Inferring OpenVPN State Machines Using Protocol State Fuzzing

Intern Lesly-Ann Daniel
Supervised by Erik Poll - Joeri de Ruiter

Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen, The Netherlands

From May 15th, 2017 to August 13th, 2017
Presentation: September 5th, 2017

Context

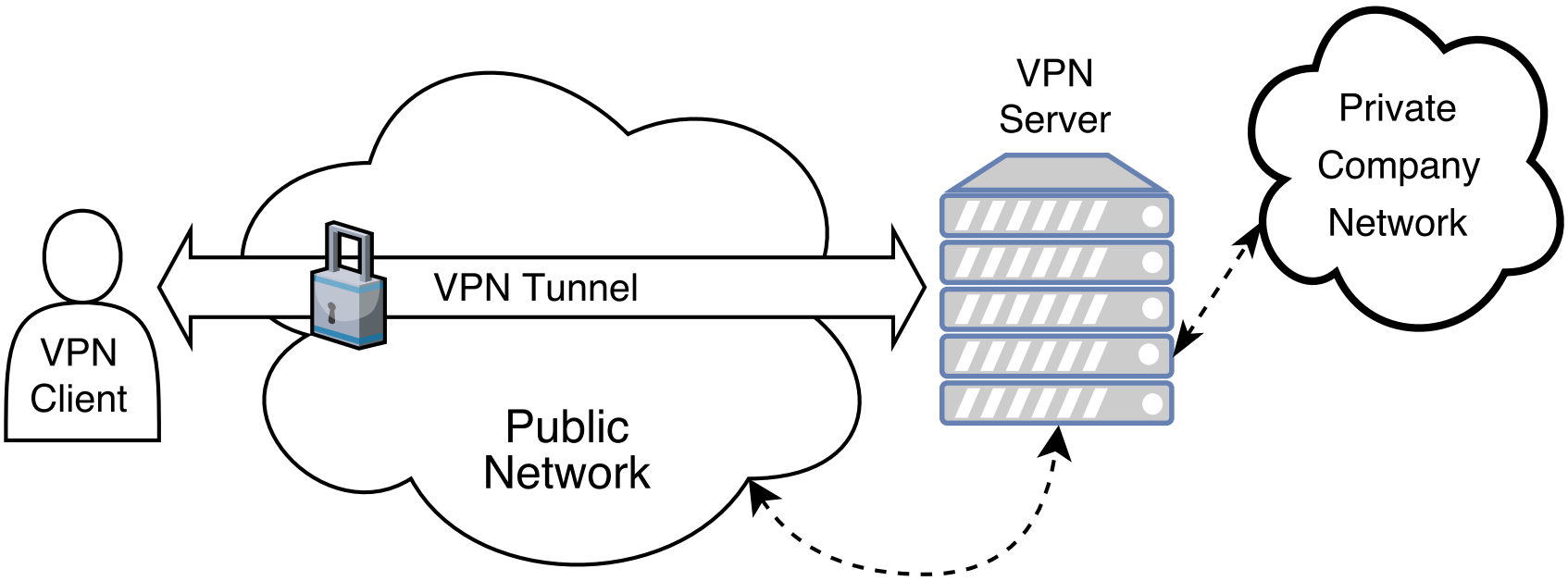

Figure: A VPN connexion between a client and a server.

Context

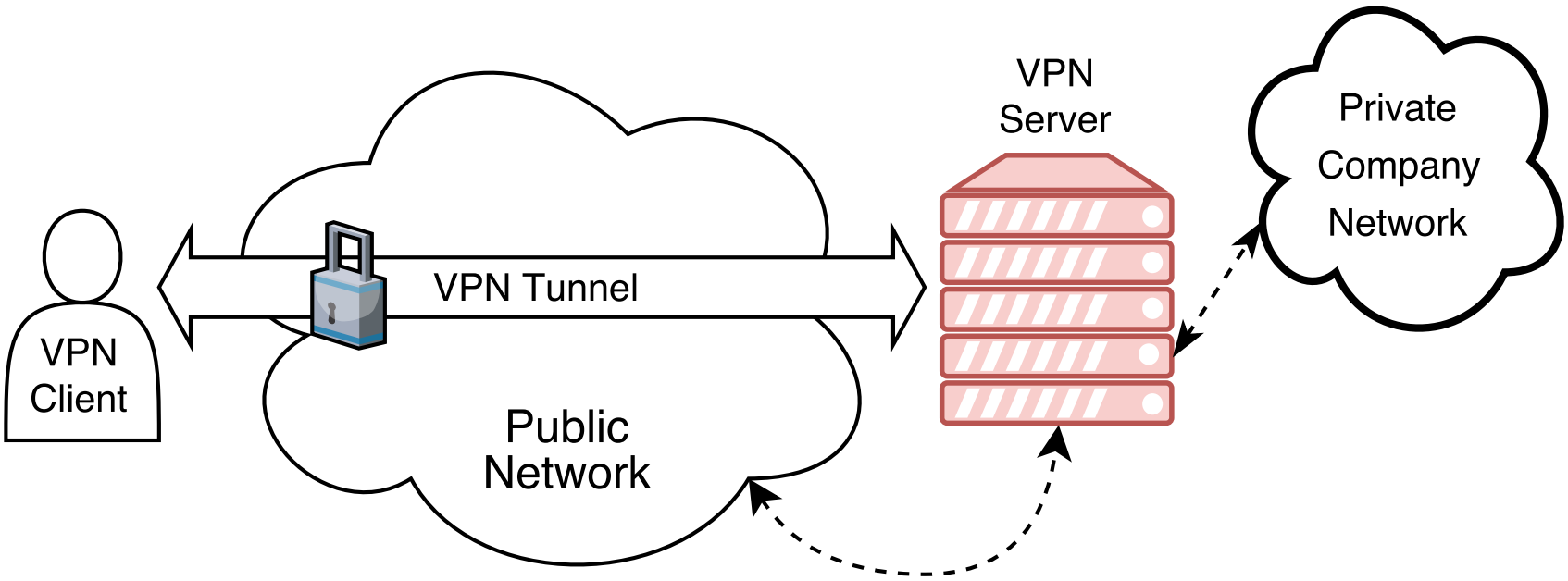

Figure: A VPN connexion between a client and a server.

Motivation

Figure: An OpenVPN server.

OpenVPN widely used VPN but:

- No formal specification and not subject to a lot of research.
- No documentation about the sequence of messages.
- No doc about the conduct to adopt in case of unexpected or erroneous message.

Goal

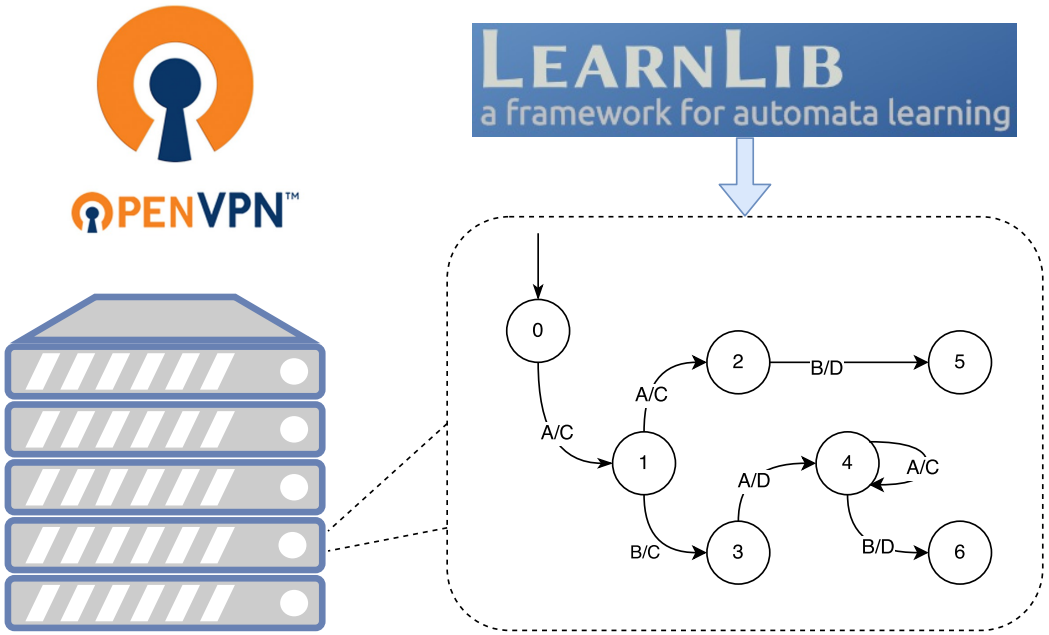

Figure: Inferring a state machine of an OpenVPN server with LearnLib.

- \+ Detect logical flaws
- \+ Detect superfluous states
- \+ Get information about the implementation
- \+ Infer a formal specification

Outline

Outline

Security

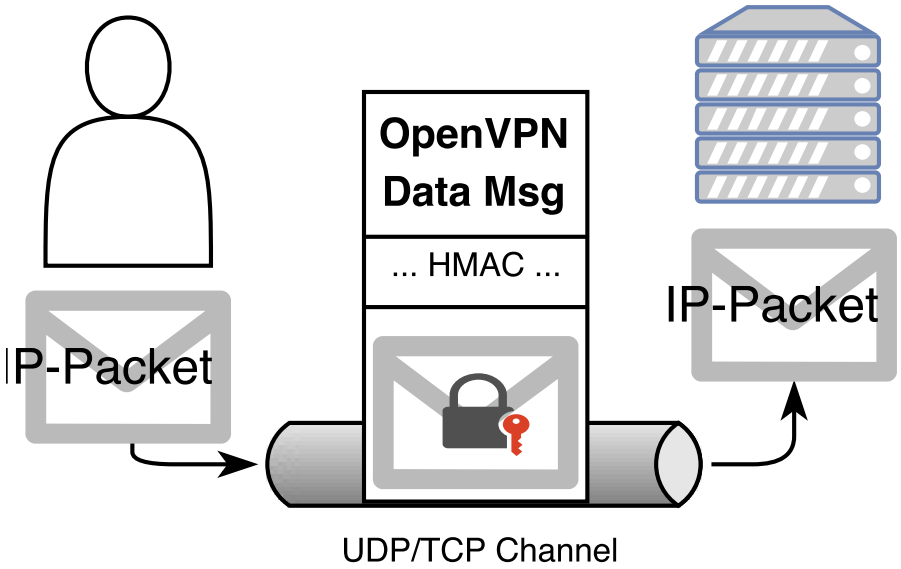

Figure: VPN tunneling of an IP packet.

Confidentiality

End-to-end encryption

Authentication

Certificates for both peers

Integrity

Hash-based Message Authentication Code (HMAC)

OpenVPN initialization sequence

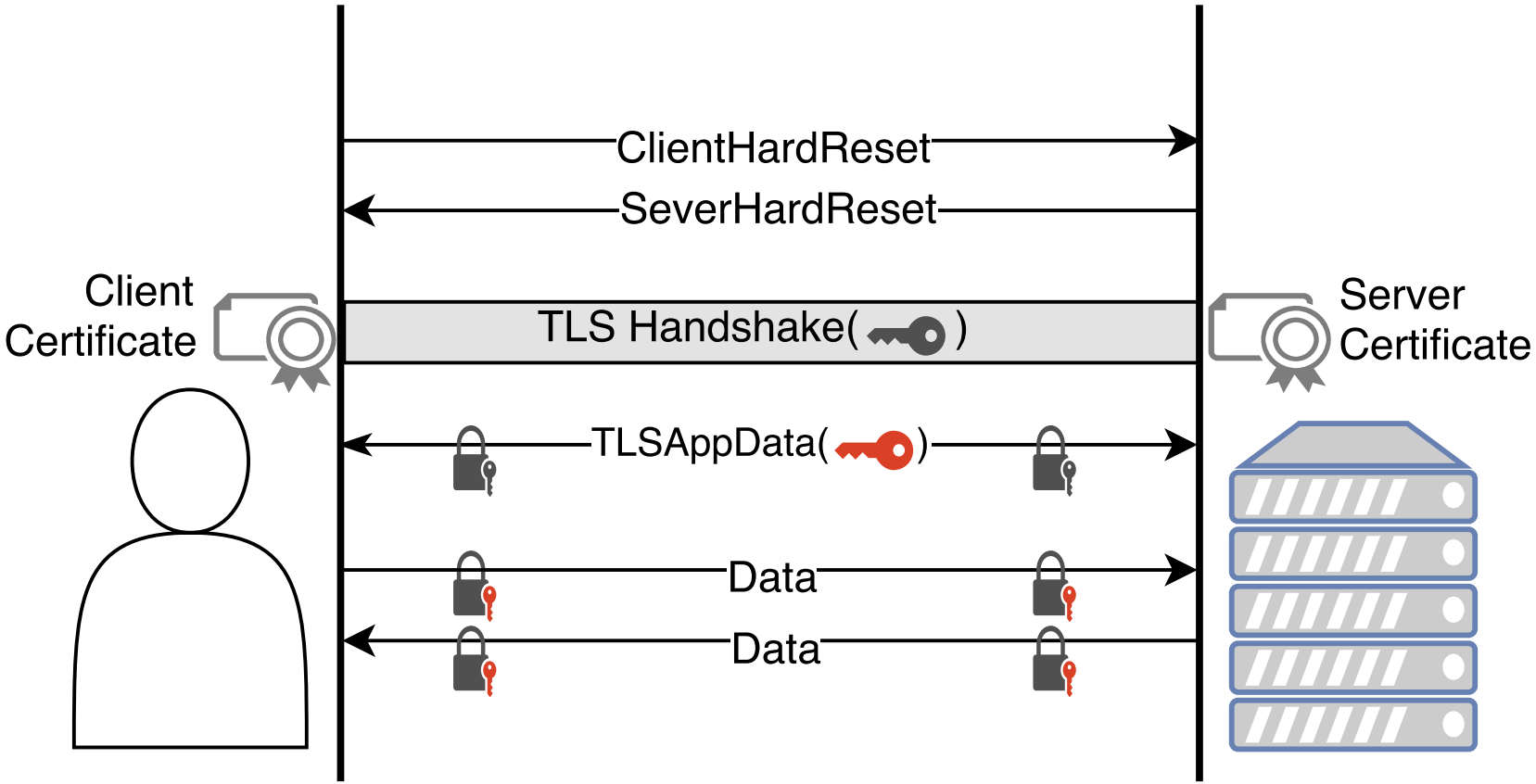

Figure: Sequence of messages initiating a OpenVPN connexion between a client and a server.

Outline

Protocol State Fuzzing

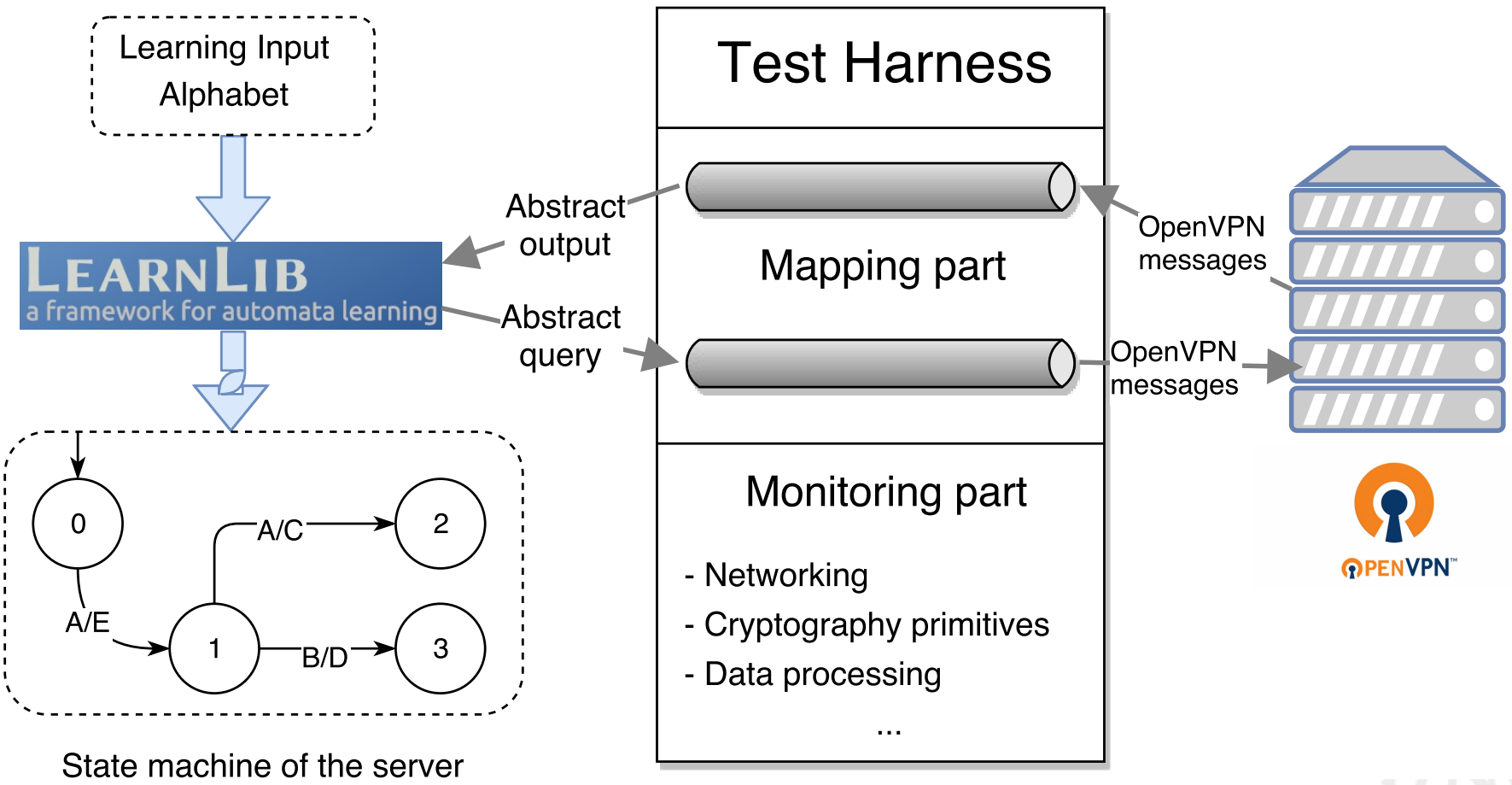

Figure: Inferring a state machine of an OpenVPN server.

Outline

Nondeterminism Issues

Querry

CHR TlsInit DataPingReq

Expected answer

SHR Success DataPingRep

Nondeterministic answers

SHR Fail ϵ

ϵ SHR Success

Causes of nondeterminism

- Network related
  - Packet lost or delayed.
- Time related
  - TCP/UDP timeouts.
  - Time-dependent events (e.g. TLS window).

Experimental Setup

- 1 Test-Harness
- 3 Input alphabets (focus on different phases of the protocol)
- Analysis of OpenVPN and OpenVPN-NL
- Analysis of UDP and TCP versions

Outline

The OpenVPN Session Initialization

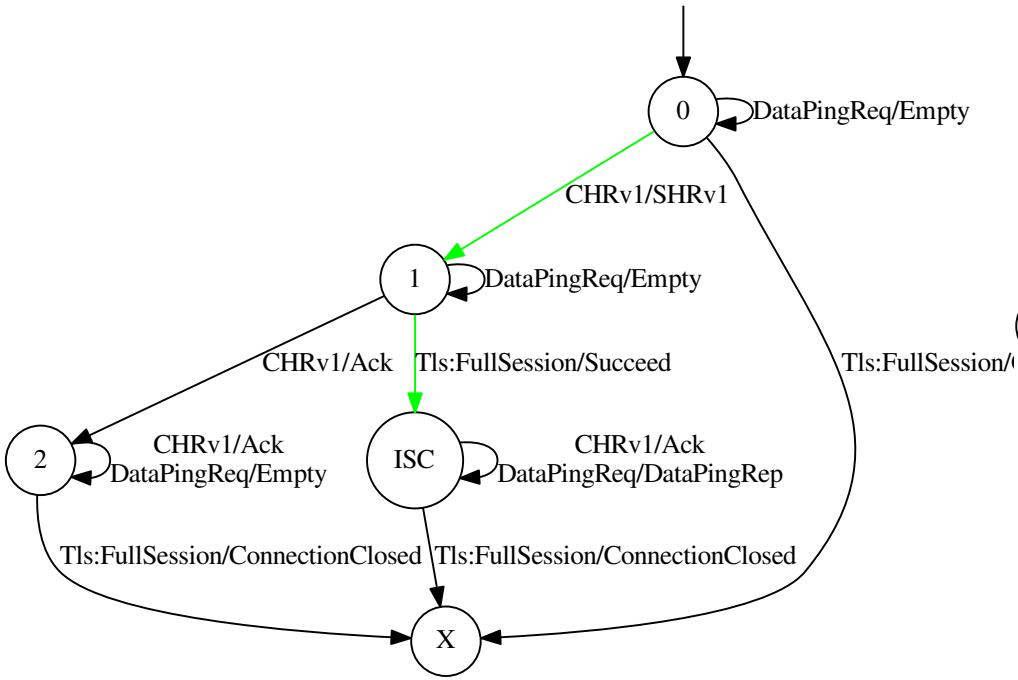

Figure: OpenVPN server running in TCP mode.

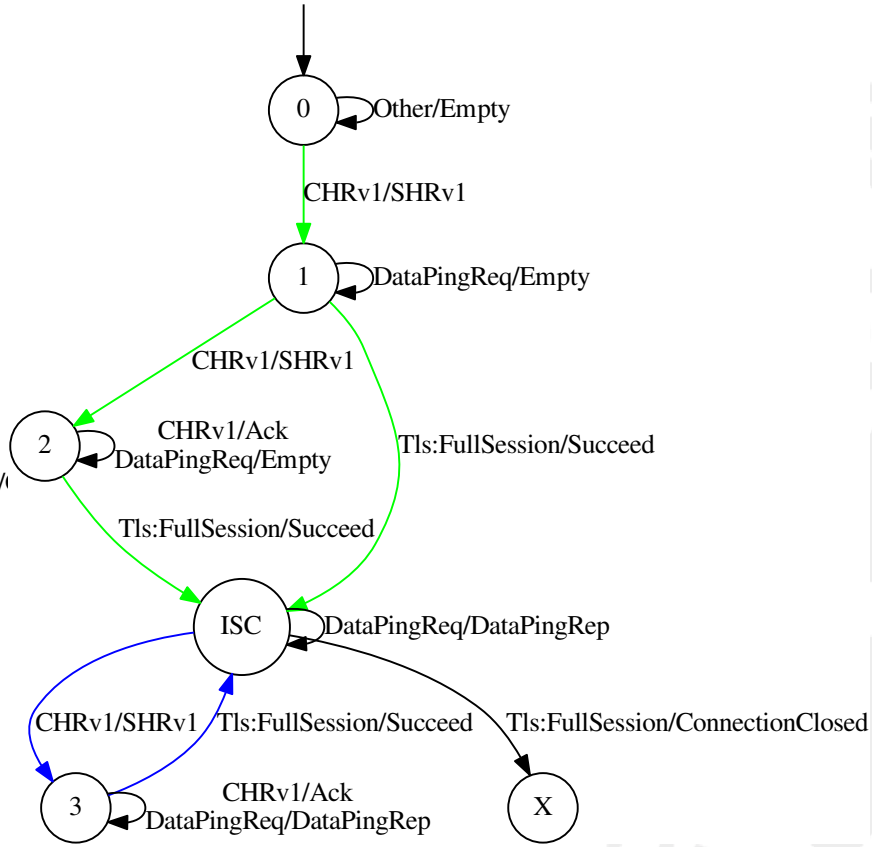

Figure: OpenVPN server running in UDP mode.

The TLS Handshake

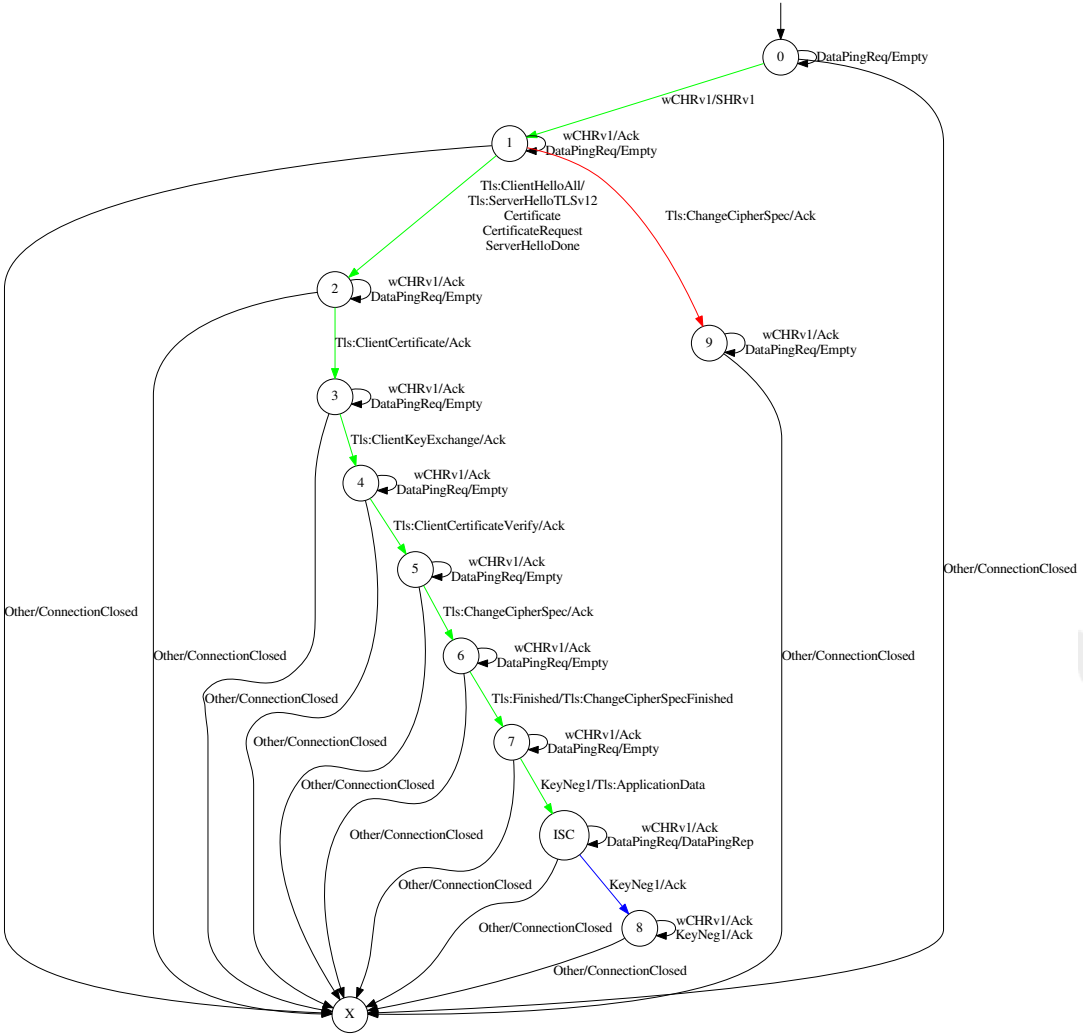

Figure: OpenVPN server running in TCP mode.

The TLS Handshake

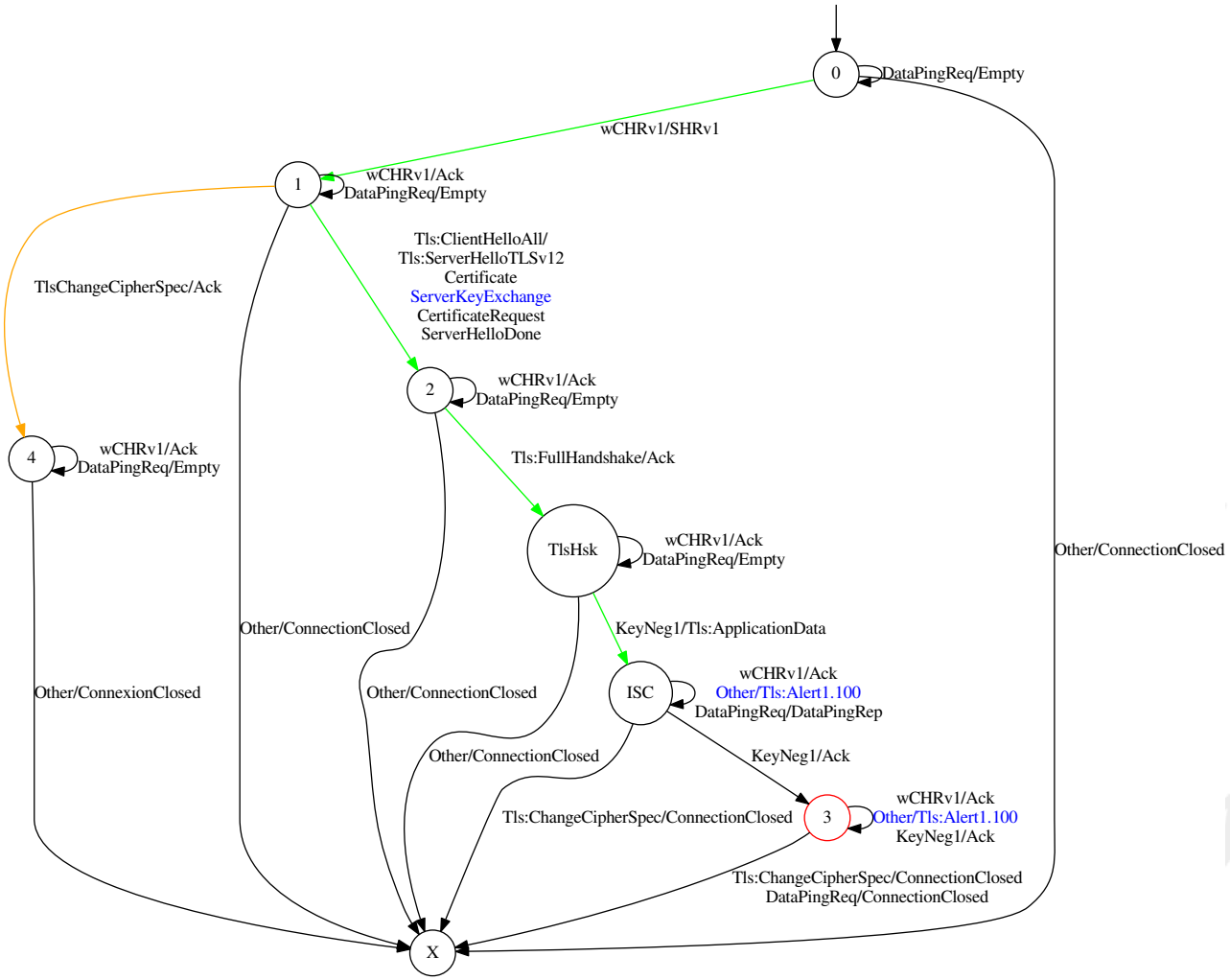

Figure: Servers running in TCP mode: the blue parts are specific to OpenVPN-NL and the orange part is specific to OpenVPN.

The Key Renegotiation Mechanism

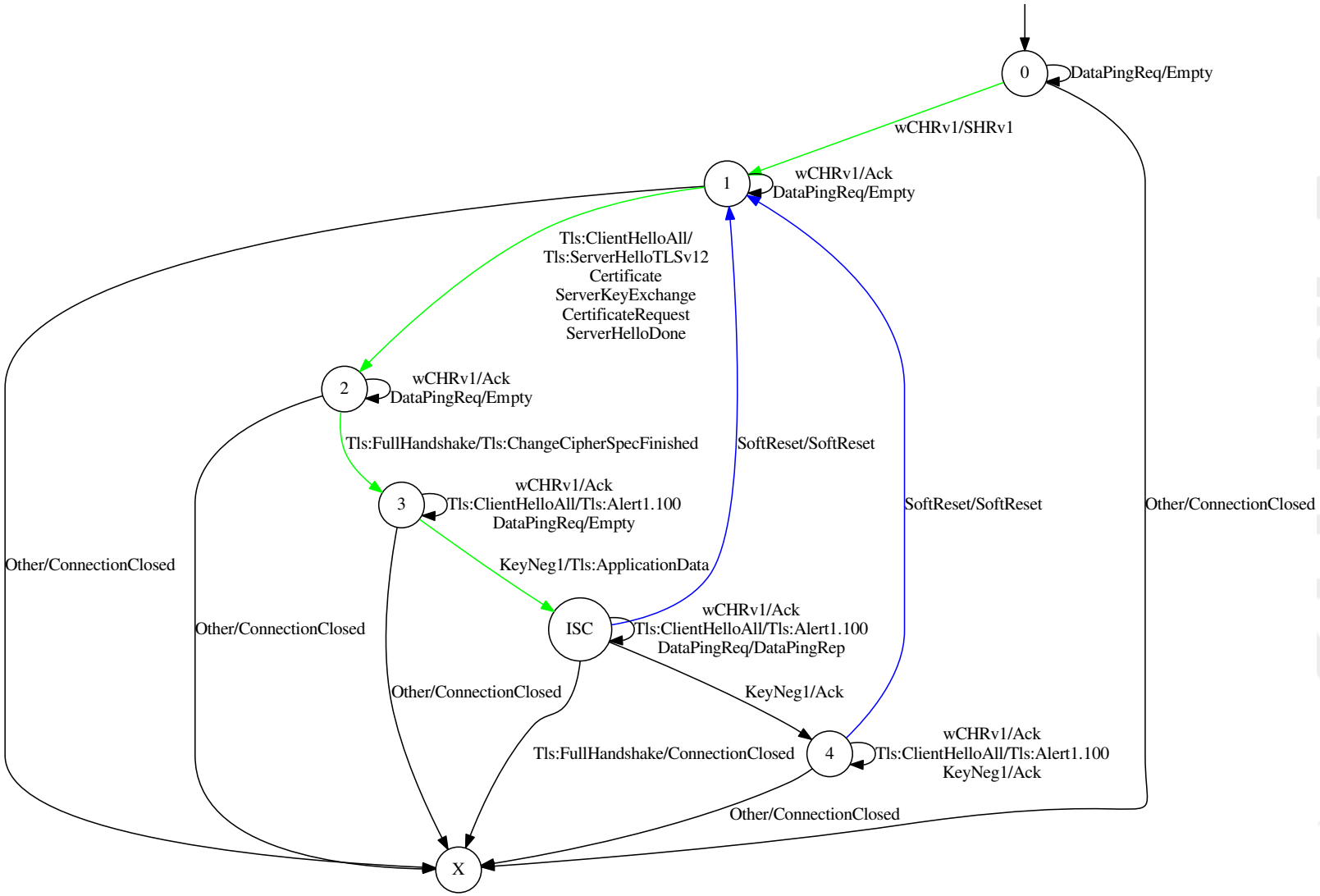

Figure: OpenVPN-NL server running in TCP mode.

Outline

Conclusion

- − Coarse analysis, can only detect logical flaws.
- − Timing-related events (learning time from 40 min to 50 hours).
- \+ No vulnerability found but some unexpected differences between UDP and TCP.
- \+ The state machines give a good insight into the implementation and can be used to infer a formal specification of the protocol.

Regular Inference

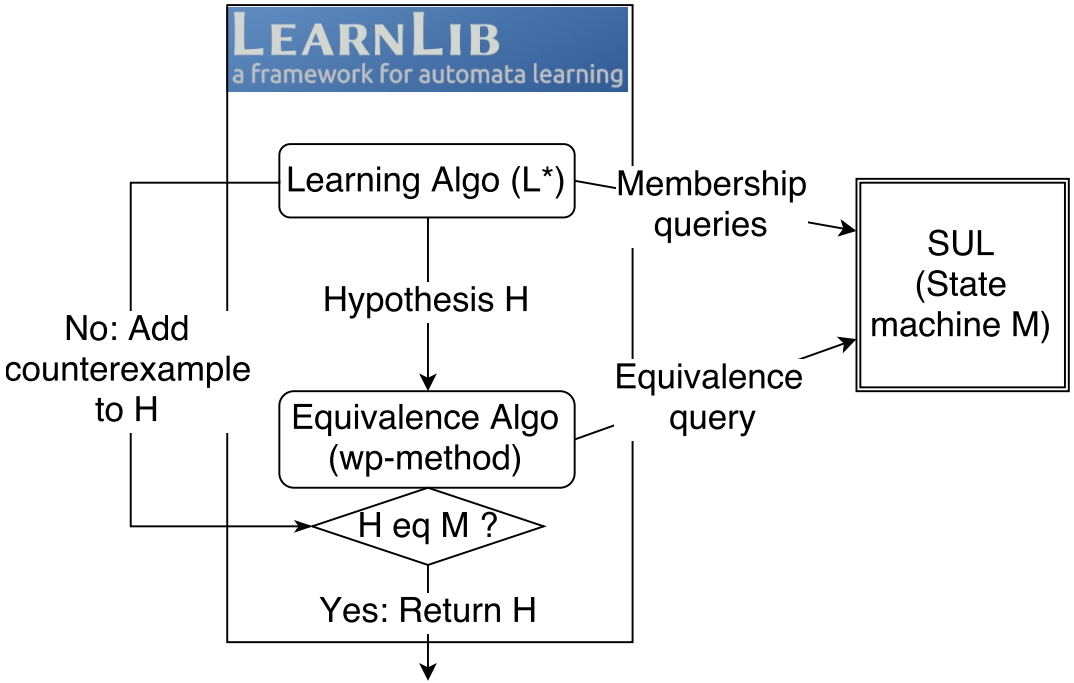

Figure: Regular inference process.

Membership querry

What is the response to a sequence of input symbols ?

Equivalence querry

Is an hypothesized automaton H equivalent to M ?

Protocol State Fuzzing of TLS Implementations

Largely automated analysis of TLS client- and server- side implementations (9 impl.) by Joeri de Ruiter.
Learning time: from 9 min to over 8 hours.
Vulnerabilities found:

- GnuTLS: HeartbeatRequest resets the handshake buffer. This can be exploited to bypass the handshake integrity-check.
- Java Secure Socket Extension: Finish message can be sent without ChangeCipherSpec, allowing the client to send unencrypted data (or use the old keys).
- OpenSSL: Sending an additional ChangeCipherSpec after successfully completing a handshake changes the client keys to the server keys.