ETH zürich

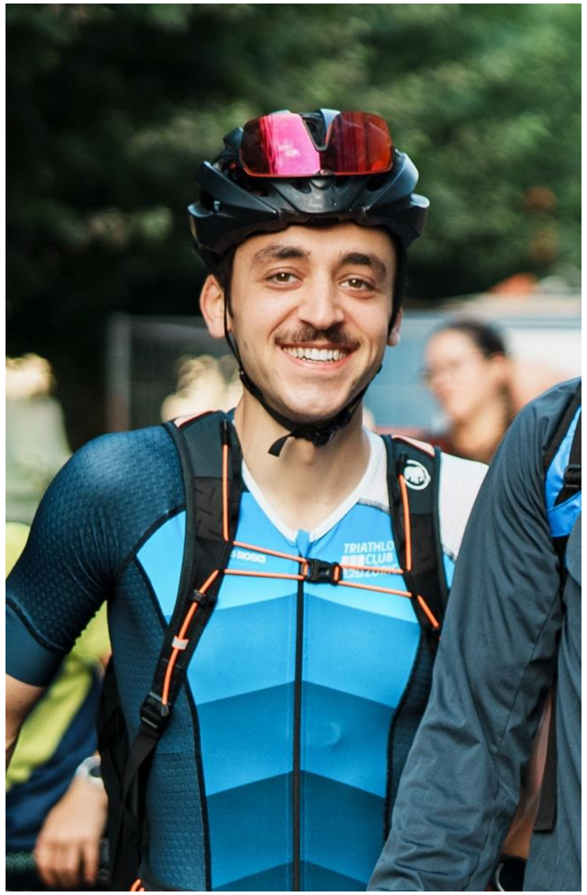
Department of Computer Science

Tamarin Workshop
Automated Protocol Verification

Felix Linker, Xenia Hofmeier
PhD Student, ETH Zurich

Who am I?

- Felix Linker
  - Doctoral Student at ETH Zurich and independent consultant
  - Active at the IETF
- Worked on and with Tamarin for the last five years
  - Formal verification of the iMessage PQ3 protocol
  - Formal verification of the SecureDrop whistleblowing protocol
  - Implemented new induction scheme for Tamarin



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

The EMV Standard: Break, Fix, Verify

David Basin, Ralf Sasse, and Jorge Toro-Pozo
Department of Computer Science
ETH Zurich, Switzerland

***Abstract*—EMV is the international protocol standard for smartcard payment and is used in over 9 billion cards worldwide. Despite the standard's advertised security, various issues have been previously uncovered, deriving from logical flaws that are hard to spot in EMV's lengthy and complex specification, running over 2,000 pages.**

We formalize a comprehensive symbolic model of EMV in Tamarin, a state-of-the-art protocol verifier. Our model is the first that supports a fine-grained analysis of all relevant security guarantees that EMV is intended to offer. We use our model to automatically identify flaws that lead to two critical attacks: one that defrauds the cardholder and a second that defrauds the merchant. First, criminals can use a victim's Visa contactless card to make payments for amounts that require cardholder verification, without knowledge of the card's PIN. We built a proof-of-concept Android application and successfully demonstrated this attack on real-world payment terminals. Second, criminals can trick the terminal into accepting an unauthentic offline transaction, which the issuing bank should later decline, after the criminal has walked away with the goods. This attack is possible for implementations following the standard, although we did not test it on actual terminals for ethical reasons. Finally, we propose and verify improvements to the standard that prevent these attacks, as well as any other attacks that violate the considered security properties. The proposed improvements can be easily implemented in the terminals and do not affect the cards in circulation.

I. Introduction

EMV, named after its founders Europay, Mastercard, and Visa, is the worldwide standard for smartcard payment, developed in the mid 1990s. As of December 2019, more than 80% of all card-present transactions globally use EMV, reaching up to 98% in many European countries. Banks have a strong incentive to adopt EMV due to the *liability shift*, which relieves banks using the standard from any liability from payment disputes. If the disputed transaction was authorized by a PIN then the consumer (EMV terminology for the payment-card customer) is held liable. If a paper signature was used instead, then the merchant is charged.

EMV: 20 Years of Vulnerabilities

Besides the liability shift, EMV's global acceptance is also attributed to its advertised security. However, EMV's security has been challenged numerous times. Man-in-the-middle (MITM) attacks [1], card cloning [2], [3], downgrade attacks [3], relay attacks [4]–[8], and card skimming [9], [10] are all examples of successful exploits of the standard's shortcomings. The MITM attack reported by Murdoch *et al.* [1] is believed to have been used by criminals in 2010–11 in France and Belgium to carry out fraudulent transactions for ca. 600,000 Euros [11]. The underlyi *al.*'s attack is that the card's response PIN verification request is not authen

Some of the security issues iden implementations of the standard. flaws whose repairs would require infrastructure. Identifying such fl to the complexity of EMV's exe flexible in terms of card authe verification methods, and onlin raises the question of how we possible executions and improv twenty years of attacks.

Approach Taken: Break, Fix,

In this paper we focus on the EMV protocol design. model for the symbolic ana is written in Tamarin [12] tool that has been used to including TLS 1.3 [14] supports protocol verifi versaries and unbounde

Our model supports hold in any EMV tra three most relevant

1) *Bank accepts* action accepte bank.
2) *Authentication to the terminal*: by the terminal are authenticated by the authorized online, the bank.
3) *Authentication to the bank*: All transactions accepted by the bank are authenticated by the card and the terminal.

Our model faithfully considers the three roles present in an EMV session: the bank, the terminal, and the card. Previous symbolic models merge the terminal and the bank into a single agent [16]–[18]. This merging incorrectly entails that the terminal can verify all card-produced cryptographic proofs that the bank can. This is incorrect as the card and the bank share a symmetric key that is only known to them.

Using our model, we identify a critical violation of authentication properties by the Visa contactless protocol: the cardholder verification method used in a transaction, if any, is neither authenticated nor cryptographically protected against modification. We developed a proof-of-concept Android application that exploits this to **bypass PIN verification** by

IEEE COMPUTER SOCIETY

IEEE

42nd IEEE Symposium on Security and Privacy

Best Practical Paper

Sponsored by Intel and IBM

Presented to

David Basin, Ralf Sasse, Jorge Toro-Pozo

for

The EMV Standard: Break, Fix, Verify

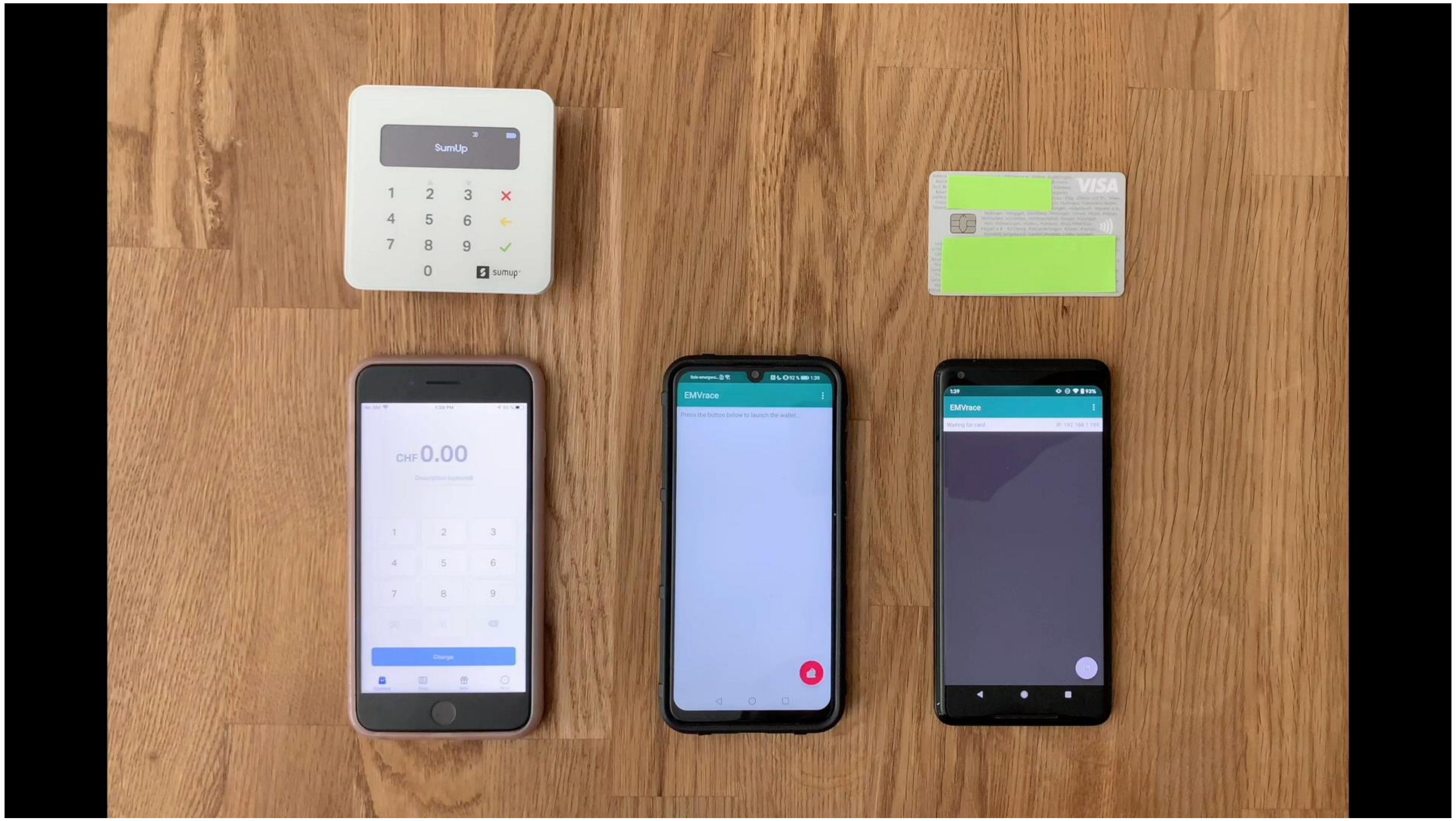
May 24-27, 2021

Alina Oprea
Alina Oprea, Program Chair SP 21

Alvaro Cardenas
Alvaro Cardenas, General Chair SP 21

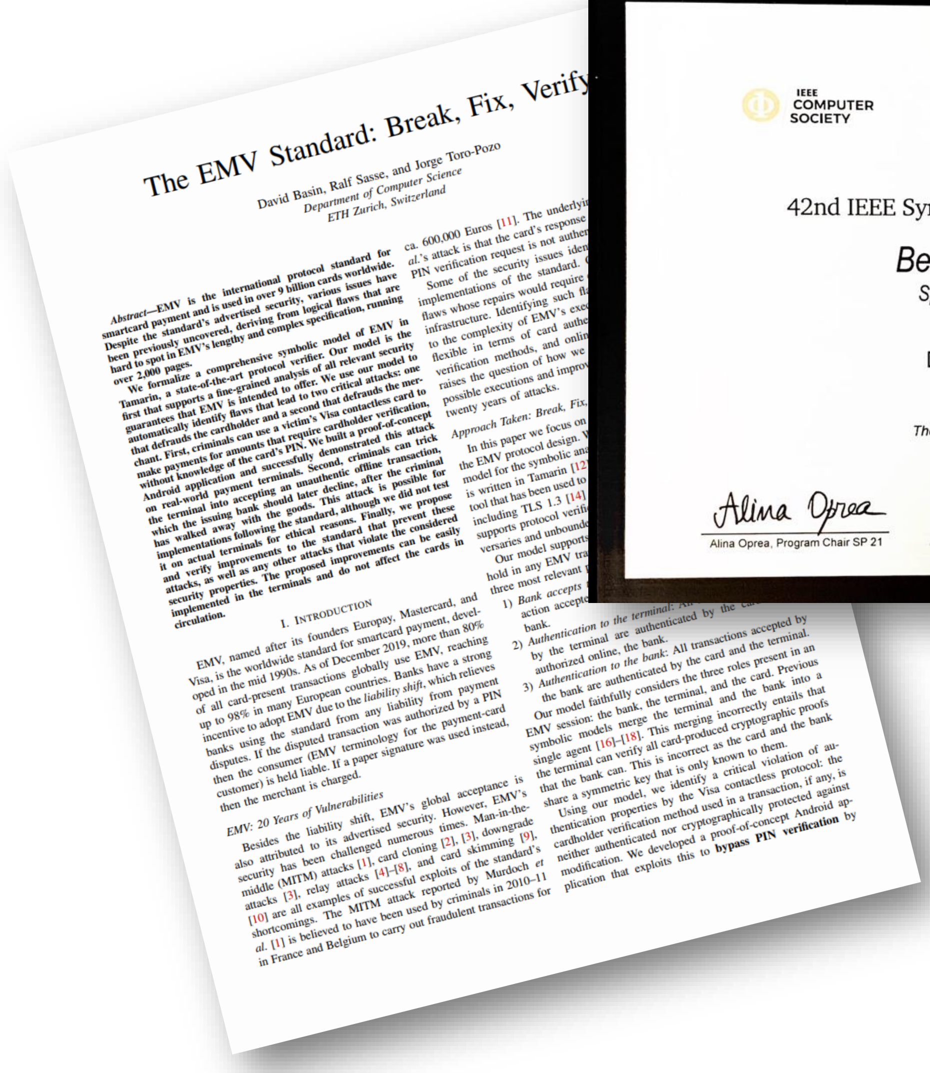
[illegible]
Thorsten Holz, Program Chair SP 21

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!



The EMV Standard: Break, Fix, Verify

David Basin, Ralf Sasse, and Jorge Toro-Pozo
Department of Computer Science
ETH Zurich, Switzerland

***Abstract*—EMV is the international protocol standard for smartcard payment and is used in over 9 billion cards worldwide. Despite the standard's advertised security, various issues have been previously uncovered, deriving from logical flaws that are hard to spot in EMV's lengthy and complex specification, running over 2,000 pages.**

We formalize a comprehensive symbolic model of EMV in Tamarin, a state-of-the-art protocol verifier. Our model is the first that supports a fine-grained analysis of all relevant security guarantees that EMV is intended to offer. We use our model to automatically identify flaws that lead to two critical attacks: one that defrauds the cardholder and a second that defrauds the merchant. First, criminals can use a victim's Visa contactless card to make payments for amounts that require cardholder verification, without knowledge of the card's PIN. We built a proof-of-concept Android application and successfully demonstrated this attack on real-world payment terminals. Second, criminals can trick the terminal into accepting an unauthentic offline transaction, which the issuing bank should later decline, after the criminal has walked away with the goods. This attack is possible for implementations following the standard, although we did not test it on actual terminals for ethical reasons. Finally, we propose and verify improvements to the standard that prevent these attacks, as well as any other attacks that violate the considered security properties. The proposed improvements can be easily implemented in the terminals and do not affect the cards in circulation.

I. Introduction

EMV, named after its founders Europay, Mastercard, and Visa, is the worldwide standard for smartcard payment, developed in the mid 1990s. As of December 2019, more than 80% of all card-present transactions globally use EMV, reaching up to 98% in many European countries. Banks have a strong incentive to adopt EMV due to the *liability shift*, which relieves banks using the standard from any liability from payment disputes. If the disputed transaction was authorized by a PIN then the consumer (EMV terminology for the payment-card customer) is held liable. If a paper signature was used instead, then the merchant is charged.

EMV: 20 Years of Vulnerabilities

Besides the liability shift, EMV's global acceptance is also attributed to its advertised security. However, EMV's security has been challenged numerous times. Man-in-the-middle (MITM) attacks [1], card cloning [2], [3], downgrade attacks [3], relay attacks [4]–[8], and card skimming [9], [10] are all examples of successful exploits of the standard's shortcomings. The MITM attack reported by Murdoch *et al.* [1] is believed to have been used by criminals in 2010–11 in France and Belgium to carry out fraudulent transactions for ca. 600,000 Euros [11]. The underlyi
al.'s attack is that the card's response
PIN verification request is not authen

Some of the security issues iden
implementations of the standard. O
flaws whose repairs would require
infrastructure. Identifying such fl
to the complexity of EMV's exec
flexible in terms of card authe
verification methods, and onlin
raises the question of how we
possible executions and improv
twenty years of attacks.

Approach Taken: Break, Fix,

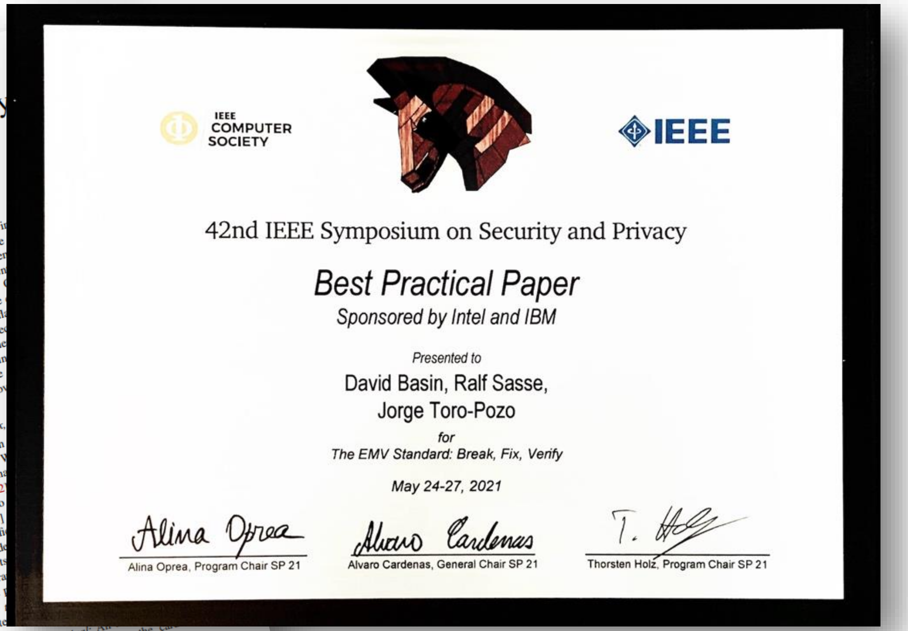
In this paper we focus on
the EMV protocol design. W
model for the symbolic ana
is written in Tamarin [12
tool that has been used to
including TLS 1.3 [14]
supports protocol verifi
versaries and unbounde

Our model supports
hold in any EMV tra
three most relevant p

1) *Bank accepts* t
action accepte
bank.
2) *Authentication to the terminal*: A
by the terminal are authenticated by the ca
authorized online, the bank.
3) *Authentication to the bank*: All transactions accepted by the bank are authenticated by the card and the terminal.

Our model faithfully considers the three roles present in an EMV session: the bank, the terminal, and the card. Previous symbolic models merge the terminal and the bank into a single agent [16]–[18]. This merging incorrectly entails that the terminal can verify all card-produced cryptographic proofs that the bank can. This is incorrect as the card and the bank share a symmetric key that is only known to them.

Using our model, we identify a critical violation of authentication properties by the Visa contactless protocol: the cardholder verification method used in a transaction, if any, is neither authenticated nor cryptographically protected against modification. We developed a proof-of-concept Android application that exploits this to **bypass PIN verification** by



IEEE COMPUTER SOCIETY

IEEE

42nd IEEE Symposium on Security and Privacy

Best Practical Paper

Sponsored by Intel and IBM

Presented to

David Basin, Ralf Sasse, Jorge Toro-Pozo

for

The EMV Standard: Break, Fix, Verify

May 24-27, 2021

Alina Oprea
Alina Oprea, Program Chair SP 21

Alvaro Cardenas
Alvaro Cardenas, General Chair SP 21

[illegible]
Thorsten Holz, Program Chair SP 21

What is Tamarin?

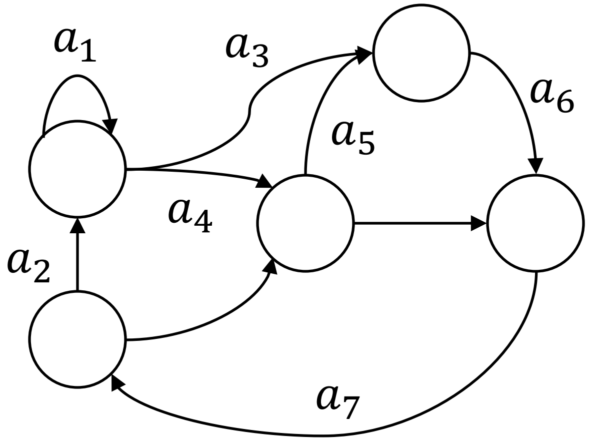
- Our world is powered by security-critical protocols
  - You want certain things to not happen
    - *Your neighbor reads your WhatsApp messages*
  - You want certain things to happen
    - *When you receive a message from me, I had sent it to you*
- Protocols are complex!
- People make mistakes!

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

The Tamarin Prover

Multiset-Rewriting Rules…

- Define labelled state transition system
- Model participant steps
- Model environment



An equational Theory…

- Defines cryptographic operations
- Defines adversary capabilities

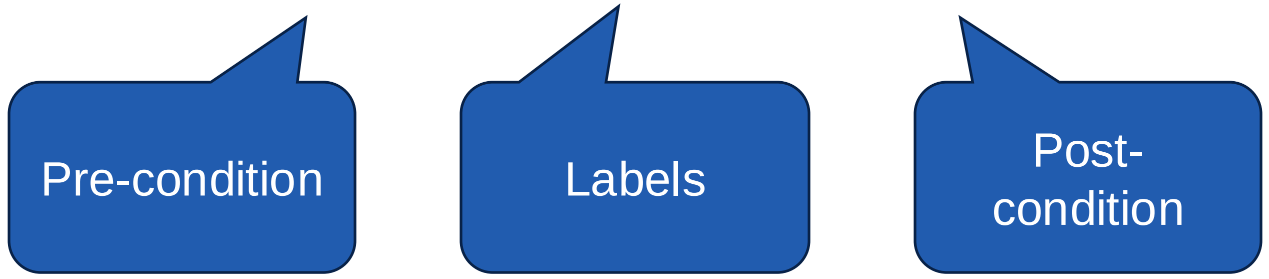
```
verify(sign(m, sk), m, pk(sk)) = true

sdec(senc(m, k), k) = m
```

Multiset Rewriting Rules

- You write a protocol **model**
- The model admits a set of **traces**
- A **multiset-rewriting rule** defines a state transition
  - Multiset-rewriting rules use multisets of **facts**
  - Pre-condition: Which state is required to apply the rule?
  - Labels: For reference in properties
  - Post-condition: Which state is added when the rule is applied?

```
rule NAME:
  [ ... ] --[ ... ]-> [ ... ]
```

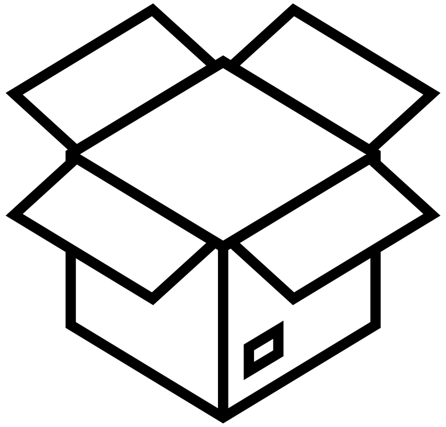


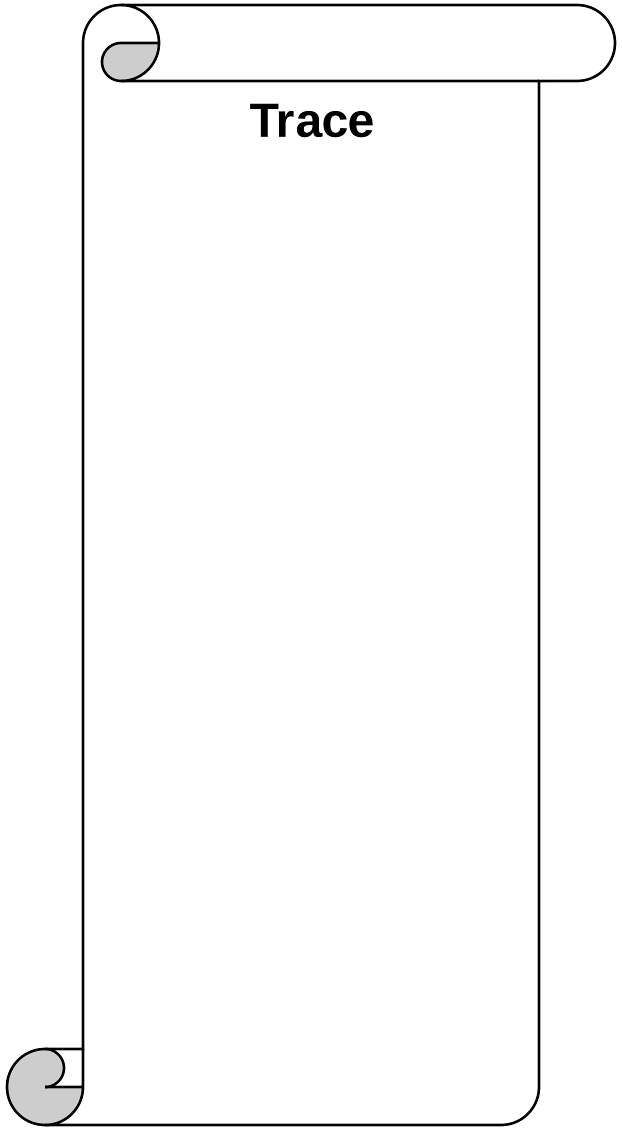
Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

Multiset Rewriting Rules - Example

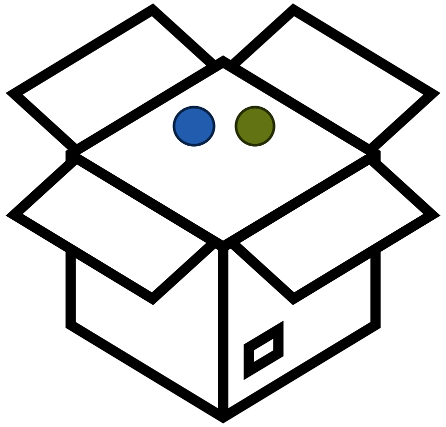
```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```





Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

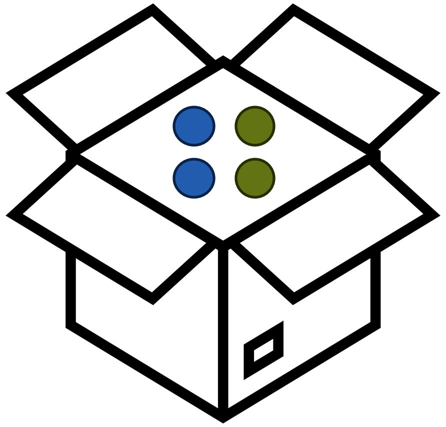


Trace

`Start()`

Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

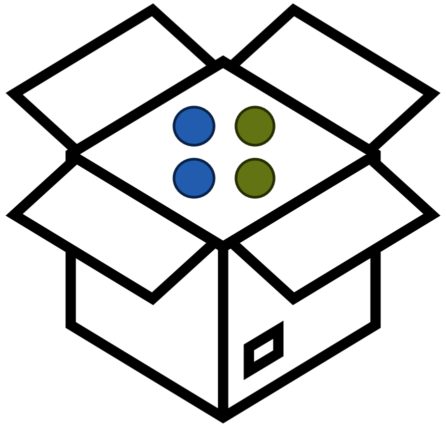


Trace

```
Start()
Start()
```

Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

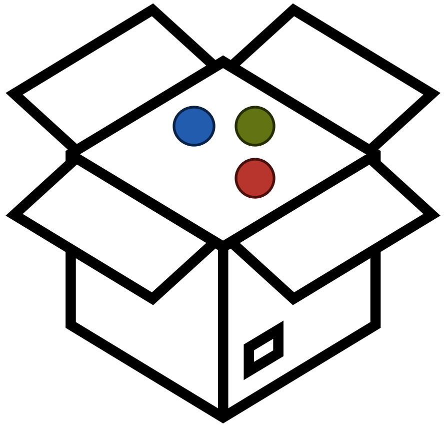


Trace

```
Start()
Start()
```

Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

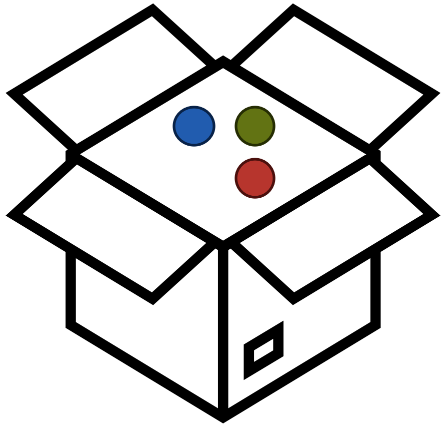


Trace

```
Start()
Start()
RedAdded()
```

Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule AddYellow:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```

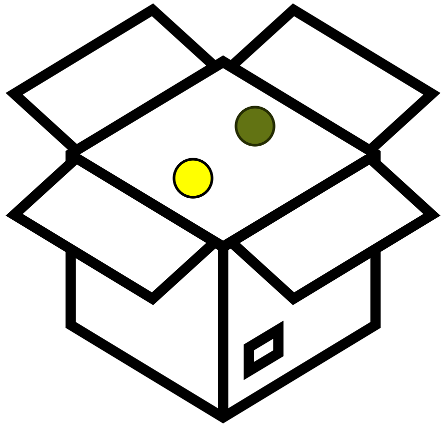


Trace

```
Start()
Start()
RedAdded()
```

Multiset Rewriting Rules - Example

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] --[ RedAdded() ]-> [ Red() ]
rule **AddYellow**:
  [ Blue(), Red() ] --[ YellowAdded() ]-> [ Yellow() ]
```



Trace

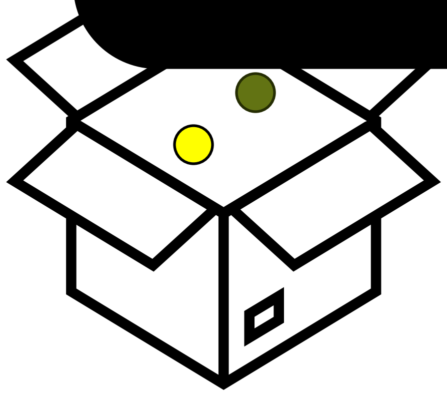
```
Start()
Start()
RedAdded()
YellowAdded()
```

Multiset Rewriting Rules - Example

Properties talk about **traces**

```
rule Start:
  [] --[ Start() ]-> [ Green(), Blue() ]
rule AddRed:
  [ Green(), Blue() ] -
rule AddYellow:
  [ Blue(), Red() ] --
```

Demo



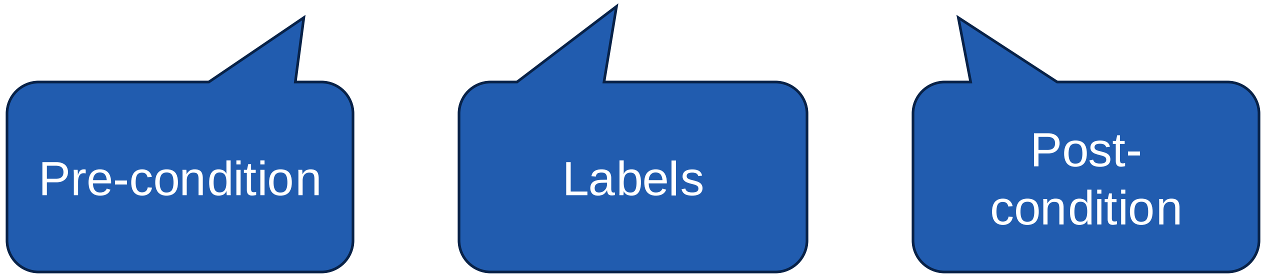
Trace

```
Start()
Start()
RedAdded()
YellowAdded()
```

Multiset Rewriting Rules

- You write a **model**
- The model admits a set of **traces**
- A **multiset-rewriting rule** defines a state transition
  - Multiset-rewriting rules use multisets of **facts**
  - Pre-condition: Which state is required to apply the rule?
  - Labels: For reference in properties
  - Post-condition: Which state is added when the rule is applied?

```
rule NAME:
  [ ... ] --[ ... ]-> [ ... ]
```

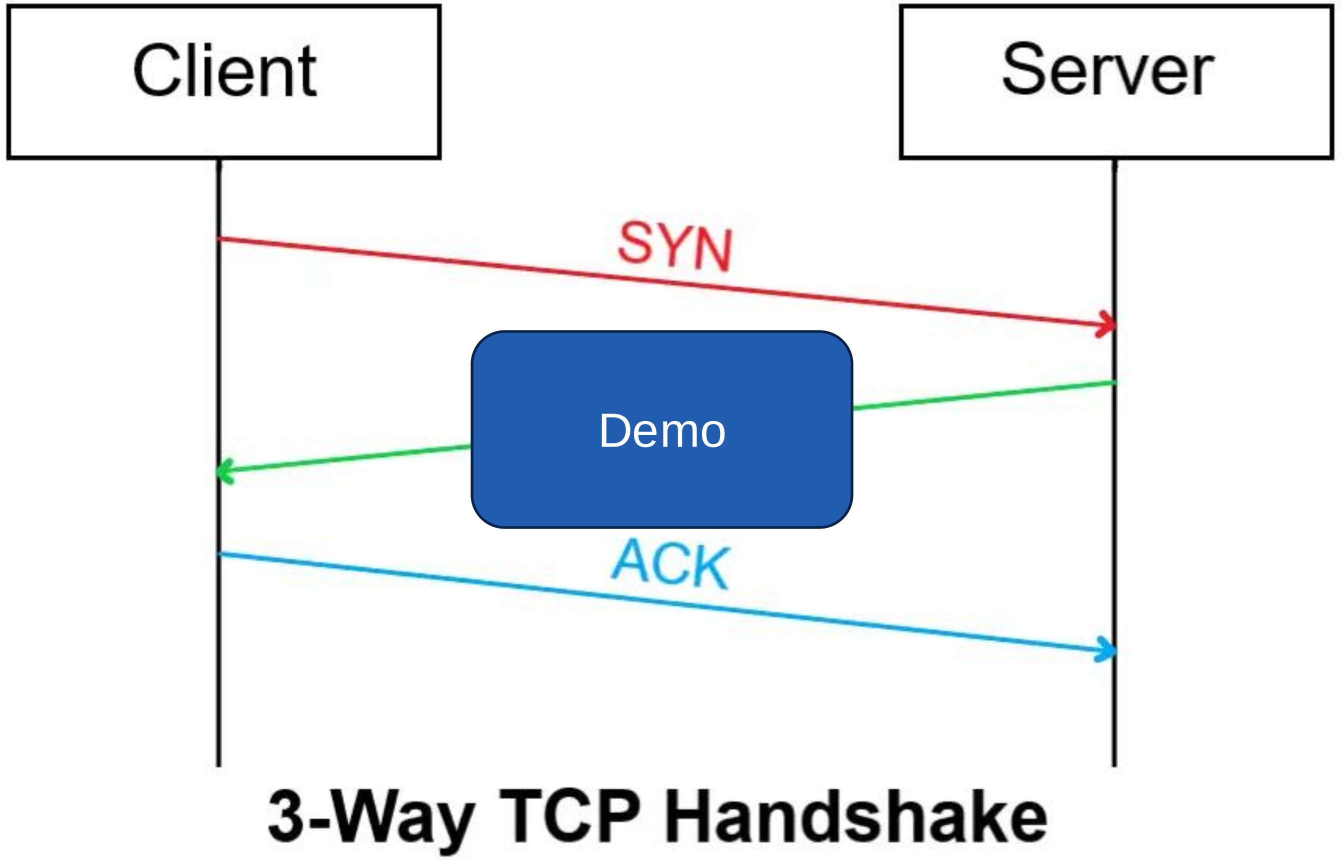


Values in Tamarin

- Facts can have **parameters**, which are **terms**
  - `Out(x)`
- Terms can be:
  - Constants: `‘g’`
  - Unguessable (fresh) values: `~k`
  - Public values: `$P`
  - Function application: `f(t1, t2)`
- A variable `x` can be any term (also called **message**)
- **Equational theory** defines semantics of functions

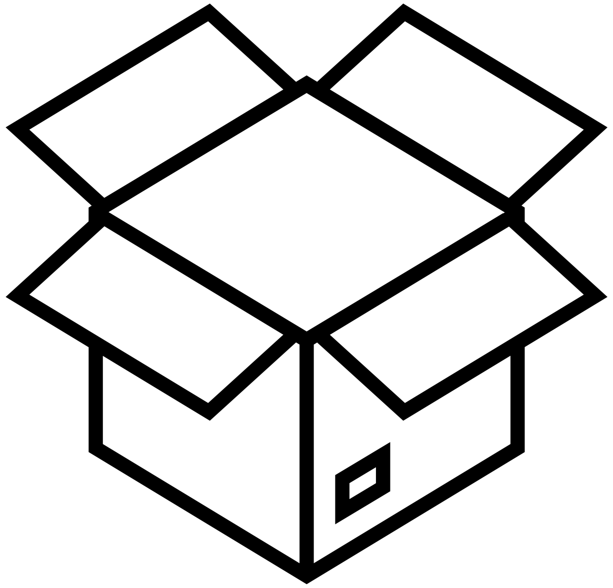
```
functions: sign/2, verify/3, pk/1, true/0
equations: verify(sign(m, sk), m, pk(sk)) = true
```

Example: TCP



Example: TCP – What happens under the hood?

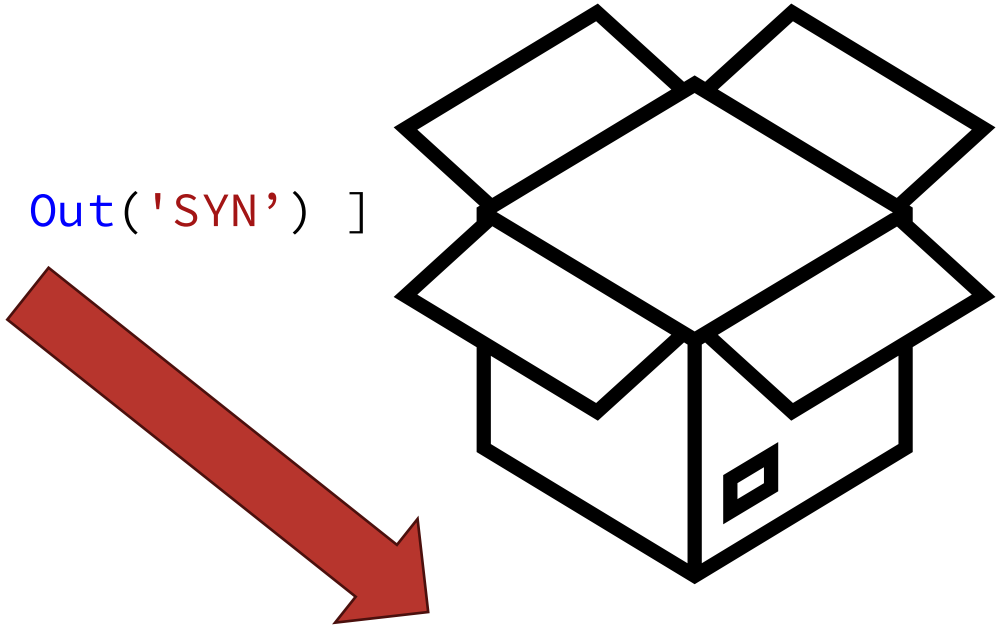
```
rule SYN:
  []
  --[ Begin() ]->
  [ St_AliceWait(), Out('SYN’) ]
```



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

Example: TCP – What happens under the hood?

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



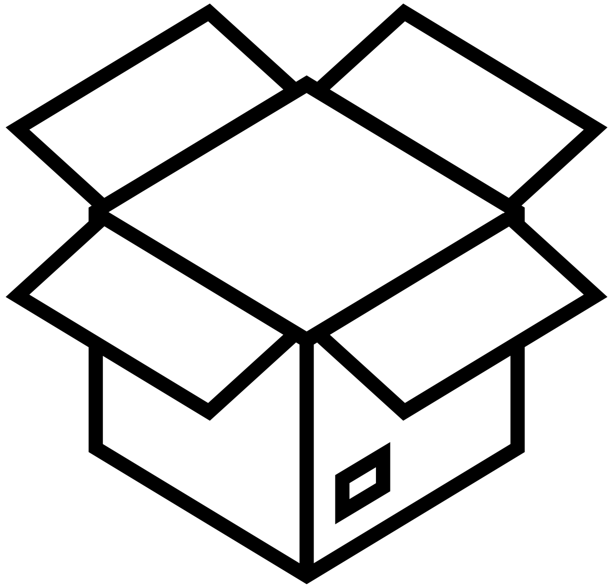
```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

```
St_AliceWait()

Out('SYN')
```

Example: TCP – What happens under the hood?

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



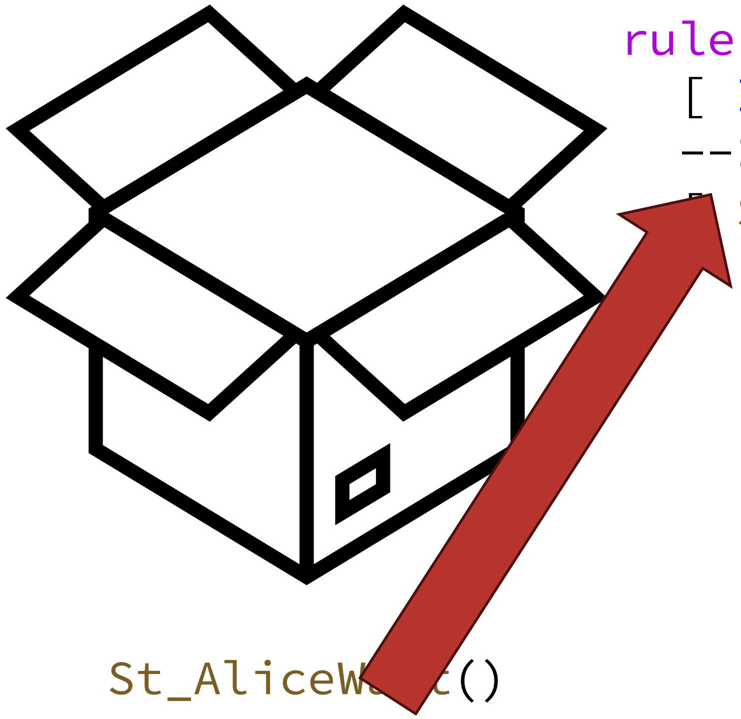
```
rule SYNACK:
  [ In('SYN') ]
  -->
  [ St_BobWait(), Out('SYNACK') ]
```

St_AliceWait()

In('SYN')

Example: TCP – What happens under the hood?

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



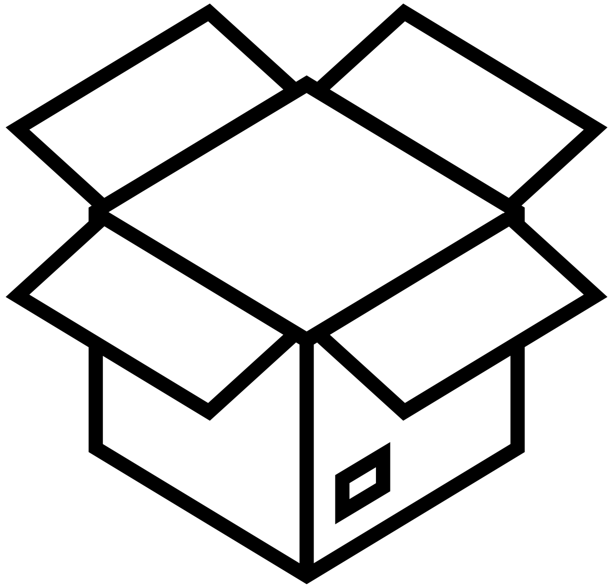
```
St_AliceWait()

In('SYN')
```

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

Example: TCP – What happens under the hood?

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



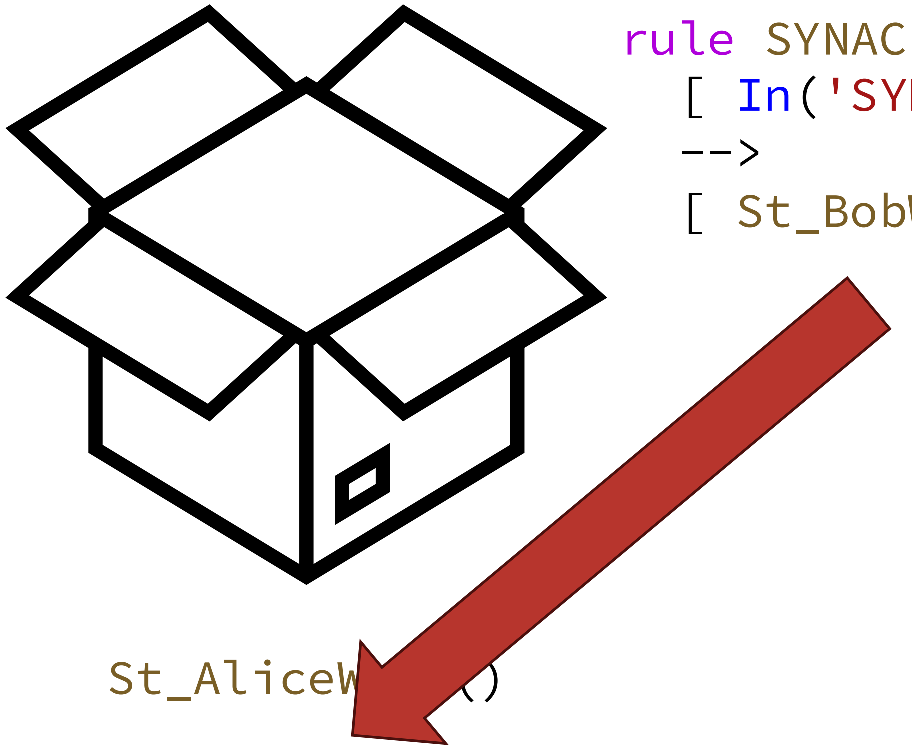
St_AliceWait()

```
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')

Recap and Exercises

State read

Message in

State write

Message out

Unification

```
rul[illegible]:
  [ St_X0(...), In(term1) ]
--[ Begin() ]->
  [ St_X1(...), Out(term2) ]


functions: function/1


rule Memorize:
  [] --> [ Fact(function('x')) ]

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

1. Go to github.com/felixlinker/tamarin-workshop/
2. Clone or download
3. Install Tamarin
4. Do exercises 1+2 (there is a syntax cheatsheet)

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 messages*
  - You want certain things to a
    - *Merchant receives paym*
- Protocols are complex!
- People make mistakes!

Tamarin proof = thing is secure

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

What is Tamarin?

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

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

Specifications vs Formal Analysis

Specification

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

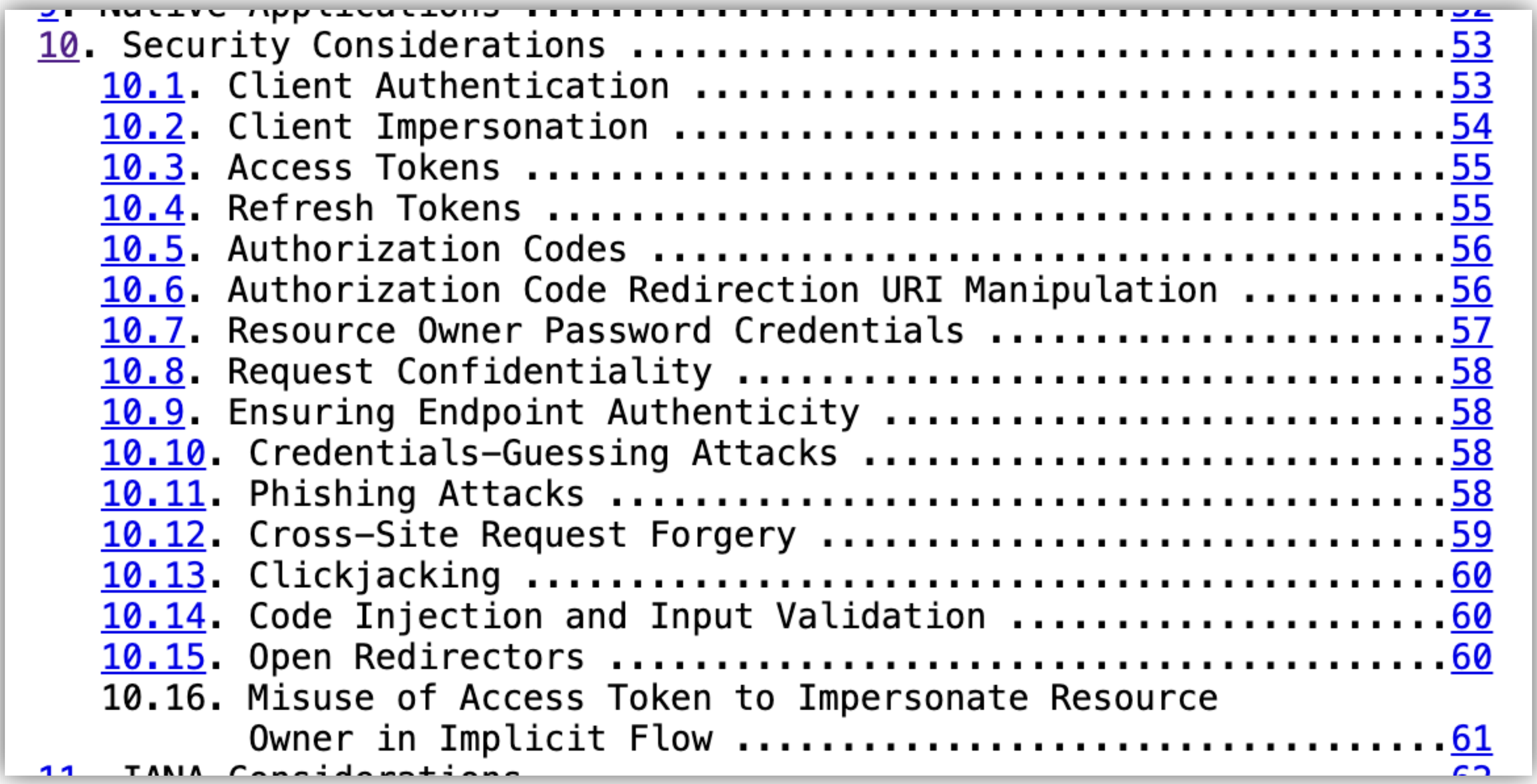
We did X so attack Y is not possible

Formal Analysis

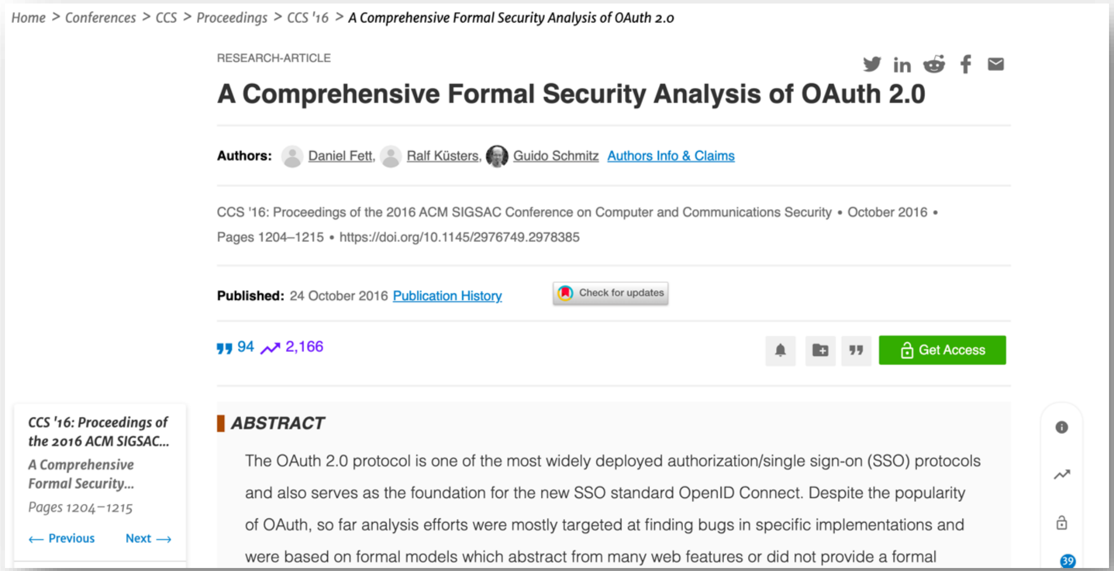
- A structured way to approach security
  - A positive definition of security properties
  - A list of explicit assumptions

Ideally…

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

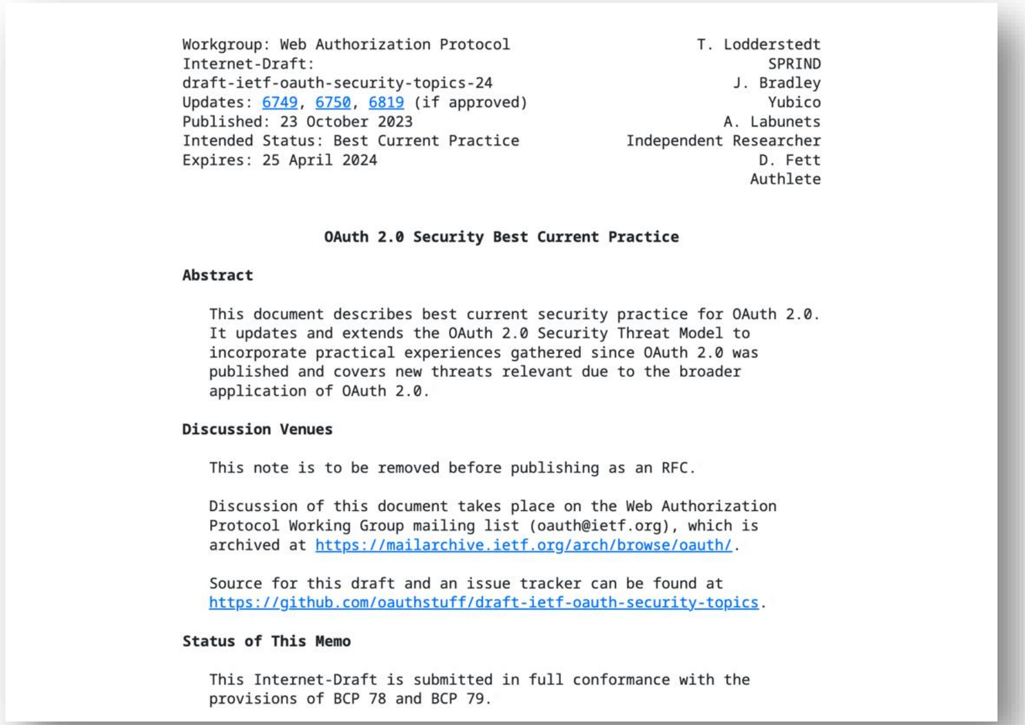
94 2,166

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

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

Check for updates
Get Access
← Previous
Next →


```
Workgroup: Web Authorization Protocol                    T. Lodderstedt
Internet-Draft:                                                 SPRIND
draft-ietf-oauth-security-topics-24                         J. Bradley
Updates: 6749, 6750, 6819 (if approved)                          Yubico
Published: 23 October 2023                                 A. Labunets
Intended Status: Best Current Practice          Independent Researcher
Expires: 25 April 2024                                         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.

Fett, Küsters, Schmitz. CCS'16.

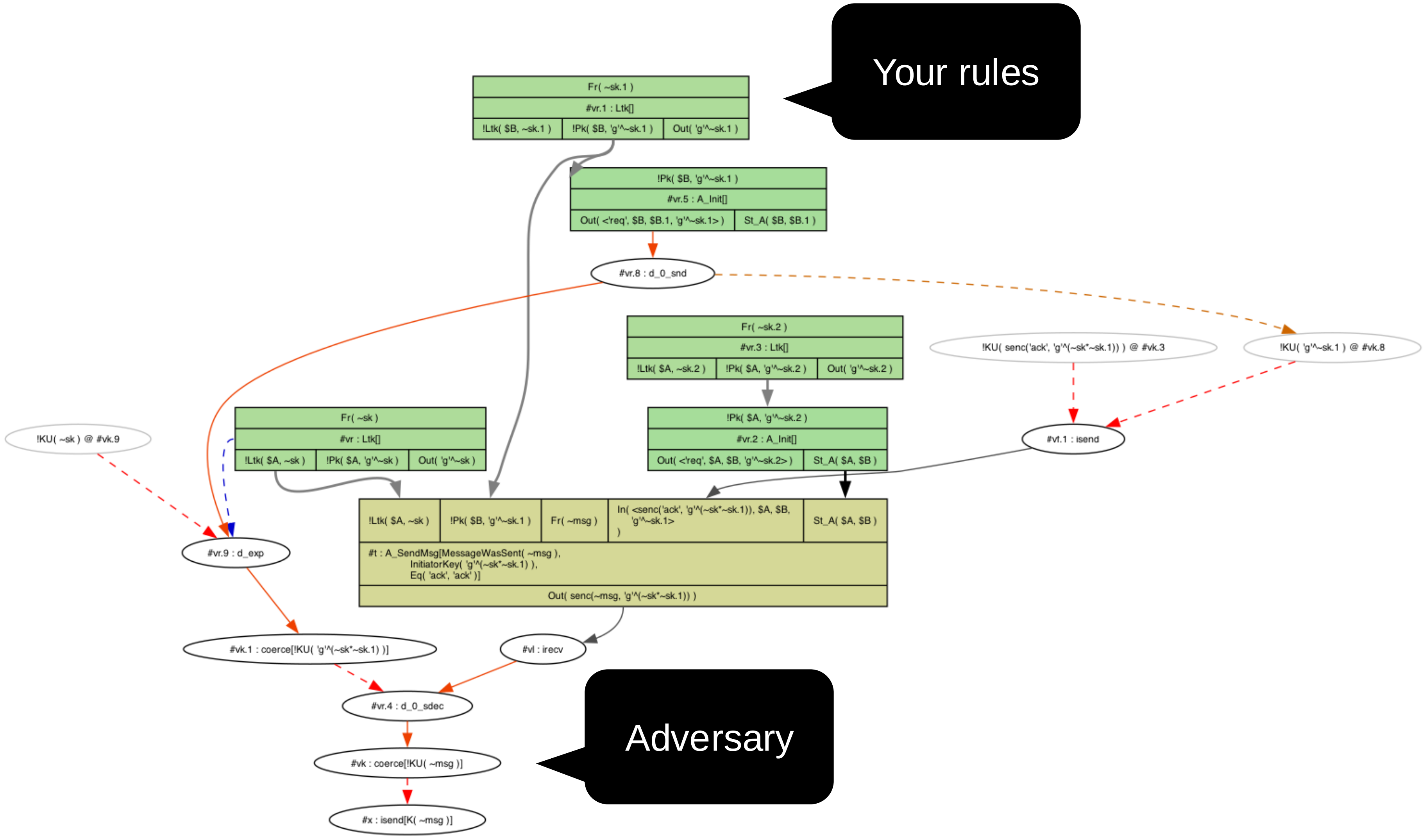
- But: Also doesn't list desired properties

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

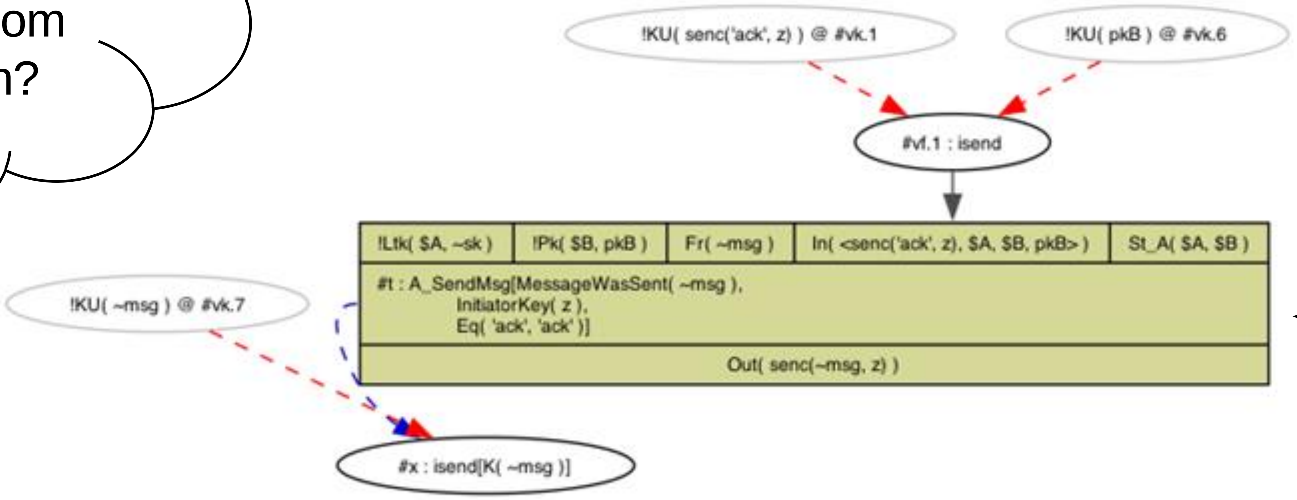
But how analyze a specification?



But how analyze a specification?

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

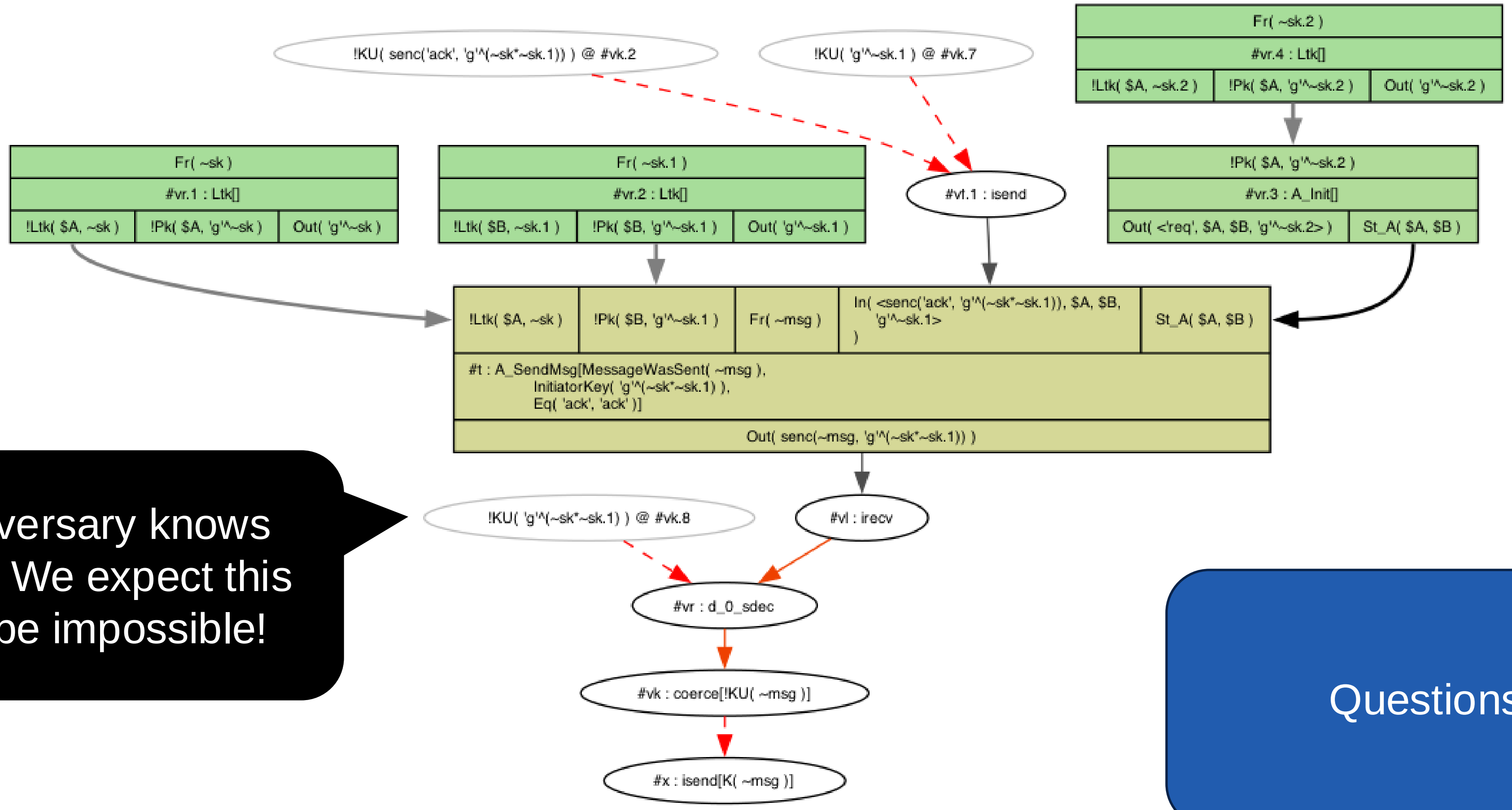
Where does the model deviate from our expectation?



Message was sent…

…and adversary knows it

But how analyze a specification?

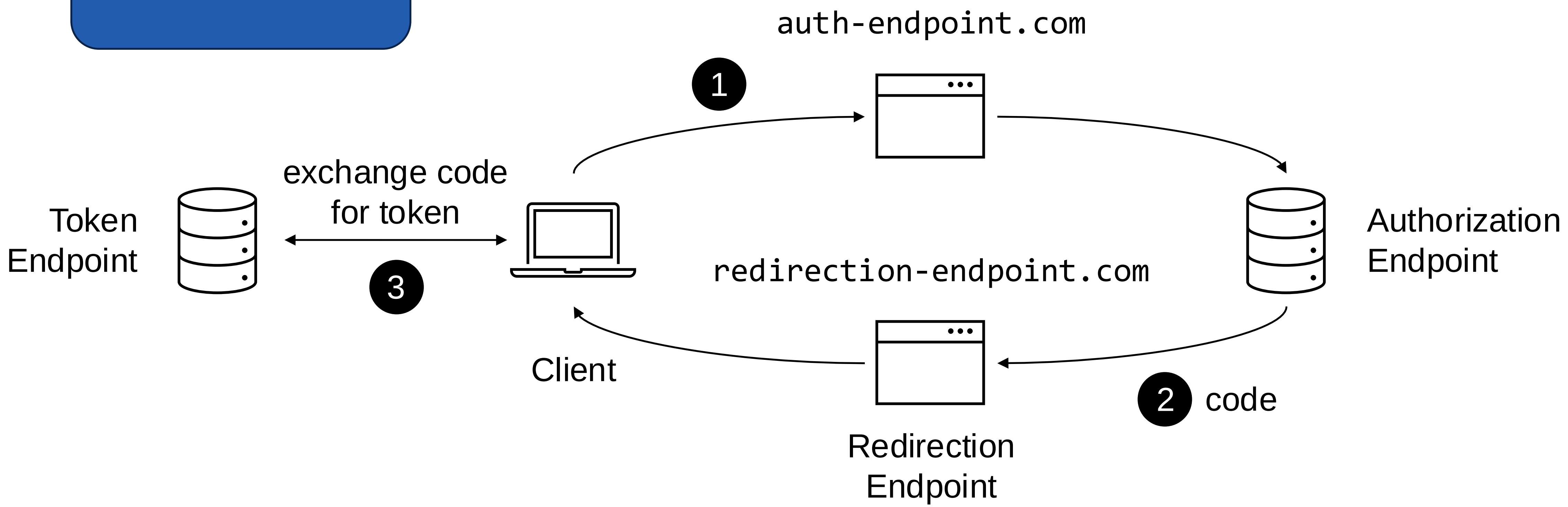


Adversary knows key! We expect this to be impossible!

Questions?

Case Study: OAuth 2.0 – Authorization Code Flow

Let's do it!



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