

Algorithm 6 $O^b_{\textsc{Recv-DG}}(c)$

if $b = 0$ **then**
  $(\mathtt{st}_R, m) \leftarrow \textsc{Recv}(\mathtt{st}_R, c)$
  if $c \notin C \land m \neq \perp \land m \neq \top$ **then**
    return m
  else
    return $\perp$
return $\perp$

7 Relations Between Datagram Notions

In this section we present relations between our security notions for datagram channels in Theorems 6, 7, and 8. Proofs of these relations appear in the full version [21]. As with the datastream notions, our datagram security definitions do not imply and are not implied by confidentiality properties because plaintext values may be leaked via the ciphertext lengths.

Theorem 6. *If a Datagram channel satisfies FEP-CCA, it satisfies INT-CTXT.*

Theorem 7. *If a Datagram channel satisfies FEP-CPA and INT-CTXT, it satisfies FEP-CCA.*

To obtain confidentiality from our datagram FEP notions, we require a property that ensures message lengths do not leak information about the plaintext content. We define Datagram Channel Length Regularity in Definition 8.

Definition 8. *Let M^0 and M^1 be n-length sequences of elements in $\mathcal{M}_{\ell_{in}} \cup \{\top\}$ such that, for all i, either $|M_i^0| = |M_i^1|$ or $M_i^0 = M_i^1 = \top$. Let P be an n-length integer sequence. Let $(\mathtt{st}_S^0, C_i^0) \leftarrow$ Send$(\mathtt{st}_S^0, P_i, M_i^0)$ and $(\mathtt{st}_S^1, c_i^1) \leftarrow$ Send$(\mathtt{st}_S^1, P_i, M_i^1)$, where in both cases Send is initialized with Init and is then called sequentially as $i = 1..n$, updating its state with each call. A datagram channel is* length regular *if, for any such M^0, M^1, and P, and for all i, either $|C_i^0| = |C_i^1|$ or $C_i^0 = C_i^1 = \perp$.*

Theorem 8. *If a Datagram channel satisfies FEP-CCA and Datagram Channel Length Regularity, it satisfies IND-CCA.*

8 A Fully Encrypted Datagram Construction

In the following section we introduce our datagram construction of a Fully Encrypted Protocol, and we prove it satisfies satisfies correctness, the desired security properties, and Traffic Shaping.

8.1 The Construction

We give a simple construction of a protocol that satisfies our two obfuscation security definitions and Traffic Shaping for datagram channels. To satisfy most of our security and obfuscation properties, we can simply use atomic encryption and decryption since datagram messages are atomic by construction. However, the Traffic Shaping property presents a challenge: not all lengths are possible outputs of an AEAD scheme, and since the messages are not ordered, they must each include a nonce as well.

In our construction, we assume the same AEAD scheme from Section 2. For simplicity, we further assume that Enc includes in its outputs each nonce as a prefix of the ciphertext when it is called, and that Dec extracts the first ℓ_{Nonce} bytes from its input as the nonce to use for decryption. Let $\ell_{\text{Overhead}} = \ell_{\text{Nonce}} + \ell_{\text{Tag}}$ represent the encryption overhead since our construction uses fresh nonces for each ciphertext. We give an explicit encoding scheme for our message space into bytes as follows: we encode the null message $\top$ as a single zero byte, and prefix all other messages (including the empty string ϵ) with a leading 1 byte. We set $\ell_{\text{out}} = 65507$ (the maximum size UDP payload length), and we include in our message space all byte strings of length up to $\ell_{\text{in}} = \ell_{\text{out}} - \ell_{\text{Overhead}} - 3$, reserving two bytes for the plaintext length and one for the message type. We denote the length of a ciphertext containing the null message as $\ell_{\text{null}} = 1 + \ell_{\text{Overhead}}$. We abuse notation and write $|m|$ to mean both the size of m and the two byte unsigned integer that represents that size, depending on the context.

Our protocol is stateless, so each function simply returns the symmetric key k as the new state after each call. The Send function generates a fresh nonce, and if Traffic Shaping is off, directly applies Enc to produce the datagram. If the desired output length p is too small for an authenticated message, we produce random bytes (Lines 6–7). We produce arbitrary length chaff messages (Line 9) or padded plaintext messages (Line 13), and return with an error if p is too large to fit in a datagram or m is too large to fit within $p \geq 0$ (Line 10). Recv interprets all small messages as chaff, and then decrypts larger messages, returning the plaintext, chaff symbol, or error where appropriate.

8.2 Properties

We prove in Theorem 9 that our datagram construction satisfies all of the desired properties presented in Section 6. The proofs are in the full version [21]. Our approach is to establish correctness, Traffic Shaping, datagram channel length regularity, FEP-CPA, and INT-CTXT, and then apply the theorems from Section 7 to establish the remaining properties.

Theorem 9. *If the AEAD scheme (Gen, Enc, Dec) in the channel construction in Figure 3 satisfies IND\$-CPA, INT-CTXT, and length additivity (see Sec 2) then the channel construction satisfies correctness for Datagram channels, Traffic Shaping, FEP-CCA, INT-CTXT, and IND-CCA.*

9 Existing Fully Encrypted Protocols

Since our work is motivated the lack of security definitions for existing Fully Encrypted Protocols, we identified a set of protocols that make an attempt to appear fully encrypted above the transport layer for analysis. Our goal in analyzing these protocols is to determine if these protocols satisfy our definitions, and to what extent they have identifying features that our security definitions are designed to address. We excluded closed-source protocols but note that many existing implementations of FEPs that do not appear in our list nonetheless adapt the approach of one of the protocols we analyze [37, 41, 49]. We did not perform a thorough audit of the security engineering of the protocols' implementations, which we consider outside the scope of our paper.

9.1 Methodology

In each case we analyzed the protocol source code and documentation (if available) in order to identify the protocol behavior. To determine whether the protocols satisfy passive FEP security, we examined if the protocol outputs were all either random bytes or pseudorandom ciphertext. For active FEP security, we identified whether these fragments or messages were also authenticated, and for datastream protocols, whether the close behavior of the protocol satisfied our definition of a secure close function.

Protocol source code and documentation also informed our analysis of techniques for length obfuscation and of the minimum-length messages of the protocols. We examined these protocol aspects to understand how well Traffic Shaping was satisfied. Existing FEPs typically accomplish length obfuscation with padding, where extra bytes are added to messages. Padding can only increase the length of the output for a given input, and thus we expect to find that existing protocols possess minimum message lengths that can serve an undesirable identifying feature of the protocol. We determined whether padding was included in the protocols and how it was added by the code, and we confirmed the presence of padding in the experiments described below. In addition, we identified protocol message formats and layouts from these sources, which we used to form hypotheses on the minimum message lengths for each protocol, which we verified via experimentation. We note that the minimum lengths we determine technically apply to the amount of data written to the socket buffer by a Send call, and the network stack may further fragment the message. However, such fragmentation is not common, and the amount written to the buffer is typically the amount sent in a network packet (*e.g.*, we observed no fragmentation in our experiments).

We observed that channel-close behavior was rarely documented and in practice varied significantly between implementations, and so we relied almost entirely on our experiments to determine when protocols terminated a connection. We note that many protocols also include time-based close behavior, which we did not analyze.

Open-source FEPs are designed to be flexible, and so all protocols in our list can be run under many configurations. Our primary goal was to identify the intended behavior of the protocol designers under recommended or default settings. Therefore, in our experiments we selected defaults, used recommended settings, and followed official examples to configure each protocol.

$\textsc{Send}(k, m, p)$:

$\mathsf{nonce} = \textsc{Rand}(\ell_{\text{Nonce}})$
if $p < 0$ and $m = \top$ **then**
  return $(k, \textsc{Enc}_k(\mathsf{nonce}, 0))$
if $p < 0$ and $|m| \leq \ell_{\text{in}}$ **then**
  return $(k, \textsc{Enc}_k(\mathsf{nonce}, 1 \| |m| \| m))$
if $m = \top$ and $p < \ell_{\text{null}}$ **then**
  return $(k, \textsc{Rand}(p))$
if $m = \top$ and $\ell_{\text{null}} \leq p \leq \ell_{\text{out}}$ **then**
  return $(k, \textsc{Enc}_k(\mathsf{nonce}, 0 \| 0^{p-\ell_{\text{null}}}))$
if $p > \ell_{\text{out}}$ or $|m| > p - \ell_{\text{Overhead}} - 3$ **then**
  return $(k, \perp)$
$\mathsf{pt} = 1 \| |m| \| 0^{(p-|m|-\ell_{\text{Overhead}}-3)} \| m$
return $(k, \textsc{Enc}_k(\mathsf{nonce}, \mathsf{pt}))$

$\textsc{Recv}(k, c)$:

if $|c| < \ell_{\text{null}}$ **then**
  return $\top$
$\mathsf{buf} \leftarrow \textsc{Dec}_k(c)$
if $\mathsf{buf} = \perp_{\textsc{Dec}}$ **then**
  return $\perp$
if $\mathsf{buf}[1] = 0$ **then**
  return $\top$
$i \leftarrow |\mathsf{buf}| - \mathsf{buf}[2,3] + 1$ ▷ Two-byte unsigned int
return $(k, \mathsf{buf}[i..])$

$\textsc{Init}(1^\lambda)$:

$k \leftarrow \textsc{Gen}(1^\lambda)$
return (k, k)

Figure 3: Our datagram channel construction

Datastream Protocol	Close Behavior	FEP-CPFA	FEP-CCFA	Length Obfuscation	Min Size
Shadowsocks-libev (request)	Never	✓	✓*	None	35
Shadowsocks-libev (response)	Auth Fail	✓	X	None	35
V2Ray-Shadowsocks (request)	Drain*	✓	X	None	35
V2Ray-Shadowsocks (response)	Auth Fail	✓	X	None	35
V2Ray-VMess	Drain*	✓	X	Padding	18†
Obfs4	Auth Fail	✓	X	Padding	44*
OpenVPN-XOR	Auth Fail	X	X	None	42†
Obfuscated OpenSSH	Auth Fail	X	X	None	16
Obfuscated OpenSSH-PSK	Auth Fail	✓	X	None	16
kcptun	Never	✓	X	None	52†
Construction (Sec 5.1)	Never	✓	✓	Traffic Shaping	1
Datagram Protocol		**FEP-CPA**	**FEP-CCA**	**Length Obfuscation**	**Min Size**
Shadowsocks-libev		✓	✓	None	55*
Wireguard-SWGP (paranoid)		✓	✓	Padding	75†*
OpenVPN-XOR		X	X	None	40†
Construction (Sec 8.1)		✓	✓	Traffic Shaping	0

Table 1: Our definitions applied to FEPs. *See discussion for significant nuances. † The smallest message is a keepalive. ✓ We have strong evidence the protocol satisfies the definition from documentation, code, and our experiments (properties are proven for our constructions). X We have demonstrated by experiment that the definition is not satisfied, or that fact is clear from the protocol design.

Many assessed protocols are designed around a request-response architecture, and thus their message format, cryptographic structure, and general behavior may differ depending on whether the messages are outgoing (Client → Server) or incoming (Server → Client). Our protocol framework is unidirectional, and so we model each of these directions as a distinct protocol, combining them only when our results for both directions are identical.

9.2 Results

Our results appear in Table 1. For padding approaches, we include only padding for the purpose of countering traffic analysis (*e.g.*, not padding required to align a plaintext message to a block boundary before encryption), and padding that could apply to all messages (*e.g.*, not padding only in the initial handshake phase). We highlight the following results:

(1) Nearly all existing FEPs satisfy passive FEP security, but no datastream FEP satisfies active FEP security (Shadowsocks

does in one direction). Also, the datastream FEPs are identifiable to an active attacker based on their close behavior.
(2) No existing protocol, whether datastream or datagram, satisfies Traffic Shaping, and all protocols are identifiable by their minimum message length.

We summarize our results in general and highlight specific results of interest below, providing the detailed descriptions of our methodology, analysis, and experiments in the full version [21].

We first describe the eight FEP implementations studied: (1) Shadowsocks [47] is a fully encrypted SOCKS5 proxy with many implementations and the ability to proxy both datastream and datagram traffic. We examined the Shadowsocks-libev [53] implementation in both TCP and UDP configurations. (2) We also studied a Shadowsocks implementation in the censorship-circumvention software suite V2Ray [40]. (3) VMess [50] is a custom FEP also implemented inside V2Ray for censorship circumvention. (4) Obfs4 [34] is a FEP designed for Tor [16]. (5) OpenVPN [35] is open-source VPN software that functions in datastream or datagram mode, and there is a well-known patch available designed to obfuscate its traffic called the XOR patch [36]. (6) Obfuscated SSH is modification of OpenSSH that fully encrypts its handshake messages, and it can be configured with a Pre-Shared Key. (7) kcptun is a FEP designed to be reliable and fast over very noisy networks. (8) Wireguard-SWGP [48] is a proxy for Wireguard [18] that fully encrypts its traffic.

Datastream protocols largely terminate the connection immediately when an authentication tag failed to validate ("Auth Fail" in Table 1). This behavior cannot be realized by any secure close function when the protocol ciphertext lengths are hidden (*e.g.*, with a variable-length ciphertext and an encrypted length field, as is the case for Shadowsocks, VMess, Obfs4, and Obfuscated OpenSSH) because this means the close function, without access to encryption keys, must be able to identify a ciphertext boundary within the concatenation of all ciphertexts fragments from the sender.

V2Ray protocols additionally include a "Drain" initial behavior (the protocol changes this behavior later in the connection) which upon decryption error delays terminating the connection until a predetermined number of total bytes have been received, randomized based on a user and per connection. The drain approach does not satisfy our notion of a secure close function, as it does not apply if the pre-determined amount has been exceeded before the error. Thus, the V2Ray protocols cannot satisfy FEP-CCFA.

Most datastream protocols satisfy FEP-CPFA. Only Obfuscated OpenSSH, which transmits symmetric key material in the clear, and the OpenVPN XOR patch, which doesn't satisfy confidentiality [57], do not. On the other hand, only Shadowsocks-libev in one direction has a close behavior compatible with FEP-CCFA: the server holds the connection open indefinitely upon error as an intentional design feature. It does report errors to the application layer, though, technically violating the FEP-CCFA definition. However, this behavior could plausibly satisfy FEP-CCFA with a reasonable and minor modification to the notion (allowing errors) or to the transport protocol (suppressing errors).

Most datagram protocols satisfy FEP-CCA, since there are no closures in this setting. Only OpenVPN-XOR does not, and it also violates FEP-CPA in this setting as well.

Few protocols include padding of any sort (only Obfs4, VMess, and Wireguard-SWGP). The Shadowsocks datastream protocol had the same minimum message length in both implementations. However, other protocols varied significantly, with several (kcptun, OpenVPN-XOR, VMess, and Wireguard-SWGP) sending minimum-length messages as keepalives, which are thus transmitted whenever the application is quiescent. Such keepalives represent a novel identifying feature for FEPs, and they highlight the value of Traffic Shaping. Datastream FEPs typically had higher minimum message lengths because they all included a per-message nonce.

Three determinations of the minimum length require some care: Obfs4 servers upon initialization select a random set of random possible output lengths, which persists. Thus, each Obfs4 server *has its own minimum message length*, with 44 bytes as the absolute minimum across all such setups. Shadowsocks-libev in the datagram setting can send messages of length 52 if directed to produce messages with invalid addresses. Wireguard-SWGP has a minimum length that depends on the configured MTU of the protocol, which the documentation suggests be set carefully, but the minimum message length is 75 with the default MTU of 1500.

We also experimentally observed that the protocols in the V2Ray framework fail to satisfy datastream integrity (and thus cannot satisfy FEP-CCFA), dropping isolated messages silently when certain ciphertext bytes were modified in transit. We reported this as a security vulnerability to the project maintainers on January 4, 2024 and the issue was resolved in version 5.14.1.[2]

10 Discussion

Modeling limitations. Our modeling choices limit our results in some ways. We follow existing models for stateful communication channels, which, for simplicity, provide unidirectional rather than bidirectional communication and omit time. Bidirectional communication can be achieved within our model by composing two unidirectional channels, one in each direction. However, such channels are not allowed to share state, including keys and other initialization parameters, and not observing this restriction can violate the security guarantees. Such a limitation reduces potential efficiency over the truly bidirectional setting. Similarly, omitting time from our model precludes some convenient protocol features, such as timeouts leading to channel closures and time-based replay protections. This limits both the design of new protocols and the analysis of existing ones. However, security models have been designed that include time [5, 14], and our models and constructions can be extended to work within such a setting.

FEP indistinguishability. The ability of an adversary to distinguish FEPs from one another (as opposed to distinguishing them from non-FEPs) is an important security concern. Identifying the use of a specific FEP, implementation, or software version enables an adversary to deploy and tailor exploits. It can also enable user profiling, which may be particularly sensitive in the context of censorship circumvention, where software distribution may be performed via physical or social networks, and so the use of a specific implementation or software version can imply membership in such a network.

[2] The bug was caused by a failure to propogate errors properly when decryption fails during the data transport phase.

Unfortunately, as we have shown in Section 9, existing FEPs and their implementations can be effectively distinguished in practice because of a failure to satisfy one or more of the security definitions we present in this work. We consider our definitions a major step towards articulating and achieving this goal for FEPs, but using them to guarantee FEP indistinguishability requires some further determinations to be made that we consider outside the scope of our paper. In particular, a secure close function must be selected and a Traffic Shaping schedule or distribution must be determined. Further, some variability in these selections may be necessary among FEPs for practical functionality and efficiency reasons.

Secure close functions. Secure close functions can present practical concerns since they constrain protocol behavior. For example, they may require channels to leave connections open indefinitely on error (*e.g.*, $C \equiv 0$ as in Shadowsocks and our construction), deploy a less-efficient message format (*e.g.*, the fixed-length ciphertexts of InterMAC [12]), or terminate connections even if errors are not introduced into the ciphertext stream (*e.g.*, if connections are closed after a fixed number of bytes or after a fixed byte sequence appears in the random ciphertext stream). However, some developers have already chosen in practice to make these tradeoffs in favor of security, both in Shadowsocks [53], which is the most widely deployed FEP in use, and libInterMAC [2], which has been thoroughly evaluated for efficiency. Additionally, the secure use of timeouts (which are outside our current model) to close connections may alleviate these concerns in some cases.

Traffic Shaping. The Traffic Shaping property presents a tradeoff between security against traffic analysis and efficiency, to be set as appropriate for a given application. A thorough analysis of how best to use Traffic Shaping in particular use cases is important but requires the use of very different tools than we apply in this work. One simple option would be to send fixed-size messages at a constant rate, requiring predictable bandwidth and providing a maximum message latency. Alternately, one could sample message times and sizes from a known distribution for a target application (the transported application to maximize efficiency, or an uncensored application in the context of censorship circumvention). The main goal of our Traffic Shaping definition is to formalize a maximally flexible functionality *at the transport layer*, providing a powerful interface that can be used to make traffic from any two distinct applications appear as similar as efficiency goals allow.

Recommendations for FEP designers. We intend our results to inform the design of existing and future FEPs. Many of the existing FEPs could satisfy our novel security definitions with straightforward changes to the message formats and protocol implementation. We make the following specific recommendations regarding the two novel protocol features we introduce, secure close functions and Traffic Shaping:

(1) We suggest ideally adopting the close function of InterMAC, where the receiver closes after error only at a multiple of n bytes, for constant n, by adding a MAC at those positions. These MACs would be in addition to any MACs needed to support variable-length records and could occur at relatively low frequency, where such frequency should be chosen to be acceptable across applications and protocols to maximize FEP indistinguishability (*e.g.* every 10K bytes). We further suggest closing the connection after a certain amount of time has passed with no ciphertext received, which is technically outside our model. The combination of these close behaviors would permit connections to eventually close after an error is observed without adding too much bandwidth overhead. An alternative suggestion is simply to adopt a connection timeout after not receiving any ciphertext. This behavior does permit an active adversary to keep a connection open by continually sending traffic, but it is a more minor change to implement.
(2) To implement Traffic Shaping, datastream FEPs should be extended to buffer content that does not fit within the prescribed output length p and to produce padding when p exceeds the amount necessary for the message. Datagram FEPs should be extended to include the use of null messages and to refuse to send messages that cannot fit within the prescribed length. FEP developers should thus implement Traffic Shaping, although existing applications need not necessarily use it and instead await more work on appropriate traffic patterns.

11 Related Work

We build on the model of Fischlin *et al.* [24] to study data streams in the presence of fragmentation, a problem also been studied elsewhere [1–3, 12]. Fragmentation is a problem for FEPs, as plaintext length fields cannot be used. The notion of Boundary Hiding [12] is related to but does not imply our FEP notions because, for one, it does not enforce random-appearing outputs.

Some aspects of Fully Encrypted Protocols have been studied, such as active probing [4, 26], and attacks on confidentiality [29, 38]. Much research has focused on detecting FEPs [27, 52, 56, 57] using a variety of approaches. Fifield [23] outlined a series of implementation weaknesses in existing FEPs. Bernstein *et al.* [10] developed a method to encode elliptic curve points as uniformly random strings.

On channel closures, Boyd and Hale [13] consider channels with in-band intentional termination signals that generate channel closures, and Marson and Poettering [33] provide security definitions for fully bidirectional channels. Hansen [28] and Albrecht *et al.* [2] discuss similar issues to the channel closures we discuss in our work around the difficulty of realizing active boundary hiding, where applications themselves leak information to an adversary through their behavior.

Fischlin *et al.* [25] analyze dTLS [43] and QUIC [31] in the datagram setting, considering especially robustness to message drops. Our datagram model can also apply to other encrypted transport protocols like IPSec [19]. Stateful encryption models similar to our datagram model are given by Bellare et al. [7] and Kohno et al. [30].

Some FEPs designs have come from elsewhere than the open-source community. Obfs4 [34] is based on Scramblesuit [54] and has been forked under the new name Lyrebird [32]. InterMAC, which is a near-FEP, has been both analyzed [12] and implemented [2]. The IETF proposal for a Pseudorandom Extension of cTLS [46] suggest a FEP encoding of TLS to improve protocol security and privacy rather than for censorship circumvention.

12 Conclusion and Future Work

A natural enhancement to our constructions would be a fully encrypted key exchange for forward secrecy. Obfs4 [34] uses Elligator [10] for this purpose, but it remains to prove its security as well as investigate other techniques. Relatedly, one could formulate a notion of forward metadata secrecy, where the FEP properties may still be preserved even if a long-term key is compromised. In addition, FEP notions should be explored in models where securely creating shared state is not trivial but bidirectional communication is possible.

There are also other practical concerns to consider. Active probing attacks are a common problem for real-world FEPs, which we do not address in this work. We also do not optimize efficiency in our FEP protocol constructions. Finally, while we introduce the Traffic Shaping capability, we do not answer the distinct and important question of how traffic *should be shaped*.

Acknowledgments

This work was supported by the Office of Naval Research (ONR) and the Defense Advanced Research Projects Agency (DARPA). We would also like to thank Ben Robinson, who helped perform the Wireguard experiments, and David Fifield and the anonymous referees who reviewed this paper for their helpful suggestions and advice.

References

[1] Martin R Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G Paterson. 2016. A surfeit of SSH cipher suites. In *ACM Conference on Computer and Communications Security (CCS)*.
[2] Martin R Albrecht, Torben Brandt Hansen, and Kenneth G Paterson. 2019. libInterMAC: Beyond Confidentiality and Integrity in Practice. *IACR Transactions on Symmetric Cryptology* (2019).
[3] Martin R Albrecht, Kenneth G Paterson, and Gaven J Watson. 2009. Plaintext recovery attacks against SSH. In *IEEE Symposium on Security and Privacy (S&P)*.
[4] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. 2020. How China detects and blocks Shadowsocks. In *ACM Internet Measurement Conference (IMC)*.
[5] Michael Backes, Praveen Manoharan, and Esfandiar Mohammadi. 2014. TUC: time-sensitive and modular analysis of anonymous communication. In *IEEE Computer Security Foundations Symposium (CSF)*.
[6] Diogo Barradas and Nuno Santos. 2017. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017, 4 (2017).
[7] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. 2002. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In *ACM Conference on Computer and Communications Security (CCS)*.
[8] Mihir Bellare and Chanathip Namprempre. 2008. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of cryptology* 21, 4 (2008).
[9] David Benjamin. 2020. *RFC 8701: Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility.* Technical Report. Internet Engineering Task Force.
[10] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *ACM conference on Computer and Communications Security (CCS)*.
[11] Cecylia Bocovich and Ian Goldberg. 2016. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *ACM Conference on Computer and Communications Security (CCS)*.
[12] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G Paterson, and Martijn Stam. 2012. Security of symmetric encryption in the presence of ciphertext fragmentation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.
[13] Colin Boyd and Britta Hale. 2017. Secure channels and termination: The last word on TLS. In *International Conference on Cryptology and Information Security in Latin America*.
[14] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. 2017. A universally composable treatment of network time. In *IEEE Computer Security Foundations Symposium (CSF)*.
[15] Roger Dingledine and Nick Mathewson. 2023. *Tor Protocol Specification.* Technical Report. https://raw.githubusercontent.com/torproject/torspec/main/tor-spec.txt.
[16] Roger Dingledine, Nick Mathewson, and Paul F Syverson. 2004. Tor: The second-generation onion router. In *USENIX Security Symposium*.
[17] Lucas Dixon, Thomas Ristenpart, and Thomas Shrimpton. 2016. Network traffic obfuscation and automated internet censorship. *IEEE Symposium on Security and Privacy (S&P)* (2016).
[18] Jason A Donenfeld. 2017. Wireguard: next generation kernel network tunnel. In *Network and Distributed System Security Symposium (NDSS)*.
[19] Naganand Doraswamy and Dan Harkins. 2003. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional.
[20] Ellis Fenske and Aaron Johnson. 2023. Security notions for Fully Encrypted Protocols. *Free and Open Communications on the Internet (FOCI)* (2023).
[21] Ellis Fenske and Aaron Johnson. 2024. Bytes to Schlep? Use a FEP: Hiding Protocol Metadata with Fully Encrypted Protocols. arXiv:2405.13310 [cs.CR] https://arxiv.org/abs/2405.13310
[22] David Fifield. 2017. The limits of timing obfuscation in obfs4. https://lists.torproject.org/pipermail/tor-dev/2017-June/012310.html. Accessed October 7, 2022.
[23] David Fifield. 2023. Comments on certain past cryptographic flaws affecting fully encrypted censorship circumvention protocols. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1362
[24] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G Paterson. 2015. Data is a stream: Security of stream-based channels. In *Annual Cryptology Conference (Crypto)*.
[25] Marc Fischlin, Felix Günther, and Christian Janson. 2020. Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3. Cryptology ePrint Archive. https://eprint.iacr.org/2020/718
[26] Sergey Frolov, Jack Wampler, and Eric Wustrow. 2020. Detecting Probe-resistant Proxies. In *Network and Distributed System Security Symposium (NDSS)*.
[27] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *Network and Distributed System Security Symposium (NDSS)*.
[28] Torben Brandt Hansen. 2020. *Cryptographic Security of SSH Encryption Schemes*. Ph. D. Dissertation. Royal Holloway, University of London.
[29] Qingbing Ji, Zhihong Rao, Man Chen, and Jie Luo. 2022. Security analysis of shadowsocks (R) protocol. *Security and Communication Networks* (2022).
[30] Tadayoshi Kohno, Adriana Palacio, and John Black. 2003. Building Secure Cryptographic Transforms, or How to Encrypt and MAC. Cryptology ePrint Archive. https://eprint.iacr.org/2003/177
[31] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Conference of the ACM special interest group on data communication (SIGCOMM)*.
[32] Lyrebird. 2024. https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/lyrebird. Accessed October 14, 2022.
[33] Giorgia Azzurra Marson and Bertram Poettering. 2017. Security notions for bidirectional channels. *Cryptology ePrint Archive* (2017).
[34] obfs4 (The obfourscator). 2024. https://github.com/Yawning/obfs4. Accessed October 14, 2022.
[35] OpenVPN Project. 2024. https://openvpn.net/. Accessed April 24, 2024.
[36] OpenVPN XORPatch. 2018. https://github.com/clayface/openvpn_xorpatch.
[37] Outline Client. 2024. https://github.com/Jigsaw-Code/outline-client. Accessed October 14, 2022.
[38] Zhiniang Peng. 2020. Redirect Attack on Shadowsocks Stream Ciphers. https://github.com/edwardz246003/shadowsocks.
[39] Trevor Perrin. 2018. The noise protocol framework. https://noiseprotocol.org
[40] Project V. 2024. https://www.v2ray.com/en/. Accessed October 14, 2022.
[41] Psiphon. 2024. https://psiphon.ca/.
[42] Jean-François Raymond. 2001. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*.
[43] E Rescorla, H Tschofenig, and N Modadugu. 2022. RFC 9147: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3.
[44] Phillip Rogaway. 2002. Authenticated-encryption with associated-data. In *ACM Conference on Computer and Communications Security (CCS)*.
[45] Phillip Rogaway. 2011. Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan* (2011).
[46] Benjamin M. Schwartz and Christopher Patton. 2022. *The Pseudorandom Extension for cTLS*. Internet-Draft draft-cpbs-pseudorandom-ctls-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-cpbs-pseudorandom-ctls/01/ Work in Progress.
[47] Shadowsocks. 2022. https://shadowsocks.org/.
[48] Simple Wireguard Proxy. 2024. https://github.com/database64128/swgp-go.
[49] Tunnelblick. 2023. https://tunnelblick.net/.
[50] VMess. 2024. https://www.v2ray.com/chapter_02/protocols/vmess.html. Accessed October 14, 2022.
[51] Ryan Wails, George Arnold Sullivan, Micah Sherr, and Rob Jansen. 2024. On Precisely Detecting Censorship Circumvention in Real-World Networks. In *Network*

and Distributed System Security Symposium (NDSS).

[52] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *ACM Conference on Computer and Communications Security (CCS).*

[53] Clow Windy, Max Lv, and Linus Yang. 2020. Shadowsocks-libev. https://github.com/shadowsocks/shadowsocks-libev/.

[54] Philipp Winter, Tobias Pulls, and Juergen Fuss. 2013. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *ACM Workshop on Privacy in the Electronic Society (WPES).*

[55] Charles V Wright, Scott E Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis.. In *Network and Distributed System Security Symposium (NDSS).*

[56] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. 2023. How the Great Firewall of China detects and blocks fully encrypted traffic. In *USENIX Security Symposium.*

[57] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. 2022. OpenVPN is Open to VPN Fingerprinting. In *USENIX Security Symposium.*