

The Work of Dean Rosenzweig

A Tribute to a Scientist and an Innovator

Andre Scedrov

University of Pennsylvania

Dean Rosenzweig (1949 – 2007)



- ▶ Distinguished mathematician and computer scientist
 - Significant contributions to *logic, computer security, and foundations of software engineering*
- ▶ Professor at the University of Zagreb
 - leader of research groups in *theoretical computer science* and in *logic and foundations of mathematics*
- ▶ Pivotal role in building up the *information technology* used in the Zagreb Stock Exchange

Program Specification and Verification

- ▶ Using *Abstract State Machines (Evolving Algebras)* introduced by Gurevich
- ▶ 1990 – 1995, with Börger: Mathematical definitions of the *Warren Abstract Machine* and *full Prolog*
- ▶ 1995, with Börger and Gurevich: Specification and verification of *Lamport's Bakery Algorithm*
- ▶ 2006, with Blass, Gurevich, and Rossman: foundations of *interactive small-step algorithms*

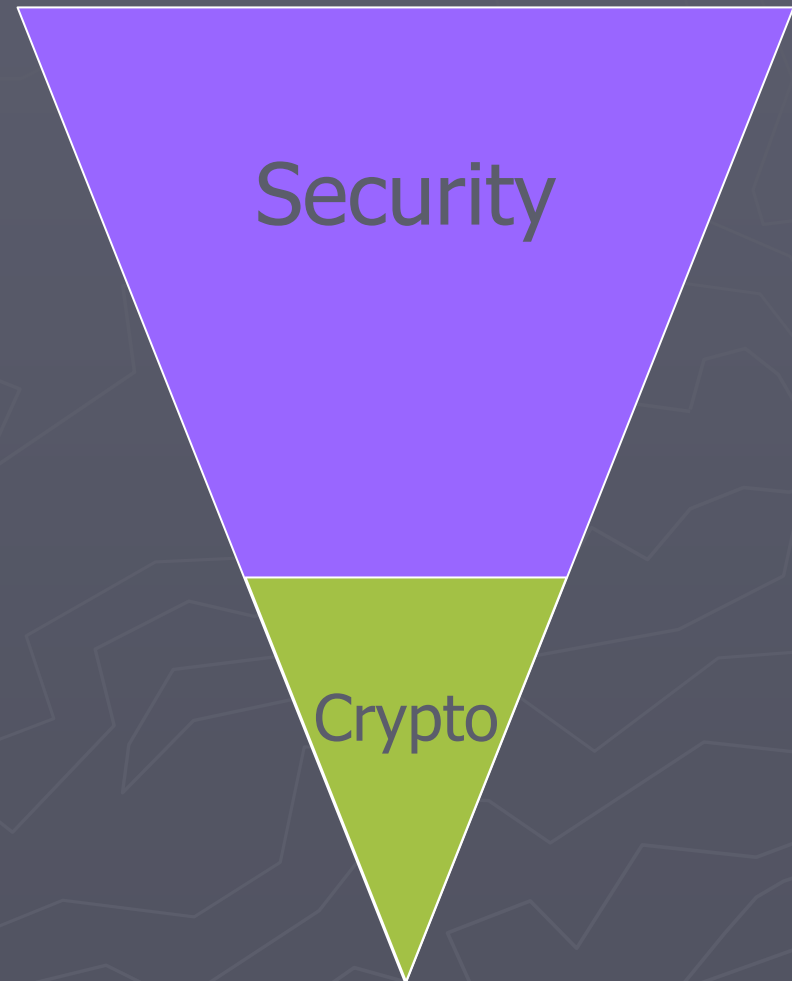
Security Protocol Verification

- ▶ 2003, with Runje, Slani: Privacy, Abstract Encryption and Protocols: an ASM Model
- ▶ 2004, with Runje: The Cryptographic Abstract Machine
- ▶ 2005, with Runje and Schulte: Model-based Testing of Cryptographic Protocols

Computer Security

Goal: protection of computer systems and digital information

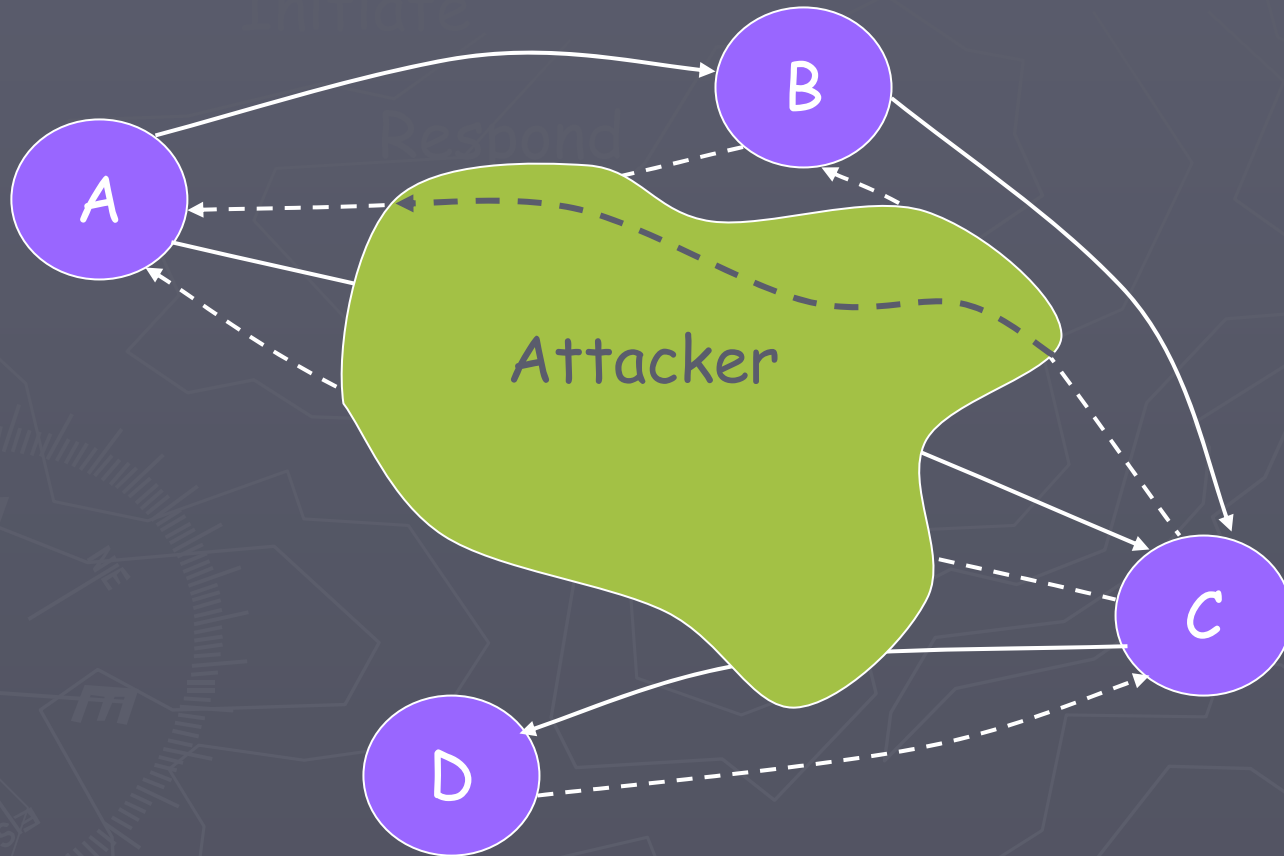
- ▶ Access control
- ▶ OS security
- ▶ Network security
- ▶ Cryptography
- ▶ ...



Protocol Security

- ▶ Cryptographic Protocol
 - Program distributed over network
 - Use cryptography to achieve goal
- ▶ Attacker
 - Read, intercept, replace messages, and remember their contents
- ▶ Correctness
 - Attacker cannot learn protected secret or cause incorrect protocol completion

Run of protocol

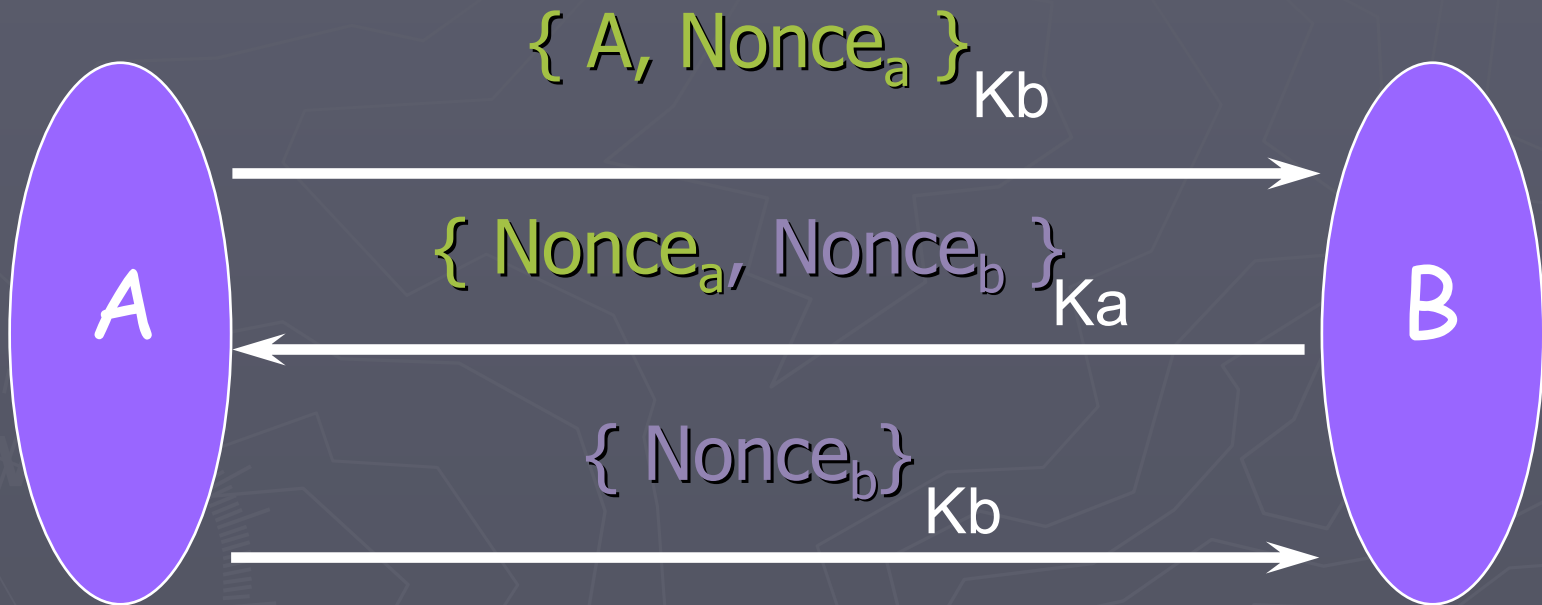


Correct if no security violation in any run

Correctness vs Security

- ▶ Program or System Correctness
 - Program satisfies specification
 - ▶ For reasonable input, get reasonable output
- ▶ Program or System Security
 - Program resists attack
 - ▶ For unreasonable input, output not completely disastrous
- ▶ Main differences
 - Active interference from environment
 - Refinement techniques may fail

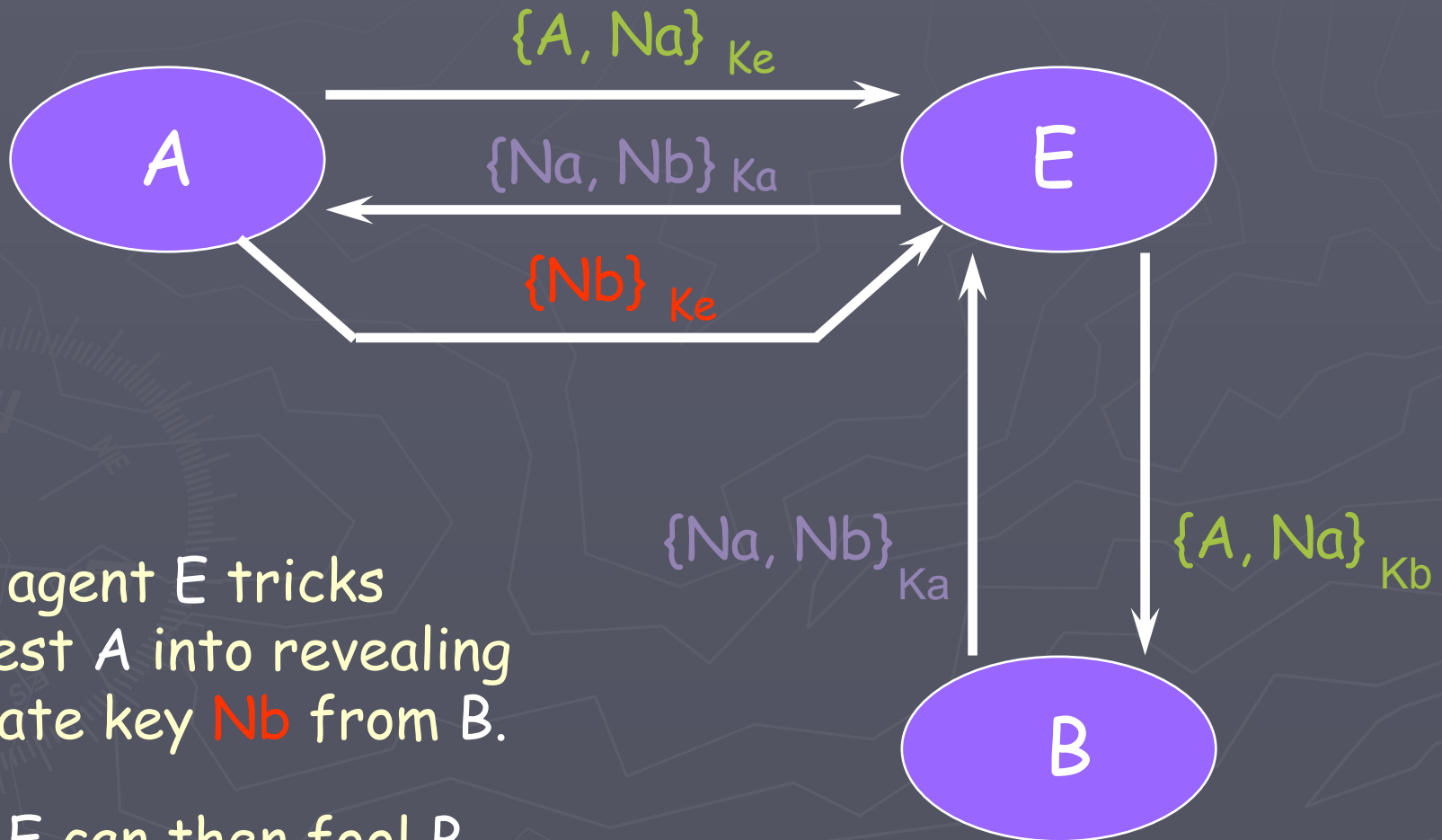
Needham-Schroeder Key Exchange



Result: A and B share two private numbers not known to any observer without K_a^{-1}, K_b^{-1}

Anomaly in Needham-Schroeder

[Lowe]



Evil agent E tricks honest A into revealing private key **Nb** from B.

Evil E can then fool B.

From computational cryptography ...

- ▶ Example: Symmetric probabilistic encryption scheme is a triple:

- K is key generator with security parameter η
- E is encryption
- D is decryption

such that

- $\Pr[k \leftarrow_R K(\eta), e \leftarrow_R E(k,m) \text{ in } D(k,e) = m] = 1$

But note:

- $\Pr[k_1 \leftarrow_R K(\eta), k_2 \leftarrow_R K(\eta) \text{ in } D(k_2, E(k_1, m)) = m]$ is negligible

- ▶ Computational security is expressed in terms of probabilistic PTime algorithms as a function of η

... to abstract cryptography

- ▶ Abstraction from computational cryptography
 - Abstract from security parameter η
 - Represent strings, coins, nonces by atoms
 - Guarantee that unlikely events become impossible
- ▶ Abstract security is absolute
 - If both key k and encryption e under k are accessible
 - ▶ $\text{Decrypt}(k, E(k, m)) = m$
 - Otherwise
 - ▶ $\text{Decrypt}(k_1, E(k_2, m)) = \text{undef}$

Security Protocol Verification Timeline

Thanks to A. Gordon

- ▶ 1988: Abstract (Dolev-Yao) and computational models known to separate communities; Burrows-Abadi-Needham logic
- ▶ 1992-95: WAN scale networking finally takes off; design flaws in early versions of SSH, SSL, and Lowe's insider attack on Needham/Schroeder motivate the need for foundational models
- ▶ 1997: To analyze models one must trade between automated bug detection in bounded models (Meadows, Millen), and hand or interactive proofs of realistic, unbounded models
- ▶ 2002: Automated proofs in the abstract model w.r.t. unbounded models (Paulson, Blanchet); Relation between abstract and computational models actively investigated (Abadi-Rogaway, Backes-Pfitzmann-Waidner, Micciancio-Warinschi, ...)
- ▶ 2007: Mechanized and automated proofs in the computational model; Relation between formal and computational models being worked out

Abstract State Machines Perspective

- ▶ Definitions (taken from abstract state machines)
 - Element of a state is *accessible* if it is the value of some term
 - States X and Y are *similar*, if they induce the same equality relation on terms: $\text{Val}_X(t_1) = \text{Val}_X(t_2)$ iff $\text{Val}_Y(t_1) = \text{Val}_Y(t_2)$
- ▶ Theorems [Rosenzweig-Runje-Schulte 2005]:
 - A small step algorithm can create an encryption only if subject and encryption key is accessible
 - Similarity \Rightarrow Computational Indistinguishability (generalization of Abadi-Rogaway)

Model-Based Testing

- ▶ Requires model of system behavior (aka. functional specifications)
- ▶ Behavior is often reactive/nondeterministic
 - Implementation is multi-threaded or distributed
 - Thread scheduling, msg. delivery hard to control
- ▶ State space is typically infinite
 - State variables, method parameters, objects

Approach

- ▶ Behavior of reactive systems can be viewed as a game between two players
 - Test tool, which can control certain moves
 - Implementation under test, which moves can only be observed
- ▶ Game is specified by a *model program*
 - Written in *Spec#* or *AsmL*
 - Meaning is given by *interface automata*
- ▶ Testing uses *game theory*
 - *Strategies* tell what move the test tool should make in a state where its moves are enabled
 - *Alternating simulation* is used for conformance notion
- ▶ Model exploration, validation and model-based testing are provided by the *Spec Explorer* tool

Model-Based Testing of Security Protocols

Basic idea

- ▶ Use symbolic model (Dolev-Yao) instead of computational model to analyze security protocols
- ▶ Agents are ordinary interactive small step abstract state machines
 - Honest agents: execute the protocol as specified (observer)
 - Intruder: can do whatever he wants (controller)
 - Intra-step interaction: import from reserve or background
- ▶ Use Spec Explorer tool as an efficient engine for attack analysis

Spec Explorer finds Lowe's anomaly in Needham-Schroeder

- ▶ What the test showed:
 - Security bug found automatically
 - Model reusable for other symmetric protocols
 - Guarantees for models with finite number of roles
 - General purpose tool used for protocol testing

IT for the Zagreb Stock Exchange

Thanks to D. Ruševljan

► Pivotal role

- Theoretical analysis of trading paradigms, principles and rules
- Architectural design of main exchange trading and information systems
- Actual implementation of many key components of trading system MOST and real-time information system MOSTich, which has been in use by the exchange for many years

IT for the Zagreb Stock Exchange

Thanks to D. Ruševljan

- ▶ Designing secure and authenticated communication protocols
- ▶ Encryption implementation
- ▶ Stream compression
- ▶ Writing the parser and interface repository for CORBA ORB
- ▶ Writing the server algorithms for order matching and trading

Dean Rosenzweig (1949 – 2007)



- ▶ Distinguished mathematician and computer scientist
 - Significant contributions to *logic, computer security, and foundations of software engineering*
- ▶ Professor at the University of Zagreb
 - leader of research groups in *theoretical computer science* and in *logic and foundations of mathematics*
- ▶ Pivotal role in building up the *information technology* used in the Zagreb Stock Exchange

How it works

- ▶ Modeling
 - Define (infinite) interface automata IA_p through program P
- ▶ Exploration
 - Reduce IA_p to a finite test graph G
- ▶ Test generation
 - Generate test cases from G
- ▶ Test execution
 - Run the test cases using the model as the oracle

To avoid state space explosion, exploration, test generation and execution can be combined into a single on-the-fly algorithm[VCST05]

Empirical Evidence of Effectiveness

▶ Used

- for Webservice infrastructure, BCL, device drivers, file replication,...
- by approx 80 testers within MSFT, approx 30 projects

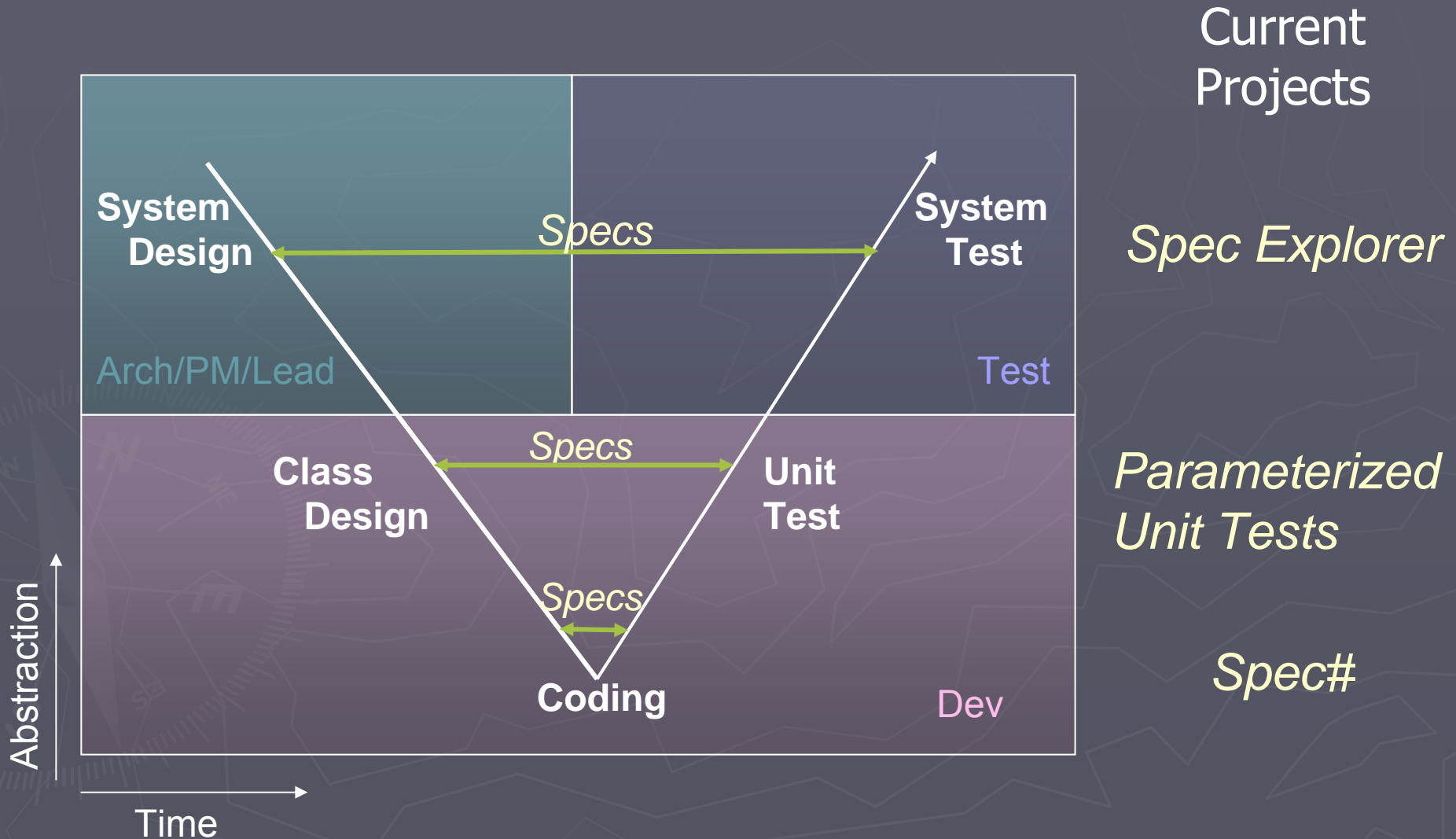
▶ Modeling effort

- Models from 2-8 pages
- Version 1 same amount than traditional
- Version $i > 1$ substantial benefits, 30-70% of the time

▶ Fault revealing capability

- 5-10 times more and “deeper” bugs revealed than traditional automatic tests
- 10-15 times more bugs revealed during design time than without models
- 50% of bugs are in code/design, 50% are in the model
- Same code coverage than with traditional techniques achievable

Software development process



Overview

▶ Part I

- System testing with model programs ...

▶ Part II

- ... of cryptographic protocols

Modeling

- ▶ *States* are mappings of variables to values
(Semantics: ASM states or first-order structures)
- ▶ *Initial state* is given by initial assignment to variables
- ▶ *Actions* are defined by method invocations
- ▶ *Preconditions* of methods and model invariants define action enabling conditions
- ▶ *Transition function* is defined by method execution

Exploration [GGSV02]

- ▶ Exploration is the process of unfolding a model program into interface automaton (IA)
 - Actions move the system from state to state
 - State comes from variables containing values
 - Objects are identities
- ▶ In general the IA is infinite, but we can impose limits to create a state space of manageable size that satisfies a given testing goal
- ▶ In Spec Explorer we
 - Restrict action parameter domains to interesting values
 - Restrict the state space to interesting states

Test generation and strategies

In Spec Explorer we separate *what* we want to test from *how* we want to do it.

- ▶ What is the conformance relation[VCST05]?
 - *Alternating refinement* says that
 - ▶ all controllable actions of the TT must be accepted by the IUT and
 - ▶ all observable actions of the IUT must be accepted by the TT
- ▶ How to check for conformance relation [NVSTG05, BGNV05]?
 - *Game strategies*
 - ▶ say what controllable action should be chosen by the TT in active states
 - ▶ are computed by value iteration of Markov decision procedure (as reachability games)

TT stands for test tool, IUT for implementation under test

Test generation with accepting states

- ▶ Accepting states are specified through a Boolean state expression φ_{acc}
 - s is an *accepting state* if s satisfies φ_{acc}
 - a *trap state* is a state from which an accepting state can not be reached
- ▶ All test cases must terminate in accepting states

Test execution

► Preparation

- Call backs are introduced in the implementation
- Model and implementation methods are bound to another

► Execution

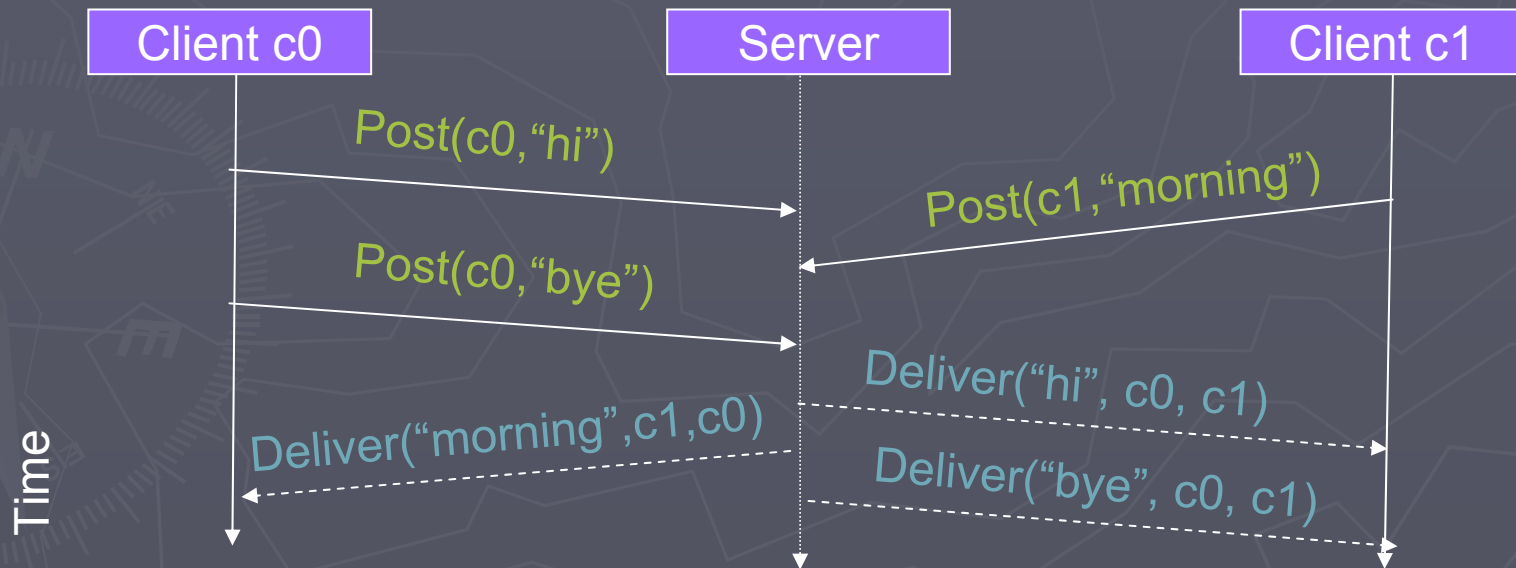
- Generated test cases are executed in *lockstep* by the model, which acts as the oracle, and the implementation
- Objects are bound at runtime to another
- A conformance failure occurs if an enabling condition is violated or expected results don't match

Example: Chat web service

Service should guarantee FIFO delivery of messages with local consistency:

Post is *controllable* action

Deliver is *observable* action



Chat Server State

```
class Client {}  
type SenderQueues = Map<Client, Seq<string>>;  
type MemberState = Map<Client, SenderQueues>;
```

```
MemberState Members = new Map();
```

Example MemberState

<i>Receiver</i>		<i>SenderQueues</i>	
[C0	->	[C1 -> ["hi", "bye"],	C2 -> ["morning"]]
[C1	->	[C0 -> ["hello"],	C2 -> ["morning"]]
[C2	->	[C0 -> [],	C1 -> ["hi", "bye"]]

Chat Server Actions

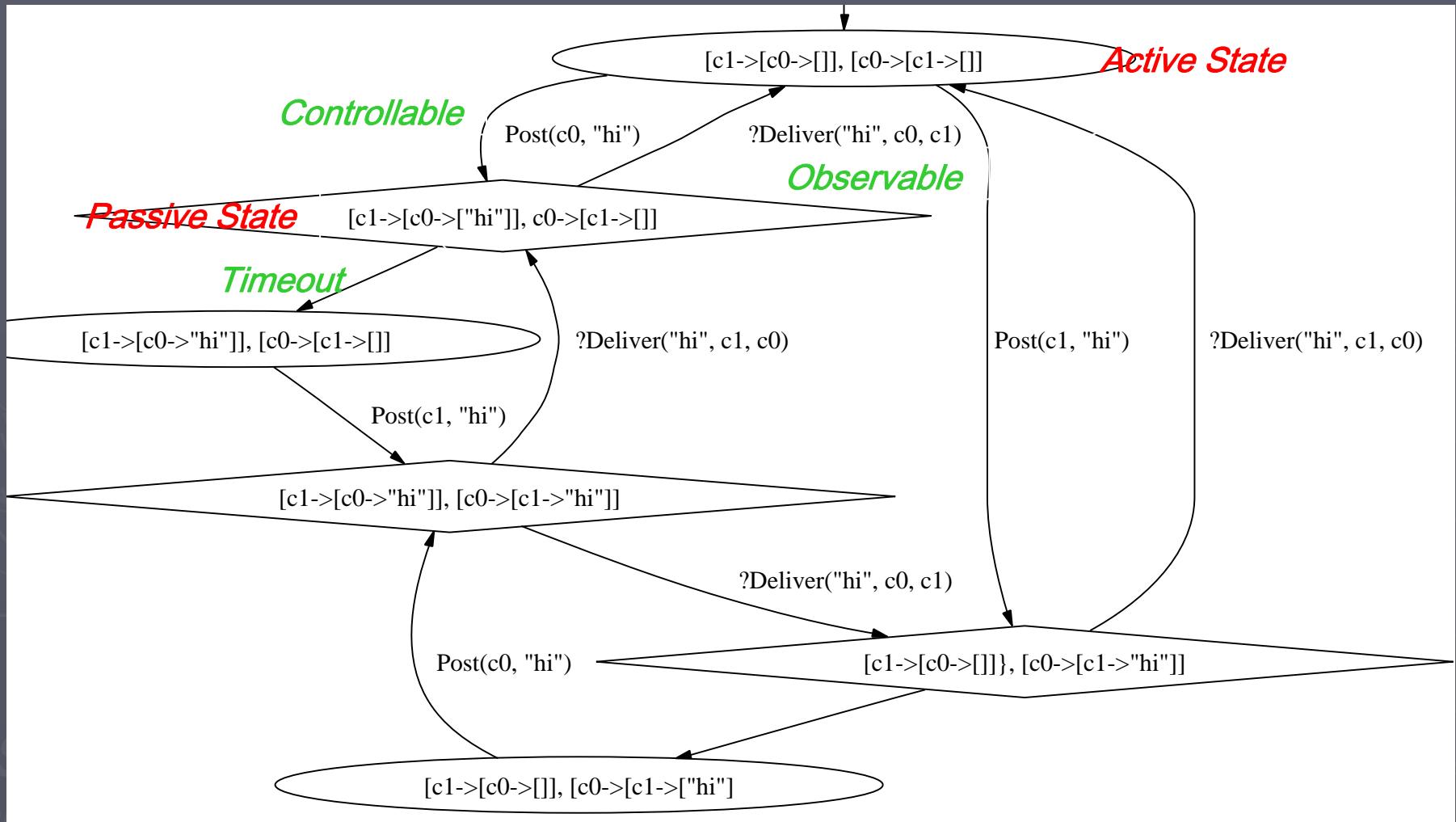
[Control lable]

```
void Post(Client sndr, string msg)
    requires sndr in Members && Members.Size > 1;
{
    foreach (rcvr in Members)
        if (rcvr != sndr) Members[rcvr][sndr].Add(msg);
}
```

[Observabl e]

```
void Deliver(string msg, Client sndr, Client rcvr)
    requires sndr in Members[rcvr] && rcvr in Members;
    requires Members[rcvr][sndr].Length > 0 &&
        Members[rcvr][sndr].Head == msg;
{
    Members[rcvr][sndr] = Members[rcvr][sndr].Tail;
}
```

Chat Server Interface Automaton



Imposed restrictions: 2 clients, one message "hi", each senders queue length ≤ 1

Spec Explorer demo of a chat server

What the demo showed:

- Model programs are useful abstractions, not only for test but also for design analysis
- Different test objectives produce different automata from the same model program
- Strategies for coverage and stress testing
- Conformance testing finds bugs

Lowe's attack on Needham Schroeder Protocol

Involved:

- Honest agents: A(lice) and B(ob)
- Intruder: E(ve)
- Nonces, private and public keys from A,B,E
- Operations: Pairing, Encrypt, Decrypt

Lowe's attack:

<u>Agent</u>	<u>Input</u>	<u>Output</u>
A to E(B)	:	$\rightarrow \{A, n_A\}_E$
E(B) to B	$:\{A, n_A\}_E$	$\rightarrow \{A, n_A\}_B$
B to E(A)	$:\{A, n_A\}_B$	$\rightarrow \{n_A, n_B\}_A$
E(A) to A	$:\{n_A, n_B\}_A$	$\rightarrow \{n_A, n_B\}_A$
A to E(B)	$:\{n_A, n_B\}_A$	$\rightarrow \{n_B\}_E$
E(B) to B	$:\{n_B\}_E$	$\rightarrow \{n_B\}_B$

Modeling infinite choices

Problem (1)

- ▶ Creation of unique keys, nonces

Observation

- ▶ Modeled via creation of objects (has same effect: no accessibility, isomorphism)

Problem (2)

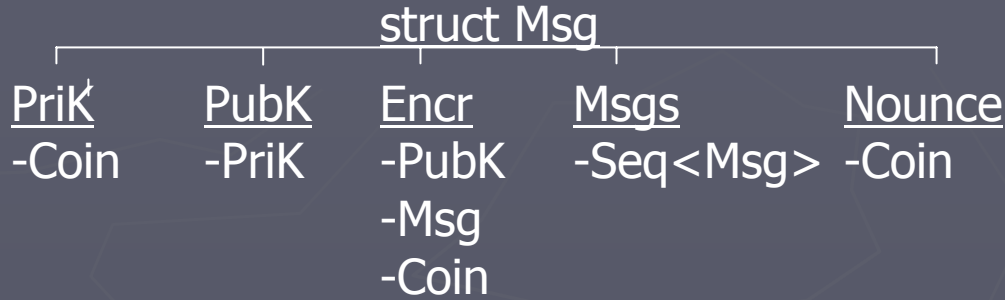
- ▶ Honest agent following a protocol, accepts an infinite set of input msgs
- ▶ Intruder can generate an infinite set of messages
- ▶ The intersection of these two sets is also infinite

Observation

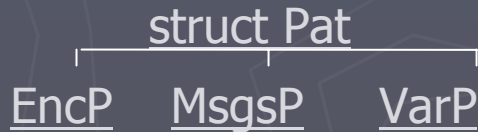
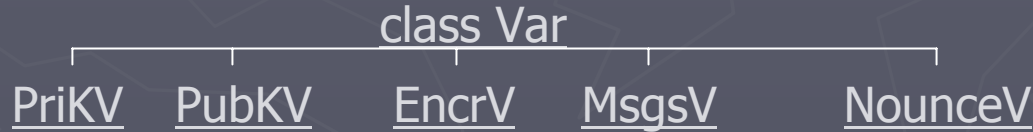
- ▶ For certain classes of protocols, this infinite number of messages can be generated with a *finite set of patterns*

Spec Explorer model for asymmetric encryption [RRS03]

class Coin



struct Env
-Var ↦ Msg



MatchMsg(Env, Pat, Msg)
CreateMsg(Env, Pat)
CreateIntruderPats(Msg)



void RunRole(Role)
bool CheckGuarantee()

The checking algorithm

1. Initialization: create honest and corrupted agents, create roles
2. Intruder asks each role for the set of possible input messages
3. Intruder runs all roles with representatives of all possible input messages
4. If the protocol guarantee is violated stop the search
5. If no move is possible – there is no attack with so many roles

Lowe's attack on Needham Schroeder Protocol

Involved:

- Honest agents: A(lice) and B(ob)
- Intruder: E(ve)
- Nonces, private and public keys from A,B,E
- Operations: Pairing, Encrypt, Decrypt

Lowe's attack:

<u>Agent</u>	<u>Input</u>	<u>Output</u>
A to E(B)	:	$\rightarrow \{A, n_A\}_E$
E(B) to B	$:\{A, n_A\}_E$	$\rightarrow \{A, n_A\}_B$
B to E(A)	$:\{A, n_A\}_B$	$\rightarrow \{n_A, n_B\}_A$
E(A) to A	$:\{n_A, n_B\}_A$	$\rightarrow \{n_A, n_B\}_A$
A to E(B)	$:\{n_A, n_B\}_A$	$\rightarrow \{n_B\}_E$
E(B) to B	$:\{n_B\}_E$	$\rightarrow \{n_B\}_B$

Recent work

Testing

- ▶ Optimal strategies for testing nondeterministic systems, Nachmanson, Veanes, Schulte, Tillmann, Grieskamp, *ISSTA04*
- ▶ Play to test, Blass (U. of Michigan), Gurevich, Nachmanson, Veanes, MSR-TR-2005-04
- ▶ On-the-fly testing of reactive systems, Veanes, Campbell, Schulte, Kohli (Univ. Oxford) MSR-TR-2005-05
- ▶ Multiplexing of partially ordered events, Campbell, Veanes, Huo (McGill Univ), Petrenko (CRIM), *TestCom 2005*
- ▶ State exploration with multiple state groupings, Campbell, Veanes, *ASM 2005*

Cryptography

- ▶ Privacy, Abstract Encryption and Protocols: an ASM Model, Rosenzweig, Runje, Slani, ASM 2003
- ▶ Model-based Testing of Cryptographic Protocols, Rosenzweig, Runje, Schulte, forthcoming