ETH zürich

Department of Computer Science

Tamarin Workshop
Automated Protocol Verification

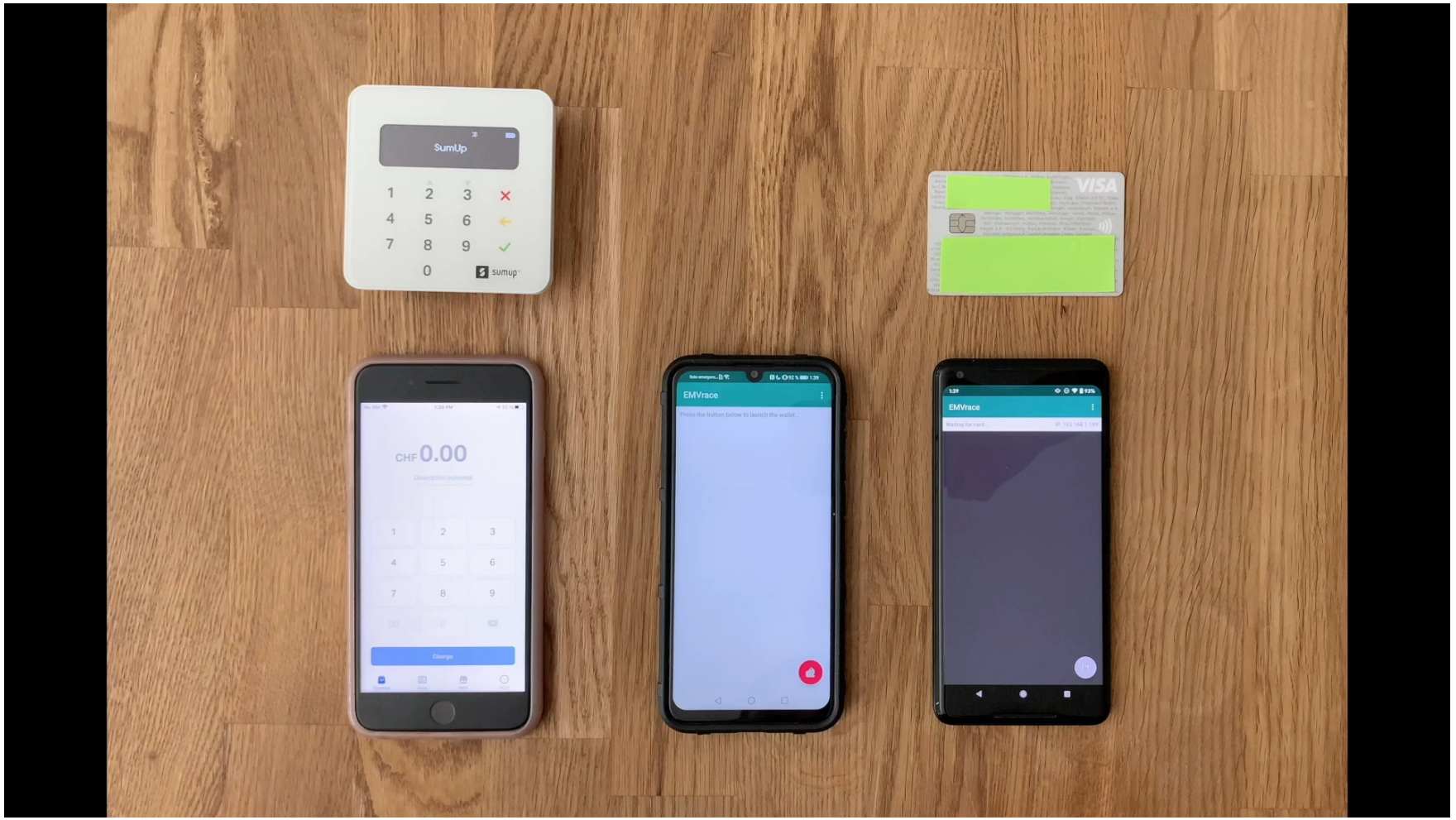
Felix Linker
PhD Student, ETH Zurich

Part 1: An Introduction to Tamarin

The EMV Standard: Break, Fix, Verify

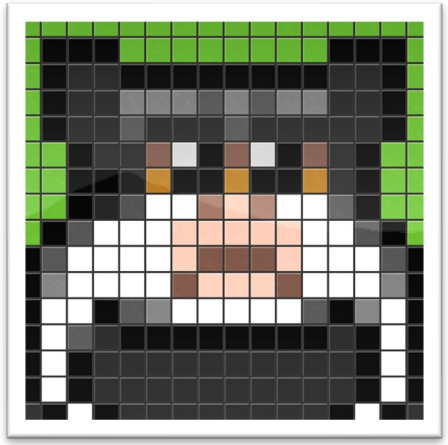
- S&P21 paper showed how to:
  - Pay with stolen credit card
  - Without ever needing the PIN

Attack Video



The EMV Standard: Break, Fix, Verify

- S&P21 paper showed how to:
  - Pay with stolen credit card
  - Without ever needing the PIN
- How did they find this attack?
- Used Tamarin!



What is Tamarin?

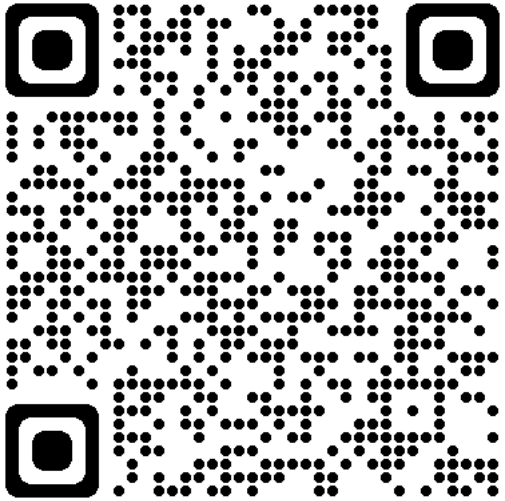
- Our world is powered by security-critical protocols
  - You want certain things to not happen
    - *NSA reads your WhatsApp messages*
  - You want certain things to always happen
    - *Merchant receives payment upon confirmation*
- Protocols are complex!
- People make mistakes!

With Tamarin, you can prove that a protocol (model) guarantees security properties

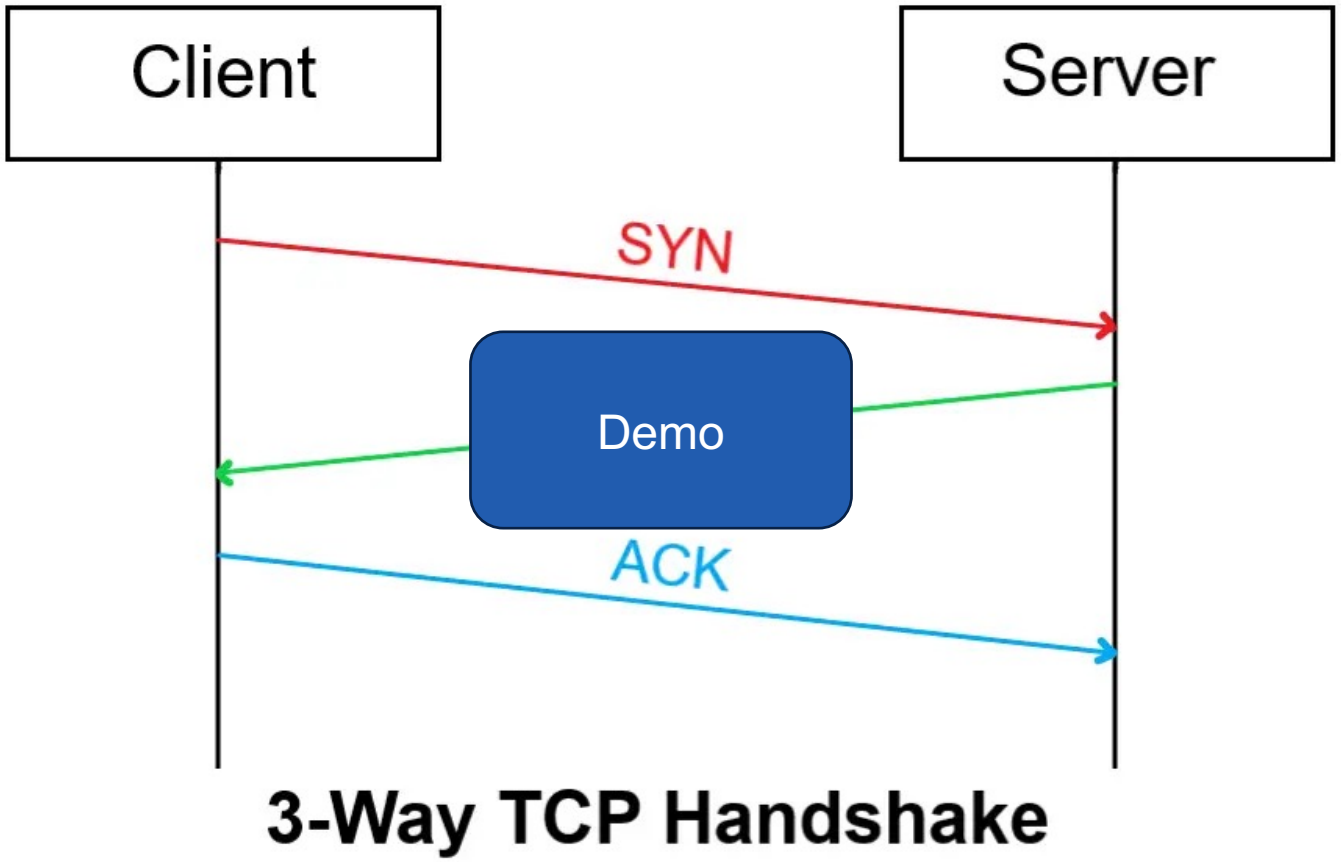
Workshop Goals

- Get your hands on Tamarin
- Tamarin is easy! (except when it isn't)

1. Go to github.com/felixlinker/tamarin-workshop/
2. Clone or download
3. Install Tamarin

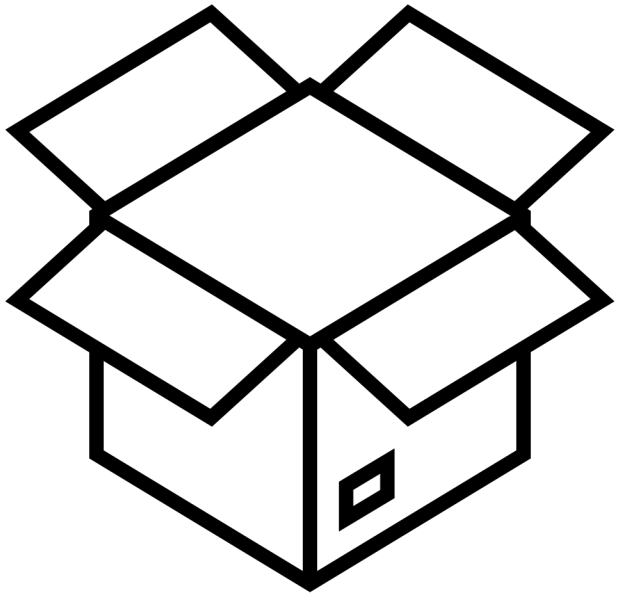


Example: TCP



Example: TCP – What happens under the hood?

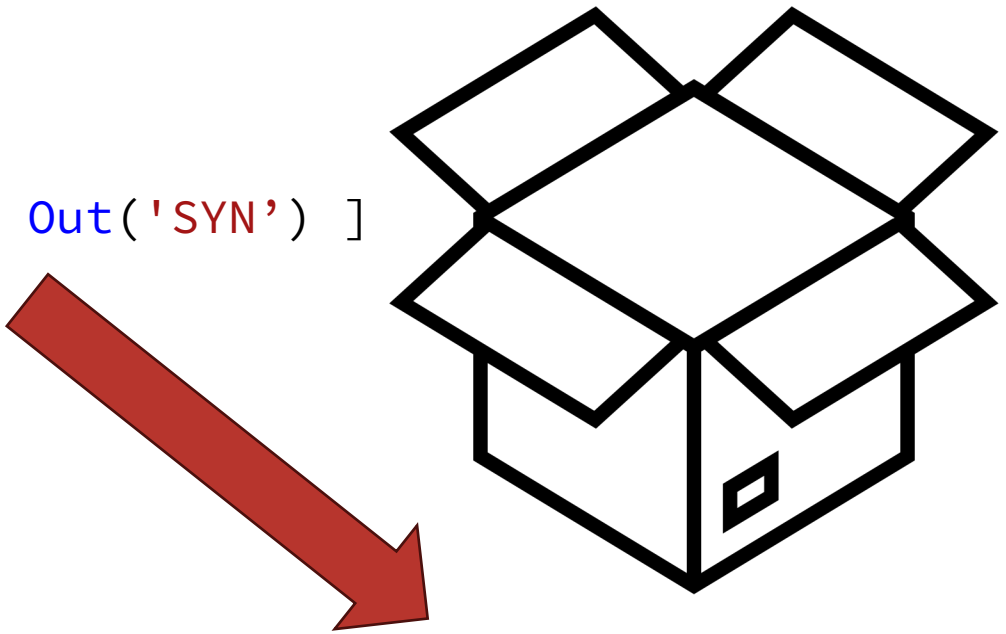
```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```



```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```



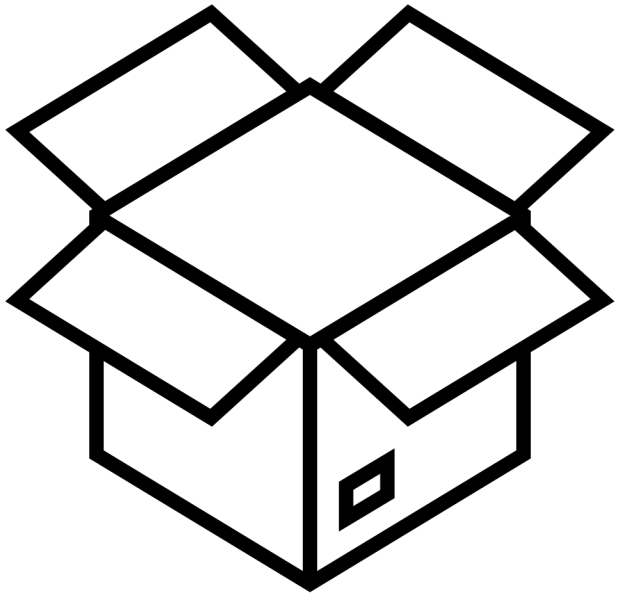
```
St_AliceWait()

Out('SYN')
```

```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```



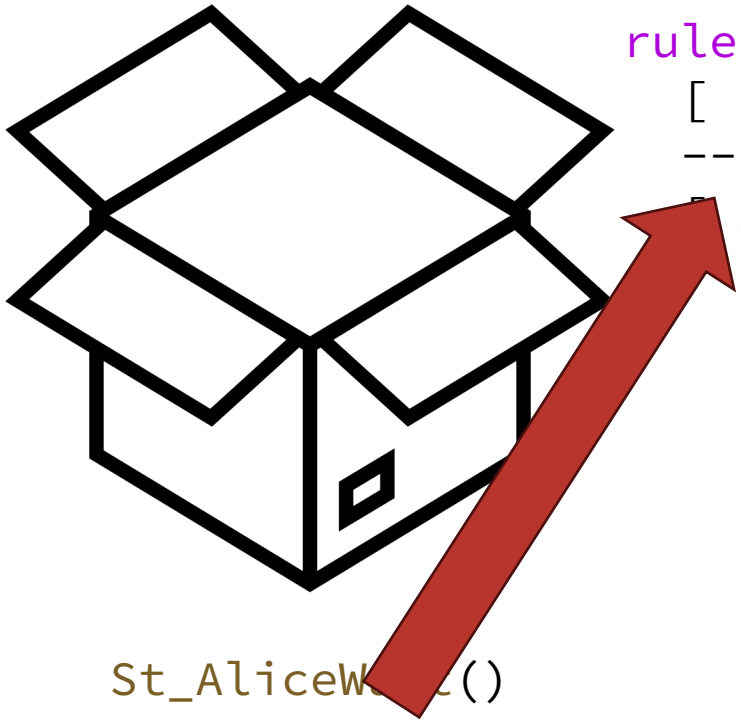
```
St_AliceWait()

In('SYN')
```

```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

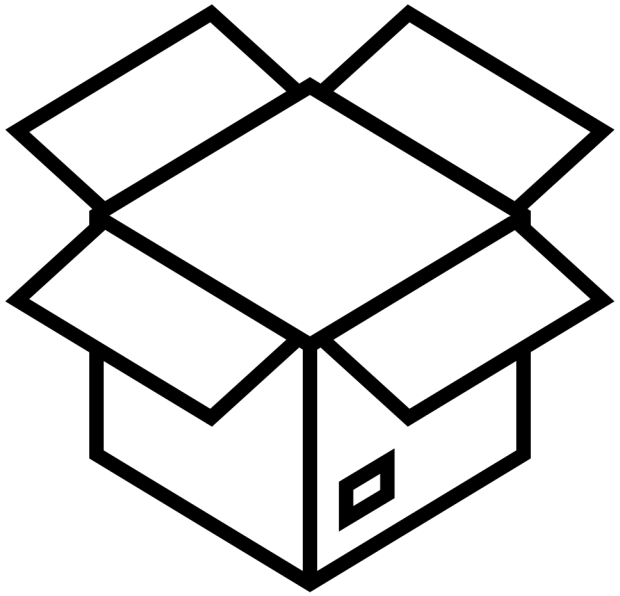
```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```




```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```

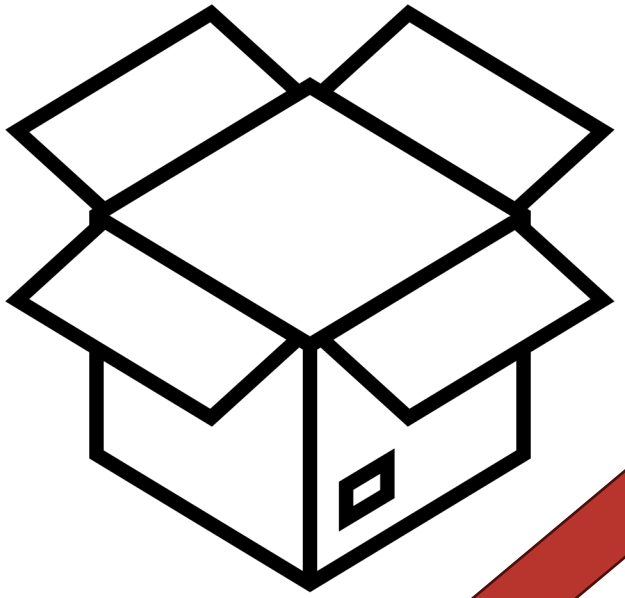


St_AliceWait()

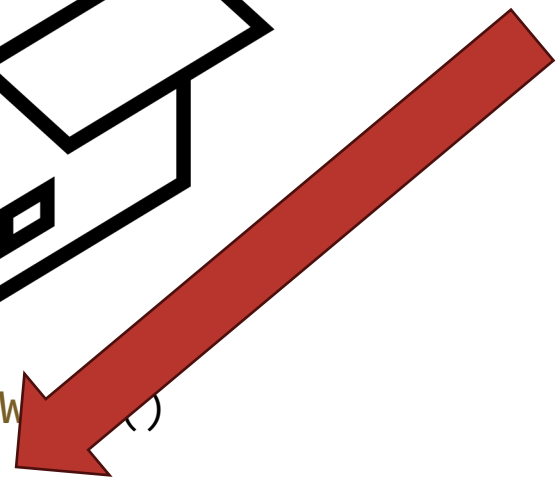
```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN') ]
```



```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```



```
St_AliceW      ()

St_BobWait()      Out('SYNACK')
```

Values in Tamarin

- Values can be:
  - Constants: `'constant'`
  - Unguessable (fresh) values: `~k`
  - Public values: `$P`
  - Function application: `f(t1, t2)`
- A variable `x` can be any of the above (also called message)
- Equational theory gives symbols semantics

```
functions: sign/2, verify/3, pk/1, true/0
equations: verify(sign(m, sk), m, pk(sk)) = true
```

Take-Aways

Cheatsheet!

Exercise 1 + 2!

```
functions: f/1
```

State read

Message in

```
rule [redacted]:
  [ St_X0(), In('...something...') ]
  --[ Begin() ]->
  [ St_X1(), Out('...something...') ]
```

State write

Message out

```
rule Me
  []
  -->
  [ MemorizeSomething(f('x')) ]
```

Fact

Function

```
rule LookUpAndSend:
  [ MemorizeSomething(v) ]
  -->
  [ Out(v) ]
```

Pattern-match

Summary – Part 1

- So far you learned
  - Modelling in Tamarin
  - State-read/message-in + state-write/message-out pattern
  - The symbolic model
- Interested in more? Documentation is quite good
- Also:
  - Manual proofs
  - Custom proof heuristics
  - Induction

Part 2: Analyzing Specifications with Tamarin

What is Tamarin?

- Our world is powered by security-critical protocols
  - You want certain things to not happen
    - *NSA reads your WhatsApp message*
  - You want certain things to
    - *Merchant receives paym*
- Protocols are complex!
- People make mistakes!

Tamarin proof = thing is secure

With Tamarin, you can prove that a protocol (model) guarantees security properties

What is Tamarin?

- Our world is powered by security-critical protocols
  - You want certain things to not happen
    - *NSA reads your WhatsApp*
  - You want certain things to
    - *Merchant receives paym*
- Protocols are complex!
- People make mistakes!

With Tamarin, you can prove that a protocol (model) guarantees security properties

What is Tamarin?

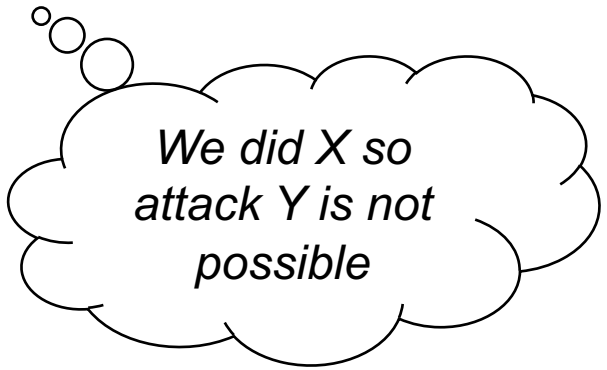
- Our world is powered by security-critical protocols
  - You want certain things to not happen
    - *NSA reads your WhatsApp messages*
  - You want certain things to always happen
    - *Merchant receives payment upon confirmation*
- Protocols are complex!
- People make mistakes!

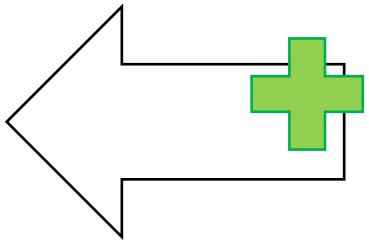
With Tamarin, you can prove that a protocol (model) guarantees certain security properties under certain assumptions

Specifications vs Formal Analysis

Specification

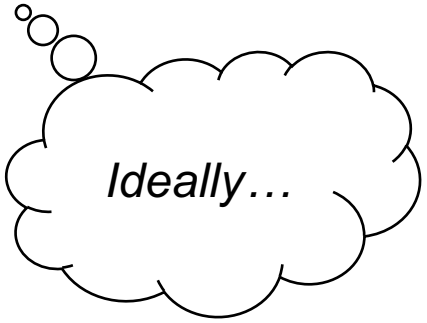
- Designed to foster compatible implementations
- Often deliberately underspecified
- Security considerations often ad-hoc



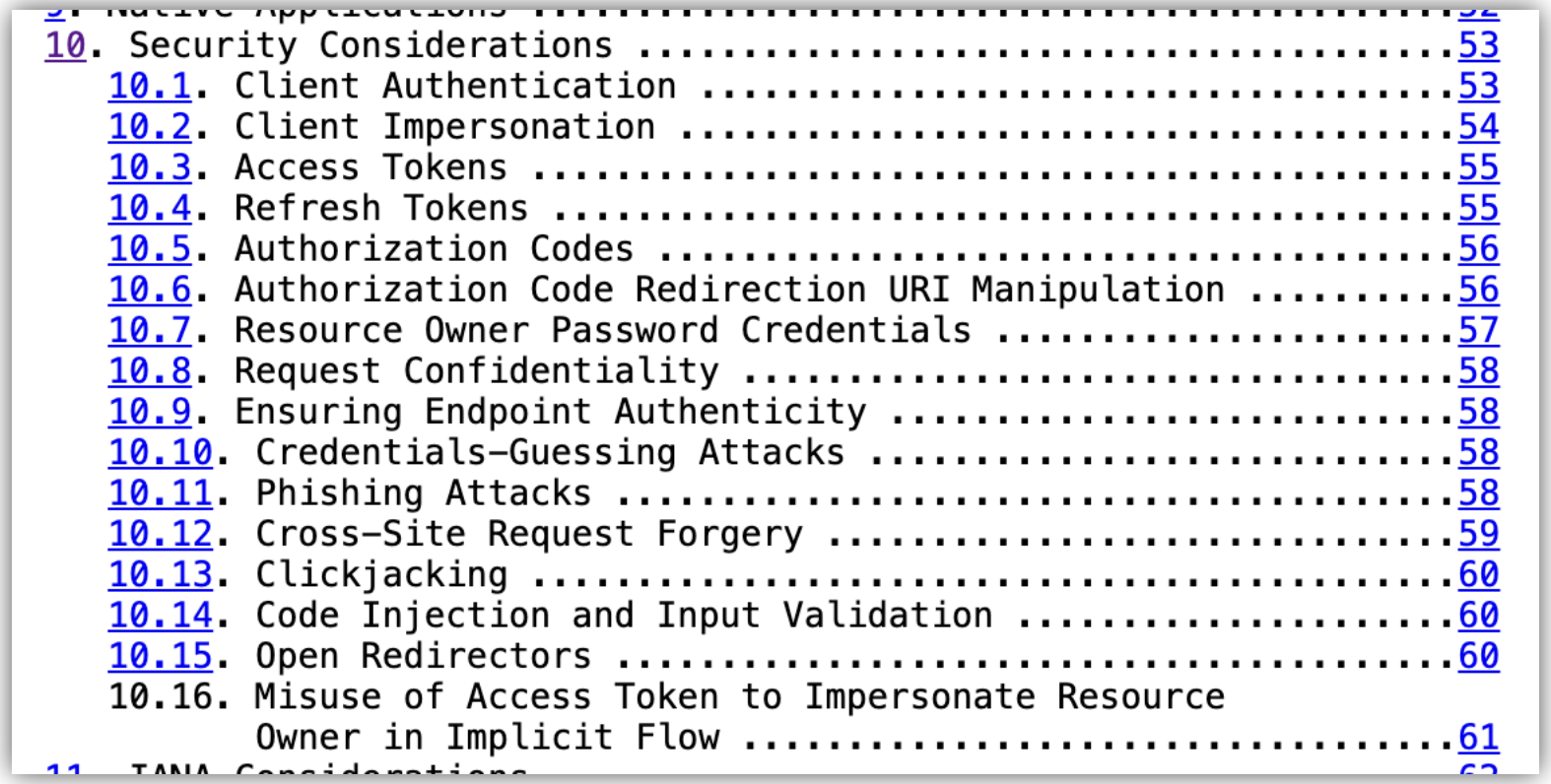


Formal Analysis

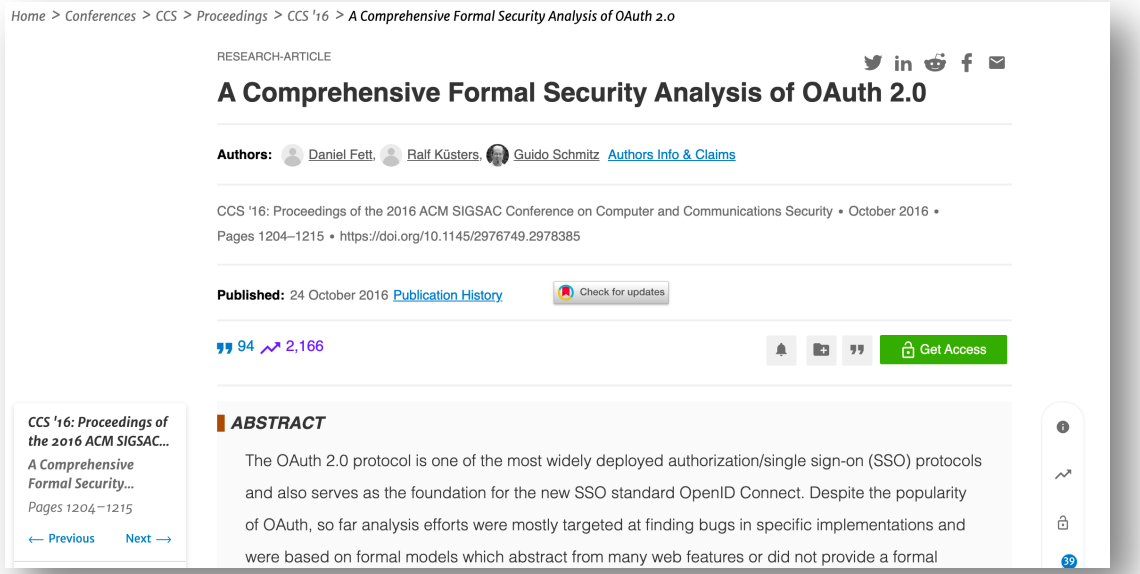
- A structured way to approach security
  - A positive definition of security properties
  - A list of explicit assumptions



Case Study: OAuth 2.0



Case Study: OAuth 2.0 – Prior Work



Home > Conferences > CCS > Proceedings > CCS '16 > A Comprehensive Formal Security Analysis of OAuth 2.0

RESEARCH-ARTICLE

A Comprehensive Formal Security Analysis of OAuth 2.0

Authors: Daniel Fett, Ralf Küsters, Guido Schmitz Authors Info & Claims

CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security • October 2016 • Pages 1204–1215 • https://doi.org/10.1145/2976749.2978385

Published: 24 October 2016 Publication History

Check for updates

94 2,166

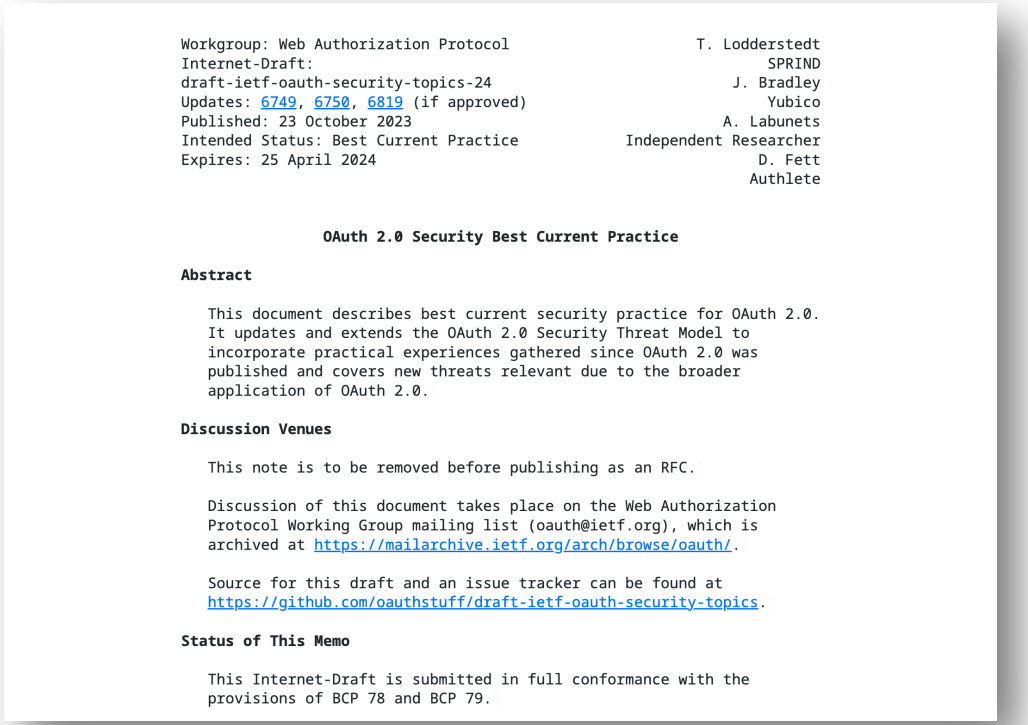
Get Access

CCS '16: Proceedings of the 2016 ACM SIGSAC...
A Comprehensive Formal Security...
Pages 1204–1215
← Previous Next →

ABSTRACT

The OAuth 2.0 protocol is one of the most widely deployed authorization/single sign-on (SSO) protocols and also serves as the foundation for the new SSO standard OpenID Connect. Despite the popularity of OAuth, so far analysis efforts were mostly targeted at finding bugs in specific implementations and were based on formal models which abstract from many web features or did not provide a formal

Fett, Küsters, Schmitz. CCS'16.



Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-security-topics-24
Updates: 6749, 6750, 6819 (if approved)
Published: 23 October 2023
Intended Status: Best Current Practice
Expires: 25 April 2024

T. Lodderstedt
SPRIND
J. Bradley
Yubico
A. Labunets
Independent Researcher
D. Fett
Authlete

OAuth 2.0 Security Best Current Practice

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

Discussion Venues

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

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/oauthstuff/draft-ietf-oauth-security-topics.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

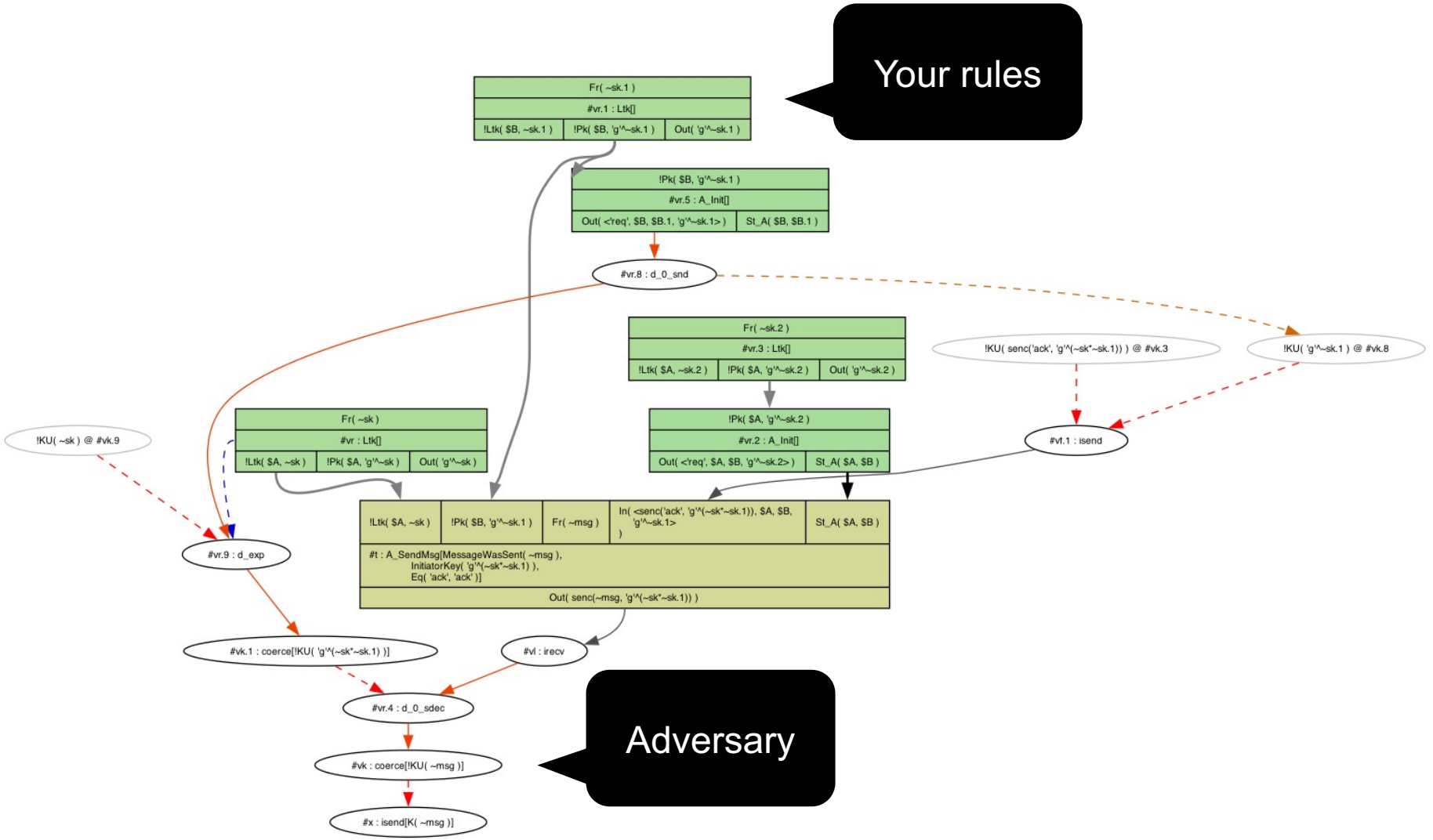
- But: Also doesn't list desired properties

Case Study: OAuth 2.0 – But how analyze a specification?

1. Implement an initial specification
2. Model security properties
   - It's okay if they are trivially true
3. Make your model more realistic
   - Now the properties are hopefully false
4. Refine everything
   - Let your understanding guide you
   - Let Tamarin tell you why your understanding is wrong

Use the GUI

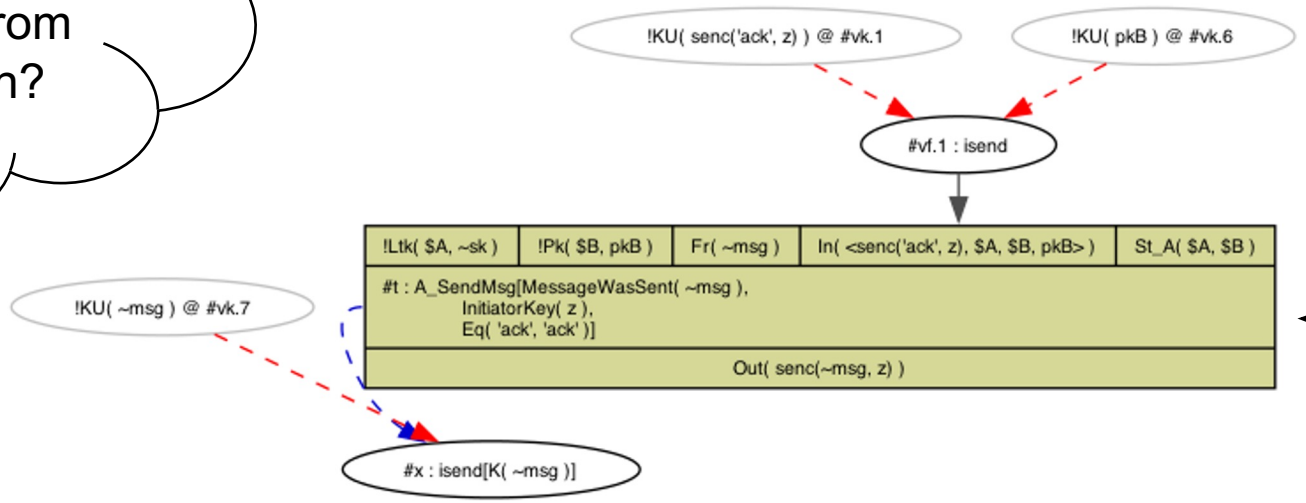
But how analyze a specification?



But how analyze a specification?

```
lemma SecrecyMessage:
  "All m #t. MessageWasSent(m) @ #t
   ==> not Ex #x. K(m) @ #x"
```

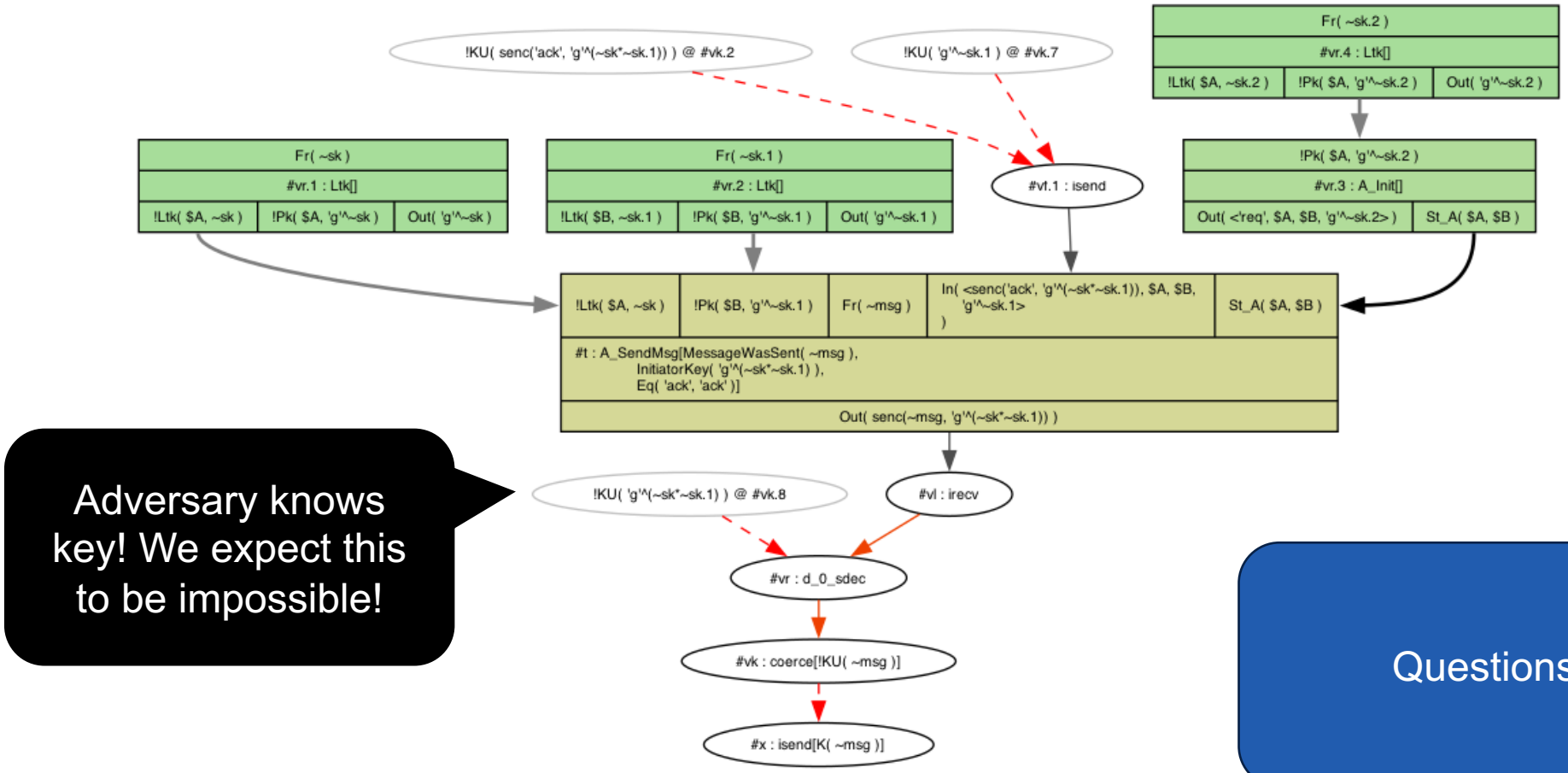
Where does the model deviate from our expectation?



Message was sent…

…and adversary knows it

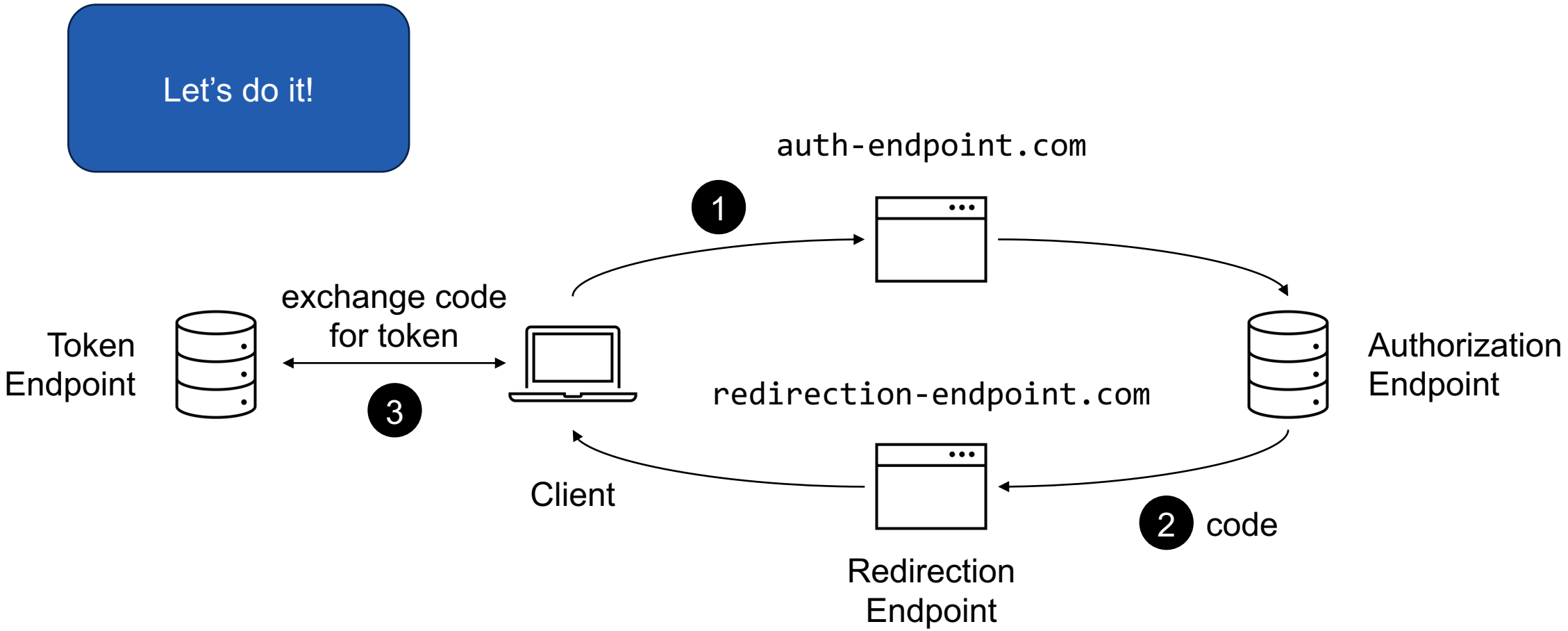
But how analyze a specification?



Adversary knows key! We expect this to be impossible!

Questions?

Case Study: OAuth 2.0 – Authorization Code Flow



Further Reading

C. Herley and P. C. Van Oorschot, "SoK: Science, Security and the Elusive Goal of Security as a Scientific Pursuit," 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2017, pp. 99-120, doi: 10.1109/SP.2017.38.

Daniel Fett, Ralf Küsters, and Guido Schmitz. 2016. A Comprehensive Formal Security Analysis of OAuth 2.0. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 1204–1215. https://doi.org/10.1145/2976749.2978385