

TLS WG
Internet-Draft
Intended status: Experimental
Expires: 13 October 2022

B. Schwartz
Google LLC
C. Patton
Cloudflare, Inc.
11 April 2022

The Pseudorandom Extension for cTLS
draft-cpbs-pseudorandom-ctls-01

Abstract

This draft describes a cTLS extension that allows each party to emit a purely pseudorandom bitstream.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at <https://github.com/bemasc/pseudorandom-ctls>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 13 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

Internet-Draft

Pseudorandom cTLS

April 2022

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Conventions and Definitions	2
2.	Introduction	3
2.1.	Background	3
2.2.	Goal	3
2.2.1.	Requirements	3
2.2.2.	Non-requirements	4
2.2.3.	Experimental status	4
3.	The Pseudorandom Extension	4
3.1.	Form	4
3.2.	Use	5
3.2.1.	Key Derivation	6
3.2.2.	With Streaming Transports	6
3.2.3.	With Datagram Transports	7
3.3.	Protocol confusion defense	8
4.	Plaintext Alerts	9
5.	Operational Considerations	9
5.1.	Multiple profiles and key rotation	9
5.2.	Computational cost	10
6.	Security Considerations	10
7.	Privacy Considerations	11
8.	IANA Considerations	11
8.1.	TSPRP Registry	11
8.2.	cTLS Template Key registry	11
8.3.	TLS ContentType Registry	12
9.	References	12
9.1.	Normative References	12
9.2.	Informative References	13
	Acknowledgments	13
	Authors' Addresses	13

[1.](#) Conventions and Definitions

The contents of a two-party protocol as perceived by a third party are called the "wire image".

A Tweakable Strong Pseudorandom Permutation (TSPRP) is a variable-

input-length block cipher that is parameterized by a secret "key" and a public "tweak". Also known as a "super-pseudorandom permutation" or "wide block cipher".

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

[2.](#) Introduction

[2.1.](#) Background

Compact TLS [[cTLS](#)] is a compact representation of TLS 1.3 (or later), intended for uses where compatibility with previous versions of TLS is not required. It defines a pre-configuration object called a "template" that contains a profile of the capabilities and behaviors of a TLS server, which is known to both client and server before they initiate a connection. The template allows both parties to omit information that is irrelevant or redundant, allowing a secure connection to be established while exchanging fewer bits on the wire.

Every cTLS template potentially results in a distinct wire image, with important implications for user privacy and ossification risk.

One interesting consequence of conventional wire formats (i.e. not pseudorandom) is the risk of protocol confusion attacks. For example, in the NAT Slipstreaming attacks [[SLIPSTREAM](#)], a web server causes a browser to send HTTP data that can be confused for another protocol (e.g. SIP) that is processed by a firewall. Because firewalls are typically focused on attacks arriving from outside the network, malicious payloads sent from a trusted client can trick some firewalls into disabling their own protections.

[2.2.](#) Goal

The goal of this extension is to enable two endpoints to agree on a TLS-based protocol whose wire image is purely pseudorandom.

[2.2.1.](#) Requirements

- * Privacy: A third party without access to the template cannot tell whether two connections are using the same pseudorandom cTLS template, or two different pseudorandom cTLS templates.
- * Ossification risk: Every byte sent on the underlying transport is pseudorandom to an observer who does not know the cTLS template.
- * Efficiency: Zero size overhead and minimal CPU cost in the simplest case. Support for servers with many cTLS templates, when appropriately constructed.

- * Protocol confusion attack resistance: This attack assumes a malicious server or client that can coerce its peer into sending a ciphertext that could be misinterpreted as a different protocol by a third party. This extension must enable each peer to ensure that its own output is unlikely to resemble any other protocol.

[2.2.2.](#) Non-requirements

- * Efficient support for demultiplexing arbitrary cTLS templates.
- * Addressing information leakage in the length and timing of transmissions.

[2.2.3.](#) Experimental status

This specification has experimental status (INTENDED). The goals of this experiment include:

- * To assess the internet's tolerance of unrecognized protocols.
- * To gain experience with TSPRPs in a protocol context.
- * To exercise cTLS's extensibility features.
- * To support practical and theoretical research into protocol distinguishability.

[3.](#) The Pseudorandom Extension

[3.1.](#) Form

A cTLS template is structured as a JSON object. This extension is represented by an additional key, "pseudorandom", whose value is an object with at least two string-valued keys: "tsprp" (a name from the TSPRP registry (see [Section 8.1](#))) and "key" (a base64-encoded shared secret whose length is specified by the TSPRP). For example, a cTLS template might contain an entry like:

```
"pseudorandom": {  
  "tsprp": "exp-hctr2",  
  "key": "nx2kEm50FCE...Ty0hG0w477EHS"  
},
```

[3.2.](#) Use

The cTLS Record Layer protocol is comprised of AEAD-encrypted ciphertext fragments interleaved with plaintext fragments. Each record is prefixed by a plaintext header, and some records, like those containing the ClientHello and ServerHello, are not encrypted at all. The ciphertext fragments are pseudorandom already, so this extension specifies a transformation of the plaintext fragments that ensures that all bits written to the wire are pseudorandom.

Conceptually, the extension sits between the cTLS Record Layer and the underlying transport (e.g. TCP, UDP). The transformation is based on a TSPRP with the following syntax:

```
TSPRP-Encipher(key, tweak, message) -> ciphertext  
TSPRP-Decipher(key, tweak, ciphertext) -> message
```

The TSPRP specifies the length (in bytes) of the key. The tweak is a byte string of any length. The TSPRP uses the key and tweak to encipher the input message, which also may have any length. The output ciphertext has the same length as the input message.

Pseudorandom cTLS uses the TSPRP to encipher all plaintext handshake records, including the record headers. As long as there is sufficient entropy in the key_share extension or random field of the ClientHello (resp. ServerHello) the TSPRP output will be pseudorandom.

TODO: Check that the assumptions hold for HelloRetryRequest. As long as no handshake messages are repeated verbatim, it should be fine, but we need to check whether an active attacker can trigger a replay.

Pseudorandom cTLS also enciphers every record header. In addition to the header, 16 bytes of the AEAD ciphertext itself is enciphered to ensure the input has enough entropy. Any AEAD algorithm whose ciphertext overhead is less than 16 bytes is not compatible with this specification.

By default, Pseudorandom cTLS assumes that the TLS ciphertext is using an AEAD algorithm whose output is purely pseudorandom, such as AES-GCM and ChaCha20-Poly1305. If the ciphertext might not be pseudorandom (e.g. when handling hostile plaintext), the ciphertext can be "reciphered" to ensure pseudorandomness (see [Section 3.3](#)).

[3.2.1.](#) Key Derivation

To provide clear separation between data sent by the client and the server, the client and server encipher data using different keys, derived from the profile key as follows:

```
client_key = TSPRP-Encipher(key, "derive", zeros)
server_key = TSPRP-Encipher(key, "derive", ones)
```

where zeros and ones are messages the same size as key, with all bits set to zero and one respectively. This procedure guarantees that client_key and server_key are distinct and would appear unrelated to any party who does not know the profile key.

[3.2.2.](#) With Streaming Transports

When used over a streaming transport, Pseudorandom cTLS requires that headers have predictable lengths. Therefore, if a Connection ID is negotiated, it MUST always be included. Normally, when TLS runs on top of a streaming transport, Connection IDs are not enabled, so this is not expected to be a significant limitation.

The transformation performed by the sender uses TSPRP-Encipher() and client_key or server_key. The sender first constructs any CTLSPplaintext records as follows:

1. Set tweak = "hs".
2. Replace the message with TSPRP-Encipher(client/server_key, tweak, message).
3. Fragment the message if necessary, ensuring each fragment is at least 16 bytes long.
4. Change the content_type of the final fragment to ctls_handshake_end(TBD) (see [Section 8.3](#)).

Note: This procedure requires that handshake messages are at least 16 bytes long. This condition is automatically true in most configurations.

The sender then constructs cTLS records as usual, but applies the following transformation before sending each record:

1. Let prefix be the first 19 bytes of the record.
2. If the record is CTLSPplaintext, set tweak = "".

3. If the record is CTLS ciphertext, let tweak be the 64-bit Sequence Number in network byte order.
4. Replace prefix with TSPRP-Encipher(client/server_key, tweak, prefix).

OPEN ISSUE: How should we actually form the tweaks? Should we add some kind of chaining, within a stream or binding ServerHello to

ClientHello?

[3.2.3.](#) With Datagram Transports

Pseudorandom cTLS applies to datagram applications of cTLS without restriction. If there are multiple records in the datagram, encipherment starts with the last record header and proceeds back-to-front.

Given the inputs:

- * payload, an entire datagram that may contain multiple cTLS records.
- * TSPRP-Decipher() and client_key or server_key
- * connection_id, the ID expected on incoming CTLSCiphertext records

The recipient decipheres the datagram as follows:

1. Let max_hdr_length = max(15, len(connection_id) + 5). This represents the most data that might be needed to read the CTLSP Plaintext and DTLS Handshake headers (Section 5.2 of [\[DTLS13\]](#)) or the CTLSCiphertext header.
2. Let index = 0.
3. While index != len(payload):
 1. Let prefix = payload[index : min(len(payload), index + max_hdr_length + 16)]
 2. Let tweak = "datagram" + len(payload) + index.
 3. Replace prefix with TSPRP-Decipher(client/server_key, tweak, prefix).
 4. Set index to the end of this record.

CTLSP Plaintext records are subject to an additional decipherment step:

1. Perform fragment reassembly to recover the complete

Handshake.body (Section 5.5 of [[DTLS13](#)]).

2. Let tweak be "datagram hs" + Handshake.msg_type.
3. Replace Handshake.body with TSPRP-Decipher(client/server_key, tweak, Handshake.body).

[3.3](#). Protocol confusion defense

The procedure described in [Section 3.2](#) is sufficient to render the bitstream pseudorandom to a third party when both peers are operating correctly. However, if a malicious client or server can coerce its peer into sending particular plaintext (as is common in web browsers), it can choose plaintext with knowledge of the encryption keys, in order to produce ciphertext that has visible structure to a third party. This technique can be used to mount protocol confusion attacks [[SLIPSTREAM](#)].

This attack is particularly straightforward when using the AES-GCM or ChaCha20-Poly1305 cipher suites, as much of the ciphertext is encrypted by XOR with a stream cipher. A malicious peer in this threat model can choose desired ciphertext, XOR it with the keystream to produce the malicious plaintext, and rely on the other peer's encryption stage to reverse the encryption and reveal the desired ciphertext.

As a defense against this attack, the Pseudorandom cTLS extension supports two optional keys named "client-recipher" and "server-recipher". Each key's value is an integer E between 0 and 16 (inclusive) indicating how much entropy to add. When the "client-recipher" key is present, the client MUST prepend E fresh random bytes to CTLSCiphertext.encrypted_record before encipherment. The server MUST apply a similar transformation if the "server-recipher" key is present.

This transformation does not alter the Length field in the Unified Header, so it does not reduce the maximum plaintext record size. However, it does increase the output message size, which may impact MTU calculations in DTLS.

The sender MUST compute R using a cryptographically secure pseudorandom number generator (CSPRNG) whose seed contains at least 16 bytes of entropy that is not known to the peer.

In general, a malicious peer can still produce desired ciphertext with probability 2^{-8E} for each attempt by guessing a value of R . Accordingly, values of E less than 8 are NOT RECOMMENDED for defense against confusion attacks.

[4.](#) Plaintext Alerts

Representing plaintext alerts (i.e. `CTLSPlaintext` messages with `content_type = alert(21)`) requires additional steps, because Alert fragments have little entropy.

A standard TLS Alert fragment is always 2 bytes long. In Pseudorandom cTLS, senders MUST append at least 16 random bytes to any plaintext Alert fragment and increase `CTLSPlaintext.length` accordingly. Enciphering and deciphering then proceed identically to other `CTLSPlaintext` messages. The recipient MUST remove these bytes before passing the `CTLSPlaintext` to the cTLS implementation.

QUESTION: Are there client-issued Alerts in response to malformed `ServerHello`?

[5.](#) Operational Considerations

[5.1.](#) Multiple profiles and key rotation

Pseudorandom cTLS supports multiple profiles on the same server port. If all profiles share the same Pseudorandom cTLS configuration (and the same length of `connection_id` if applicable), the server simply deciphers the incoming data before reading the `profile_id` or `connection_id`.

If multiple Pseudorandom cTLS configurations are in use, the server can use trial deciphering to determine which profile applies to each new connection. A trial is confirmed as correct if the deciphered `ClientHello.profile_id` matches an expected value. To avoid false matches, server operators SHOULD choose a `profile_id` whose length is at least 8 bytes.

Pseudorandom cTLS key rotation can be represented as a transition from one profile to another. If the only difference between two profiles is the Pseudorandom cTLS configuration, the server MAY use the same `profile_id` for both profiles, relying on trial deciphering to identify which version is in use. Trial deciphering is also sufficient to determine whether the client is using Pseudorandom cTLS, so the "pseudorandom" key MAY appear in the template's "optional" section.

Pseudorandom cTLS does not support demultiplexing distinct configurations by `connection_id`. Such use would require both the client and server to perform trial deciphering on every datagram. Instead, clients that implement Pseudorandom cTLS **MUST** use a distinct transport session (e.g. UDP 5-tuple) for each cTLS profile.

[5.2.](#) Computational cost

Pseudorandom cTLS adds a constant, symmetric computational cost to sending and receiving every record, roughly similar to the cost of encrypting a very small record. The cryptographic cost of delivering small records will therefore be increased by a constant factor, and the computational cost of delivering large records will be almost unchanged.

The optional defense against ciphertext confusion attacks further increases the overall computational cost, generally at least doubling the cost of delivering large records. It also adds up to 16 bytes of overhead to each encrypted record.

[6.](#) Security Considerations

Pseudorandom cTLS operates as a layer between cTLS and its transport, so the security properties of cTLS are largely preserved. However, there are some small differences.

In datagram mode, the `profile_id` and `connection_id` fields allow a server to reject almost all packets from a sender who does not know the template (e.g. a DDoS attacker), with minimal CPU cost. Pseudorandom cTLS requires the server to apply a decryption operation to every incoming datagram before establishing whether it might be valid. This operation is $O(1)$ and uses only symmetric cryptography, so the impact is expected to be bearable in most deployments.

cTLS templates are presumed to be published by the server operator. In order to defend against ciphertext confusion attacks ([Section 3.3](#)), the client **MUST** refuse to connect unless the server provides a cTLS template with a sufficiently large "client-recipher" value.

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. The basic idea is that the "real" protocol (Pseudorandom cTLS) should be indistinguishable from some "target" protocol that the network is known tolerate. The assumption is that middleboxes would not attempt to parse packets whose contents are pseudorandom. (The same idea underlies QUIC's wire encoding format [[RFC9000](#)].) A

starting point might be the formal notion of "Observational Equivalence" (<https://infsec.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/publications/pub2015/ASP0bsEq.pdf>).

[7.](#) Privacy Considerations

Pseudorandom cTLS is intended to improve privacy in scenarios where the adversary can observe traffic to various servers but lacks access to their cTLS templates, by preventing the adversary from determining which profiles are in use by which clients and servers. If instead the adversary does have access to some cTLS templates, and these templates do not have distinctive profile_id values, Pseudorandom cTLS can reduce privacy, by enabling strong confirmation that a connection is using a specific profile.

When Pseudorandom cTLS is enabled, the adversary can still observe the length and timing of messages, so templates that differ in these can still be distinguished. Implementations MAY use TLS padding to reduce the observable patterns.

The adversary could also send random data to the server (a "probing attack") in order to learn the fraction of messages of each length that produce valid ClientHellos. This "probability fingerprint" could allow discrimination between profiles. Server operators that wish to defend against probing attacks SHOULD choose a sufficiently long profile_id that invalid ClientHellos are always rejected without eliciting a response. A 15-byte profile_id provides 128-bit security.

[8.](#) IANA Considerations

[8.1.](#) TSPRP Registry

This specification anticipates the existence of an IANA registry of Tweakable Strong Pseudorandom Permutations (TSPRPs). Until such a registry exists, the value "exp-hctr2" is reserved to indicate the HCTR2 TSPRP [[HCTR2](#)].

[8.2.](#) cTLS Template Key registry

This document requests that IANA add the following value to the "cTLS Template Keys" registry:

Schwartz & Patton Expires 13 October 2022 [Page 11]

Internet-Draft Pseudorandom cTLS April 2022

+	+	+	+
Key	JSON Type	Reference	
+	+	+	+
pseudorandom	object	(This document)	
+	+	+	+

Table 1

[8.3.](#) TLS ContentType Registry

IANA is requested to add the following codepoint to the TLS Content Types Registry

This document requests that a code point be allocated from the "TLS ContentType" registry. The row to be added in the registry has the following form:

+	+	+	+
Value	Description	DTLS-OK	Reference
+	+	+	+
TBD	ctls_handshake_end	Y	RFCXXXX
+	+	+	+

Table 2

[9.](#) References

[9.1.](#) Normative References

- [cTLS] Rescorla, E., Barnes, R., and H. Tschofenig, "Compact TLS 1.3", Work in Progress, Internet-Draft, [draft-ietf-tls-ctls-05](#), 7 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-05>>.
- [DTLS13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, [draft-ietf-tls-dtls13-43](#), 30 April 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", [RFC 9000](#), DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/rfc/rfc9000>>.

[9.2.](#) Informative References

- [HCTR2] "Length-preserving encryption with HCTR2", n.d., <<https://eprint.iacr.org/2021/1441/20211027:085150>>.
- [SLIPSTREAM] "NAT Slipstreaming v2.0", n.d., <<https://samy.pl/slipstream/>>.

Acknowledgments

TODO

Authors' Addresses

Benjamin Schwartz
Google LLC
Email: bemasc@google.com

Christopher Patton
Cloudflare, Inc.
Email: cpatton@cloudflare.com