

# Protocol Analysis: The SPYCE Perspective

Joe Halpern

# One goal of SPYCE:

Accomplish a high-level task,  
distributed among many nodes  
that do not trust each other  
(and may have incentives).

# How do we do this?

- We need good protocols
- And we need good ways of verifying that they do what they're supposed to
- But that means we also need good ways of describing and specifying protocols
  - This can be hard

# Why is describing a security protocol hard?

Standard programming languages approach:

- A protocol is a set of traces/executions/runs
- That's the SPYCE view too

But ... security protocols have special twists

- What is the intruder's protocol?
  - What capabilities does intruder have?
- Who can invoke the protocol? How often?
- What's a nonce?

# Why is specifying a security protocol hard?


Security involves many subtleties:

- What is a fresh message?
- What is a good key?
- What is a **fair** protocol?
- What is a good route?
- What is anonymity?



Once we know how to describe/specify protocols, we can

- prove specific protocols correct
- develop a deeper understanding of what makes things hard/easy
- build tools to help design/specify/verify protocols



**We have made major  
progress in all these areas.**

# (Some) SPYCE Accomplishments

## ■ MSR

- Language for describing security protocols and intruders
- Used to verify
  - Kerberos: anomalous behavior found!
  - \* Contract signing
- Deeper understanding of what makes things hard

\* = new since September

# SPYCE Accomplishments (cont'd)

- \* (Preliminary steps towards) specifying/verifying BGP
  - Path Vector Policy Systems
- \* Modeling intruders using algorithmic knowledge
  - Going beyond Dolev-Yao
- \* Analyzing protocols using knowledge
- \* General foundations for understanding nonces, “freshness”, jurisdiction

# SPYCE Accomplishments (cont'd)

- \* Relating formalisms for describing protocols
  - strand spaces to runs/systems
  - CSP to runs/systems
- \* Provided abstraction for securely exporting web services
  - Different protocol implementations
    - shared key, certificates
  - Proved correctness using Cryptyc

# SPYCE Accomplishments (cont'd)

- \* Cryptographic protocol analysis that takes into account algebraic properties of cryptographic primitives

# MSR: Background

[Butler, Cervesato, Chadha, Durgin, Jaggard, Lincoln, Mitchell, Scedrov, Shmatikov]

- Builds on CHAM [Berry&Boudol], and Concurrent Rewriting [Meseguer]

Syntax:

- Facts: Positive atomic formulas  
 $F := P(t_1, \dots, t_n)$
- State: multiset of facts  $\{ F_1, \dots, F_n \}$ 
  - Includes network messages, private state

# State Transition Rules

$$F_1, \dots, F_k \rightarrow \exists x_1 \dots \exists x_m. G_1, \dots, G_n$$

- If  $F_1, \dots, F_k \in \sigma$ , then a next state  $\sigma'$  has
  - facts  $F_1, \dots, F_k$  removed
  - $G_1, \dots, G_n$  added, with  $x_1 \dots x_m$  replaced by new symbols
  - same facts otherwise

# (Some) advantages

- Multiset can be used to capture different roles
- Existential quantification can be used to capture nonces

$$F_1, \dots, F_k \rightarrow \exists x_1 \dots \exists x_m. G_1, \dots, G_n$$

- $x_1, \dots, x_m$  can take on all possible values in the next state
- the transition is **nondeterministic**
- Transition rules can capture Dolev-Yao intruder (and others)

# Complexity-Theoretic Insights

		Bounded # of roles	Bounded use of $\exists$	Unbounded use of $\exists$
Intruder with $\exists$	$\neq, =$	NP – complete	??	Undecidable
	$=$ only		DEXP – time	
Intruder w/o $\exists$	$\neq, =$			
	$=$ only			

**Key insight:** existential quantification ( $\exists$ ) captures cryptographic nonce; main source of complexity

[Durgin, Lincoln, Mitchell, Scedrov]

# Using MSR to Analyze Kerberos

- Kerberos Goals
  - Repeatedly authenticate a client to multiple servers
  - Minimize use of client's long term key(s)
- Kerberos is a real world protocol
  - Windows 2000 (RFC 1510 + extensions)
  - User login, file access, printing, etc.
- Other methods have been used to verify Kerberos
  - Murφ, Paulson's inductive methods

# Kerberos 5 Analysis: Goals

[Butler, Cervesato, Jaggard, Scedrov]

- Give precise statement and formal analysis
- Identify and formalize protocol goals
  - This requires a good specification language!
- Discover anomalous behavior
  - Suggest possible fixes, test them

# Overview of Results

- Formalized Kerberos 5 at different levels of detail
- Observed anomalous behavior
  - Some properties of Kerberos 4 do not hold for Kerberos 5
  - Proved authentication properties that do hold for Kerberos 5
- Proofs of properties which do hold
- Interactions with Kerberos working group

# MSR: Contract Signing

[Chadha, Mitchell, Scedrov, Shmatikov]

Another important application domain.

- This one pushes the boundaries of the specification language
- A large part of the difficulty is stating the requirements:
  - fairness,
  - timeliness,
  - abuse-freedom

# Contract signing: Specification

Two adversarial parties want to exchange signatures on an agreed-upon contract text

- Both parties want to sign a contract
- Neither wants to sign first
- Fairness: each party gets the other's signature or neither does
- Timeliness: No player gets stuck
- Abuse-freedom: No party can prove to an outside party that it can control the outcome

# Abuse-Freedom

What does “no party can prove to an outside party that it can control the outcome” mean?

- Based on its observations, an outside observer will not **know** that party **P** has an advantage
  - Using epistemic logic seems critical here
  - And in lots of other security-related applications

# Contract Signing: Summary

- Several signature exchange protocols analyzed
- Formal definitions of fairness, optimism, timeliness, advantage, and abuse-freedom provided
  - abuse-freedom defined using epistemic logic
- Impossibility result
  - any fair, timely and optimistic protocol necessarily gives one participant an advantage

# Epistemic Logic

**Knowledge** plays a fundamental role in security:

- What does an intruder know?
  - Can she factor?
- Abuse-freedom:
  - What can an outsider know?
- Anonymity:
  - Can an observer know who performed an action?
- Noninterference:
  - What does Low know about High's state?

# Modeling an Intruder

[Halpern, Pucella]

- What if the intruder can't be characterized by Dolev-Yao?
  - an intruder that guesses
  - an intruder with side information

Epistemic logic is a good tool but ...

- we need to take resource limitations into account
  - intruders can't factor

# Resource-Bounded Reasoning

- Assume that an intruder has an algorithm that responds “Yes”, “No” or “?” to queries about formulas
  - The intruder **algorithmically knows**  $p$  if the algorithm responds “Yes”
- Different choices of algorithms can model different types of intruders
  - E.g., Dolev-Yao intruder, intruder with some partial information, intruders that guess
- We need both algorithmic knowledge and standard knowledge to analyze algorithms

# Capturing Security Concepts

[Halpern, van der Meyden, Pucella]

What's a "fresh" message?

- One that's unpredictable
- No one **knew** what the message would be in advance

■ What's a good key?

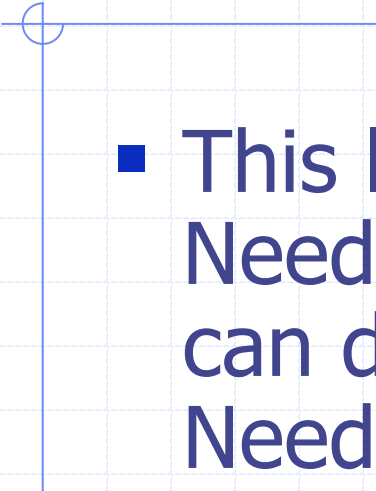
- One that only authorized users know.

Using knowledge (and probability) we can provide clean semantics for all the concepts used by BAN

# Security Protocol Logic

[Durgin, Mithcell, Petkovic]

- Logic for reasoning about security protocols that incorporates knowledge
- Typical formulas
  - $[actions]_P f$ 
    - Some role of principal  $P$  performs  $actions$ , after which formula  $f$  holds
  - $Knows(P, m)$ 
    - Principal  $P$  created  $m$  or received msg and has keys to extract  $m$  from msg
    - An instance of algorithmic knowledge

- 
- This logic has been used to verify the Needham-Schroeder-Lowe protocol, and can detect problems in the original Needham-Schroeder protocol
  - Likewise for the (related but somewhat different) Halpern-van der Meyden-Pucella approach
  - Next steps: a synthesis

# A New Direction: BGP

[Griffin, Jaggard, Ramachandran]

BGP = Border Gateway Protocol

- Used to set up routing between different Autonomous Systems
  - e.g., between MCI and Sprint
- Routing on the Internet is done locally
  - Router decides where it should forward a message based on local information
- Local routing can cause global anomalies
  - We want to avoid this

Want a language that can express a wide range of policies

- shortest-path routing vs. paths based on business relationships

Suggestion:

- Path-Vector Policy Systems to express mechanism for route info exchange
- separate policy language to express individual nodes' policies

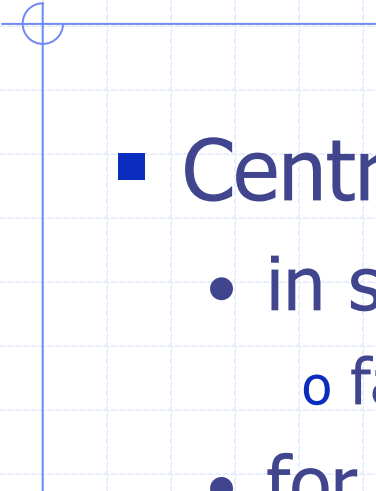
**Impossibility result:** checking policies locally not enough to guarantee “global sanity” unless **autonomy** or **transparency** are compromised.

- **transparency:** policy writers can understand the effects of their policies
- **autonomy:** policy writers can rank routes independent of their neighbors

**Bottom line:** local information cannot give us everything we want

# Summary

- We use a number of approaches for representing protocols, but they all boil down to sets of runs
  - We speak the same language
  - Results can be imported from one framework to another
- We've examined specific protocols
  - Kerberos, contract signing, BGP
- ... and proved general results about classes of protocols
  - impossibility results: no protocols with certain properties

- 
- Central role of knowledge
    - in specifying properties of interest
      - fairness, anonymity
    - for defining basic concepts
      - freshness, goodness of keys
    - in characterizing intruders

# (Some) Next Steps

- Further investigation of practical protocols
- Formal methods for cryptographic protocols
- Automating verification
- Adding utilities to specifications
- Verifying **mechanisms**
  - **mechanism** = set of rules for playing a game, designed to encourage “good” behavior
    - e.g. tax system, type of auction