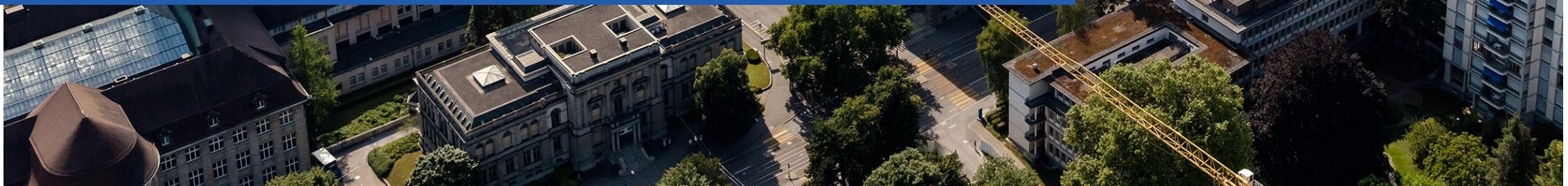




Tamarin Workshop Automated Protocol Verification

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

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 that first 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 unauthenticated off-card 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 strict standard. 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 underlying attack is that the card's response to a PIN verification request is not authen-

Some of the security issues identified in the paper include:

- 1) *Bank accepts off-card action*: The paper focuses on the EMV protocol design. We model for the symbolic analysis tool that has been used to verify the standard. The tool includes TLS 1.3 [14] and supports protocol verification and unbounded verifications.
- 2) *Authentication to the terminal*: The paper shows that the terminal is authenticated by the bank and authorized online.
- 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

42nd IEEE Symposium on Security and Privacy

Best Practical Paper
Sponsored by Intel and IBM

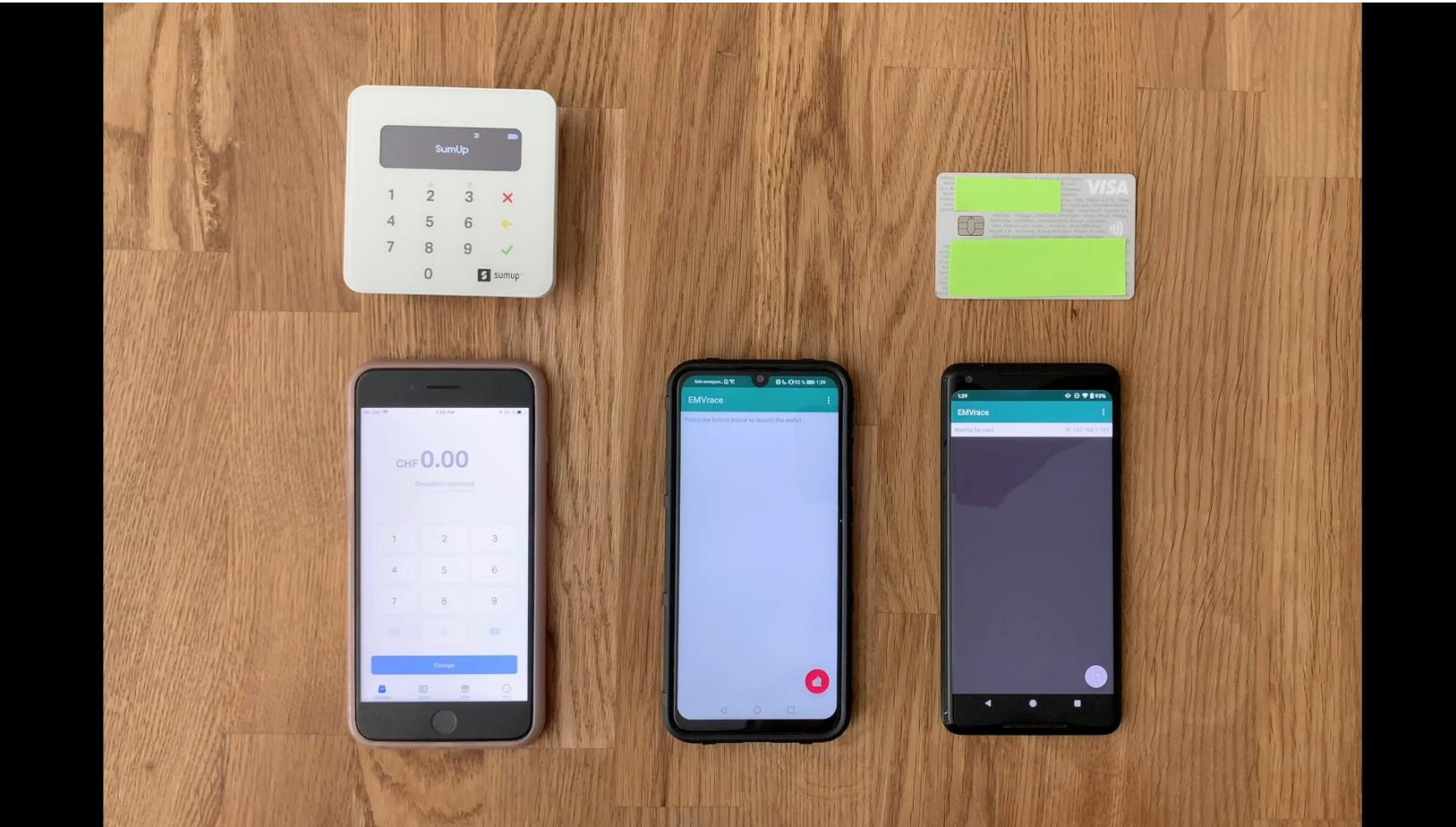
Presented to
David Basin, Ralf Sasse,
Jorge Toro-Pozo
for
The EMV Standard: Break, Fix, Verify
May 24–27, 2021

Alina Oprea
Alina Oprea, Program Chair SP 21

Alvaro Cardenas
Alvaro Cardenas, General Chair SP 21

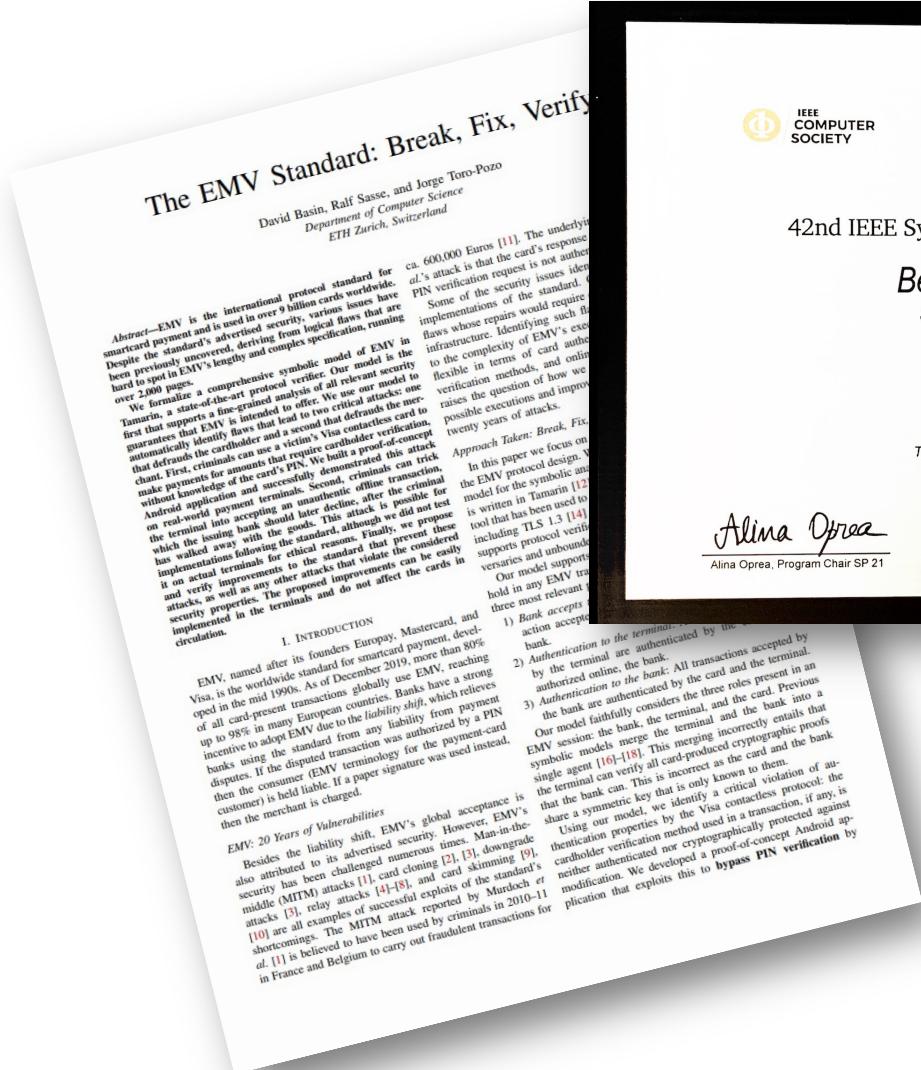
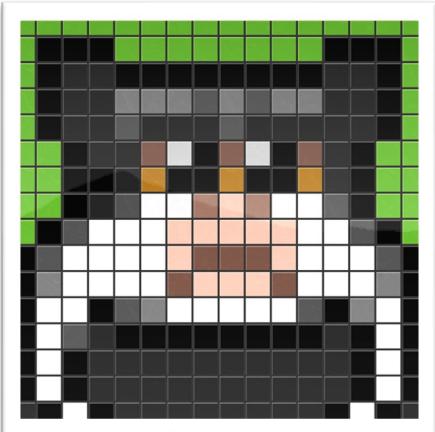
T. Holz
Thorsten Holz, Program Chair SP 21

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!



What is Tamarin?

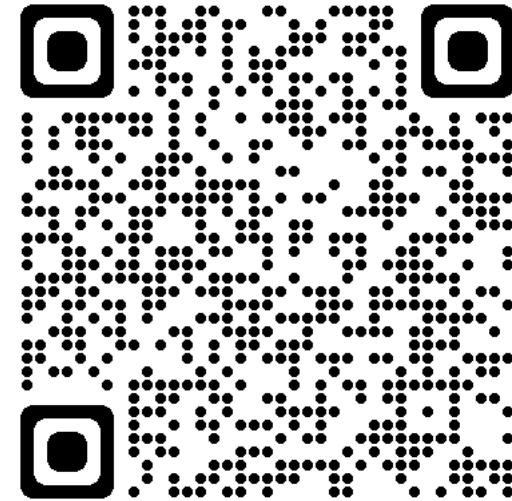
- Our world is powered by security-critical protocols
 - You want certain things to not happen
 - *NSA reads your WhatsApp messages*
 - You want certain things to always happen
 - *Merchant receives payment upon confirmation*
- Protocols are complex!
- People make mistakes!

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

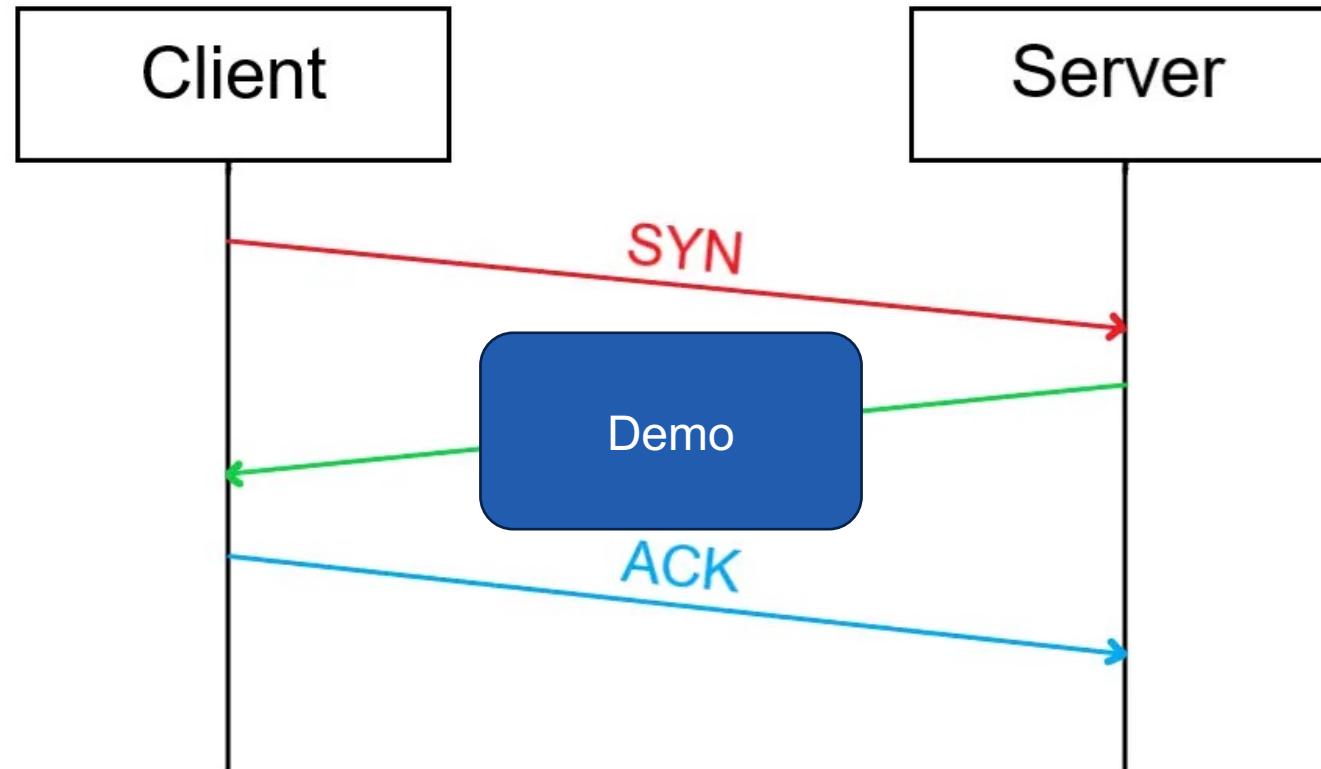
Workshop Goals

- 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



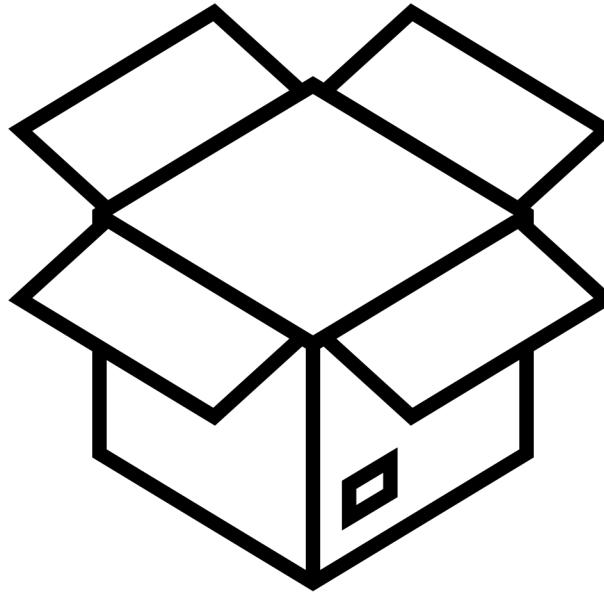
Example: TCP



3-Way TCP Handshake

Example: TCP – What happens under the hood?

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

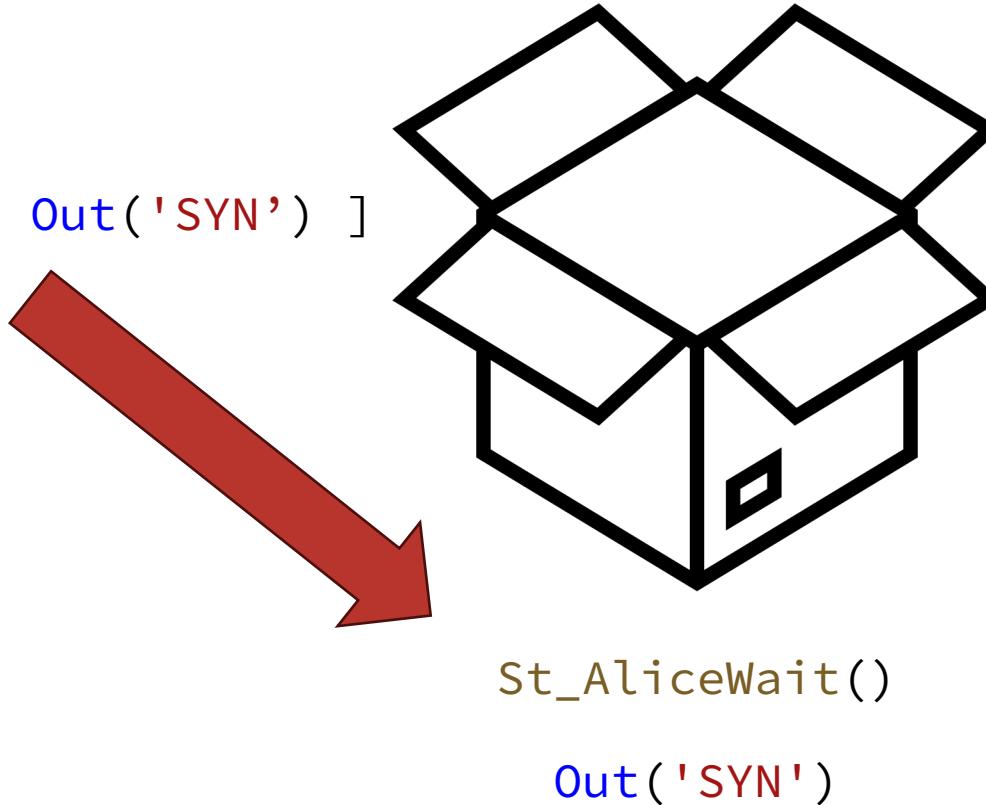


```
rule SYNACK:  
[ In('SYN') ]  
-->  
[ St_BobWait(), Out('SYNACK') ]
```

Example: TCP – What happens under the hood?

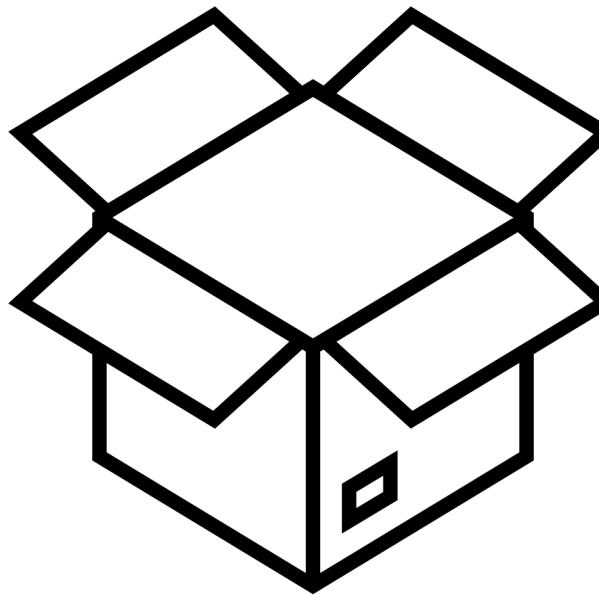
```
rule SYN:  
[]  
--[ Begin() ]->  
[ St_AliceWait(), Out('SYN') ]
```

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



Example: TCP – What happens under the hood?

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



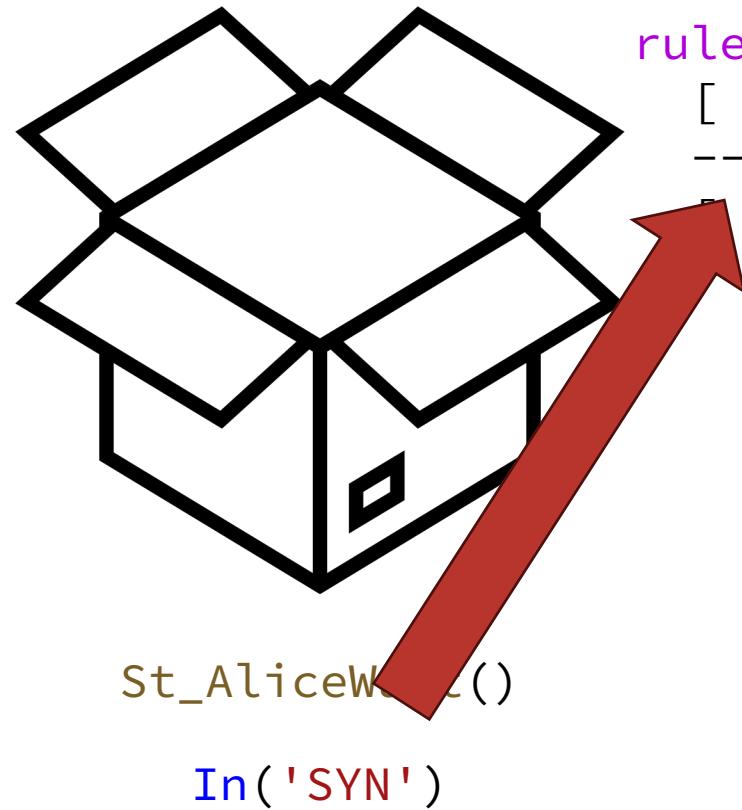
St_AliceWait()

In('SYN')

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

Example: TCP – What happens under the hood?

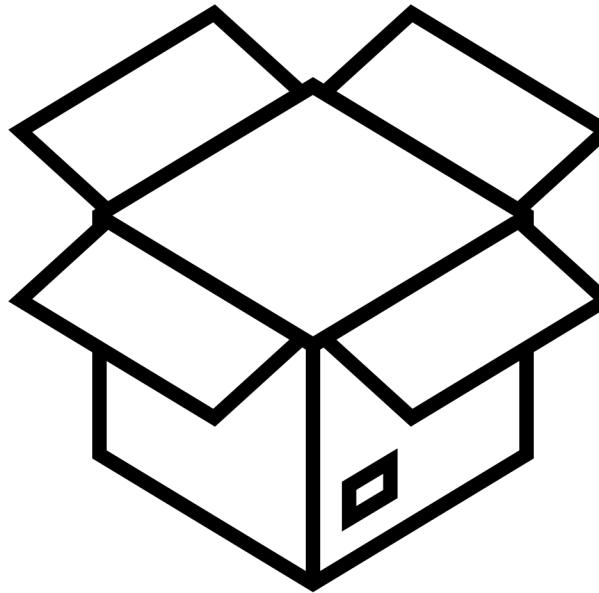
```
rule SYN:  
[]  
--[ Begin() ]->  
[ St_AliceWait(), Out('SYN') ]
```



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

Example: TCP – What happens under the hood?

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



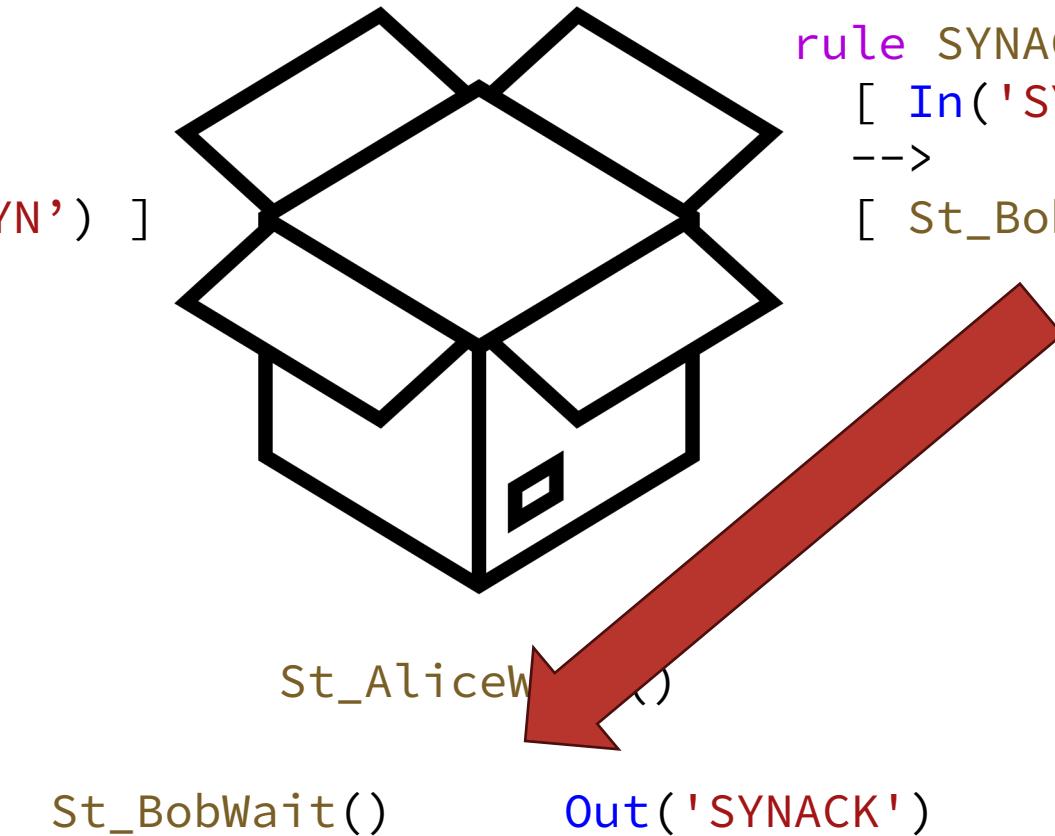
St_AliceWait()

```
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') ]
```



Values in Tamarin

- Values can be:
 - Constants: ‘constant’
 - Unguessable (fresh) values: $\text{\~{}k}$
 - Public values: $\$P$
 - Function application: $f(t_1, t_2)$
- 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

The diagram illustrates a state transition process. It starts with a blue rounded rectangle labeled "State read". An arrow points from "State read" to a blue rounded rectangle labeled "Message in". Another arrow points from "Message in" to a blue rounded rectangle labeled "State write". A final arrow points from "State write" back to "State read", completing the cycle.

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

The diagram illustrates two rule definitions and their execution paths:

- rule Me**:
 - Body: Fact
 - Body: FunctionBoth paths lead to the same result:
[MemorizeSomething(f('x'))]
- rule LookUpAndSend:**
 - Body: MemorizeSomething(v)
 - Path: --> [Out(v)]A blue arrow labeled "Pattern-match" points from the "Out(v)" step to the "LookUpAndSend" rule.

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 happen
 - *Merchant receives payment*
- 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 messages*
 - *Merchant receives payment before shipping*
- 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!

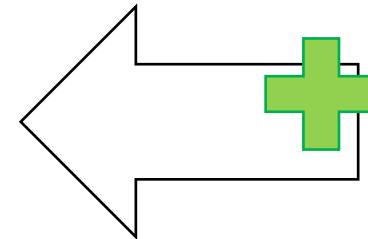
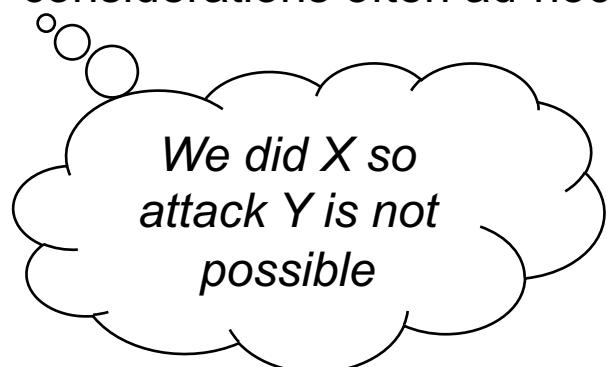
With Tamarin, you can prove that a protocol (model) guarantees certain security properties under certain assumptions

Specifications vs Formal Analysis



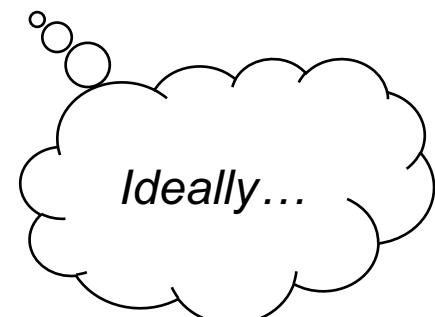
Specification

- 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

10. Native Applications	52
10. Security Considerations	53
10.1. Client Authentication	53
10.2. Client Impersonation	54
10.3. Access Tokens	55
10.4. Refresh Tokens	55
10.5. Authorization Codes	56
10.6. Authorization Code Redirection URI Manipulation	56
10.7. Resource Owner Password Credentials	57
10.8. Request Confidentiality	58
10.9. Ensuring Endpoint Authenticity	58
10.10. Credentials-Guessing Attacks	58
10.11. Phishing Attacks	58
10.12. Cross-Site Request Forgery	59
10.13. Clickjacking	60
10.14. Code Injection and Input Validation	60
10.15. Open Redirectors	60
10.16. Misuse of Access Token to Impersonate Resource Owner in Implicit Flow	61
11. TANA Considerations	62

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

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

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.

Fett, Küsters, Schmitz. CCS'16.

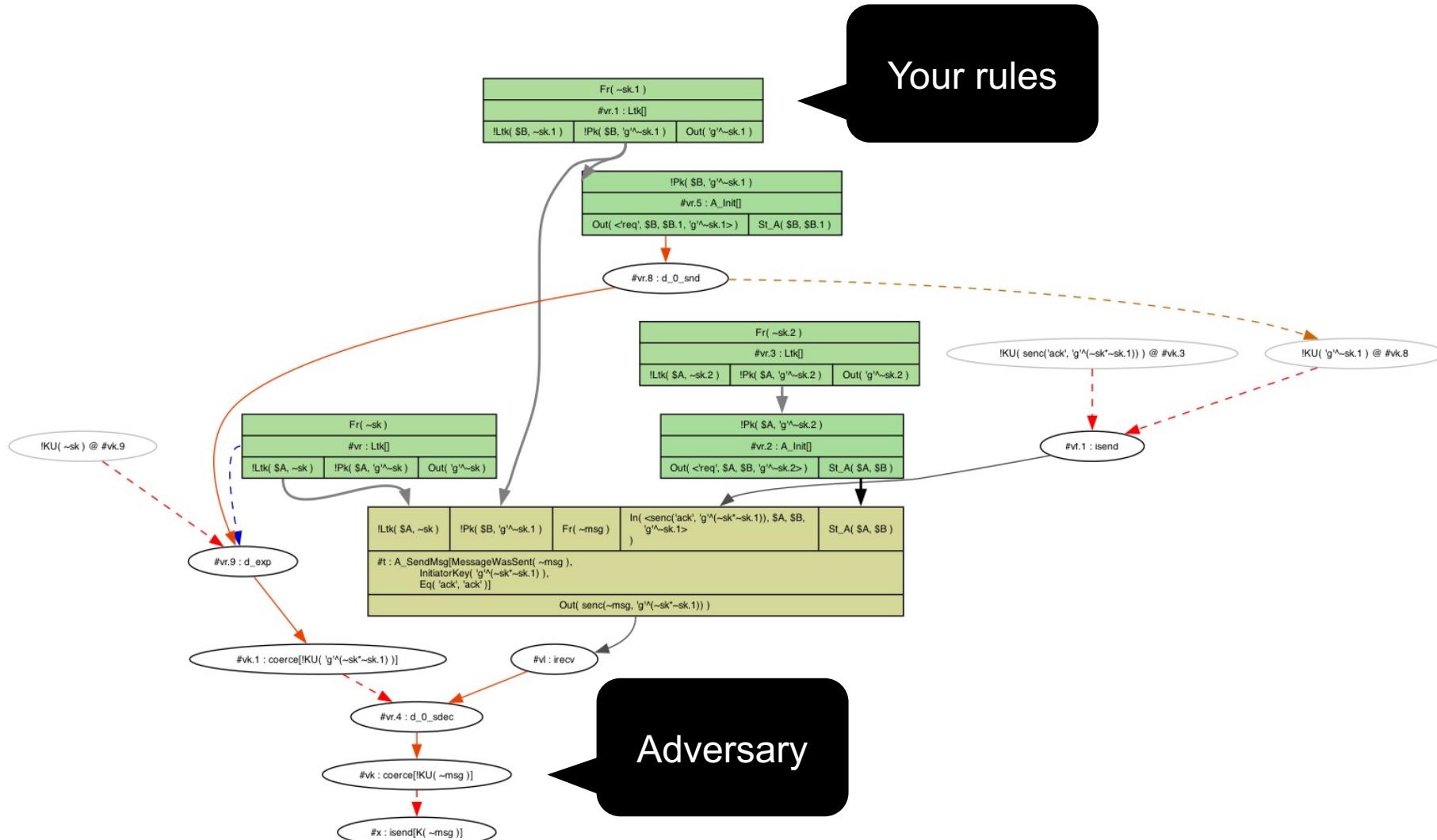
- But: Also doesn't list desired properties

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

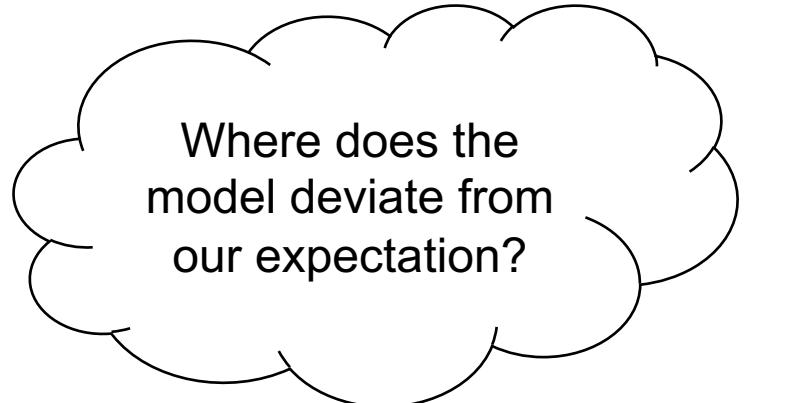
1. Implement an initial specification
2. Model security properties
 - It's okay if they are trivially true
3. Make your model more realistic
 - Now the properties are hopefully false
4. Refine everything
 - Let your understanding guide you
 - Let Tamarin tell you why your understanding is wrong

Use the GUI

But how analyze a specification?

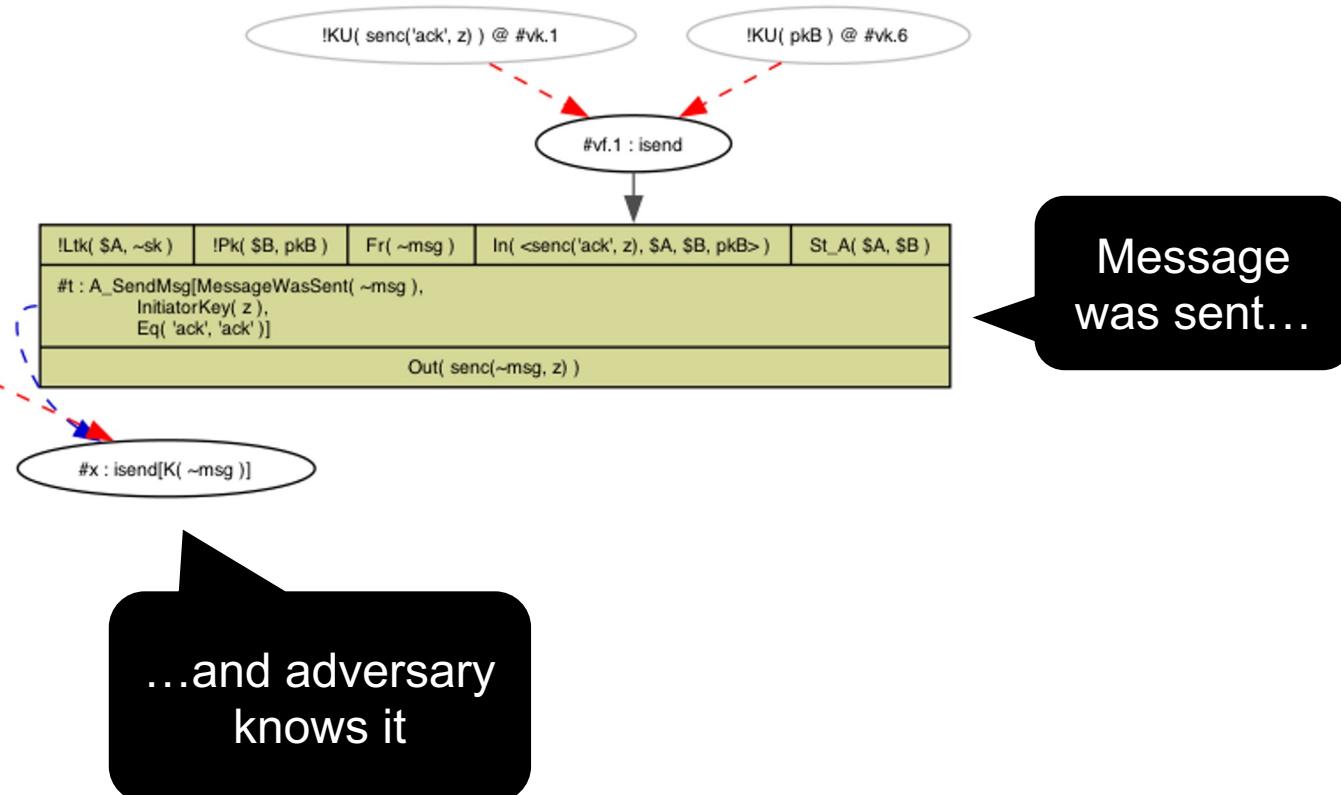


But how analyze a specification?

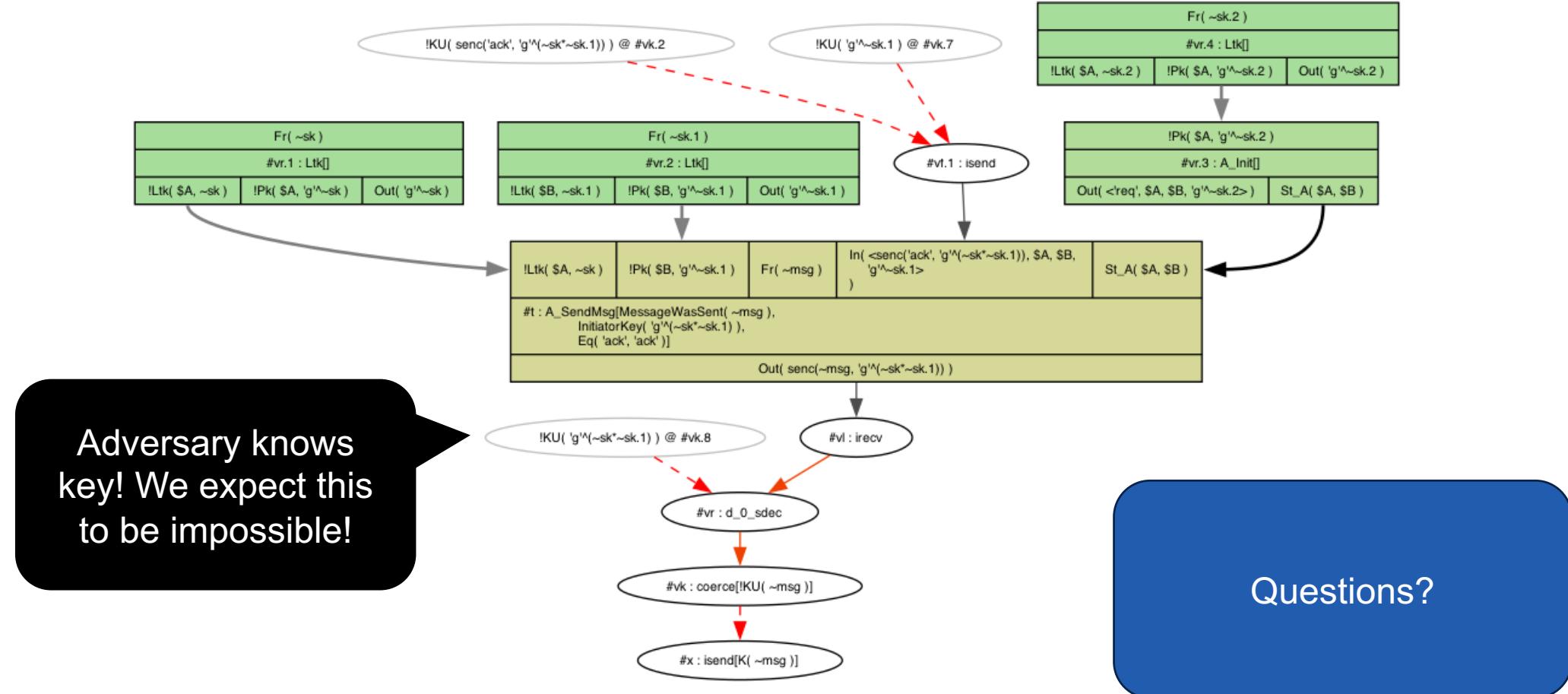


lemma SecrecyMessage:

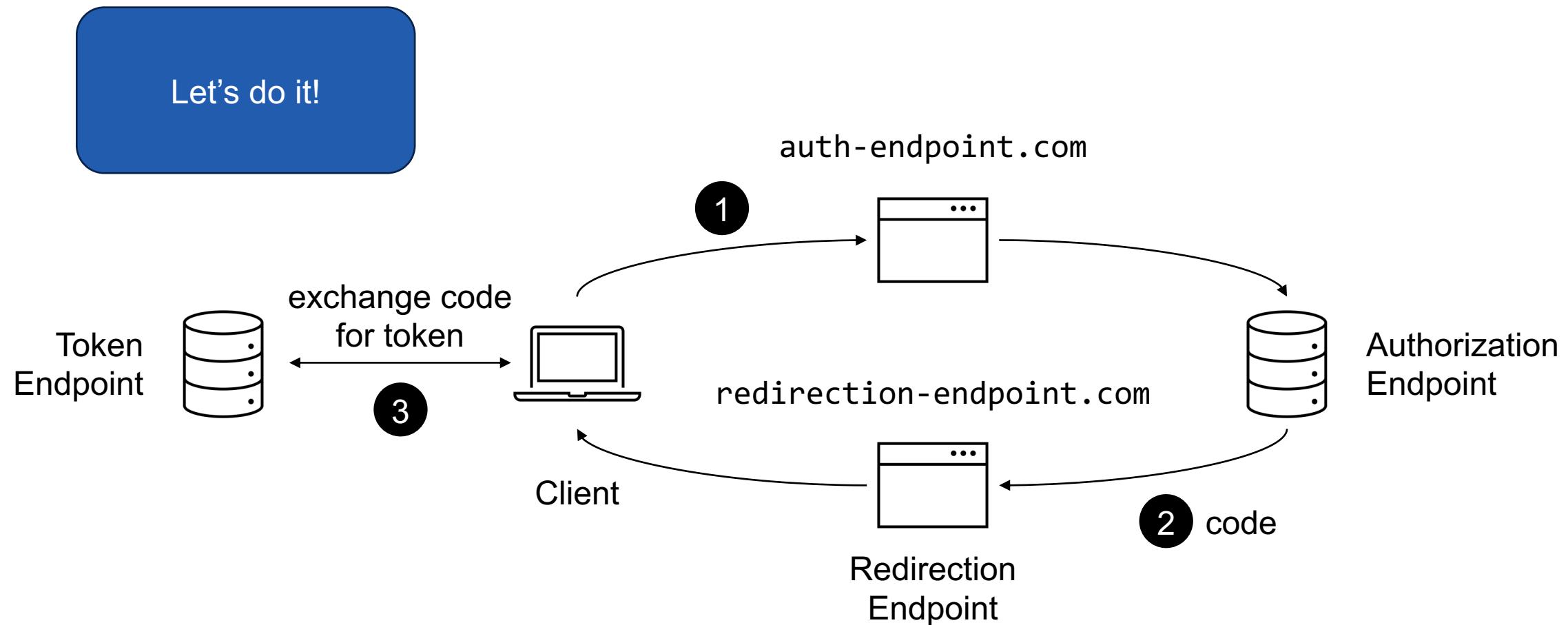
"**All** $m \ #t.$ **MessageWasSent**(m) @ $\#t$
==> **not** **Ex** $\#x.$ **K**(m) @ $\#x$ "



But how analyze a specification?



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.2017.84.

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>