Using Formal Tools To Design Secure Protocols

OSW 2019

Luca Arnaboldi & Hannes Tschofenig

Email: l.arnaboldi@ncl.ac.uk

Newcastle University, Arm Ltd

School Of Computing

March 20th, 2019

Before We begin

Newcastle University
School of Computing

Tutorial Materials Available here:

http://homepages.cs.ncl.ac.uk/l.arnaboldi/OSW2019Tutorial/tutorial.html

Protocols Are Constantly Broken

Newcastle University
School of Computing

EternalBlue (WannaCry) – SMB Protocol

- Thousands of NSH PC's
- And more

KRACK Attacks

- Breaking WPA2
- Potentially millions vulnerable

5G (CCS2018)

- Issues in authentication phase (not being fixed)

Are we just finding attacks?

Newcastle University
School of Computing

NO!

- Formal Verification is much more than breaking a protocol
- Can be a useful design tool
- Has been successfully used in other scenarios (e.g. TLS1.3)

Protocol Security

Newcastle University
School of Computing

Problem

How do we know if a protocol is secure?

- Smart people stare at it?

More structured approach

- Threat model & intended properties
  - Stare at protocol to find attack
  - Write proof of attack?
  - Alter the protocol and stare again

Can formal methods help make this simpler?

- Model checking and verification of the protocol

Solution

Idea: Encode protocol as a math formula

- e.g. to send a message from A to B this condition must hold
- Define the participants
- Have a standardized attacker model

- Encode the properties as conditions that must hold
- Automatically check all system traces and **ANYTHING*** that can go wrong
  - (limiting the staring to a minimum)

*Almost anything (see later)

Why use automated formal methods?

Newcastle University
School of Computing

- So far we have established that formal methods are a pretty neat idea (I hope)
- But why use tools instead of pen and paper?

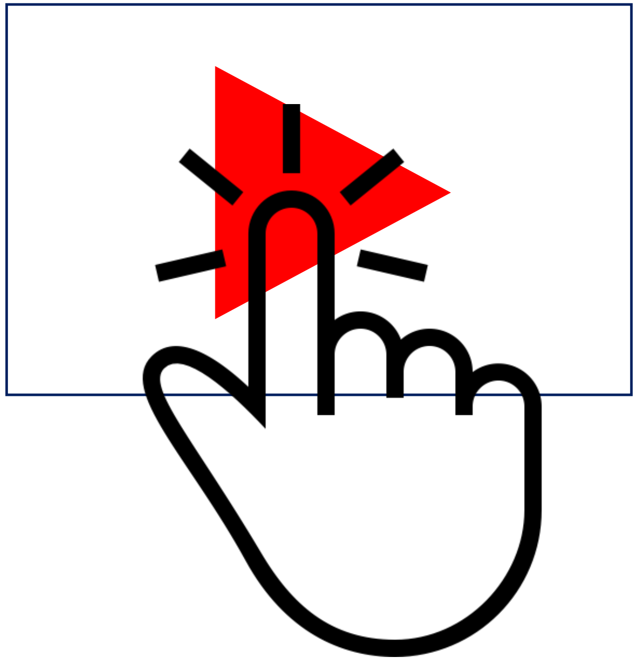
Is the proof in the next slide correct?

Proposition 2.1 Suppose there is a probabilistic algorithm AL with time bound VLI which takes for input a public key v and withstands, with probability E > 2-'+' * the identification test for a straight exam. Then the discrete logarithm of v can be computed in time O(IALI/&) and constant, positive probability.

Proof. This is similar to Theorem 5 in Feige, Fiat, Shamir (1987). The following algorithm AL' computes log,v. 1. Repeat the following steps at most 1/& times: generate x the same way as does algorithm AL, pick a random e' in (0, ..., 2 -1) and check whether AL passes the identification test for (x,e'); if AL succeeds then fix x and go to 2. 2. Probe 1/~ random numbers en in (0,...,2t-1) . If algorithm AL passes the identification test for some en that is distinct from e' then go to 3 and otherwise stop. 3. Choose the numbers y', y" which AL submits to the identification test in response to e', e". (y'-y" is the discrete logarithm of v"-~' (mod P).) 4. Output (y'-y")/(e"-e') (mod q) . t
242 We bound from below the success probability of this algorithm. The algorithm finds in step 1 a passing pair (x,e') with probability at least i. With probability at least a, the x chosen in step 1, has the property that AL withstands the identification test for at least a '$ s-fraction of all e E (0, ..., 2 -1). For such an x step 2 finds a passing number en that is distinct from e' with probability at least 1 - (l-s/2)'/" > 1 - 2.7-'/* > 0.3 . This shows that the success probability of the algorithm is at least 0.3/4.

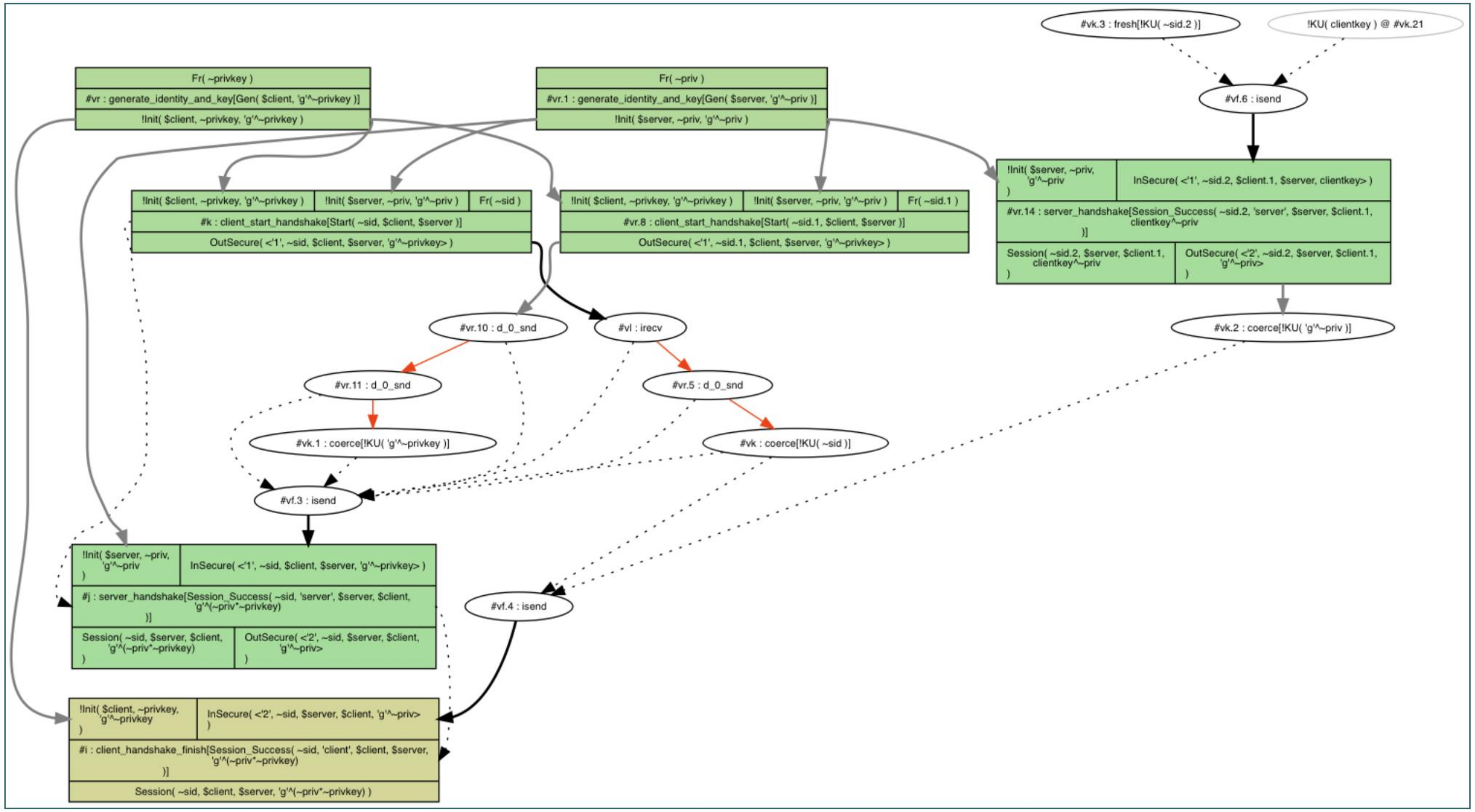
Schnorr, Claus-Peter. "Efficient signature generation by smart cards." *Journal of cryptology* 4.3 (1991): 161-174.

Alternatively…..



Correct

OR

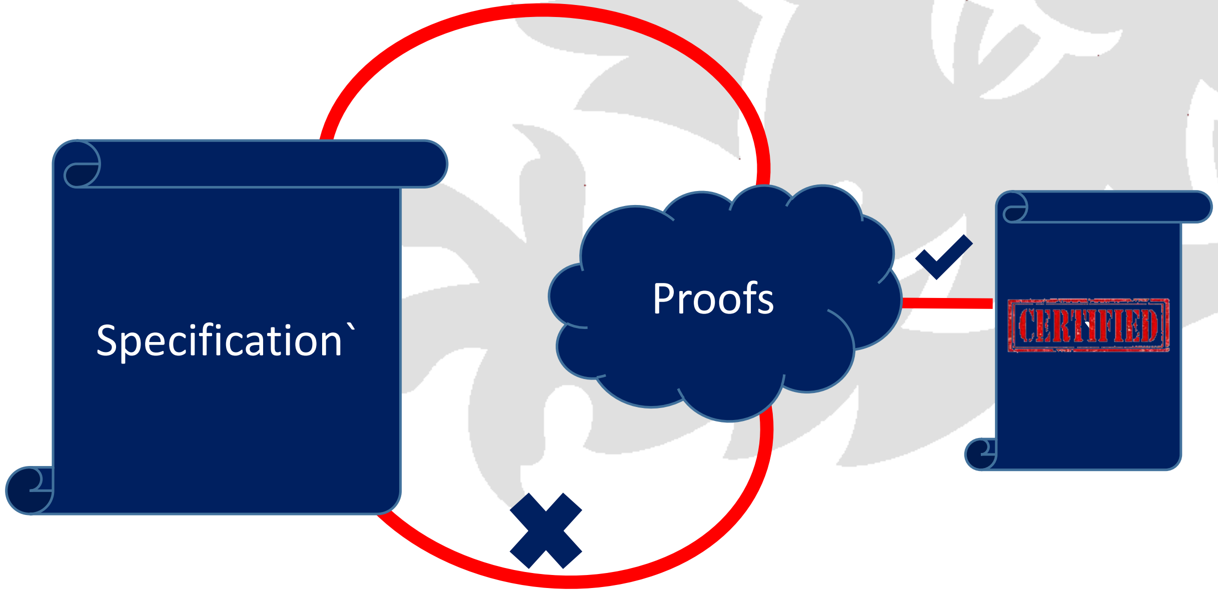


Easy to read attack trace

Automated Formal Tools

Newcastle University
School of Computing

- **They give a guarantee and assurance**
- Unlike testing it covers every single scenario
- Visualises attacks
- Interact with the protocol
- Allows rapid prototyping



Formal Protocol Verification

Newcastle University
School of Computing

Types of Protocol Models

	Symbolic model (Needham and Schroeder & Dolev and Yao)	Computational Model (Goldwasser, Micali, Rivest, Yao et al)
Messages	Literals	BitStrings
Adversary	Rules (e.g. Dolev-Yao)	All Probabilistic Polynomial Timed Adversary
Crypto Primitives	Blackbox	Complete Math Theory
Reasoning	Concurrent Process	Security Theorems (Probability + Complexity)

Different Modelling Options

Symbolic model

Pros

- Quick prototyping
- Immediate feedback
- Intuitive
- Replaces a wordy specification
- Finds attacks
- Low cost
- Low Effort
- High Reward

Cons

- Restricted expressiveness
- Hard to extend attacker capabilities
- Only looks at specification details
- Cannot look at implementation issues

Computational Model

Pros

- A protocol proven secure in the computational model is unattackable
- Can translate easily into code
- Very rigorous
- Can verify algorithms, types and almost any mathematical structure

Cons

- Slow
- Unintuitive
- Does not replace a specification
- Steep learning curve
- Not very human readable
- Doesn't find attacks**

Formal Protocol Verification

Newcastle University
School of Computing

Symbolic model

Tamarin

- Specified as multiset rewriting systems
- Built on Maude tool
- Generates protocol as set of traces (attacks and benign)
- Temporal logic for proofs
- Interactive

ProVerif

- Subset of Pi-calculus
- It uses over approximation techniques
- If a property cannot be proven, it reconstructs an attack's trace
- Fully automated

Computational Model

EasyCrypt

- Proofs by sequences of games
- Improved automation
- Uses SMT solvers
- Probabilistic Hoare logic pHL for verification
- Interactive (guided proofs)

CryptHOL

- Written as HOL (not algorithmically)
- Well established platform
- Small trusted code base
- Uses SMT's
- Game based proofs

Formal Protocol Verification

Newcastle University
School of Computing

Symbolic model

Tamarin

- Specified as multiset rewriting systems
- Built on Maude tool
- Generates protocol as set of traces (attacks and benign)
- Temporal logic for proofs
- Interactive

ProVerif

- Subset of Pi-calculus
- It uses over approximation techniques
- If a property cannot be proven, it reconstructs an attack's trace
- Fully automated

Computational Model

EasyCrypt

- Proofs by sequences of games
- Improved automation
- Uses SMT solvers
- Probabilistic Hoare logic pHL for verification
- Interactive (guided proofs)

CryptHOL

- Written as HOL (not algorithmically)
- Well established platform
- Small trusted code base
- Uses SMT's
- Game based proofs

Automatic Protocol Verification Using Tamarin

- Very intuitive to use and to get started
  - Security analysis from get go (before full protocol)
- Visual and easy to understand feedback loop
- Modular design
- Open source and active community
- Used in real world! (TLS 1.3, 5G key exchange)
- Quicker to design than computational models
- **Finds errors & Attacks!**

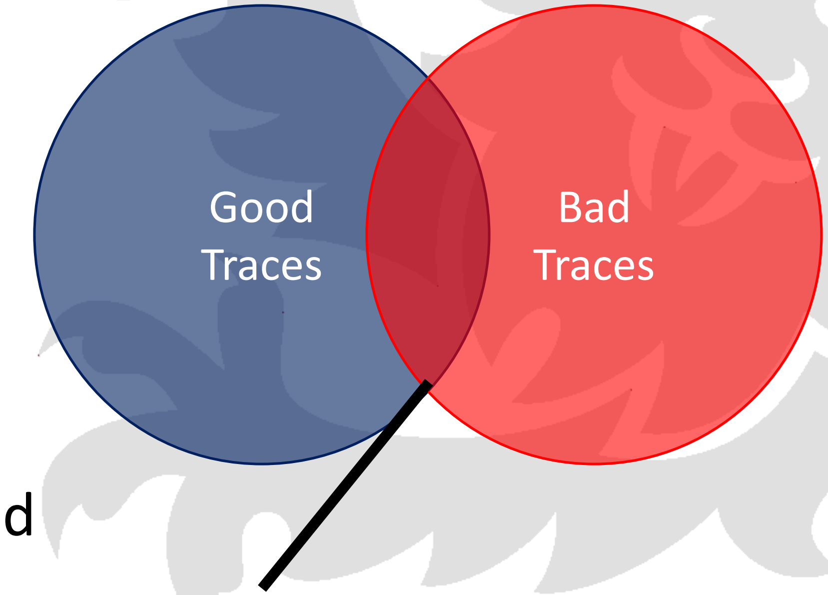
It's adorable!

TAMARIN

Tamarin Overview

Newcastle University
School of Computing

- Multiset Rewriting rules for model and adversary
  - Imagine the protocol as a set of rules
  - The rules dictate what traces can take place
  - Rules generate a transition system
- Security Goals using first order logic
  - Special rules that dictate what a good trace is
- Automatically checks all traces
  - Proves all traces are good
  - Or, shows counterexample as to why they are bad



Empty means it's secure, otherwise contains attack

Tamarin Syntax - Basics

Newcastle University
School of Computing

- Protocol Contents
  - Terms (variables through the system)
    - Term ~x denotes a fresh(like a nonce)
    - Term $x denotes a publicly available value (like id of participant)
    - Term m denotes untyped messages
  - Facts (Conditions that alter the state)
    - Special Facts In(t), Out(t), Fr(t), K(t)
    - Attacker Facts: KU(t), Isend (t), Coerce(t) and common to add Reveal (t)
  - Actions (Keeps track of state but leaves it unmodified)
- State of the system is the combination of facts
- Rules dictate what facts will be in the state

Tamarin Syntax – Rules & Lemmas

Newcastle University
School of Computing

- Rules decide how the state can change: L – [A] -> R
  - L : Facts that get consumed
  - A : Keep track of terms
  - R : Save new facts to state
- A rule which is live can be executed at any point
  - Even by an adversary!
- We dictate a correct execution by a set of lemmas
- Proofs are done making use of Actions
  - Dummy Proof :

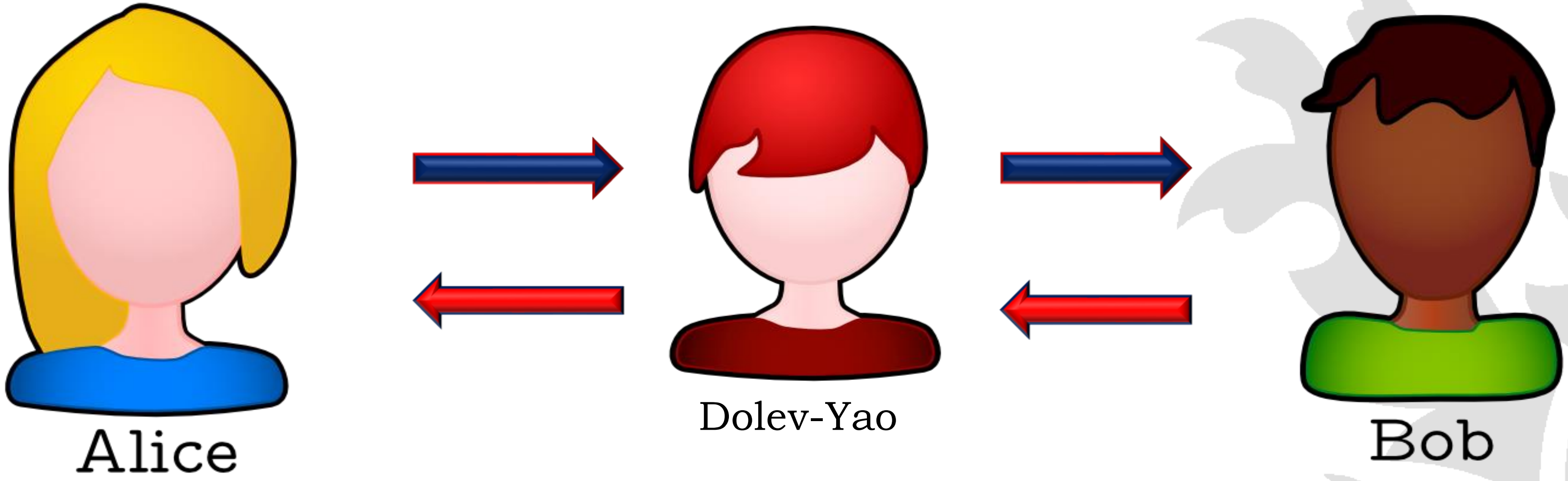
```
Lemma dummy:
    All x  #i.  A1(x) @ #i ==> Ex y #j.  A2(y)  @ #j
```

Extra Toppings

- Functions
  - Inbuilt: asymmetric encryption, symmetric encryption, hashing, signing, XOR, Diffie-Hellman
  - Defining our own functions e.g:
    - Asymmetric Encryption : adec(aenc(m,pk(secretKey),secretKey) = m
- Restrictions (axioms)
  - Set out logic that is true for the protocol
    - Single Proof of Possession Key :

```
restriction single_PoP_Key:
        "All #i #j. PoPSetup(  ) @ #i & PoPSetup(  )  @ #j ==> #i = #j"
```

Threat Model – Dolev-Yao



Channel **IS** the intruder.

Can *read*, *modify*, *fabricate* and *replay* all messages

Cannot encrypt/decrypt without the key

Security Properties

Newcastle University
School of Computing

Properties for Authentication protocols

- Aliveness
- Weak Agreement
- Non-injective Agreement
- Injective agreement

Lowe, Gavin. "A hierarchy of authentication specifications." *Computer security foundations workshop, 1997. Proceedings., 10th*. IEEE, 1997

Security Properties

Newcastle University
School of Computing

Properties for Authentication protocols

- Aliveness
- Weak Agreement
- Non-injective Agreement
- Injective agreement

Additionally (quite obviously):

- Secrecy

Lowe, Gavin. "A hierarchy of authentication specifications." *Computer security foundations workshop, 1997. Proceedings., 10th*. IEEE, 1997

Injective Agreement

Newcastle University
School of Computing

"A protocol guarantees to an initiator **A** injective-agreement with a responder **B** on a message **m** if, whenever **A** completes a run of the protocol , apparently with responder **B**, then **B** has previously been running the protocol, apparently with **A**, and **B** was acting as a responder in his run, and the two agents agreed on the full content of the message **m**, and each run of **A** corresponds to a *unique* run of **B**"

Injective Agreement

Newcastle University
School of Computing

"A protocol guarantees to an initiator **A** injective-agreement with a responder **B** on a message **m** if, whenever **A** completes a run of the protocol , apparently with responder **B**, then **B** has previously been running the protocol, apparently with **A**, and **B** was acting as a responder in his run, and the two agents agreed on the full content of the message **m**, and each run of **A** corresponds to a *unique* run of **B**"

To simplify, this often means that:

> *The communication between A and B is **authenticated, secure** and **replay protected***

Injective Agreement

School of Computing

"A protocol guarantees to an initiator **A** injective-agreement with a responder **B** on a message **m** if, whenever **A** completes a run of the protocol , apparently with responder **B**, then **B** has previously been running the protocol, apparently with **A**, and **B** was acting as a responder in his run, and the two agents agreed on the full content of the message **m**, and each run of **A** corresponds to a *unique* run of **B**"

To simplify, this often means that:

> *The communication between A and B is **authenticated, secure** and **replay protected***

Strongest level of assurance (covers aliveness and other agreements)

Lemmas and Proofs

- Describe good behaviour in the system
  - Attack is anything that breaks this behaviour

lemma secrecy:

"All x #i.

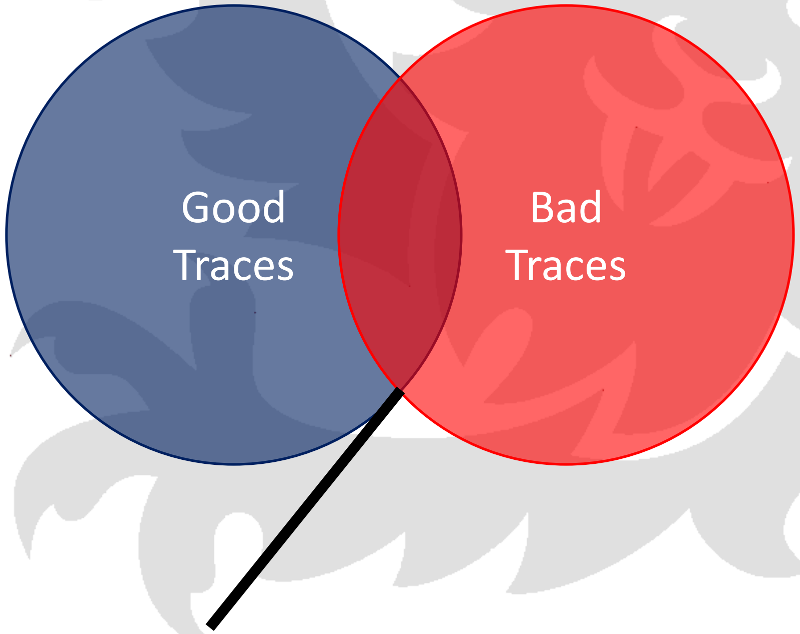
Secret (x) @ #i ==>

not (Ex #j. K(x)@#j) |

((Ex B #r. KeyReveal(B) @#r)

"

- If there is a claim that x is secret, either the attacker doesn't know x or the encryption key is leaked



Empty means it's secure, otherwise contains attack

Lemmas and Proofs

- Describe good behaviour in the system
  - Attack is anything that breaks this behaviour

lemma secrecy:

"All x #i.

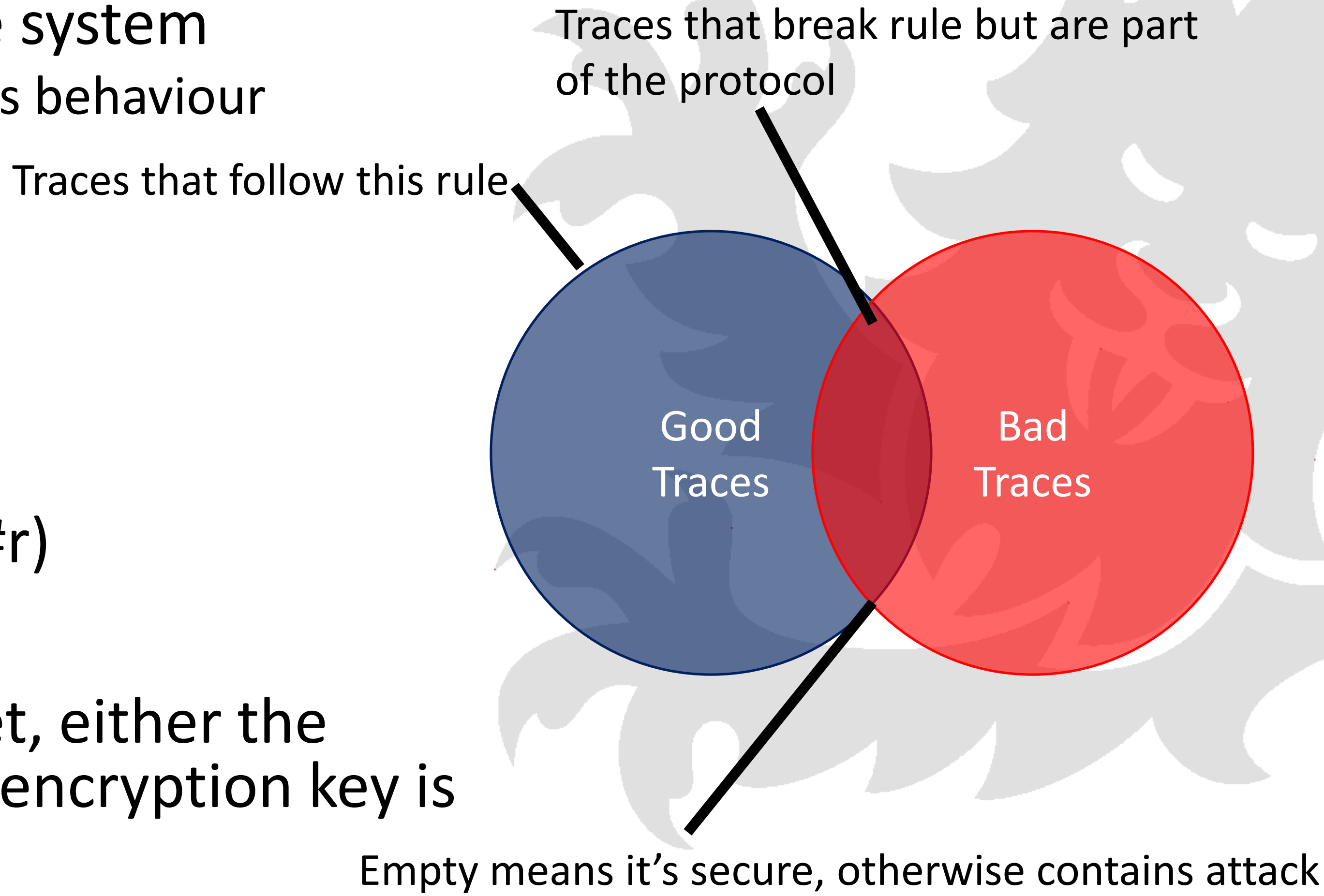
Secret (x) @ #i ==>

not (Ex #j. K(x)@#j) |

((Ex B #r. KeyReveal(B) @#r)

"

- If there is a claim that x is secret, either the attacker doesn't know x or the encryption key is leaked



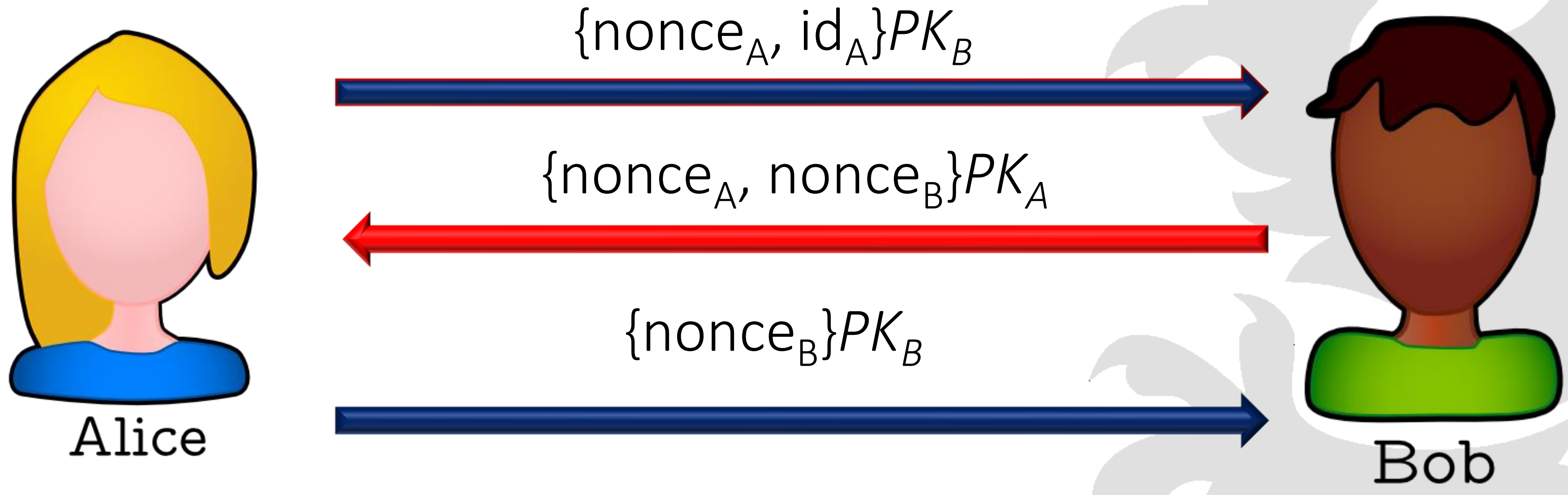
Newcastle University

School of Computing

Live Demonstration

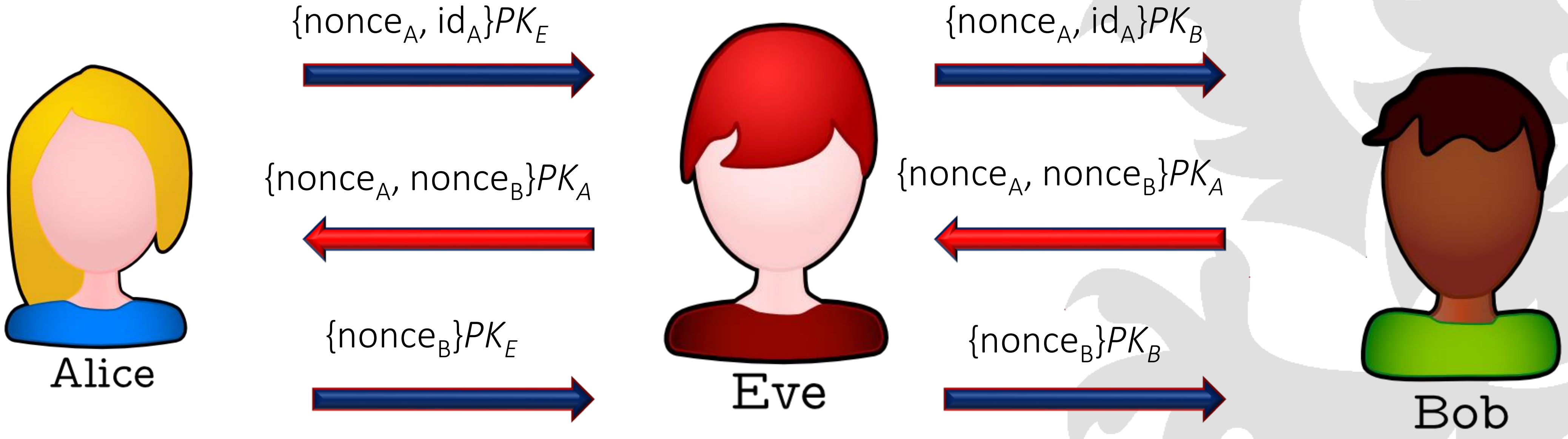
Needham Schroder

Newcastle University
School of Computing

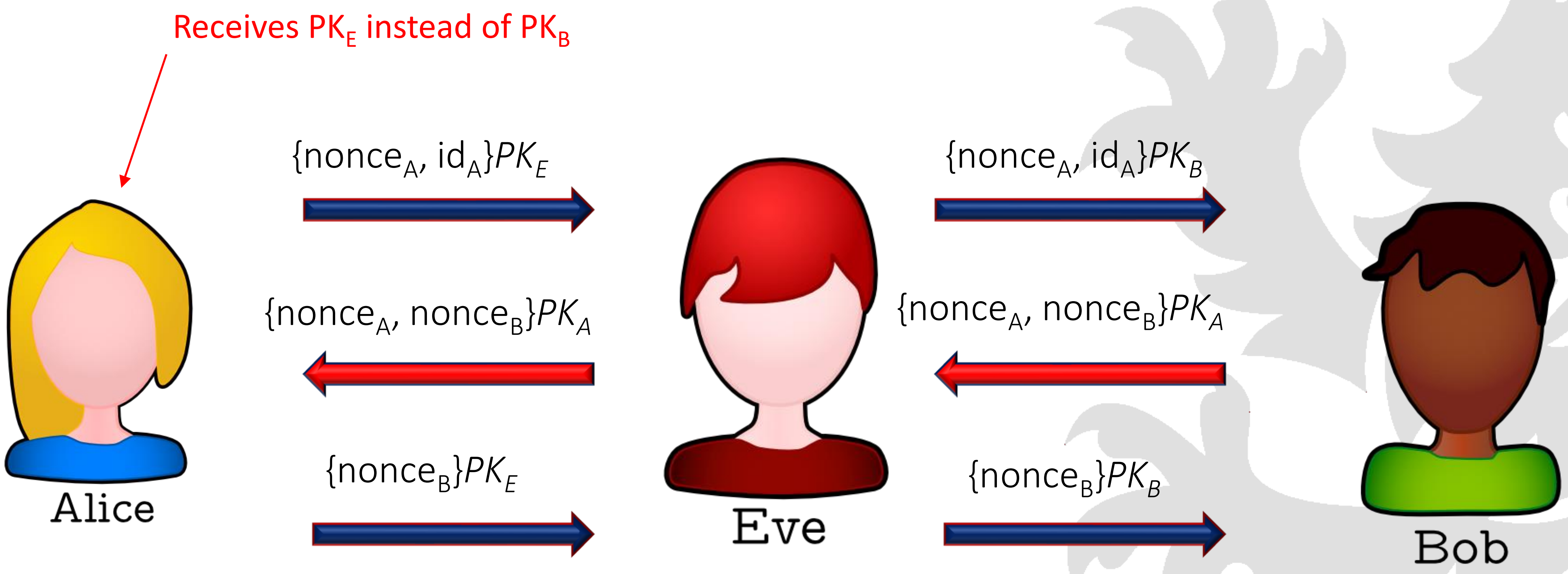


Practical Lab 1

Needham Schroder - Attack



Needham Schroder - Attack

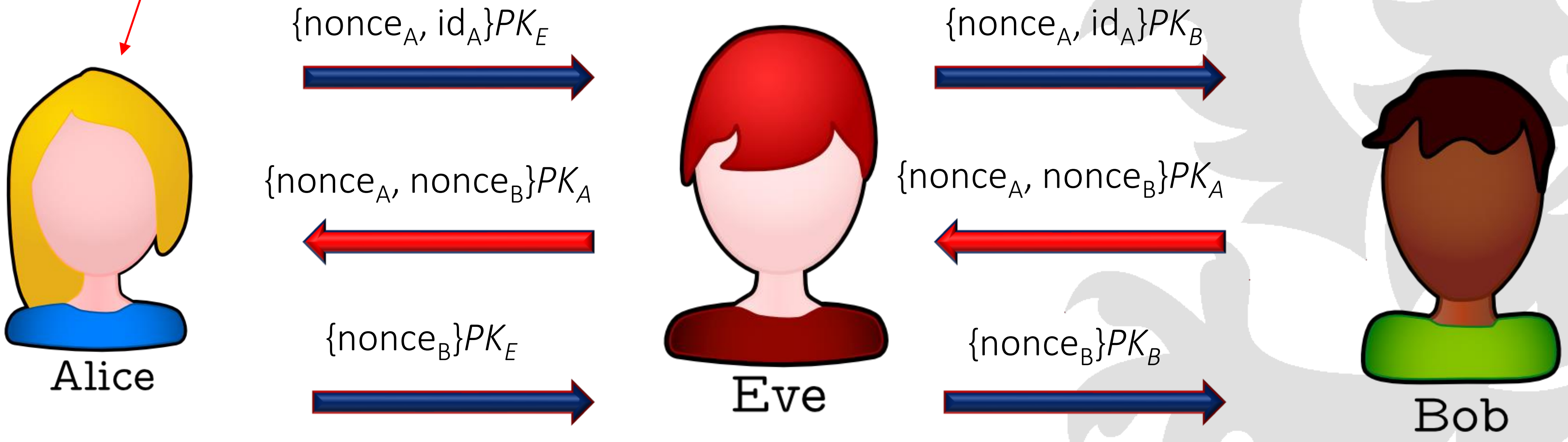


Needham Schroder - Attack

Newcastle University
School of Computing

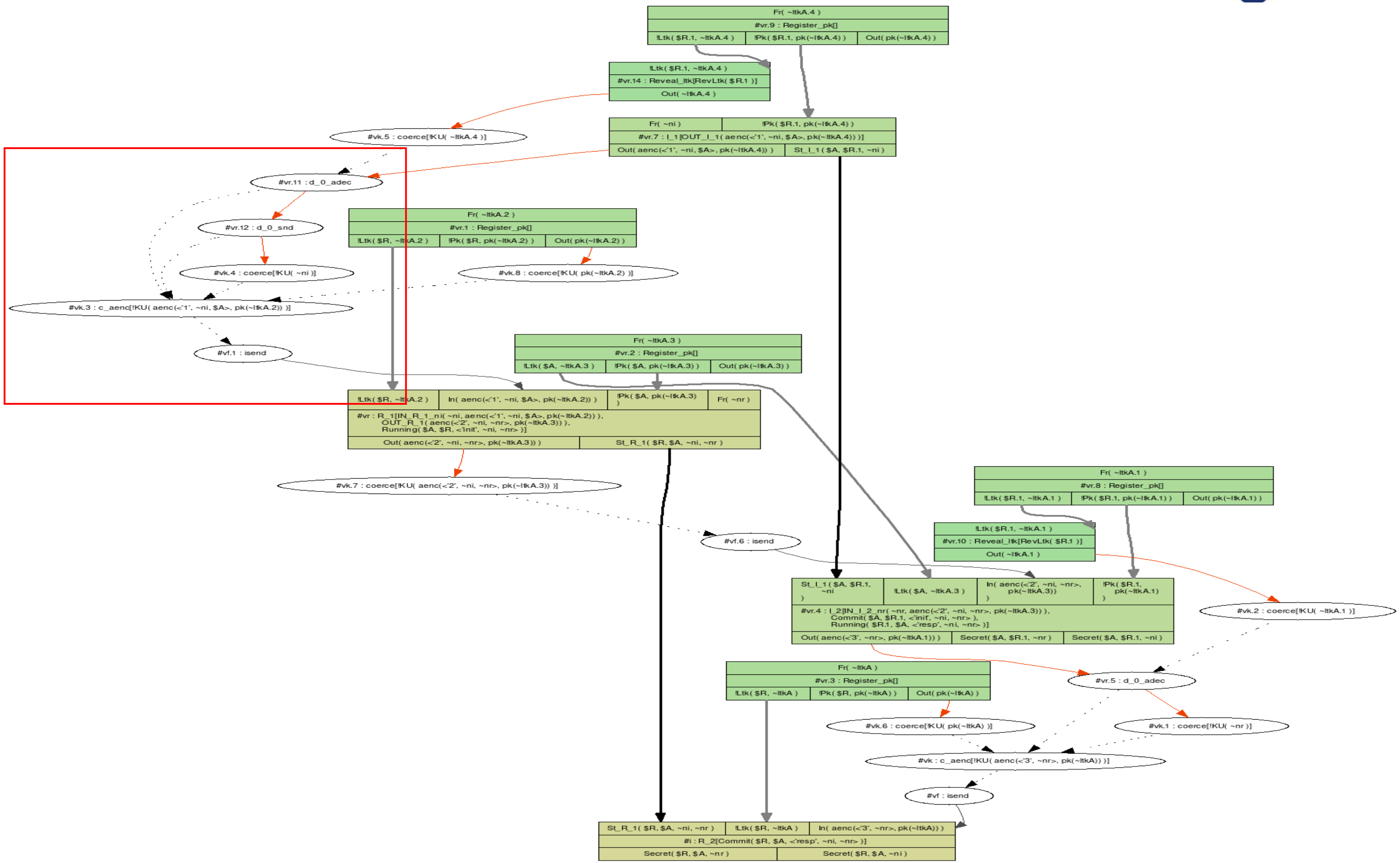
Was only discovered 17 years later!

Receives PK_E instead of PK_B



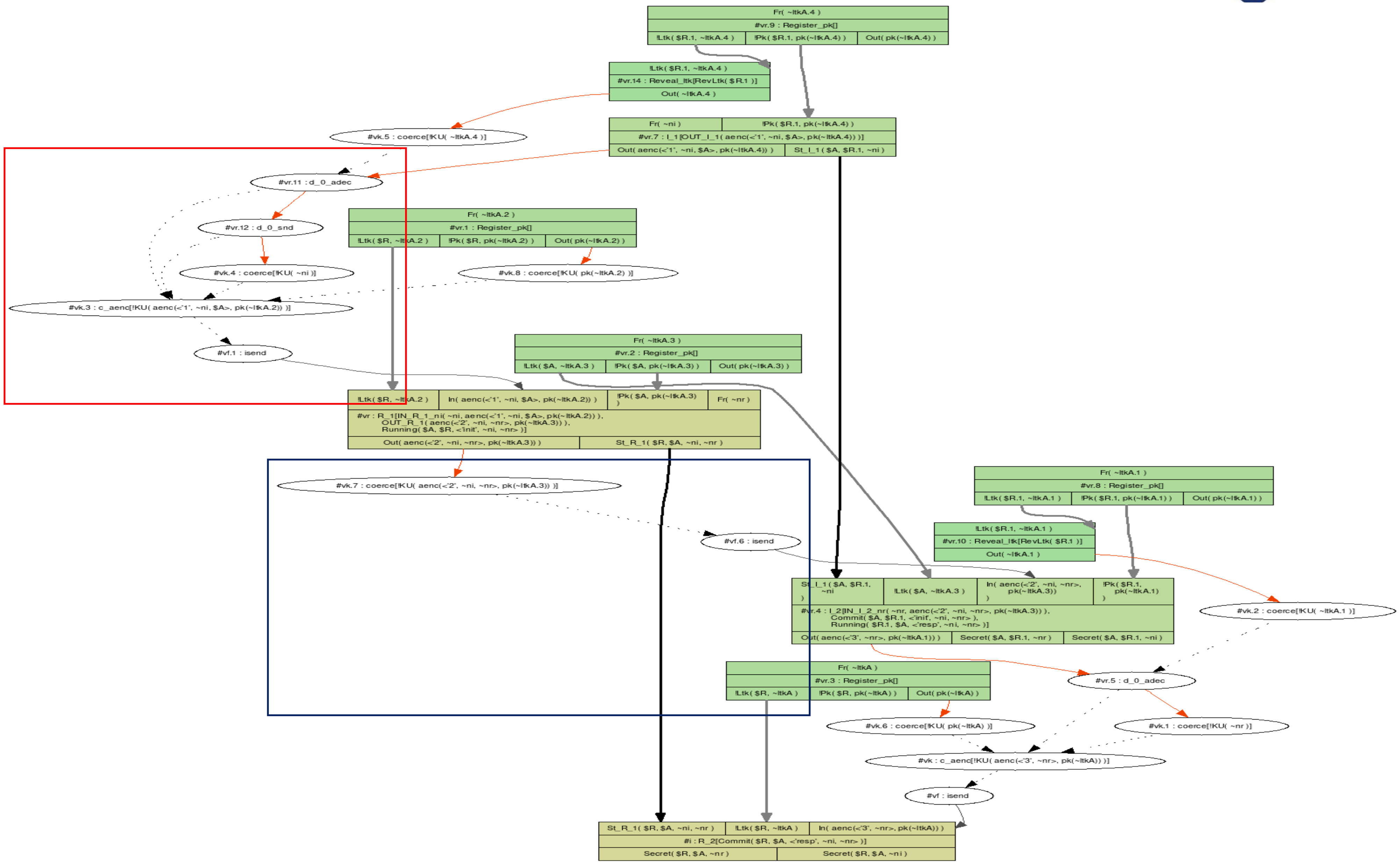
Needham Schroeder Attack – Tamarin

Example – Tamarin Output



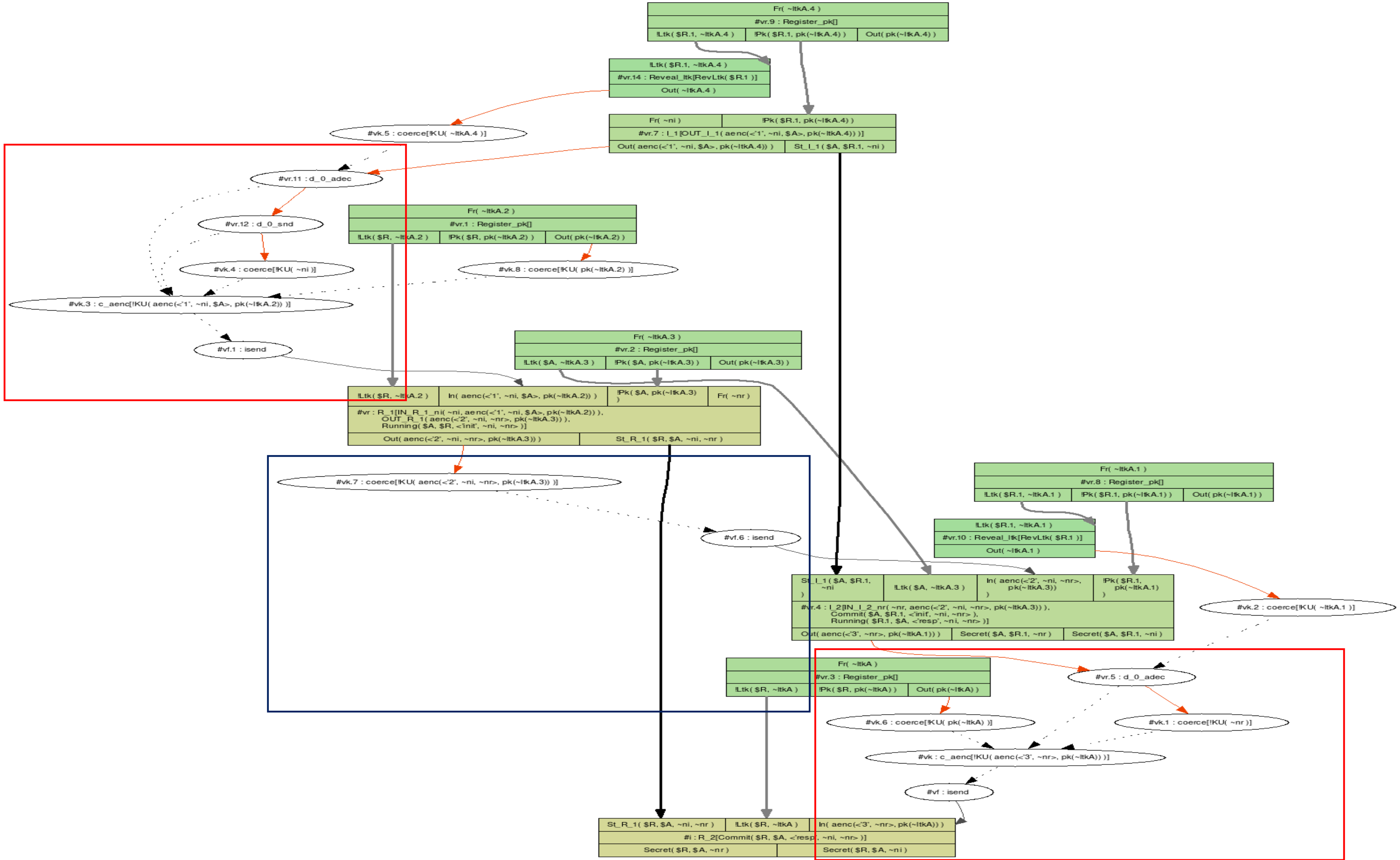
Needham Schroeder Attack – Tamarin

Example – Tamarin Output



Needham Schroeder Attack – Tamarin

Example – Tamarin Output

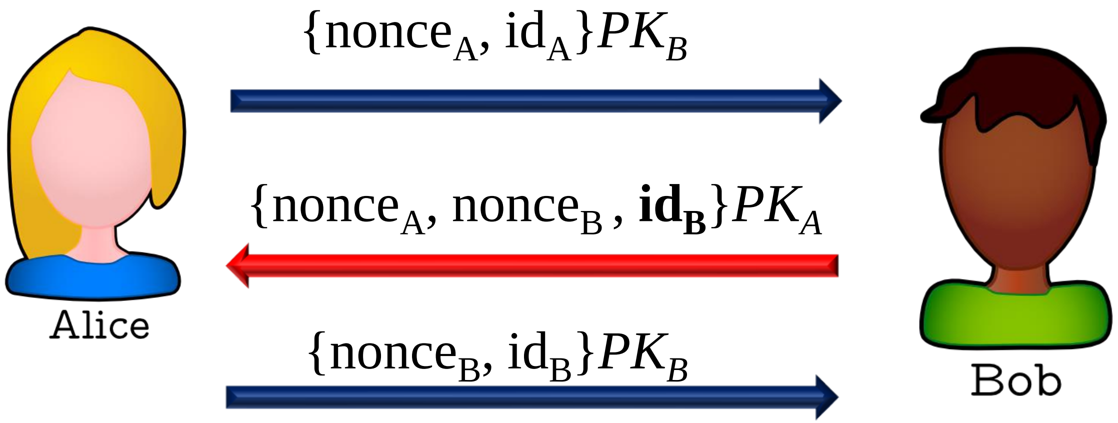


Formal Protocol Verification

Newcastle University
School of Computing

Example – Fixed and Proof

Needham Schroeder Lowe Fix




```
analyzed: nslp.spthy

  types (all-traces): verified (33 steps)
  nonce_secrecy (all-traces): verified (54 steps)
  injective_agree (all-traces): verified (92 steps)
  session_key_setup_possible (exists-trace): verified (5 steps)
```

Newcastle University
School of Computing

Live Demonstration

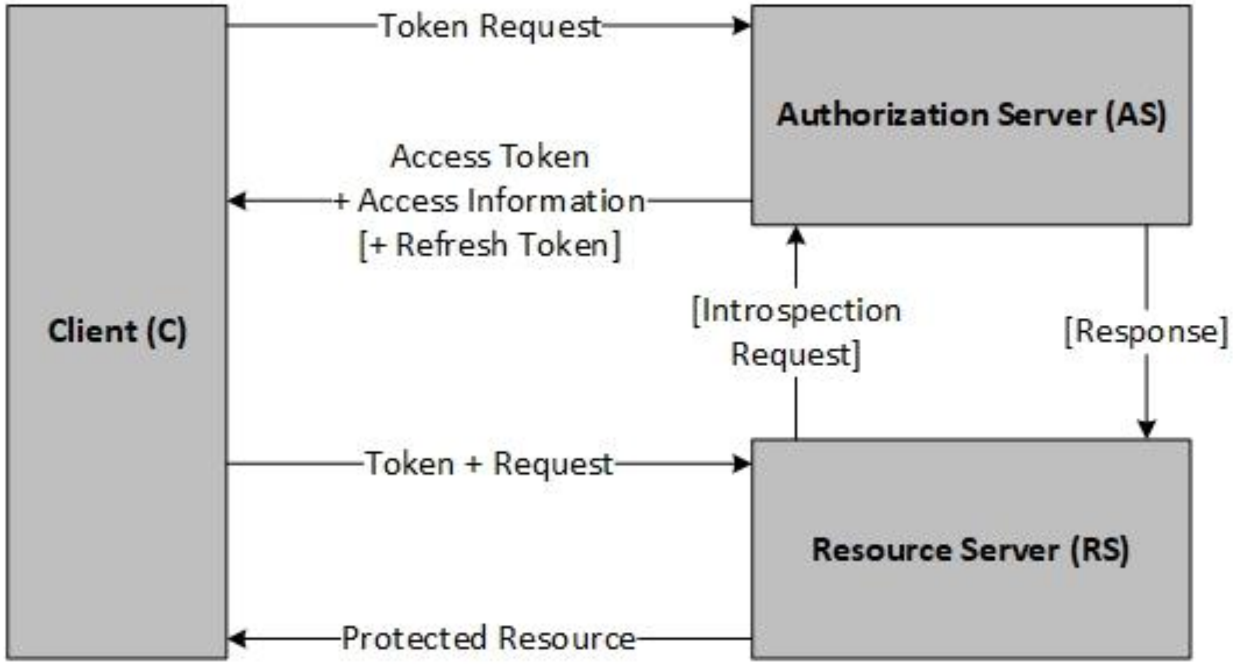
What we cannot do (yet!)

- Protocol Verification ignores physical attacks against the devices:
  - Side-channel attacks:
    - Power consumption, timing, noise, . . .
  - Faults introduced in the system in order to break its security

ACE-OAuth Flow

The protocol follows the following steps:

0. Discovery Step
1. Client requests access token
2. Token is returned to client
3. Now the token is presented to access the resource
4. The Resource server may or may not check with the AS whether it's correct
5. If the token is correct access is granted to the device



Introduce Skeleton Model

Newcastle University

School of Computing

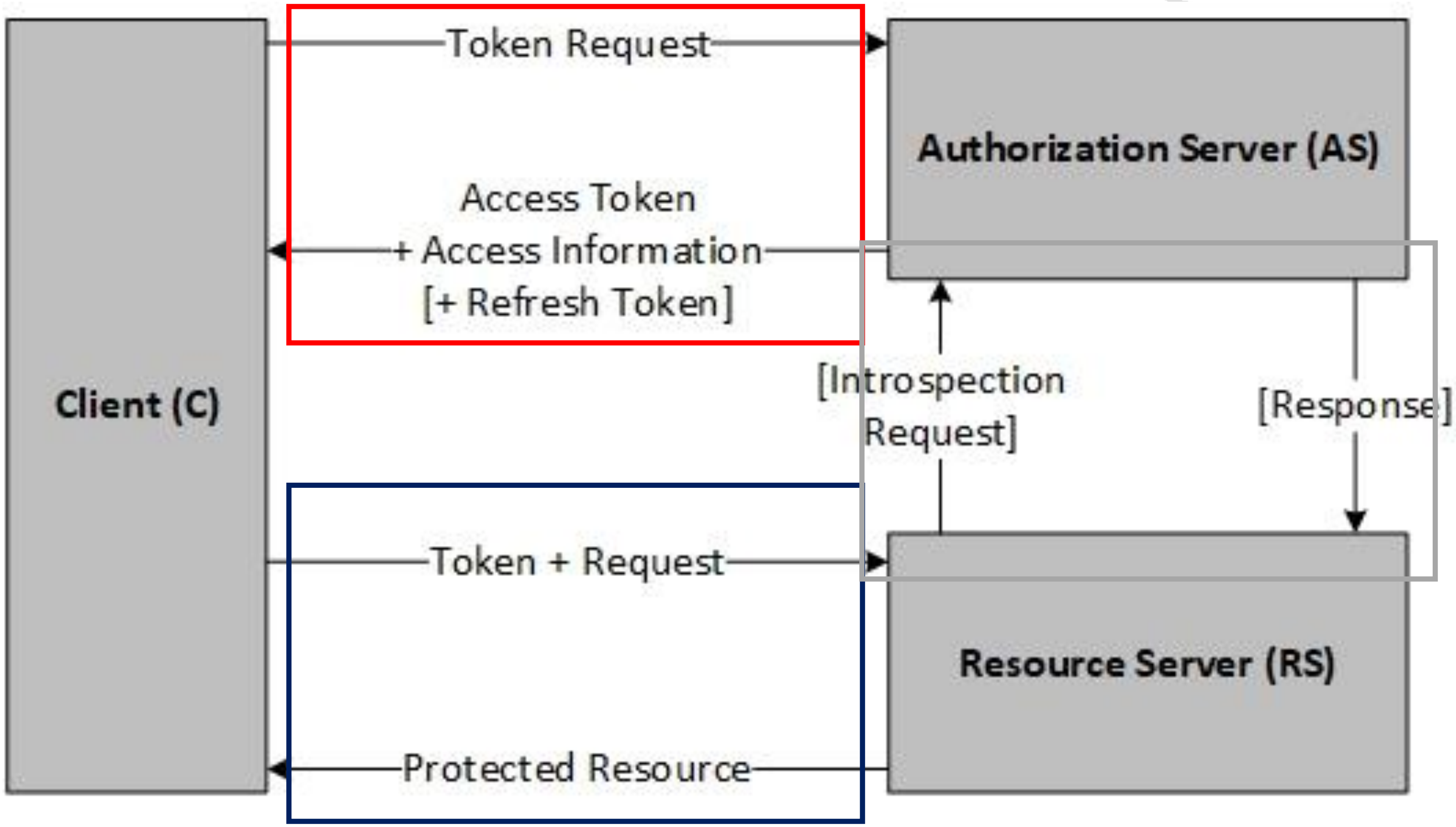
- Full flow of the protocol modelled in Tamarin
- Analysed the full specification and formalised the requirements
- Security objectives automatically checked
- Enables to test different design choices
- First effort to verify the full protocol flow
- Proved security of the protocol (flow only)

Design Choices

Newcastle University
School of Computing

- IETF drafts are generalised and not easily translated into formal
- Initial phase for formalising the security properties
  - Outlined the key pieces of information that needed to be secured
  - Outlined objectives for each exchange
- Most verification efforts model and assess security of specific implementation
- We went the opposite direction:
  - More flexible
  - Can be easily extended
  - More scalable

ACE-OAuth Flow



Formal Translation

Newcastle University
School of Computing

- Modular design
  - Can swap out different configurations
- Different extensions can be tested
- Can interactively view results
- After testing on two different extensions
  - New requirements arose
  - We had to change the implementation
  - With the interactive model we fixed mistakes and assessed risks

Newcastle University
School of Computing

The issue with composability

IETF drafts are by design flexible and allow for a wide range of different implementations (which is great), However:

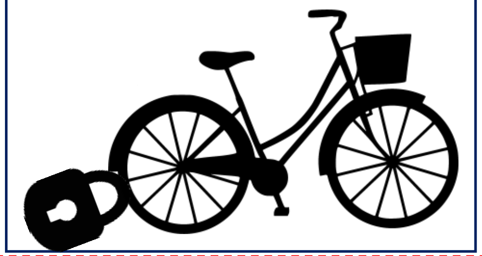
If security protocol X guarantees properties in scenario A,

DOES NOT MEAN that if used in scenario B same results hold!

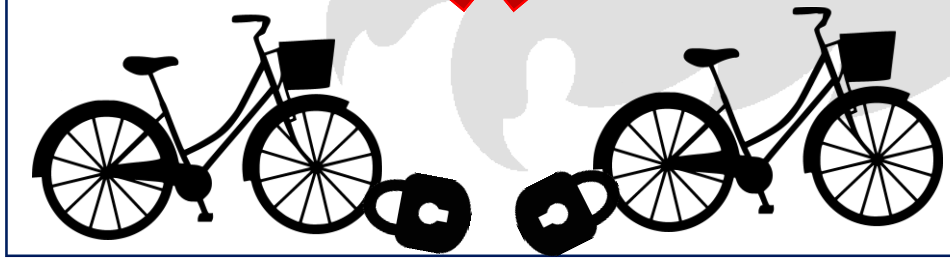
Example Bike Shop Security Protocols:

Protocol A

Protocol B



Protocol A+B



Designing an Extension

- The IoT has lots of different scenarios
- Allowing for different extensions is therefore desirable
- Extensions can cause serious security flaws

So we need a way to check this:

- Our approach:
  - Give a generalised model to start with
  - Formalise security goals so that it can be easily verified
  - Add your own extensions
  - Allow the tools to do the hard work!