Black-box security analysis of state machine implementations

Joeri de Ruiter

18-03-2019

SIDN

Agenda

1. Why are state machines interesting?
2. How do we know that the state machine is implemented correctly?
3. What can go wrong if the implementation is incorrect?

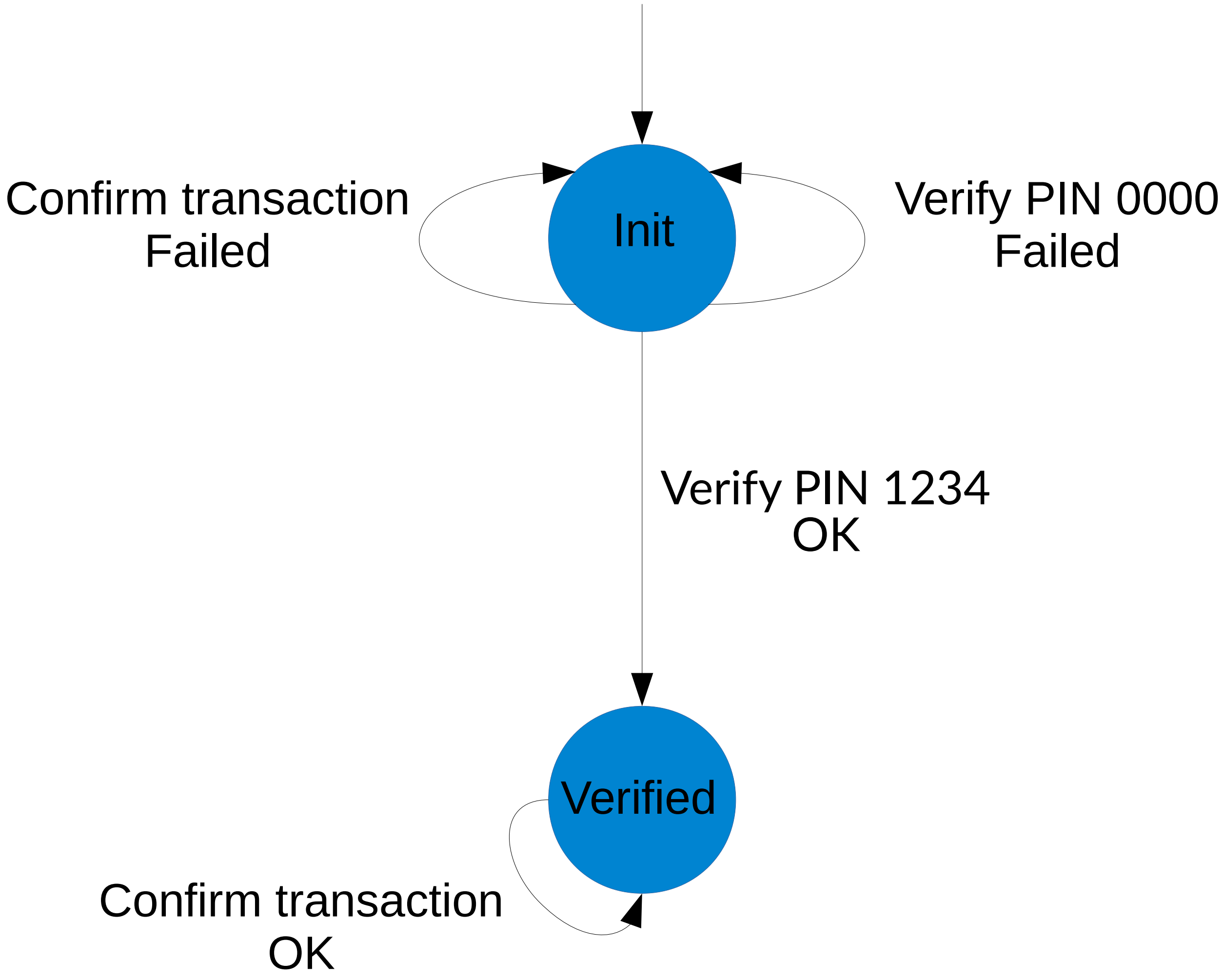
What are state machines?

- Almost every protocol includes some kind of state
- State machine is a model of the different states and the transitions between them
- When receiving a messages, given the current state:
  - Decide what action to perform
  - Which message to respond with
  - Which state to go the next

Why are state machines interesting?

- State machines play a very important role in security protocols
- For example:
  - Is the user authenticated?
  - Did we agree on keys? And if so, which keys?
  - Are we encrypting our traffic?
- Every implementation of a protocol has to include the corresponding state machine
- Mistakes can lead to serious security issues!

State machine example



State machines in specifications

- Often specifications do not explicitly contain a state machine
  - Mainly explained in lots of prose
- Focus usually on happy flow
  - What to do if protocol flow deviates from this?

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

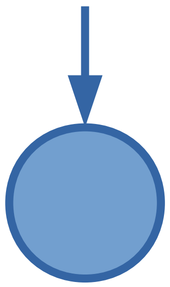
Implementation of state machines

- How do we know that a state machine is implemented correctly?
  - Often state is implicitly included
- Test whether it works against other implementations?
  - Typically only tests the happy flow
- What about invalid sequences that might lead to security vulnerabilities?
- Can we somehow extract the state machine from an implementation?
  - State machine inference

State machine inference

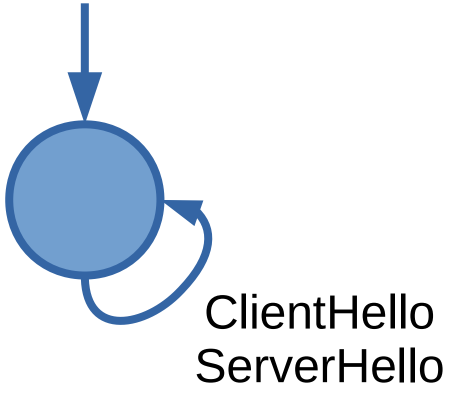
- Black-box technique to extract state machines from implementations
  - Only communication with the system
  - All we need to know is how to construct messages
- Fuzzing of message order
- Useful for security analysis
  - Discover vulnerabilities and bugs
  - Provides interesting insights in the code
  - Will not find carefully hidden backdoors
- Analysis can be done either manually of automated

State machine inference – example



State machine inference – example

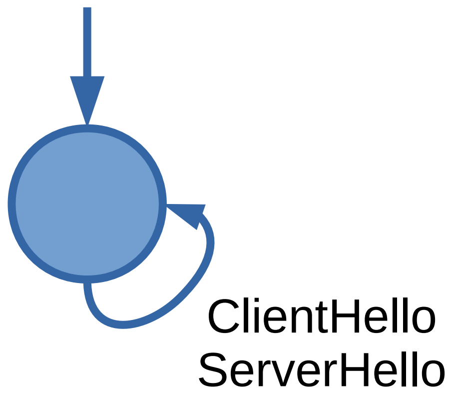
→ ClientHello
← ServerHello



State machine inference – example

→ ClientHello
← ServerHello

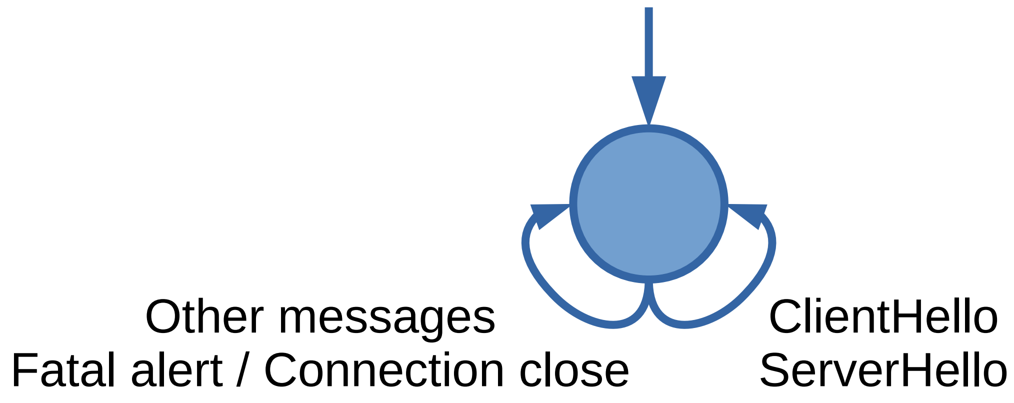
→ Other messages
← Fatal alert / Connection close



State machine inference – example

→ ClientHello
← ServerHello

→ Other messages
← Fatal alert / Connection close

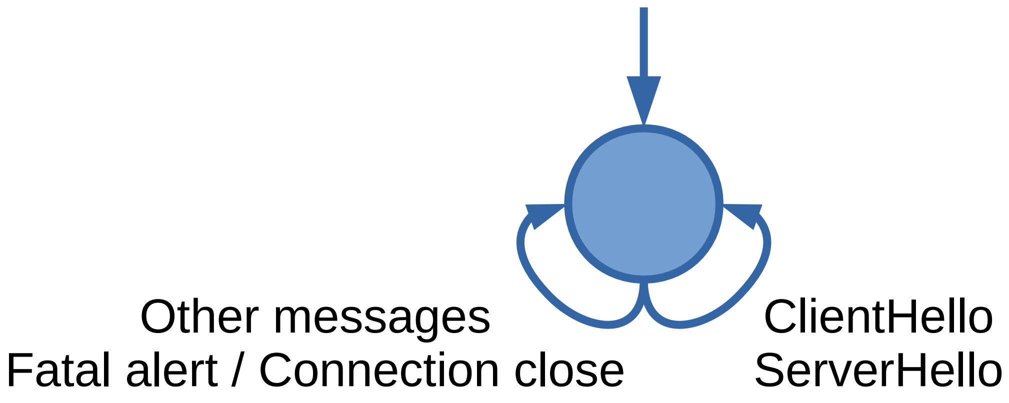


State machine inference – example

→ ClientHello
← ServerHello

→ Other messages
← Fatal alert / Connection close

→ ClientHello, ClientHello
← Fatal alert / Connection close

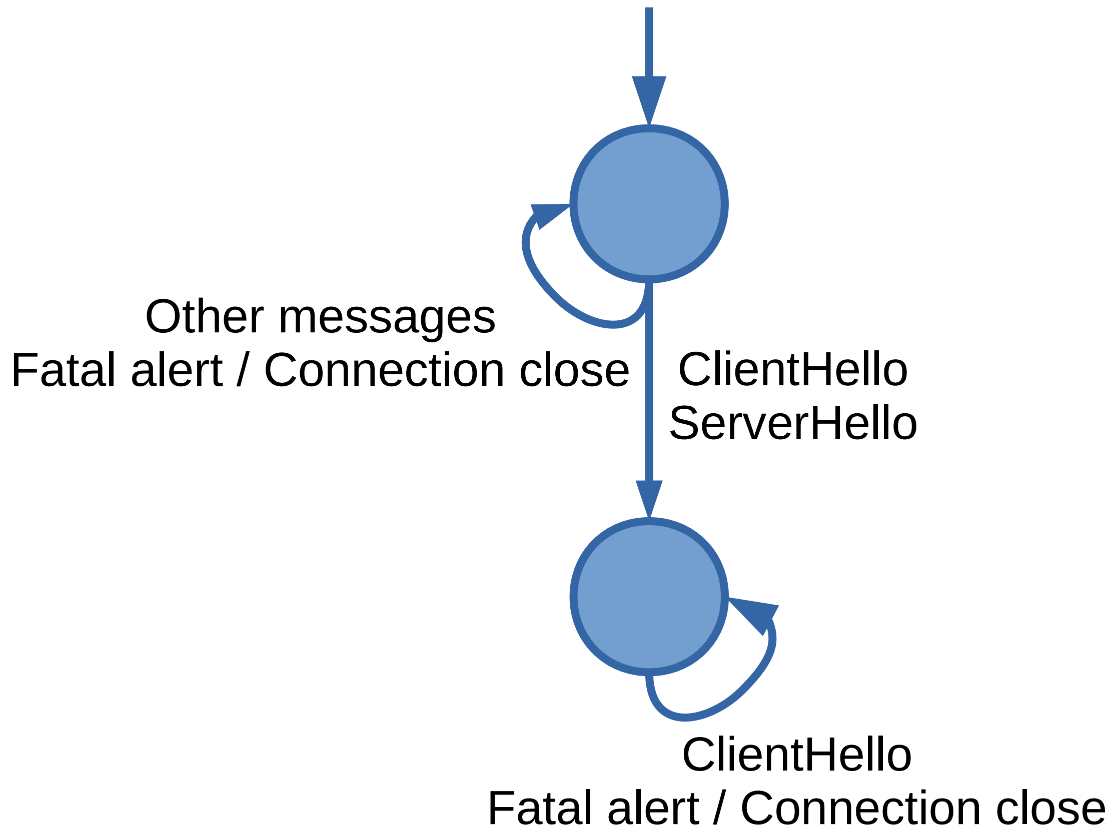


State machine inference – example

→ ClientHello
← ServerHello

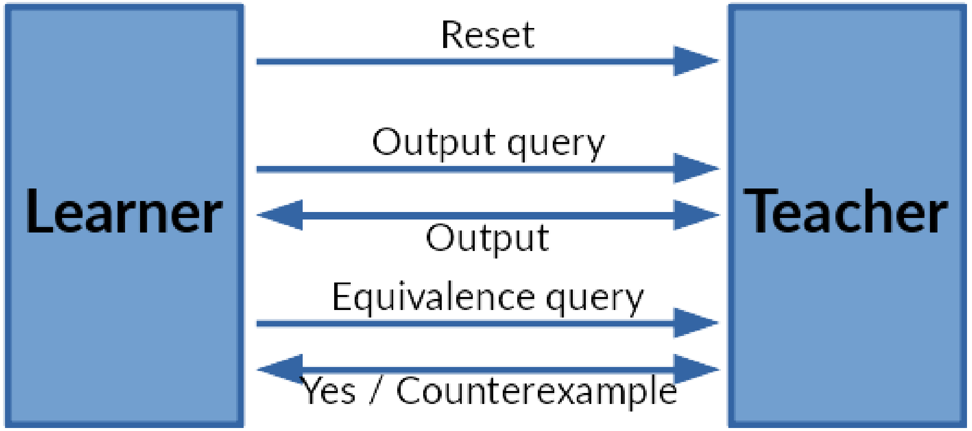
→ Other messages
← Fatal alert / Connection close

→ ClientHello, ClientHello
← Fatal alert / Connection close



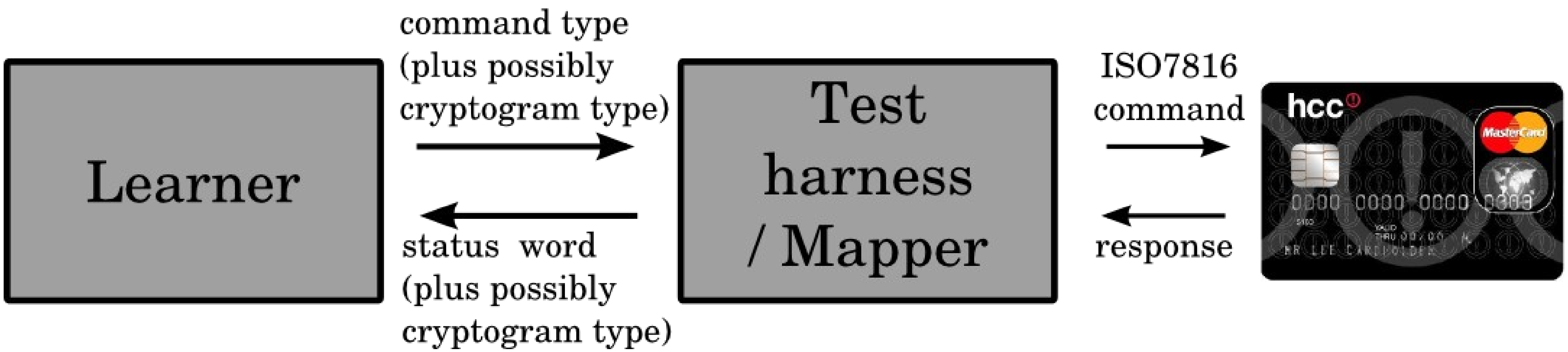
State machine inference - theory

- Deterministic Mealy machine
- Learner
  - Tries to learn the state machine of an implementation
  - Constructs a hypothesis of the state machine
- Teacher
  - Knows the state machine of the implementation
  - Answers questions about the implementation
  - Determines whether provided hypothesis is correct

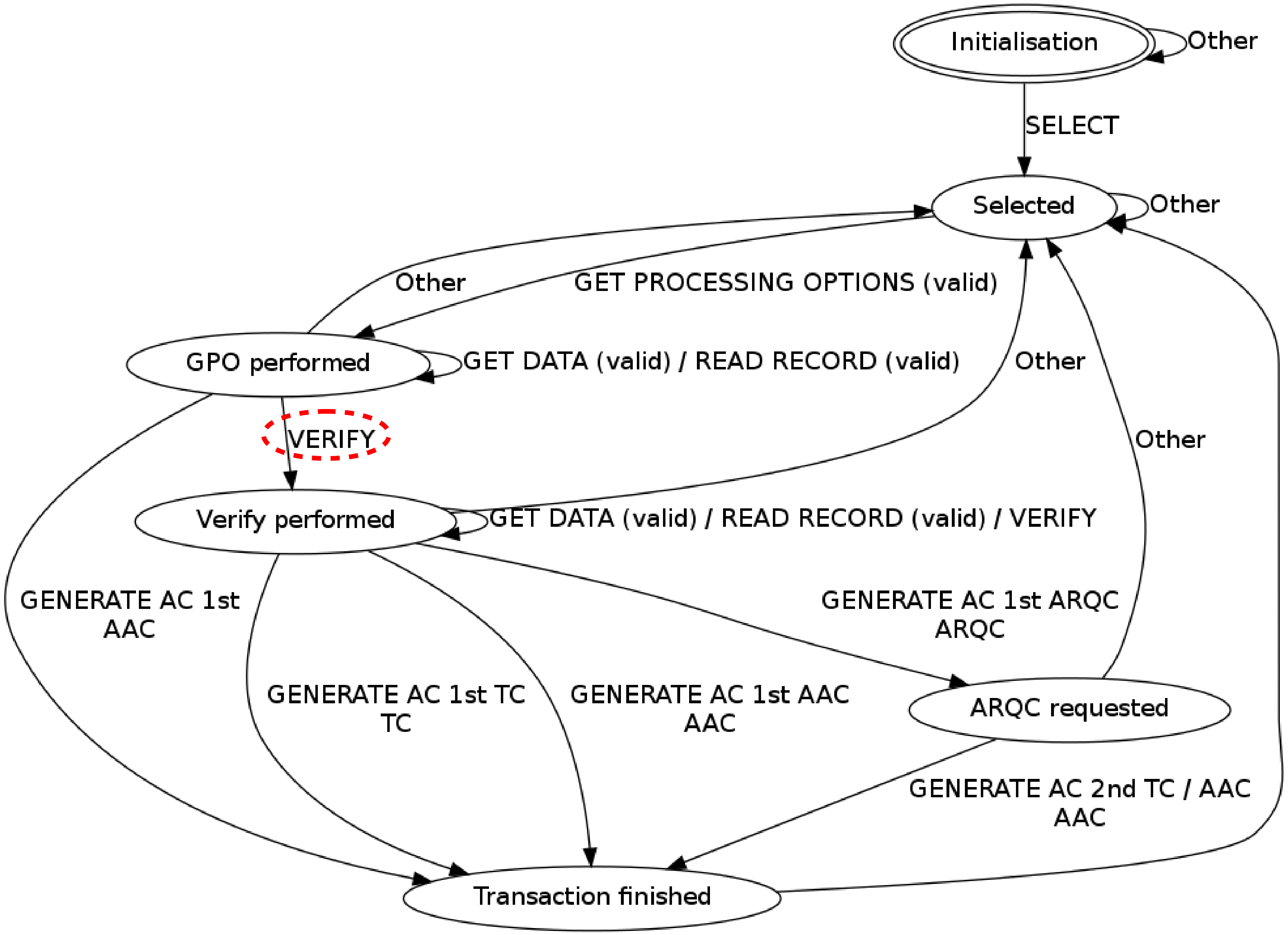


State machine inference - practice

- Convert abstract input symbols used in the algorithms to bytes
- Convert responses back to abstract symbols
- Need some way to reset the system
- Equivalence checking needs to be approximated
- Basically, you need a stateless implementation of the protocol you want to analyse



State machine inference - example



Analysed systems

- Bank cards (EMV)
- ABN-AMRO's e.dentifier2
- TLS
  - Collection of well-known implementations
  - OpenSSL versions from a 14 year period
- Wi-Fi (4-way handshake)
- OpenVPN, IPsec, TLS1.3, DTLS, and more

StateLearner

- Tool to infer state machines from implementations
- Uses LearnLib developed at TU Dortmund
  - Implementation of several learning and equivalence algorithms
- Built-in support for TLS and smart cards
- Can easily be extended to analyse other protocols

ABN-AMRO’s e.dentifier2

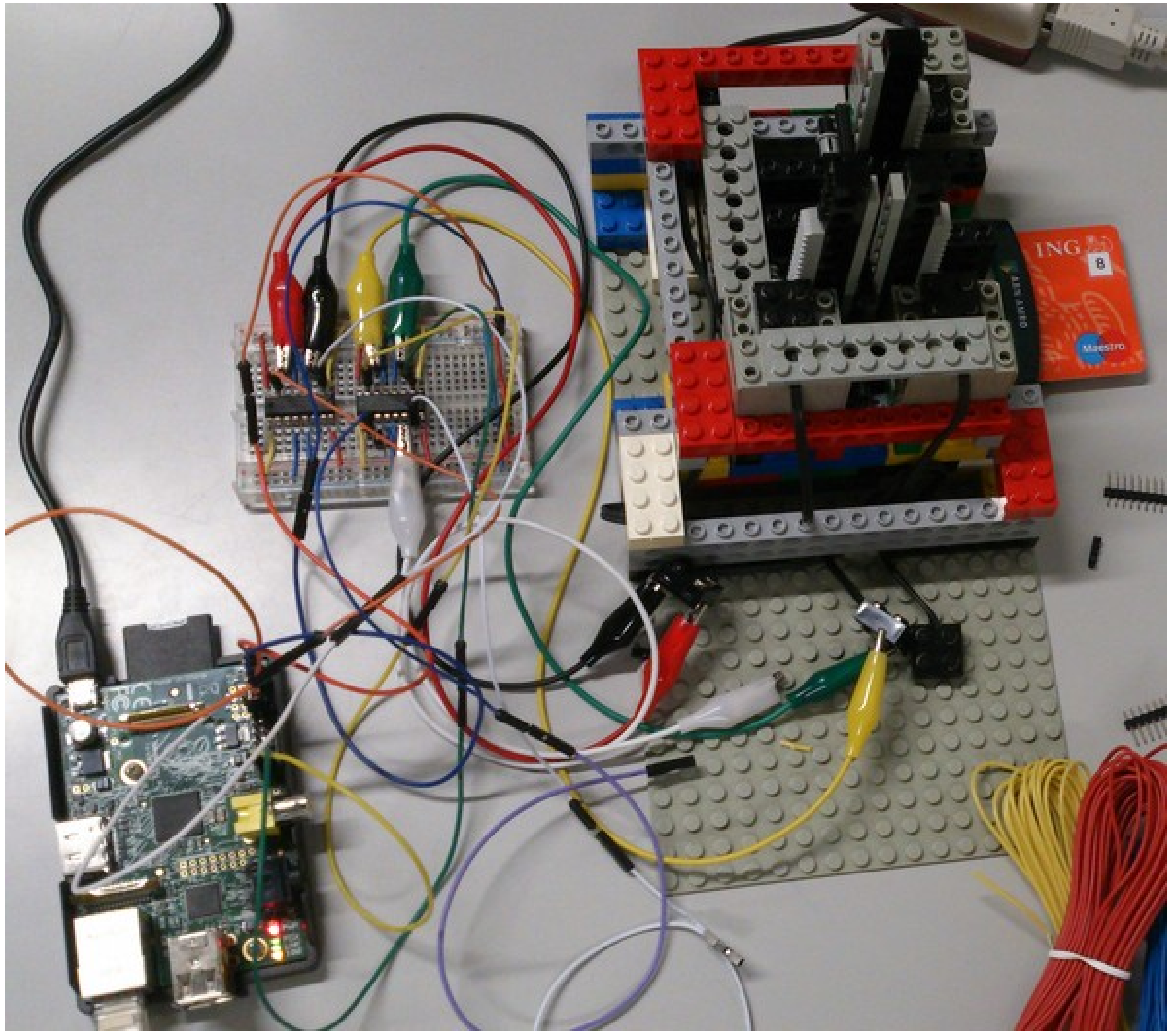
- Handheld reader used for online banking
- Provides what-you-see-is-what-you-sign functionality
- In theory a good idea…
- However, in previous manual analysis we found a serious flaw
- Can we automatically find this type of flaws?

SIDN LABS

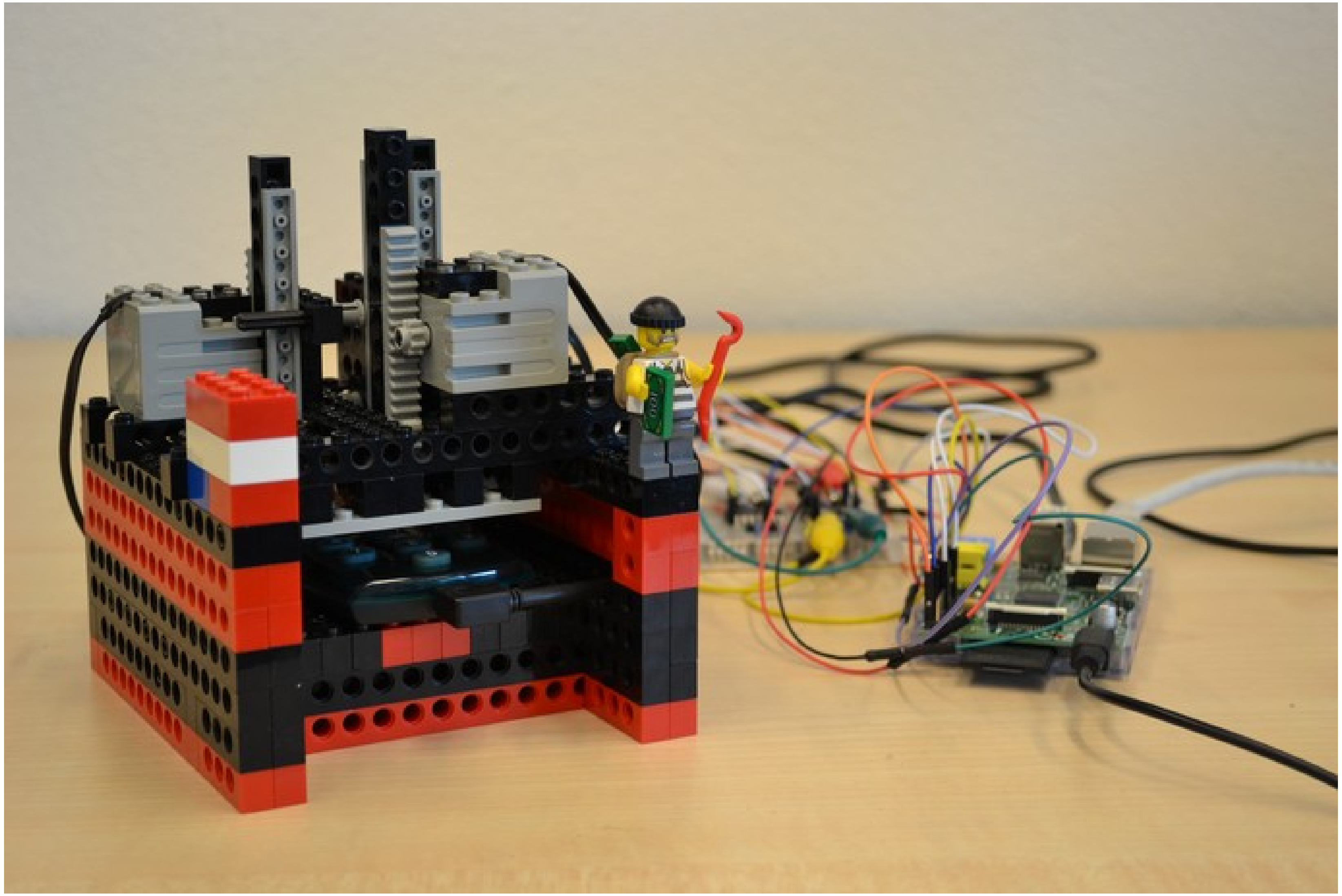
Analysing the e.dentifier2

- Problem: the protocol involves pressing buttons
- Solution: LEGO!
- Push buttons on e.dentifier2 using a Lego robot
- Controlled by Raspberry Pi
  - 3 motors: OK, Cancel, digit
  - Power USB line
- Programmed own bank card
- https://youtu.be/hyQubPvAyq4

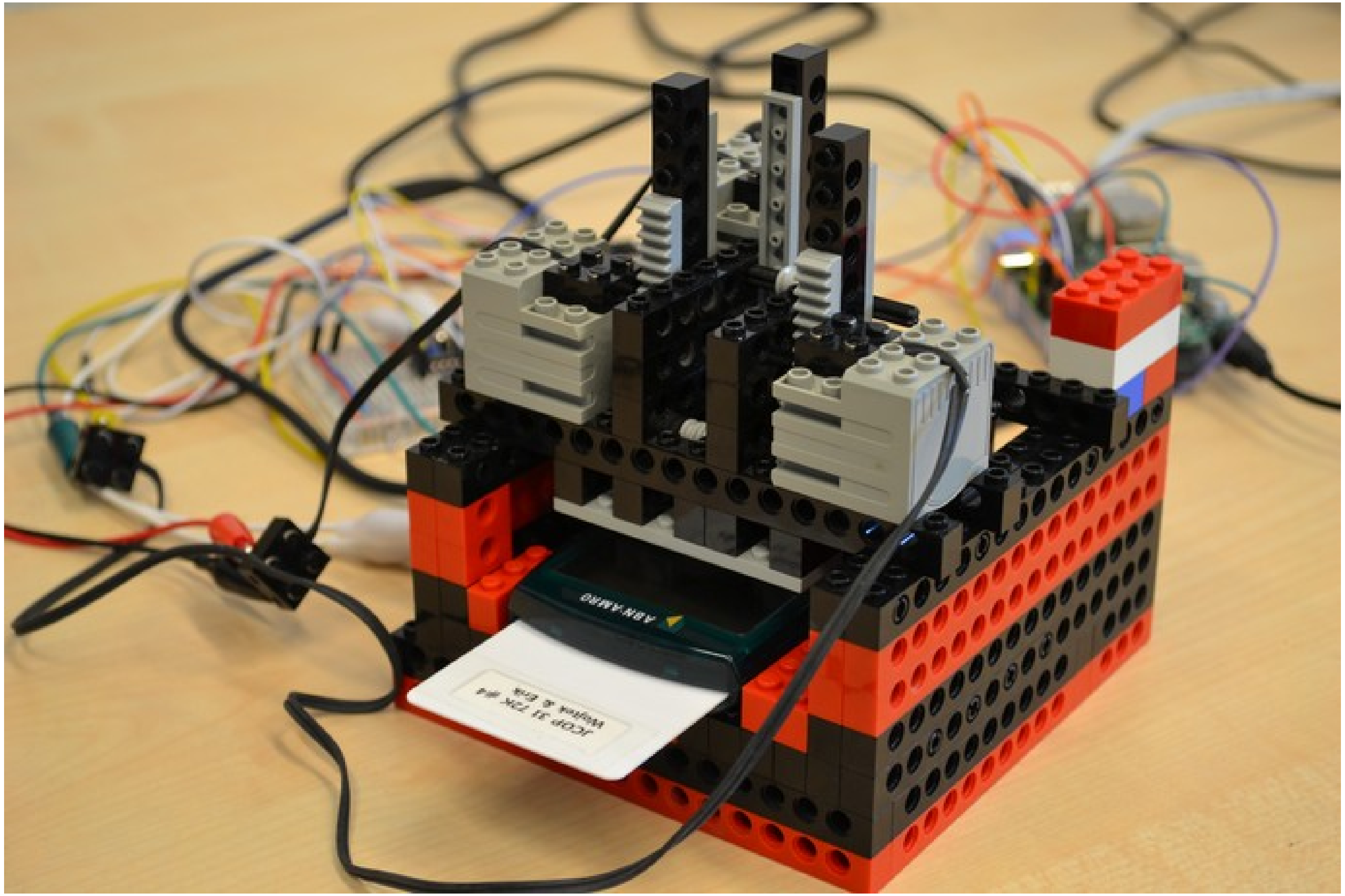
Lego robot



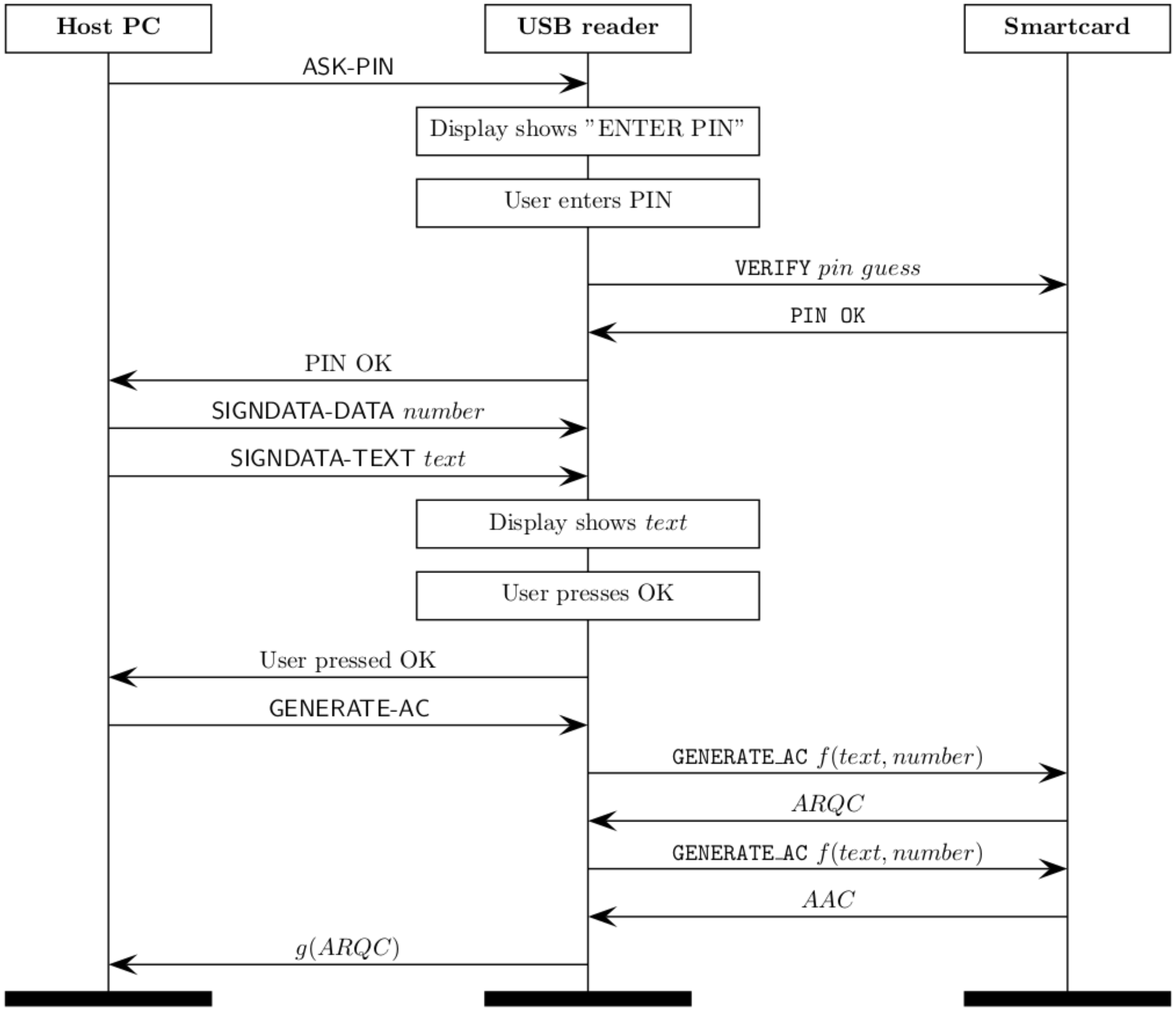
Lego robot



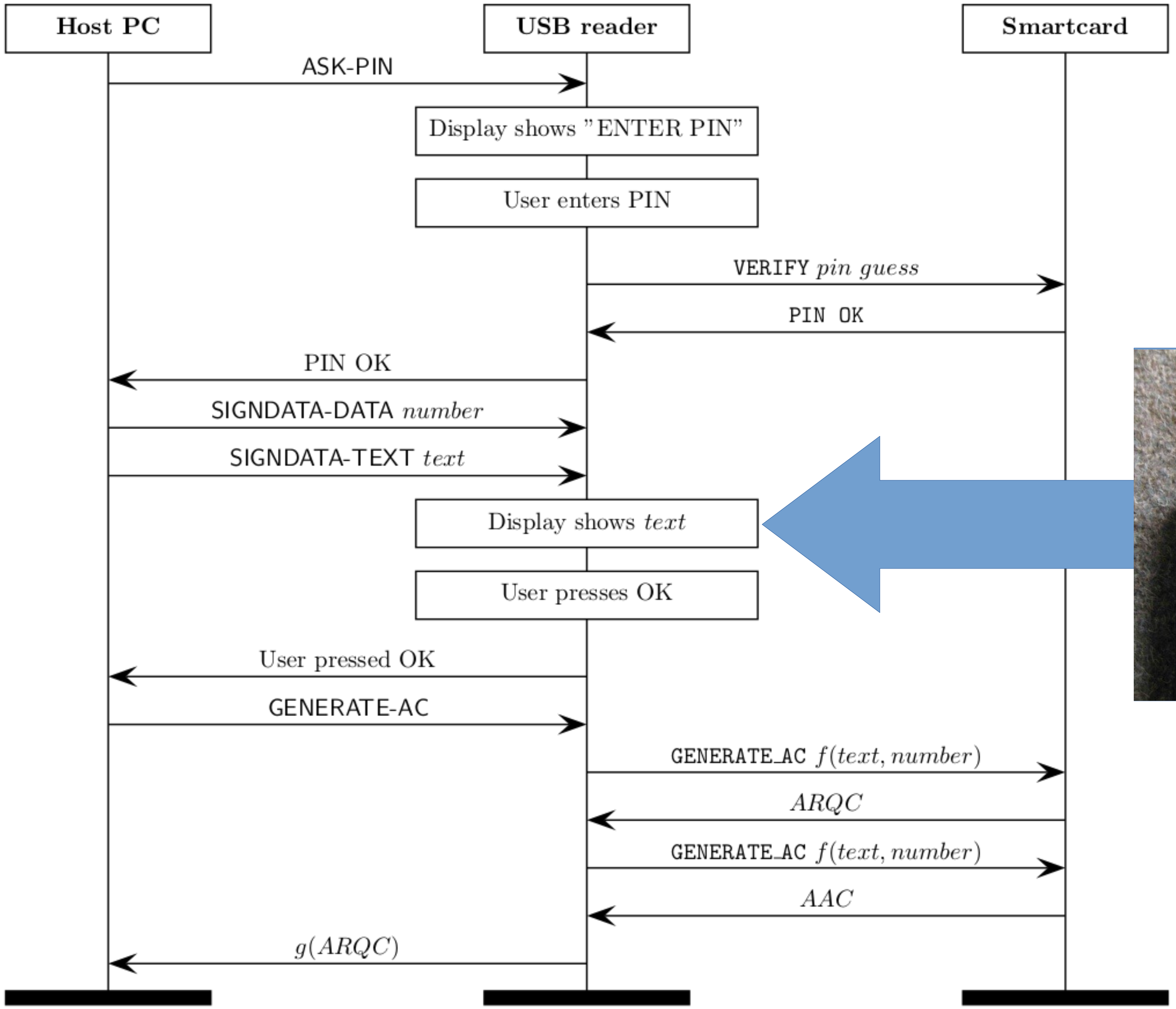
Lego robot



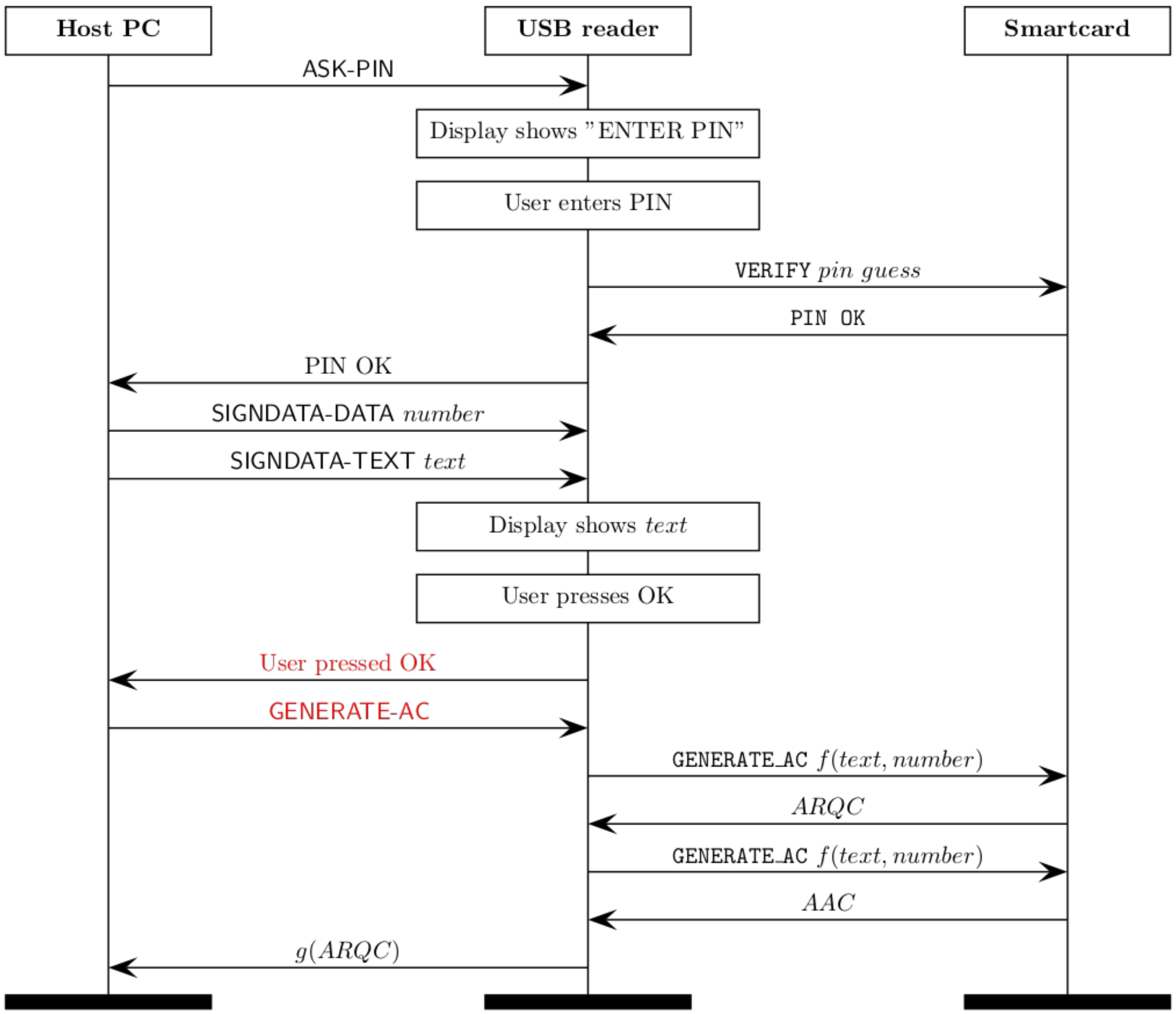
e.dentifier2 protocol



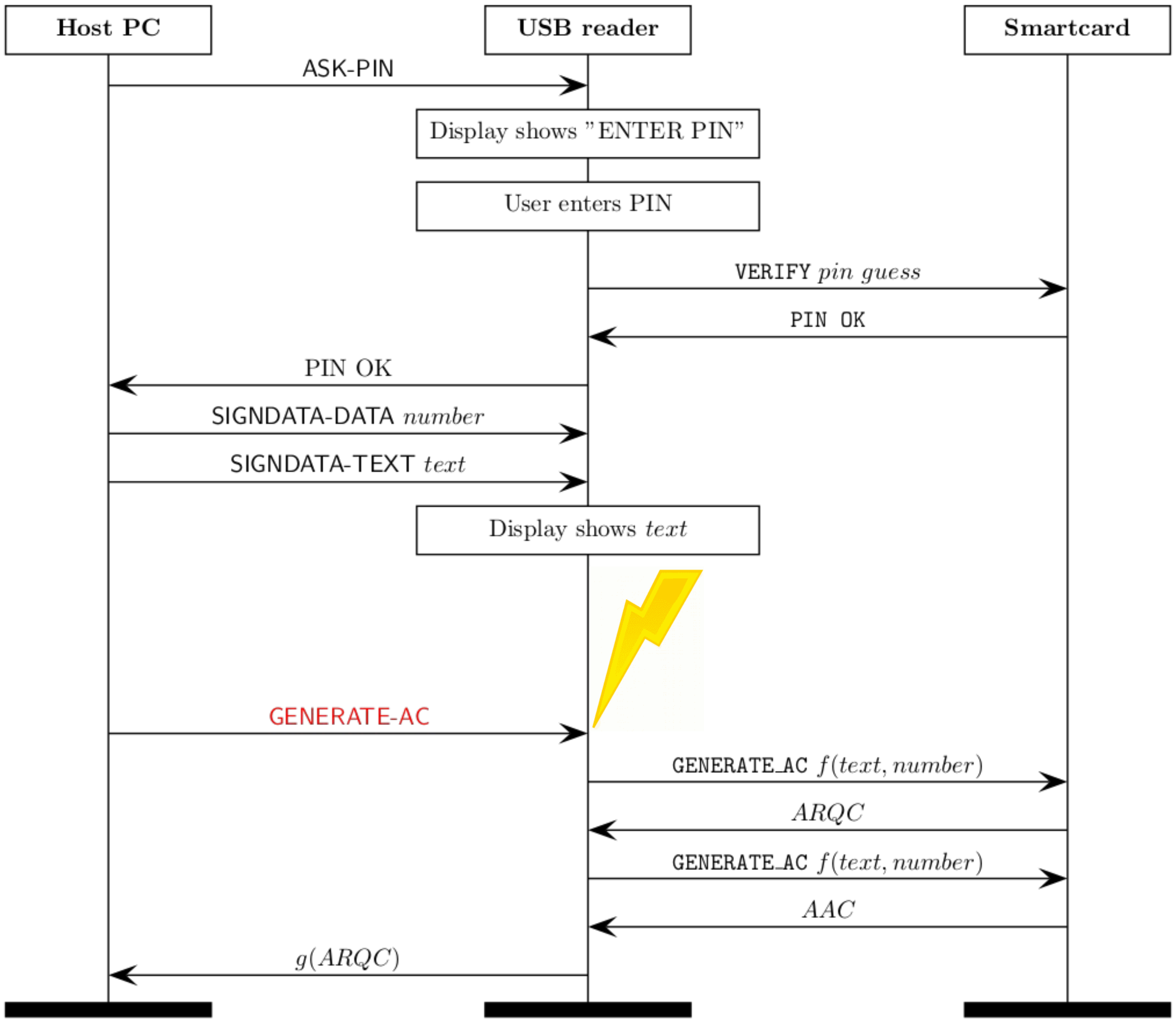
e.dentifier2 protocol



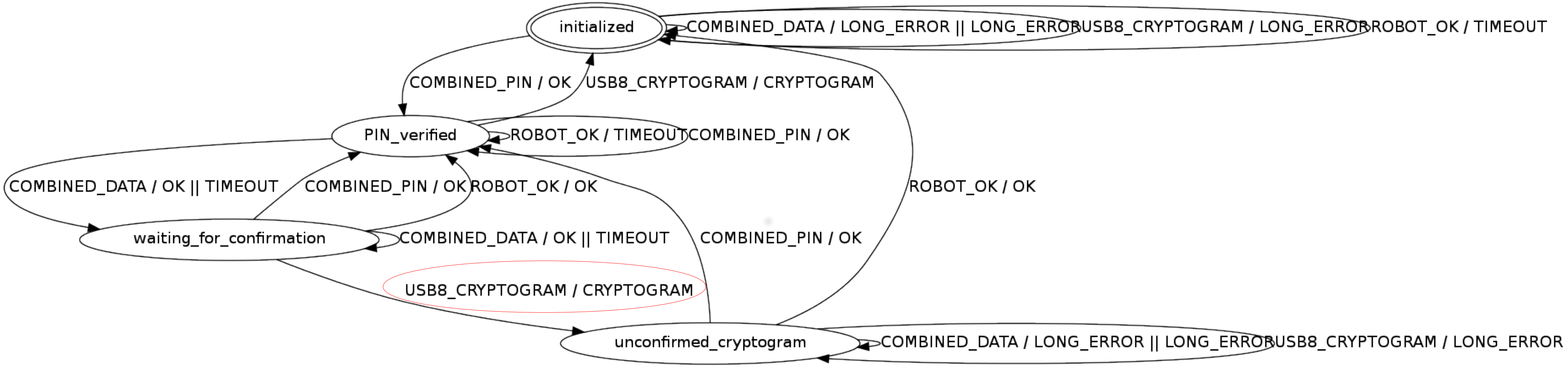
e.dentifier2 protocol

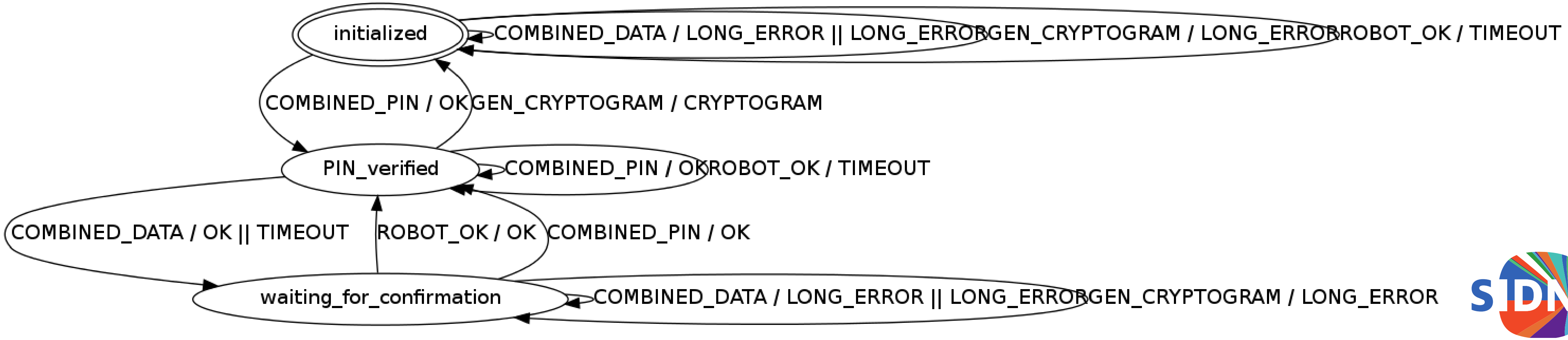


e.dentifier2 protocol



Results e.dentifier2





SIDN LABS

Analysing TLS

- (Almost) stateless TLS implementation in StateLearner
- Minimal state needed to support crypto operations
- Tested both clients and servers
- All regular TLS messages, as well as Heartbeat extension
  - RSA and DH key exchange
  - Client authentication
  - Some special symbols that correspond with exceptions in the test harness

Refreshing TLS

Client		**Server**
ClientHello	⟶	
		ServerHello
		Certificate
	⟵	ServerHelloDone
ClientKeyExchange		
ChangeCipherSpec		
Finished	⟶	
		ChangeCipherSpec
	⟵	Finished
ApplicationData	⟷	ApplicationData

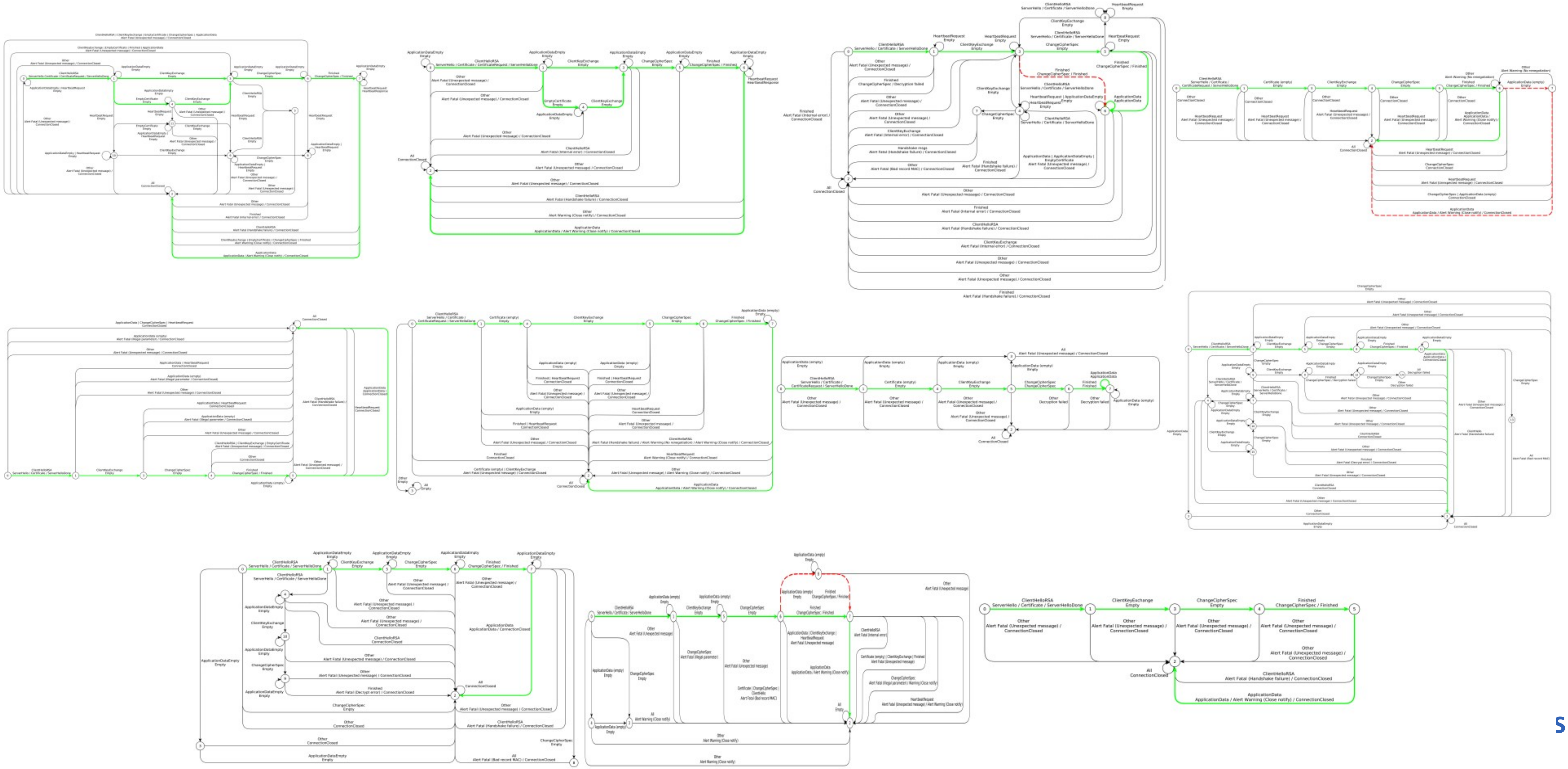
Analysing well-known TLS implementations

- Many different TLS implementations
  - OpenSSL, BoringSSL, LibreSSL
  - GnuTLS
  - Java Secure Socket Extension
  - mbed TLS (previously PolarSSL)
  - NSS
  - RSA BSAFE for C
  - RSA BSAFE for Java
  - miTLS
  - Nqsb-TLS
- Every learned model different!

Analysing TLS

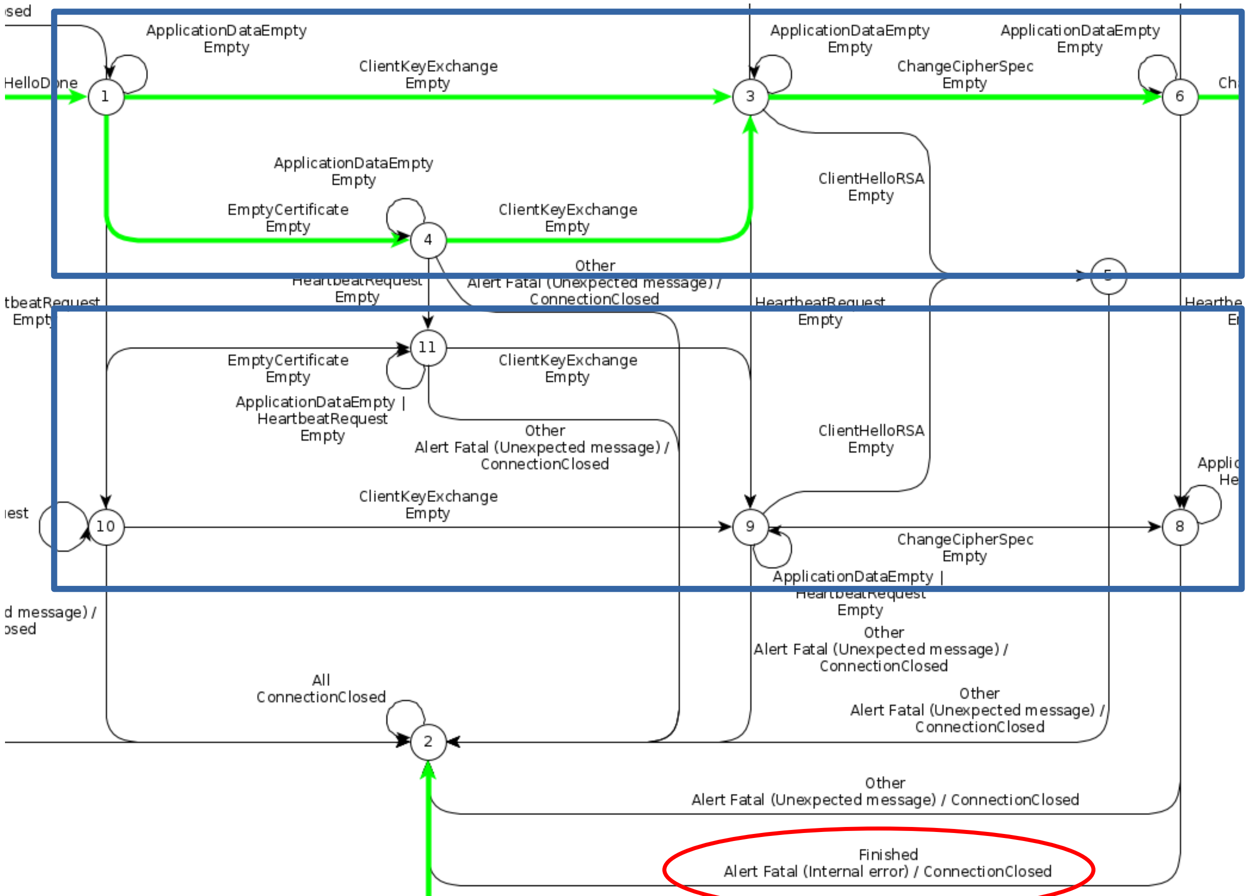
- Used demo application when available
- 6 to 16 states
- State machine learned in 6 minutes to 8 hours
  - Depends on implementation specific time-outs (100ms to 1.5s)
  - Under 1 hour if connections are properly closed
- Discovered flaws in different implementations

TLS models



GnuTLS

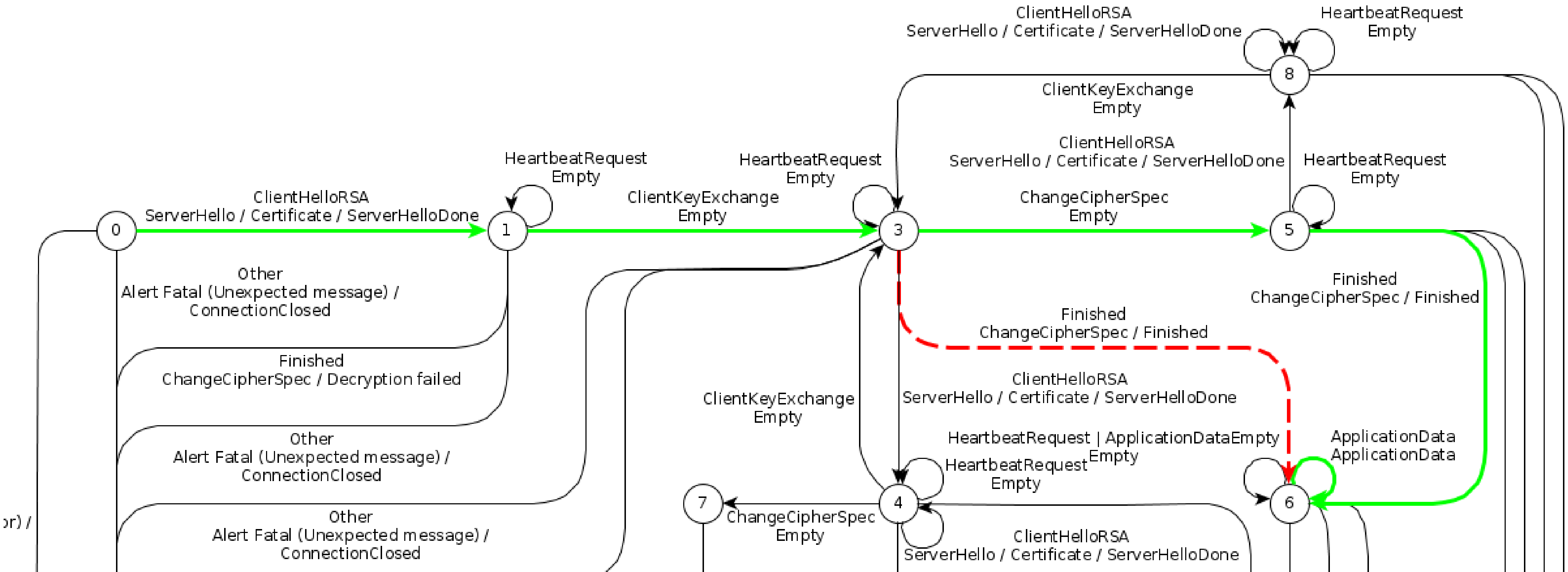
- Shadow path after sending HeartbeatRequest during handshake
- Buffer reset that contains all handshake messages to provide integrity
- Same problem present in the client



SIDN LABS

Java Secure Socket Extension

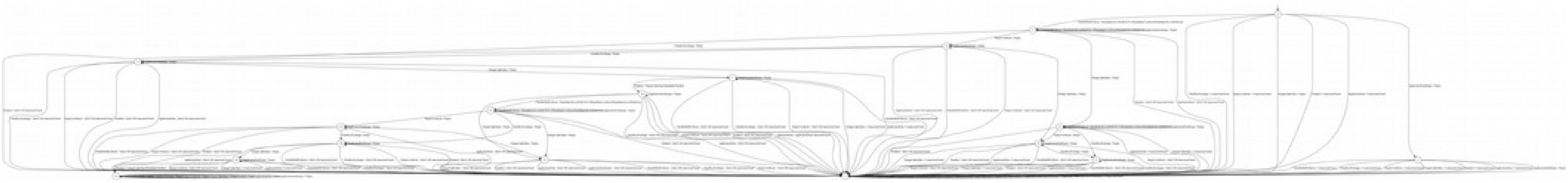
- Possible to skip ChangeCipherSpec message
- Server will accept plaintext data
- Problem also present in client
- At the same time discovered by the Prosecco group at INRIA, France



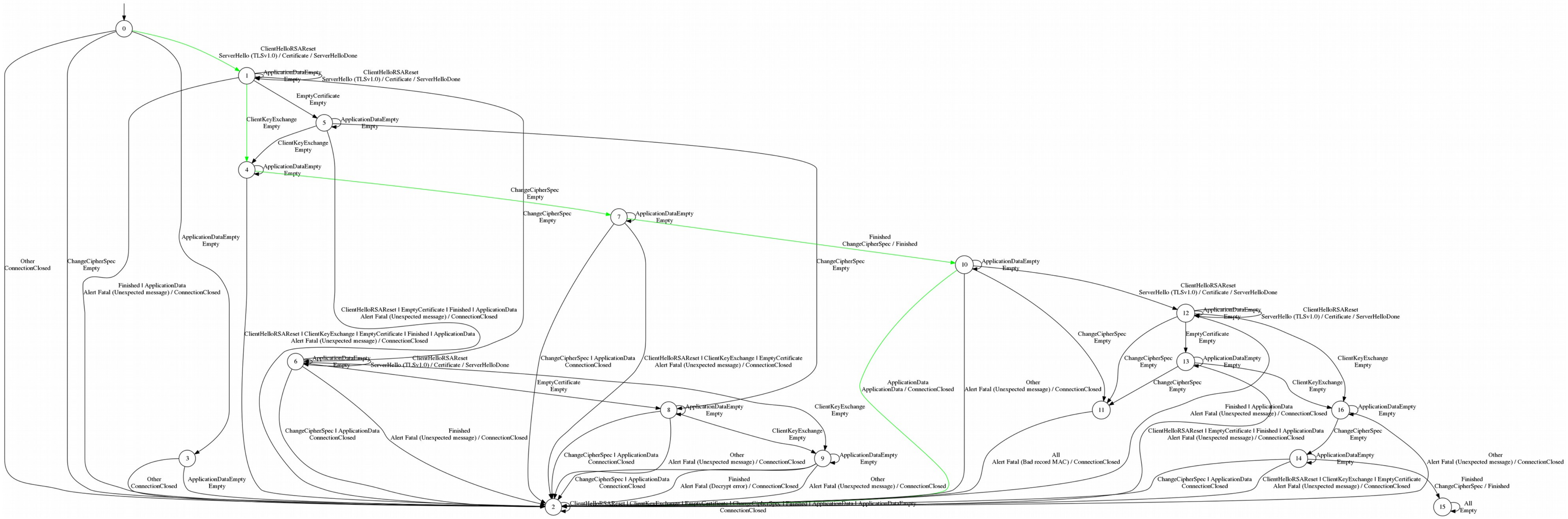
Large scale analysis of OpenSSL

- Learned 145 versions of OpenSSL and LibreSSL
- Number of unique state machines
  - Server-side: 15 for OpenSSL, 2 for LibreSSL
  - Client-side: 9 for OpenSSL, 1 for LibreSSL
- Number of states
  - Server-side: between 6 and 17
  - Client-side: between 7 and 12
- Several CVEs could be detected in older state machines
  - For example, EarlyCCS vulnerability

OpenSSL 0.9.7 (2002)

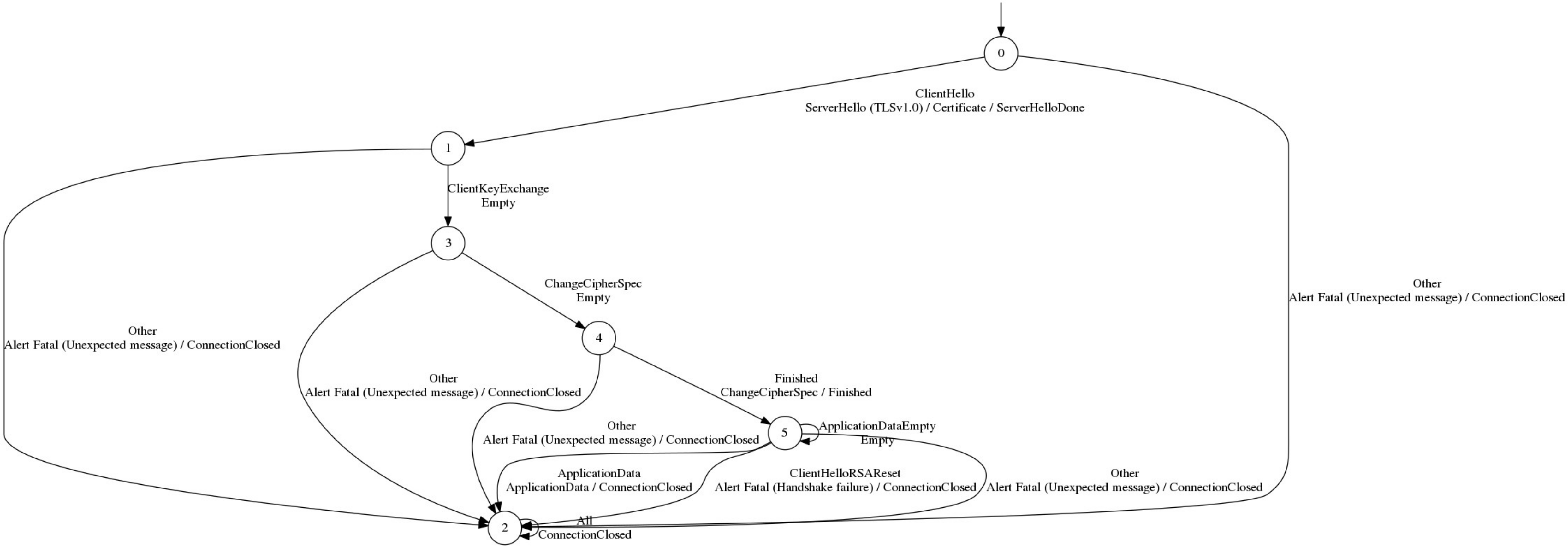


OpenSSL 0.9.7 (2002)



SIDN LABS

OpenSSL 1.1.0b (2016)



Conclusion

- State machine inference is an effective technique to discover security issues and other bugs
- Everybody interprets specifications differently
  - Including a state machine in specifications would help
- Can also be interesting to fingerprint implementations
- StateLearner is available from:
  https://github.com/jderuiter/statelearner

Thanks for your attention!

SIDN LABS