

# Abstractions for Cryptographically Faithful Proofs of Security Protocols

---

**Christoph Sprenger**, ETH Zurich  
joint work with Michael Backes, David Basin,  
Birgit Pfitzmann, and Michael Waidner

**CosyProofs Spring School**, Apr 13, 2010

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



# Security Protocol Models

## Dolev-Yao and Cryptographic Reality

---

### Symbolic approaches: Dolev-Yao model [DY81]

- messages from **free term algebra**, e.g.,  $\text{encrypt}(k, \text{pair}(NA, \text{Alice}))$
- **active attacker controls network**: may **intercept** messages, **insert** any message he can **derive** from **messages** observed on the network
- **perfect cryptography** assumption: ciphertext leaks no partial information about the cleartext

### Cryptographic approaches

- messages are **bitstrings** without a priori structure, e.g., 110100
- strong security definitions & guarantees: **negligible success probability** for attack by a **computationally bounded adversary**
- security **proofs by reduction** to underlying crypto primitives

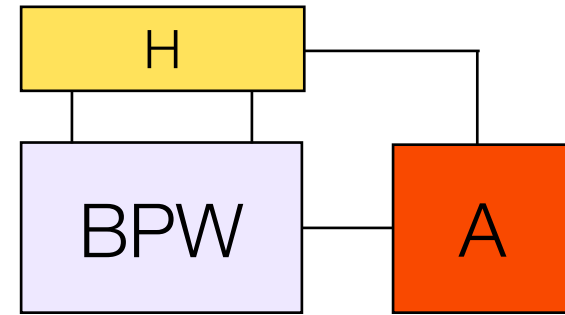


Cryptographic justification of Dolev-Yao-style models is an active research area, over the past decade.

# The Backes-Pfitzmann-Waidner Model

(The “BPW Model”, 2003)

---



## Characteristics

- component (“library”) implementing **standard crypto primitives**, used by honest users and adversary; includes public-key encryption, signatures, symmetric encryption, and message authentication (MACs)
- **local functions** for **message construction and manipulation**, and **send functions** for **message transmission**
- **stateful**: tracks **content** (messages) and **knowledge** (who knows what); messages manipulated indirectly using **handles** (pointers)
- two versions: **cryptographic (real)** and **symbolic (ideal)**

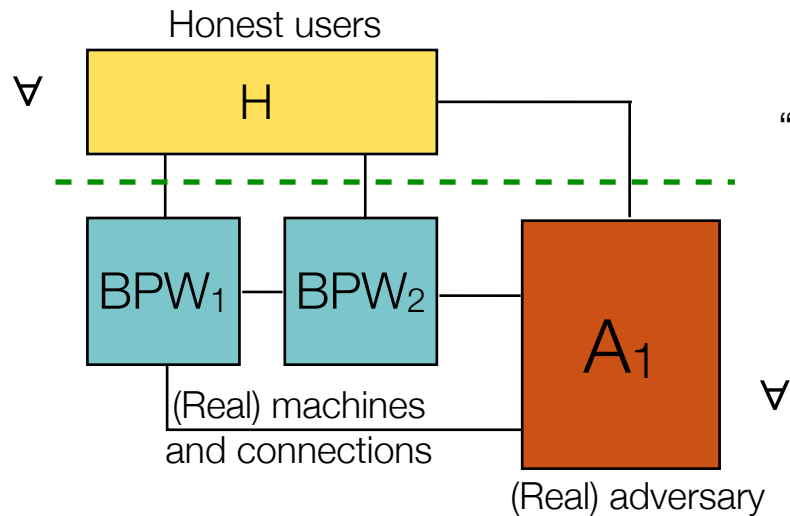
## Model Captures Security-Relevant Real-World “Imperfections”

- secure encryption and signature schemas are **probabalistic**
- encryption cannot keep the **length of cleartexts** secret
- adversary may include **incorrect messages** in encryption

# Cryptographic Soundness of BPW Model

Reactive Simulatability/UC (RSIM/UC) [BPW, Canetti]

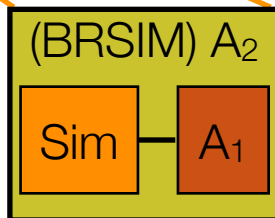
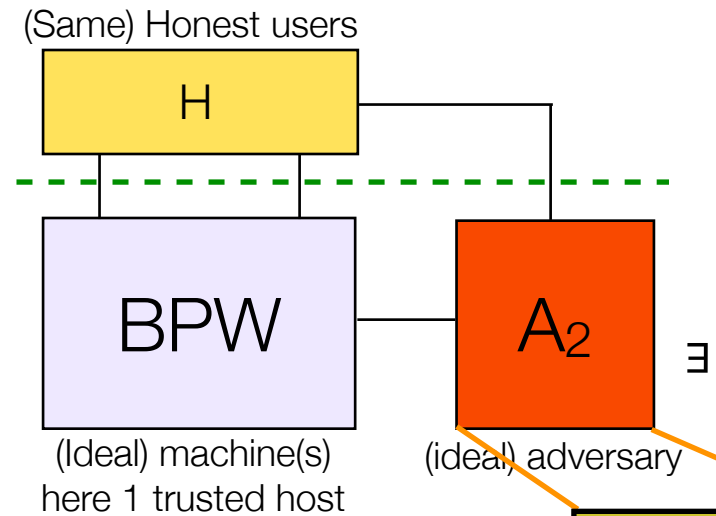
Cryptographic / Real System



“as secure as”

$\approx$

Symbolic / Ideal System



- probabilistic poly-time **indistinguishability** of respective user views; comparison enabled by uniformly using **handles** to refer to messages
- attacks on cryptographic realization translate to symbolic abstraction; equivalently: **security property preservation** from ideal to real
- **compositionality**: RSIM relation preserved in all contexts (e.g. protocols!)
- **once-and-for-all reduction proof**. Afterwards, work with symbolic version

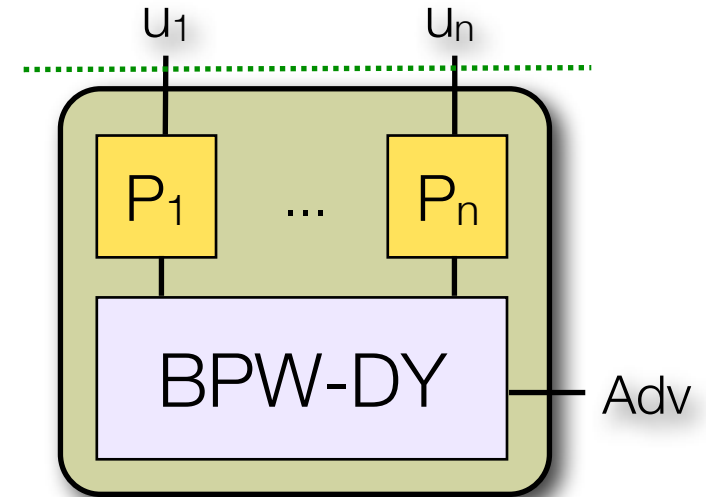
We will work with the special case of Blackbox Reactive Simulatability.

# Objectives and Results

---

## Main Objectives

1. **object-level reasoning**: cryptographically sound machine-assisted **reasoning about security protocols**
2. **meta-level reasoning**: soundly **abstract** and **simplify** BPW model and mechanize methods for such abstractions



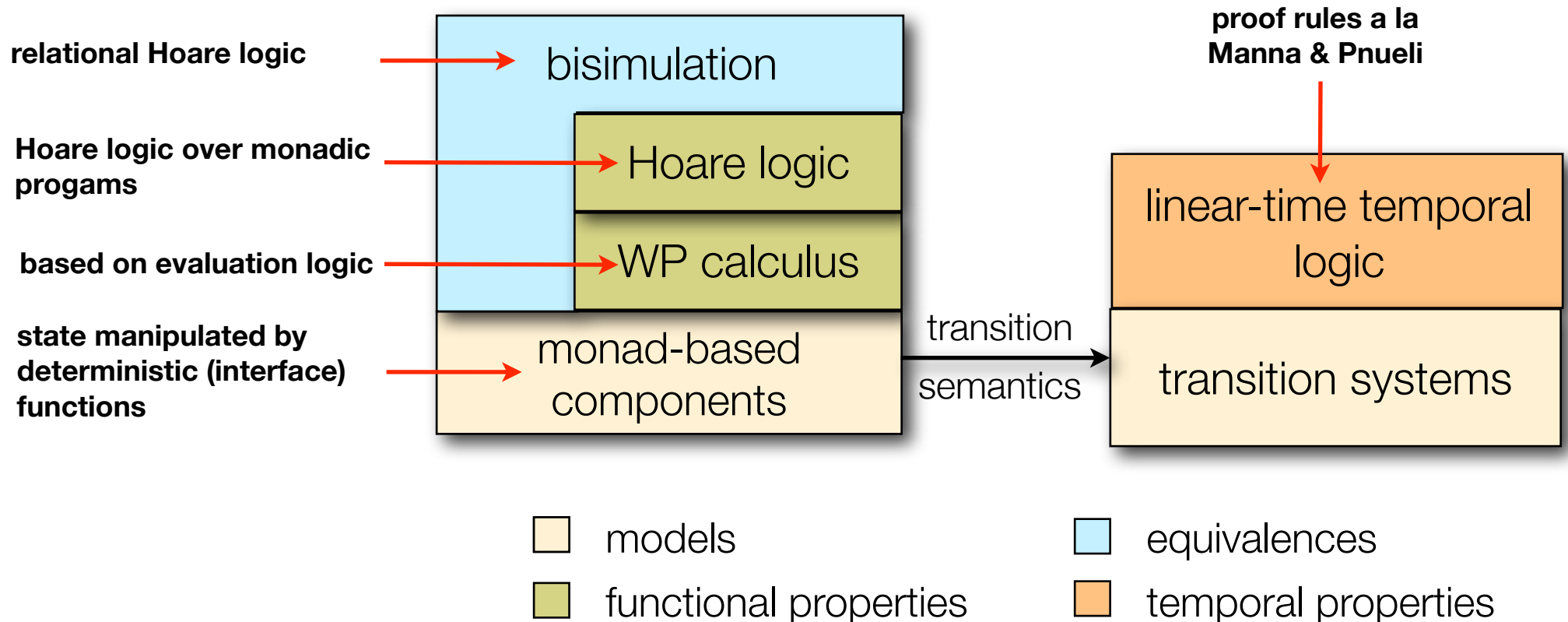
## Results

- ✓ two formalizations of BPW model, connected by **BRSIM abstraction step**
- ✓ **abstract protocol model** (also **BRSIM**) very close to standard protocol models
- ✓ **proof of concept**: cryptographically-sound verification of several protocols
- ✓ general **modeling** and **verification toolbox** for state-based systems

All formalizations carried out in the Isabelle/HOL theorem prover

# Modeling and Verification Toolbox

Overview (details: [TPHOLs 2007])



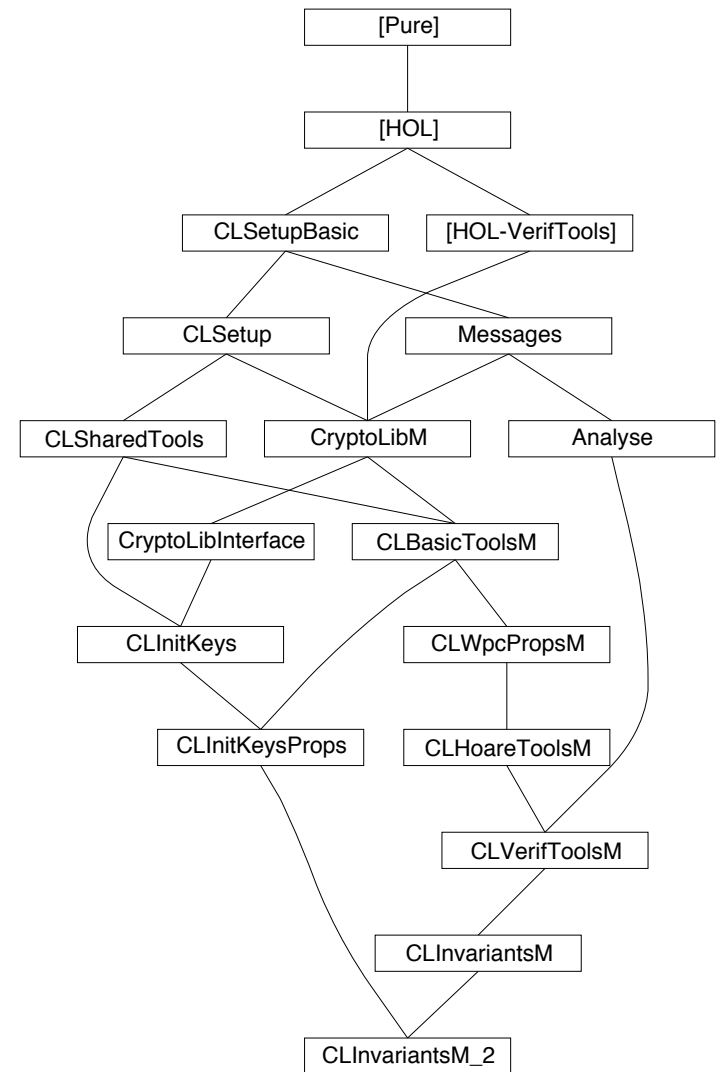
# Overview

## → Part I: Formalizations of the BPW Model

- direct abstractions of the BPW Model
- case study:  
Needham-Schroeder-Lowe public-key protocol

## Part II: A More Abstract Protocol Model (APM)

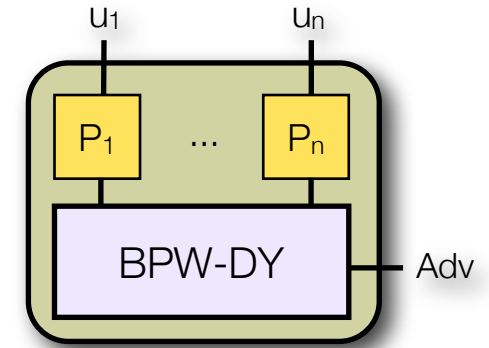
## Part III: Reasoning in the APM



BPW model version II

# Two Formalizations of BPW Model

Overview (details: [CSFW 2006])



## First version (V1): messages as DAGs

- **HOL-formalization based on paper version of BPW model of [BPW 2003]** (abstracts original component and communication model)
- faithful to original **data representation: messages represented as DAGs**
- **too concrete**: initial attempt to prove NSL protocol failed!

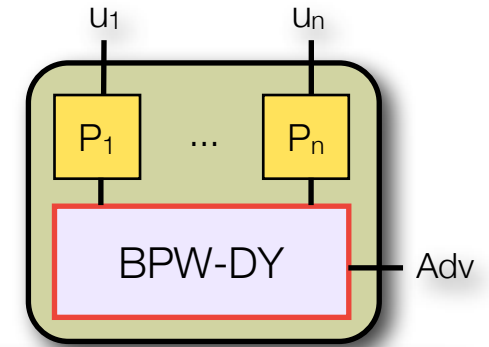
## Second version (V2): message terms

- **abstract data representation**: messages defined as **inductive data type**  
↳ recovers **inductive reasoning** over messages
- **concise property specifications**: expressed using **derivable knowledge**  
↳ enables efficient **equational reasoning**, e.g.,  $\text{parts}(G \cup H) = \text{parts}(G) \cup \text{parts}(H)$
- Overall, **simplified state** leads to substantially **better proof automation**

**Cryptographic Soundness: bisimulation** (hence **BRSIM**) with first version proved using relational Hoare logic.

# BPW Model Formalization (V2)

## Messages (partial)

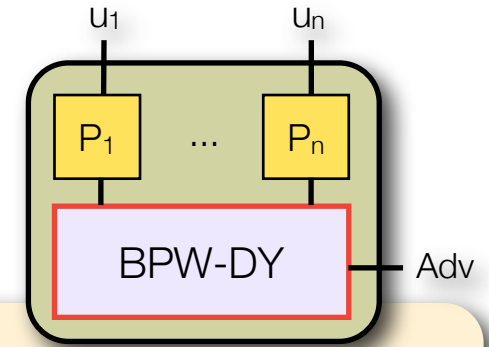


```
datatype  $\delta$  msg =  
  mNonce tag           -- nonce  
| mPke key             -- public encryption key  
| mSke key             -- private decryption key  
| mData  $\delta$          -- data item  
| mPair ( $\delta$  msg) ( $\delta$  msg) -- pair of messages  
| mEncv tag key ( $\delta$  msg) -- valid ciphertext  
| mGarbage tag len    -- adversary garbage  
| mEnci tag key len   -- invalid ciphertext
```

- inductive message type **parametrized** on type of payload data
- cryptographic messages are tagged with **randomness** (tag/key)
- special **garbage** and **invalid ciphertext** messages created only by adversary

# BPW Model Formalization (V2)

## State and interface (partial)



**record**  $\delta$  mLibState =

knowsM :: party  $\Rightarrow$  hnd  $\rightarrow$   $\delta$  msg

-- knowledge map for each party

gen\_nonce :: party  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- nonce generation

gen\_enc\_key\_pair :: party  $\Rightarrow$  (hnd  $\times$  hnd,  $\delta$  mLibState) S

-- generate asymmetric key pair

encrypt :: [party, hnd, hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- key and cleartext to ciphertext

decrypt :: [party, hnd, hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- key and ciphertext to cleartext

store :: [party,  $\delta$ ]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- store payload data

pair :: [party, hnd  $\times$  hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- pairing

first, second :: [party, hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- projections

adv\_parse :: hnd  $\Rightarrow$  ( $\delta$  pContent,  $\delta$  mLibState) S

-- adversary message parser

adv\_garbage :: nat  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- generate garbage

adv\_invalid\_ciph :: [hnd, nat]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- generate invalid ciphertext

send :: [user, party, hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- user sends message to adversary

adv\_send :: [party, user, hnd]  $\Rightarrow$  (hnd,  $\delta$  mLibState) S

-- adversary sends message to user

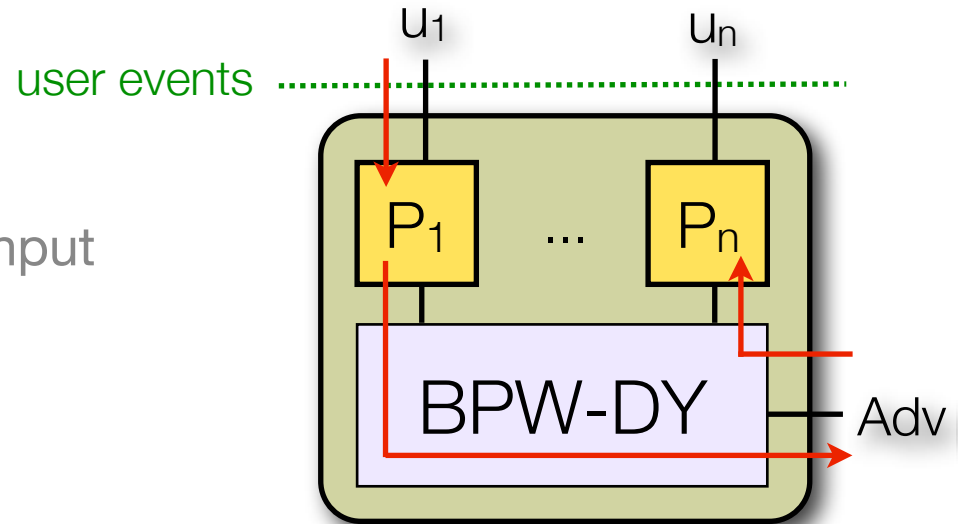
- simplified state: **knowledge map for each party** maps handles to messages
- **15 local functions** for message construction, destruction and queries
- **4 special adversary functions**, including message parsing
- **2 send functions** for message transmission between user's and adversary

# BPW Model Formalization

## Protocol system

### Protocol components $P_i$

- maintains a **local state**
- **user event handler** for user input;  
**network event handler** for adversary input
- call **BPW** model **local functions** to
  - ▶ **parse input** messages and
  - ▶ **construct reply** messages
- event handlers output to user or adversary

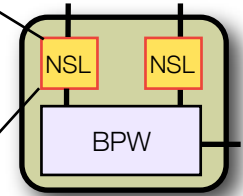


### Complete protocol system

- **system event handlers**: protocol handlers **composed with BPW send functions**
  - ▶ **send functions** exchange messages between protocol handlers and adversary/network
  - ▶ user interface events recorded in **user event trace** (for property specification)
- **local adversary functions**: message construction/destruction and queries

# Protocol Case Study: Needham-Schroeder-Lowe

$A \rightarrow B: \{N_A, A\}_{K_B}$   
 $B \rightarrow A: \{N_A, N_B, B\}_{K_A}$   
 $A \rightarrow B: \{N_B\}_{K_B}$



## Needham-Schroeder-Lowe protocol (NSL)

**record** `ustate` = nonces :: party  $\Rightarrow$  hnd set      -- local state of each protocol component

NeedhamSchroederLowe :: (party, party, party, ustate) protocol  
NeedhamSchroederLowe A  $\equiv$  (|

`proto_user_handler` =  $\lambda B$ .      -- construct the first NSL message

**do** `na`  $\leftarrow$  gen\_nonce (User A); **do** `z`  $\leftarrow$  add\_nonce A B na;  
**do** `nameA`  $\leftarrow$  store (User A) A; **do** `m`  $\leftarrow$  pair (User A) (na, nameA)  
**do** `em`  $\leftarrow$  encrypt (User A) (pke (User A) B) m;  
**return** (pToNet B em)

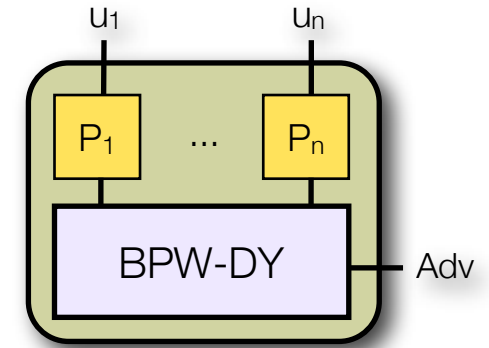
`proto_net_handler` =  $\lambda B$  m.      -- respond to incoming NSL messages

**do** `pm`  $\leftarrow$  parse\_msg A B m;  
**case** `pm` **of**  
  `msg1` nb nameB  $\Rightarrow$  mk\_msg2 A B nb  
| `msg2` na nb nameB  $\Rightarrow$  mk\_msg3 A B nb  
| `msg3` nb  $\Rightarrow$  **return** (pToUser B)

|)

# BPW Model Formalization (V2)

## Discussion



## Assessment and remaining problems

- 😊 **much better proof automation:** many properties of BPW model interface functions proved automatically (using WP calculus and equational reasoning)
  - 😞 **ad-hoc protocol specifications:** no support for defining protocol handlers
  - 😞 **fine granularity, side effects, and indirect access to messages by handles**
- Result: **proofs ca. 2 orders of magnitude larger** than Paulson's!

## Proposed solution: more abstraction!

- 👮 We present a solution for a class of protocols specified as **Role-Based Protocol Specifications** that addresses all of these problems!

Resulting model independent of BPW model and based on coarse-grained, side-effect free operations that work directly on messages.

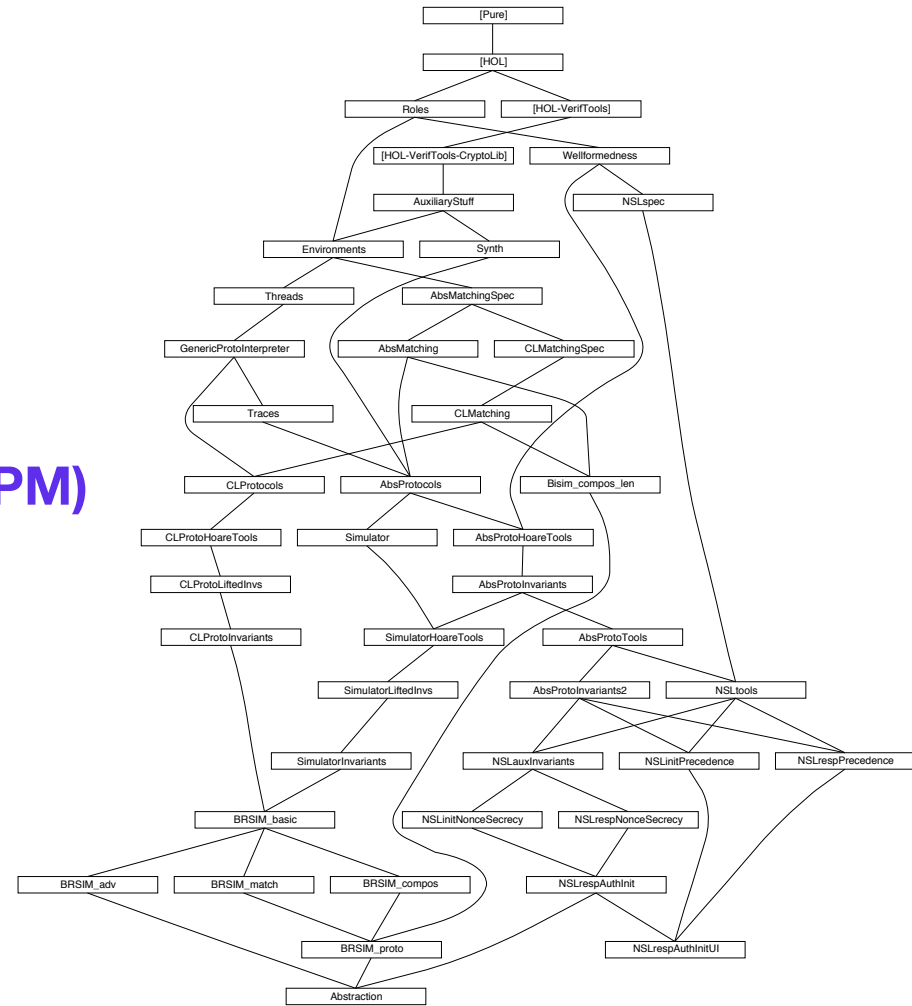
# Overview

## Part I: Formalizations of the BPW Model

## Part II: A More Abstract Protocol Model (APM)

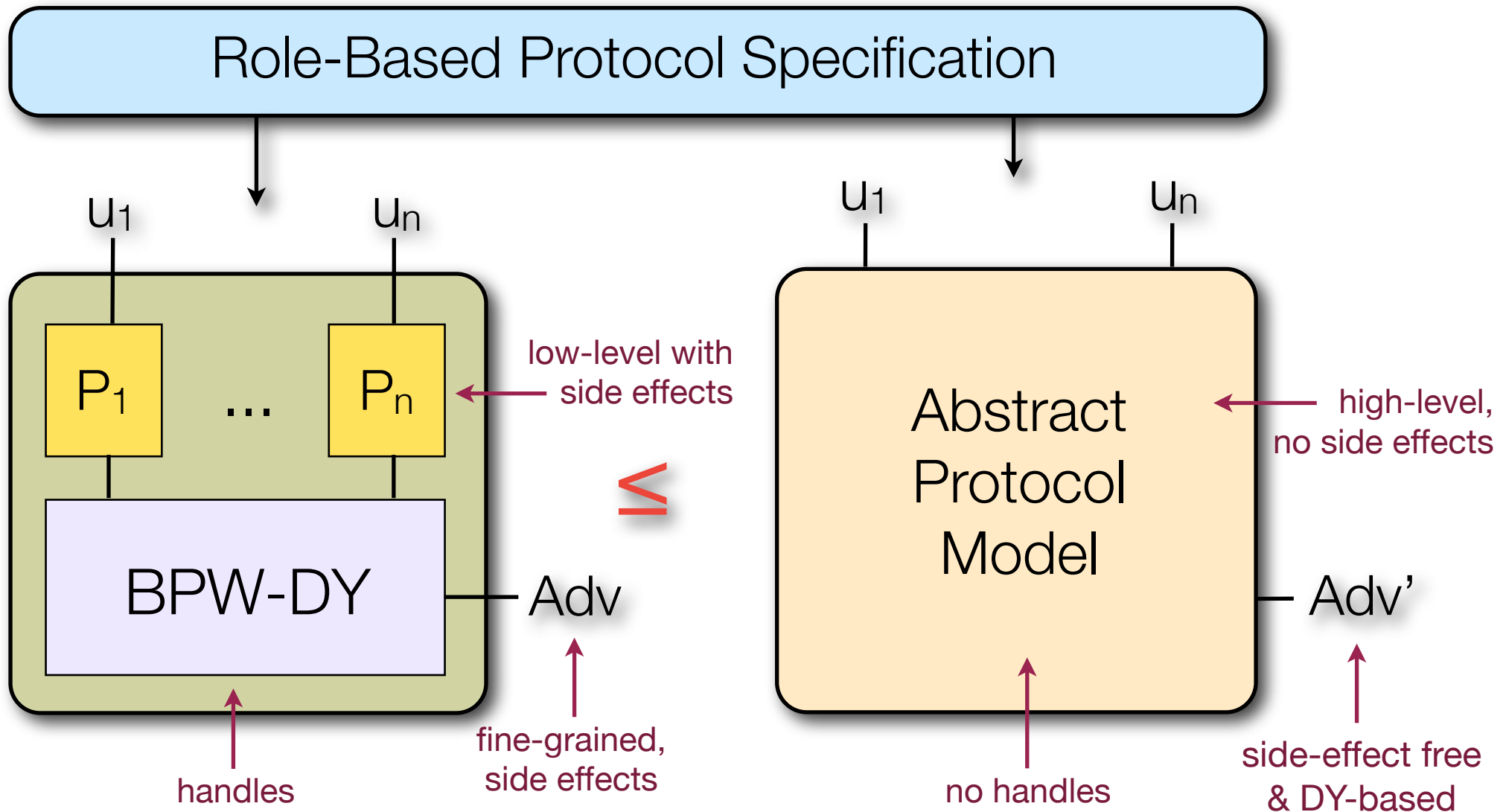
- role-based protocol specifications
- two interpretations and their relation

## Part III: Reasoning in the APM

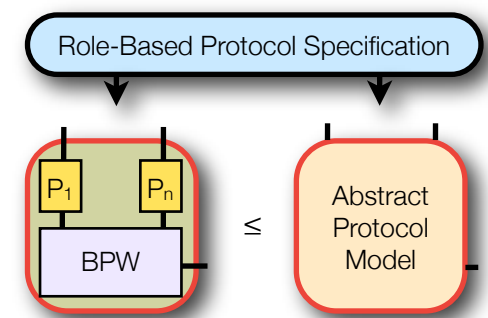


Abstract Protocol Model +  
BRSIM Proof + Case Study

# Overview of Abstraction Step & Problem Resolution



# Adversaries: Abstract and Concrete



## Concrete Adversary

- refers to messages using **handles**
- **fine granularity**: numerous interface functions, almost all with **side effects**
- reflected in **fine-grained proofs**, lacking abstraction

```
gen_nonce :: party => (hnd, δ mLibState) S
gen_enc_key_pair :: party => (hnd × hnd, δ mLibState) S
encrypt :: [party, hnd, hnd] => (hnd, δ mLibState) S
decrypt :: [party, hnd, hnd] => (hnd, δ mLibState) S
store :: [party, δ] => (hnd, δ mLibState) S
pair :: [party, hnd × hnd] => (hnd, δ mLibState) S
first, second :: [party, hnd] => (hnd, δ mLibState) S

adv_parse :: hnd => (δ pContent, δ mLibState) S
adv_garbage :: nat => (hnd, δ mLibState) S
adv_invalid_ciph :: [hnd, nat] => (hnd, δ mLibState) S

send :: [user, party, hnd] => (hnd, δ mLibState) S
adv_send :: [party, user, hnd] => (hnd, δ mLibState) S
```

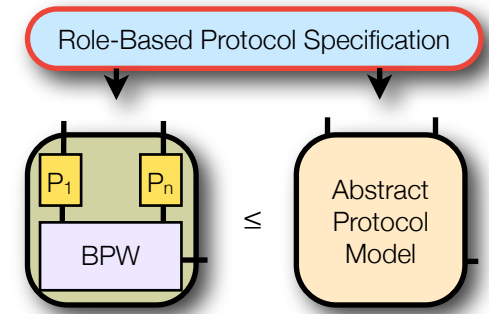
## Abstract adversary

```
adv_deriveA m ≡
  do enforce (λt. m ∈ synth (analyze (sees Adv (trace t))));
  logA (eLocal Adv m)
```

- manipulates messages terms directly **without handles**
- **side-effect-free derivability check** using standard **Dolev-Yao closure operators**  
same check performed before calling user's network input handler
- **derivable messages logged** as local (as here) or network input events **in trace**

# Role-Based Protocol Specification

## Example: Initiator Role of NSL Protocol



### Thread State

script (for role: **I**)

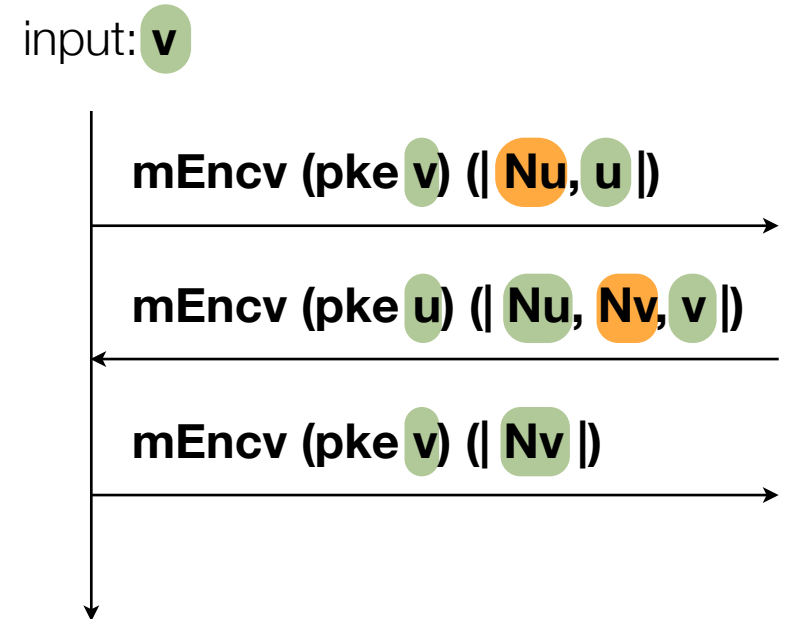
$$\left\{ \begin{array}{l} \text{rUsr } R \\ \text{rNet } R \text{ (pEnc } R \langle nI, I \rangle) \\ \text{rNet } R \text{ (pEnc } I \langle nI, nR, R \rangle) \\ \text{rNet } R \text{ (pEnc } R \ nR) \end{array} \right.$$

environment  $\rho$  (interprets roles and variables)

$\rho(I) = u$        $\rho(nI) = Nu$

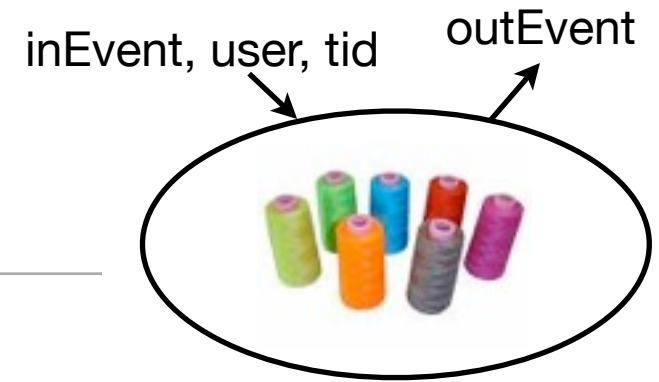
$\rho(R) = v$        $\rho(nR) = Nv$

### Event Trace



# Generic Event Handler

(aka Protocol Interpreter)



## Input/output and state

- input: **input event**, user and thread id
- output: **output event**
- state: **set of threads** for each user

## Parameters

- kind of **messages** (handles vs terms)
- **pattern matching** and **message composition** algorithms
- thread **lookup**, **state updater**

## 1) Thread lookup

- ▶ **check existence of thread** for given user and thread identifier (we assume that thread scheduling done externally)

## 2) Protocol step

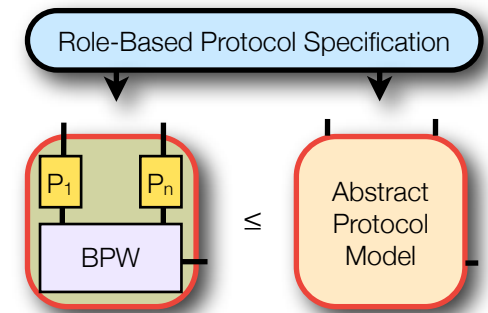
- ▶ **match input event** to input pattern
- ▶ **construct output event** according to output pattern

$$\left\{ \begin{array}{l} \underline{rNet} \ R \ (\underline{pEnc} \ I \ \langle nI, nR, R \rangle) \\ \underline{rNet} \ R \ (\underline{pEnc} \ R \ nR) \\ \vdots \\ \vdots \end{array} \right.$$

## 3) State update

- ▶ **update executed thread**: environment and remaining script

# BPW-based versus Abstract Protocol Model Message Handling



## Instantiation#1: BPW-based protocol model

- **messages** are **handles**
- **pattern matching** and **message composition** use **local BPW model functions**

### Pattern matching for ciphertexts

```

cl_match cKt rho u eh (pEnc r p) =
  do h ← decrypt (User u) (ske cKt (iusr rho r)) eh;      (* decrypt *)
  cl_match cKt rho u h p                                   (* match plaintext *)
  
```

## Instantiation #2: Abstract protocol model

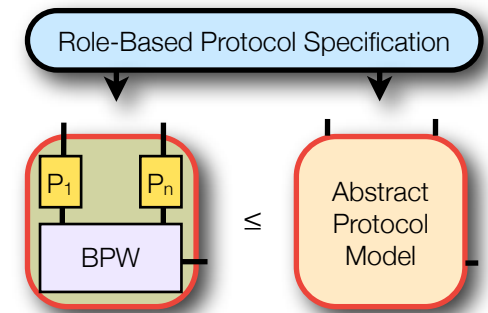
- **messages** are **algebraic terms**
- **pattern matching** and **message composition** **directly on messages**

### Pattern matching for ciphertexts

```

amatch (aKt, rho, mEncv tg k m, pEnc r p) =
  enforce (aKt (iusr rho r) = k);                          (* check key *)
  amatch (aKt, rho, m, p)                                  (* match cleartext *)
  
```

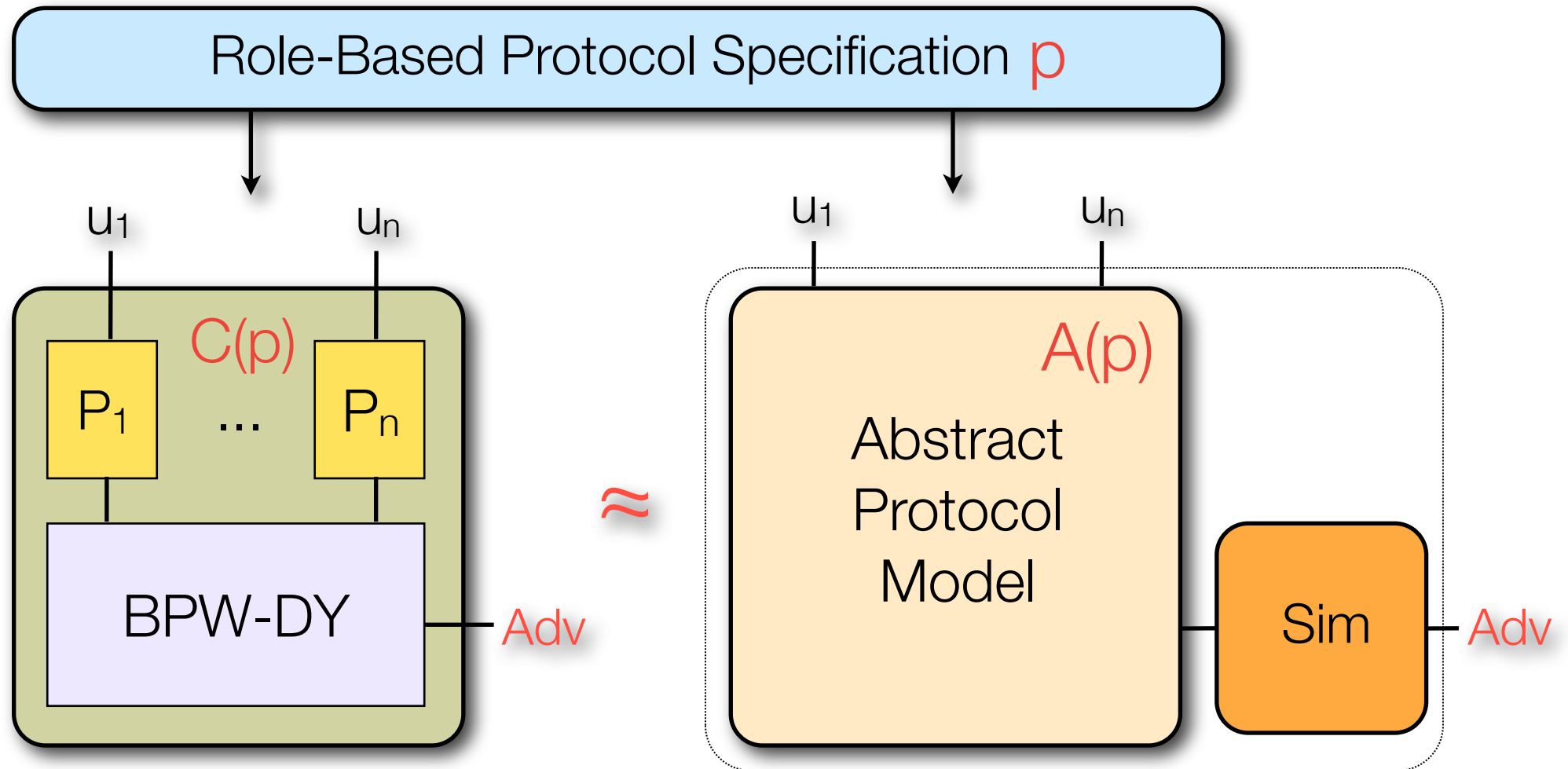
# BPW-based versus Abstract Protocol Model



	BPW-based	Abstract
messages	handles	message terms
message manipulation & adversary	fine-grained, low-level, side effects	atomic, high-level, no side effects
system-level handlers	protocol interpreter composed with send funs (additional side effect)	call protocol interpreter; derivability check <div style="border: 1px solid black; border-radius: 10px; padding: 5px; display: inline-block;"> <code>do enforce (λt. m ∈ synth (analyze (sees Adv (trace t ))));</code> </div>
trace	user I/O events (extra adversary knowledge map)	all events (replaces adversary knowledge map)
reusable general results	few, difficult	many, easy

# BRSIM Result Overview

**Theorem.** We have  $C(p) \leq A(p)$  for all well-formed, role-based protocols  $p$ .



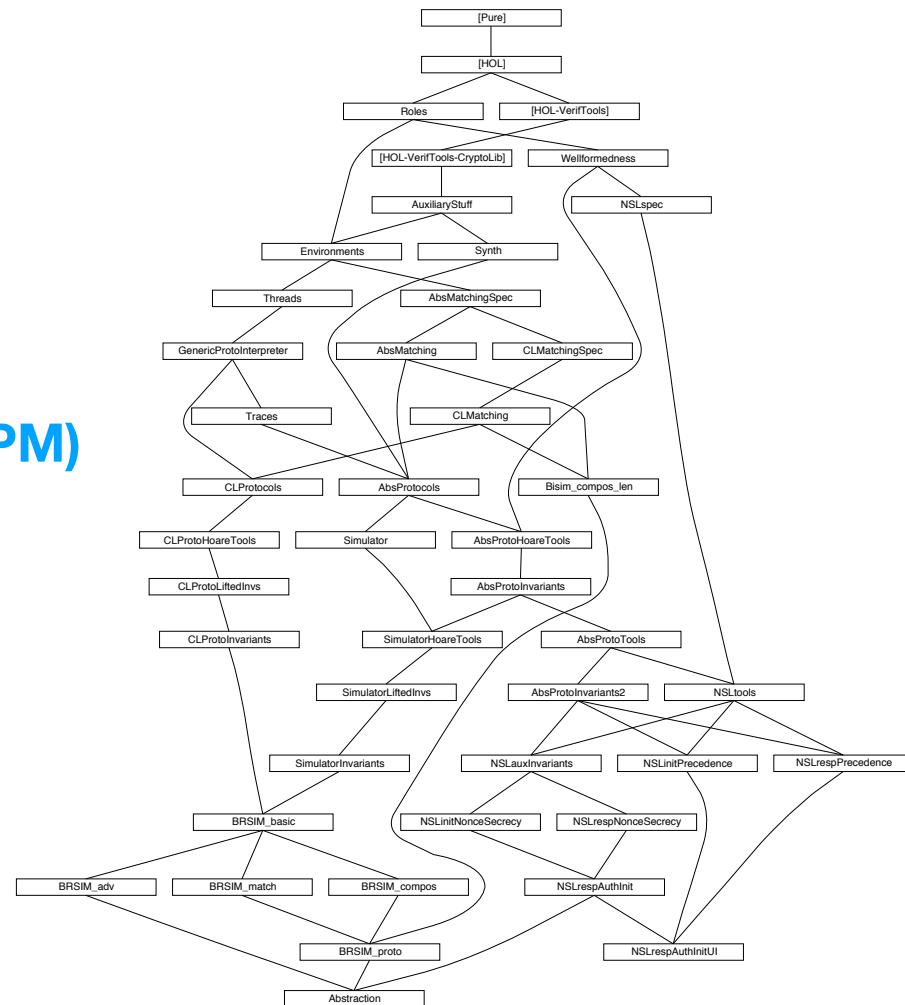
# Overview

## Part I: Formalizations of the BPW Model

## Part II: A More Abstract Protocol Model (APM)

## Part III: Reasoning in the APM

- general proof techniques
- case study: Needham-Schroeder-Lowe public-key protocol



Abstract Protocol Model +  
BRSIM Proof + Case Study

# NSL Protocol on Abstract Model

## Invariant Specifications

$$\begin{aligned} A \rightarrow B: & \{N_A, A\}_{KB} \\ B \rightarrow A: & \{N_A, N_B, B\}_{KA} \\ A \rightarrow B: & \{N_B\}_{KB} \end{aligned}$$

### Invariant: Responder Authenticates Initiator

$$\begin{aligned} \text{respAuthInitUI} \equiv & \{s \mid \forall A B. \\ & \text{eProtoOut } B \text{ (tUsr (mParty (User } A)) \in \text{set (trace } s) \\ & \longrightarrow \text{eProtoInp } A \text{ (tUsr (mParty (User } B)) \in \text{set (trace } s) \} \end{aligned}$$

- when user **B** finishes protocol, presumably with user **A**, then user **A** did indeed start a protocol run with user **B**

### Invariant: Initiator nonce secrecy

$$\begin{aligned} \text{nslInitNonceSecrecy} \equiv & \{s. \forall A B \text{ tg } N_A. \\ & \text{eProtoOut } A \text{ (tNet (User } B) \text{ (mEncv tg (aKt (User } B)) \{mNonce } N_A, \text{ mParty (User } A)\}) \\ & \in \text{set (trace } s) \\ & \longrightarrow \text{mNonce } N_A \notin \text{analyse (sees Adv (trace } s)) \} \end{aligned}$$

- whenever trace contains a log of **NSL message #1** sent by user **A** to user **B**:
- then the **nonce contained in this message is not known to the adversary**

# Overview of Invariant Proof Technique

---

## Tools for reduction of invariant proof to core lemmas

- logical characterization of protocol interpreter steps  
(proved once for APM using Hoare logic from monadic toolbox)
- use Hoare logic to decompose interface functions (event handlers & adv\_derive)

## User-side core lemma

- adding legal protocol output event to trace preserves invariant
- proved using NSL protocol step case analysis lemma  
(in HOL, proved once per protocol)

## Adversary-side core lemma

- adding message derivable by adversary to trace preserves invariant
- proved using adaptations of Paulson's theories of synth and analyse

**Reduction enabled by very limited and localized use of state.**

**Core lemmas closely correspond to inductive cases in Paulson's model.**

# NSL Protocol on Abstract Model

## Nonce Secrecy Proof (1)

$A \rightarrow B: \{N_A, A\}_{KB}$   
 $B \rightarrow A: \{N_A, N_B, B\}_{KA}$   
 $A \rightarrow B: \{N_B\}_{KB}$

### Step 1: Decomposition using Hoare Logic

```
{nslNonceSecrecy  $\cap$  auxInvariants}  
  do logA (eProtoInp u (tUsr m));  
  do thda  $\leftarrow$  lookup_thread u tid;  
  do (thdb, oev)  $\leftarrow$  proto_step NSLproto u (tUsr m) thda;  
  do update_thread u tid thdb;  
  logA (eProtoOut u oev)  
{nslNonceSecrecy}
```

user-side core lemma  
arises here

- system-level **user input handler**, unfolded down to protocol step (proto\_step)

### Proof strategy

- use **sequential composition rule of Hoare logic** used to decompose proof
- **user-side core lemma** arises from applying the **consequence rule of HL**:  
**post-condition of protocol\_step** implies **precondition of thread\_update**
- **reduction** to core lemma **completely straightforward** (could be automated)

# NSL Protocol on Abstract Model

## Nonce Secrecy Proof (2)

$A \rightarrow B: \{N_A, A\}_{KB}$   
 $B \rightarrow A: \{N_A, N_B, B\}_{KA}$   
 $A \rightarrow B: \{N_B\}_{KB}$

### Step 2: Prove User-side Core Lemma using case analysis on protocol step

$\llbracket s \in \text{proto\_step\_spec NSLproto } u \text{ (tUsr } m) \text{ thda } \text{oev} \text{ thdb};$   
 $s \in \text{nslInitNonceSecrecy}; s \in \text{auxInvariants}$   
 $e\text{ProtoInp } u \text{ (tUsr } m) \in \text{set (trace } s) \rrbracket$   
 $\implies$   
 $s(\text{ trace := } \text{oev} \# \text{ trace } s \text{ }) \in \text{nslInitNonceSecrecy}$

- `proto_step_spec`: post-condition characterizing protocol step (`proto_step`)
- also assume: invariants and trace contains `input from user`
- prove: addition of `output event` to trace preserves nonce secrecy

### Next proof step

- apply `NSL protocol step case analysis rule` (proved once per protocol)
- eliminates `proto_step_spec`, yields one `case for each role and protocol step`, i.e., case analysis on ways trace can be extended.

# NSL Protocol on Abstract Model

## Nonce Secrecy Proof (3)

$A \rightarrow B: \{N_A, A\}_{KB}$   
 $B \rightarrow A: \{N_A, N_B, B\}_{KA}$   
 $A \rightarrow B: \{N_B\}_{KB}$

### Step 3: Example subgoal for first step in initiator role

$\llbracket s \in \text{nslInitNonceSecrecy}; s \in \text{auxInvariants}$   
 $\text{eProtoInp } A (\text{tUsr } (\text{mParty } B)) \in \text{set } (\text{trace } s); \text{ wf\_aThread NSLproto } A \text{ thda};$   
 $\text{rol thdb} = I; \text{ iusr } (\text{env thdb}) R = \text{Some } B; \text{ imsg } (\text{env thda}) = \text{empty};$   
 $\text{imsg } (\text{env thdb}) \text{ nl} = \text{Some } (\text{mNonce } N_A); \text{ imsg } (\text{env thdb}) \text{ nR} = \text{None};$   
 $\text{mNonce } N_A \notin \text{parts } (\text{sees Adv } (\text{trace } s)) \rrbracket$

$\implies$   
 $s(\text{ trace } :=$   
 $\text{eProtoOut } A (\text{tNet } B (\text{mEncv } \text{tg } (\text{aKt } B) \{\text{mNonce } N_A, \text{mParty } (\text{User } A)\})) \# \text{ trace } s$   
 $) \in \text{nslInitNonceSecrecy}$

- application of **NSL protocol step case analysis rule** yields:
  - ▶ **concrete protocol messages** substituted in **input** and **output** events
  - ▶ hypotheses contain precise description of protocol step  
Here: e.g., **initiator nonce is generated in this step, meaning that it is fresh**
- subgoal above holds **by freshness of initiator nonce** (last hypothesis) and fact that **users' private keys are not known to adversary**

# NSL Case Study

## Summary and Model Comparison

$A \rightarrow B: \{N_A, A\}_{KB}$   
 $B \rightarrow A: \{N_A, N_B, B\}_{KA}$   
 $A \rightarrow B: \{N_B\}_{KB}$

### DAG-based BPW protocol model



- **infeasible** and was abandoned
- problems: irrelevant details in the model, lack of abstract property specs

### Term-based BPW protocol model



- **feasible, but tedious**; (large proof, >130 pages Isabelle/HOL PDF doc)
- problems: **fine-grained operations** with **side effects**

### Abstract protocol model



- **feasible and comparable to Paulson's proofs** in a simpler and weaker model
- enabled by straightforward **reduction of invariant proofs to two core lemmas**, very similar to **Paulson's inductive cases** for protocol steps and adversary
- **core lemmas for the 9 NSL invariants** fit on **~10 pages** (overall still ~80 pages, but lengthy reduction proofs mechanic, should be automatable)

**Chosen abstractions allow us to close the gap with standard symbolic models w.r.t. conciseness of properties and complexity of resulting proofs.**

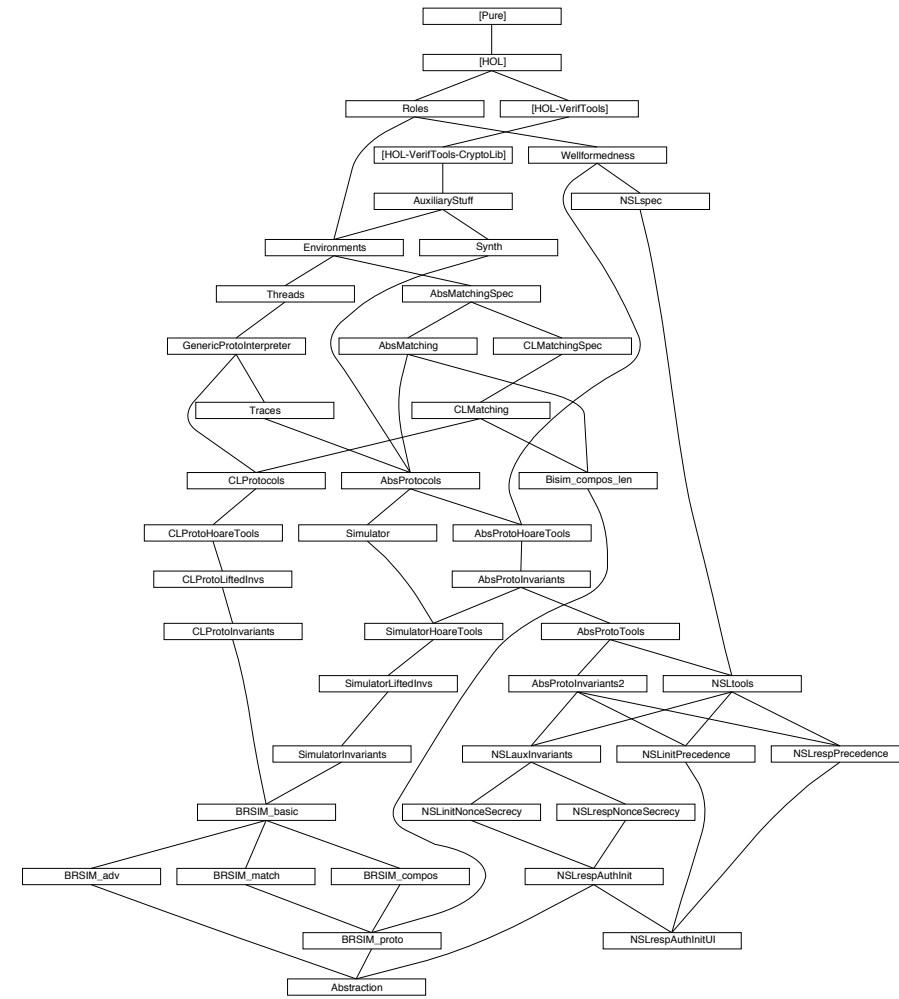
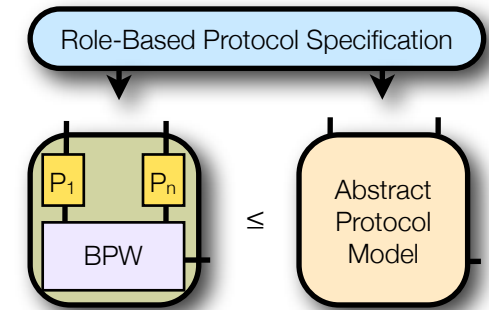
# Some Statistics

## APM+BRSIM proof & NSL case study

- 40 theories (boxes in graph)
- 150 definitions
- 1200 lemmas and theorems
- 18k lines of Isabelle/HOL
- 360 pages PDF documentation

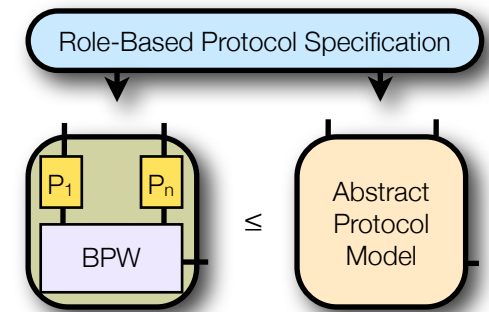
## Overall

Module	Theories	Pages	Lines	%
Modeling & verification tools	9	54	2.7k	7
DAG-based model + BRSIM	10	119	6k	16
Term-based model	17	96	4.8k	13
Term-based NSL	14	136	6.8k	18
APM + BRSIM	31	266	13.3k	35
APM-based NSL	9	83	4.2k	11
<b>Total</b>	<b>90</b>	<b>754</b>	<b>37.8k</b>	<b>100</b>



Abstract protocol model, RSim proof, and NSL case study

# Conclusions



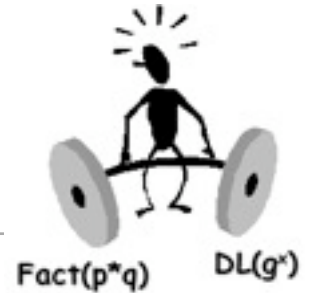
- via series of abstractions, **closed the gap** between the BPW model and **standard Dolev-Yao models**, both conceptually and practically
- **general infrastructure** supports **concise, mechanized formal proofs** in a very general, cryptographically-sound framework as well as establishing relationships between different protocol models
- case study shows that **formal cryptographically-sound proofs are actually possible and with a complexity comparable to standard Dolev-Yao models!**

## Future work

- **non-standard aspects** remain, e.g., probabilistic/tagged encryption, invalid cipher text, and message length restrictions. Which of these are **essential**? (cf. results of Cortier et al. on randomness tags).
- **role-based protocols** are a restriction. How far can this be **generalized**?
- **reasoning infrastructure**: automatic tactics and lemma generation
- **case studies**: further validate model and reasoning tools

# Questions??

---



## Collaborators

- ▶ David Basin
- ▶ Michael Backes
- ▶ Birgit Pfitzmann
- ▶ Michael Waidner

Project web page:

[www.zisc.ethz.ch/research/securityprotocolproofs](http://www.zisc.ethz.ch/research/securityprotocolproofs)