UC-PACT: Universal Composability for Preventing Adversarial Composition Techniques

Modeling the Needham-Schroeder Public-Key Protocol in EasyUC

Robert Graham

13 Aug 2025

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Naval Information Warfare Center Pacific (NIWC Pacific) under N66001-22-C-4020.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA and NIWC Pacific.

RIVERSIDE RESEARCH

OVERVIEW

- Background
  - Notation for public-key cryptographic protocols
  - The Needham-Schroeder public-key protocol
  - Analyzing Needham-Schroeder
  - Attacks on Needham-Schroeder
- EasyUC models
  - Untrusted network communication
  - EasyCrypt support theories
  - Needham-Schroeder
- Discussion
  - Analysis of the model
  - Alternative ideal functionalities
  - Revised EasyUC model of Needham-Schroeder
- Modeling the MITM attack
- References

Recurring themes

1. **What security goals are we trying to achieve?**
2. **How do we know we've achieved them?**

BACKGROUND

NOTATION FOR PUBLIC-KEY CRYPTOGRAPHIC PROTOCOLS

- Principals: **A**, **B**, …, and the adversary, **Adv**
- Nonces
  - A randomly generated integer used “once” (e.g., per session)
  - **Na** denotes a nonce generated by principal A
- Key pairs
  - **Ka** denotes the public key for principal A; **Ka^{-1}** denotes the corresponding private key
- Encryption/decryption
  - Let **M** be any plaintext message, such as A, Na
  - **{M} Ka** is the encryption of M using A’s public key
    - Anyone can do the encryption
    - Only A can decrypt it and recover M: { {M} Ka} Ka^{-1} = M
  - **{M} Ka^{-1}** is the signature containing M using A’s private key
    - Only A can sign M
    - Anyone can verify the signature and recover M: { {M} Ka^{-1}} Ka = M

THE NEEDHAM-SCHROEDER PUBLIC-KEY PROTOCOL

- Needham & Schroeder, "Using Encryption for Authentication in Large Networks of Computers," 1978
- Target functions
  - Authenticated interactive communication between two principals
    - Where *authenticated* means each principal has verified the identity of the other
  - Signed communication, in which the origin and integrity of a communication can be authenticated to a third party
- The adversary can alter or copy parts of messages, replay messages or emit false material, but cannot decrypt messages if it hasn't seen the corresponding key, guess a key, etc.
- Two principals, A and B, plus a certificate authority, S, containing public credentials

1a. A -> S: A, B
1b. S -> A: {Kb, B} Ks^{-1}

A looks up B's public key

2a. A -> B: {Na, A} Kb

3a. B -> S: B, A
3b. S -> B: {Ka, A} Ks^{-1}

B looks up A's public key

2b. B -> A: {Na, Nb} Ka

2c. A -> B: {Nb} Kb

The Needham-Schroeder Public-Key Protocol, cont.

- Now that A and B have authenticated each other, how do they carry on a conversation?
  - Double encryption

    A -> B: {{M} Ka^{-1}} Kb (or {A, {M} Ka^{-1}} Kb)

    B -> A: {{M) Kb^{-1}} Ka (or {B, {M} Kb^{-1}} Ka}

    Why bother with the protocol above, then?
  - Use the nonces (not clear from this paper or BAN89)

    A -> B: {Nb, M} Kb

    B -> A: {Na, M} Ka

Analyzing Needham-Schroeder

- Burrows, Abadi and Needham, “A Logic of Authentication,” 1989
- What security properties does the Needham-Schroeder protocol guarantee?
  - A authenticates S (Message 1b)
  - Kb is bound to B (Message 1b)
  - A authenticates B (Message 2b)
  - B authenticates S (Message 3b)
  - Ka is bound to A (Message 3b)
  - B authenticates A (Message 2c)*
  - Na and Nb are secrets between A and B (and trusted associates of A and B)
  - Ka is current (Message 1b)*
  - Kb is current (Message 2b)*
  - Na is fresh (Message 2a)
  - Nb is fresh (Message 2b)

  * Oops, not really
- The protocol further assumes that S and Ks are bound and known to A and B *a priori*

Which of these are essential and which are incidental to the use of PKE?

RIVERSIDE RESEARCH

Attacks on Needham-Schroeder

- The adversary can fool A (B) into accepting an old public key for B (A)

  1a. A -> S: A, B (Adv eavesdrops and saves)

  1b. S -> A: {B, Kb} Ks^{-1} (Adv eavesdrops and saves)

  … much later

  1a’. A -> S: A, B (Adv intercepts and replies)

  1b’. Adv -> A: {B, Kb} Ks^{-1}

  - The adversary can’t read the message, but can replay it

Result: A (B) uses a public key for B (A) that has expired or been revoked; B (A) may no longer have the corresponding secret key

Fix: Add a timestamp to Message 1b

- The adversary can fool B into believing it is A (Lowe95)

  2a. A -> Adv: {Na, A} Kadv

  2a’. Adv -> B: {Na, A} Kb

  2b’. B -> Adv: {Na, Nb} Ka

  2b. Adv -> A: {Na, Nb} Ka

  2c. A -> Adv: {Nb} Kadv

  2c’. Adv -> A: {Nb} Kb

Result (a “weird” machine):

A and Adv have a session

Adv and B have a session

but B thinks its session is with A

Both sessions use Na and Nb

Fix: Change Message 2b to {Na, Nb, B} Ka (a.k.a. Needham-Schroeder-Lowe)

Modeling Needham-Schroeder with CSP

- Gavin Lowe, “Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR,” 1996
- Defines Initiator, Responder and Intruder (not shown) processes:

```
INITIATOR(a,na) =
  user.a?b -> I_running.a.b ->
  comm!Msg1.a.b.Encrypt.key(b).na.a ->
  comm.Msg2.b.a.Encrypt.key(a)?na'.nb ->
  if na==na’ then
    comm!Msg3.a.b.Encrypt.key(b).nb ->
    I_commit.a.b -> session.a.b -> SKIP
  else
    STOP
```

```
RESPONDER(b,nb) =
  comm.Msg1?a!b.Encrypt.key(b)?na.a’ ->
  if a==a' then
    R_running.A.B ->
    comm!Msg2.b.a.Encrypt.key(a).na.nb ->
    comm.Msg3.a.b.Encrypt.key(b)?nb’ ->
    if nb==nb' then
      R_commit.a.b -> session.a.b -> SKIP
    else STOP
  else STOP
```

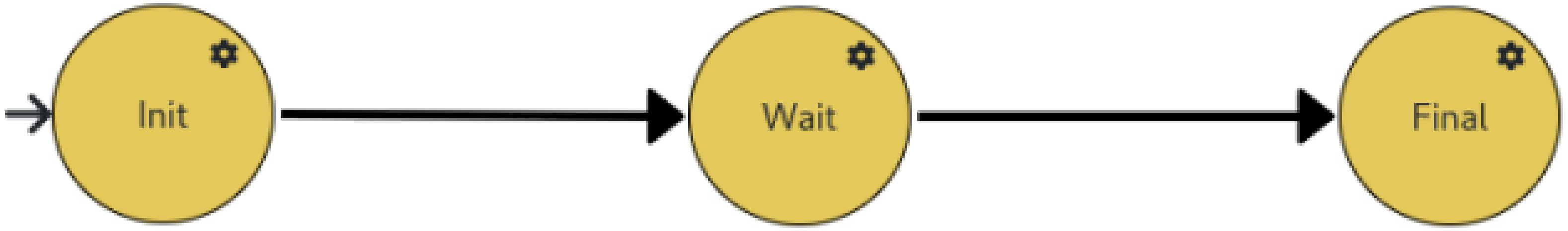
- A “specification” of authentication defines processes

```
AUTH_INIT = I_running.A.B -> R_commit.A.B -> AUTH_INIT
AUTH_RESP = R_running.A.B -> I_commit.A.B -> AUTH_RESP
```

- The FDR model checker finds a trace that violates AUTH_INIT

EasyUC Model of Needham-Schroeder

Ideal Functionality for an Untrusted Network



Init | FwdDir.D.fw_req
FwdAdv.fw_obs | Wait

if (envport pt2)

To State Arguments

pt1 (port) | pt1

Out Message Arguments

pt1 (port) | pt1
pt2 (port) | pt2
u (univ) | u

Wait | FwdAdv.fw_ok
FwdDir.D.fw_rsp | Final

pt2

Guard Description

To State Arguments

Out Message Arguments

pt1 (port) | pt1
u (univ) | u

Ideal Functionality for an Untrusted Network (Forwarding.uc)

```
direct FwDir' {
  in  pt1@fw_req (pt2 : port, u : univ).
  out fw_rsp (pt1 : port, u : univ)@pt2.
}
direct FwDir { D : FwDir' }
adversarial FwAdv {
  out fw_obs (pt1 : port, pt2 : port, u : univ)
  in  fw_ok (pt2 : port, u : univ)
}
functionality Forw implements FwDir FwAdv {
  initial state Init {
    match Message with
    | pt1@FwDir.D.fw_req (pt2, u) => {
        send FwAdv.fw_obs (pt1, pt2, u) and transition Wait (pt1)
      }
    end
  }
```


RIVERSIDE RESEARCH

IDEAL FUNCTIONALITY FOR AN UNTRUSTED NETWORK, CONT.

```
  state Wait (pt1 : port) {
    match Message with
    | FwAdv.fw_ok (pt2, u) => {
        send FwDir.D.fw_rsp (pt1, u)@pt2
        and transition Final.
      }
    | * => { fail. }
  }

  state Final {
    match Message with
    | * => { fail. }
  }
}
```

EasyCrypt "Support Theory" for Public-Key Encryption

- PKE.ec

```
type pk_t.                  (* public keys *)
type sk_t.                  (* secret keys *)
type ptxt_t.                (* plain text *)
type ctxt_t = ptxt_t.       (* cipher text/signature *)

op enc (pk: pk_t, p: ptxt_t): ctxt_t.
op dec (sk: sk_t, c: ctxt_t): ptxt_t.
op gen_pair : pk_t -> sk_t -> bool.

axiom pk_enc_dec (sk : sk_t) (pk : pk_t) (p : ptxt_t):
  gen_pair pk sk => dec sk (enc pk p) = p.
axiom pk_dec_enc (sk : sk_t) (pk : pk_t) (c : ctxt_t) :
  gen_pair pk sk => enc pk (dec sk c) = c.

hint simplify pk_enc_dec, pk_dec_enc.
```

EASYCRYPT "SUPPORT THEORY" FOR PUBLIC-KEY ENCRYPTION, CONT.

- PKE_EPDP.ec

```
require import PKE UCUniv.
(*---*) import UCEncoding.
```

EPDP = Encoding and Partial Decoding Pair

```
type ('a, 'b) epdp = {
  enc : 'a -> 'b;
  dec : 'b -> 'a option
}.
```

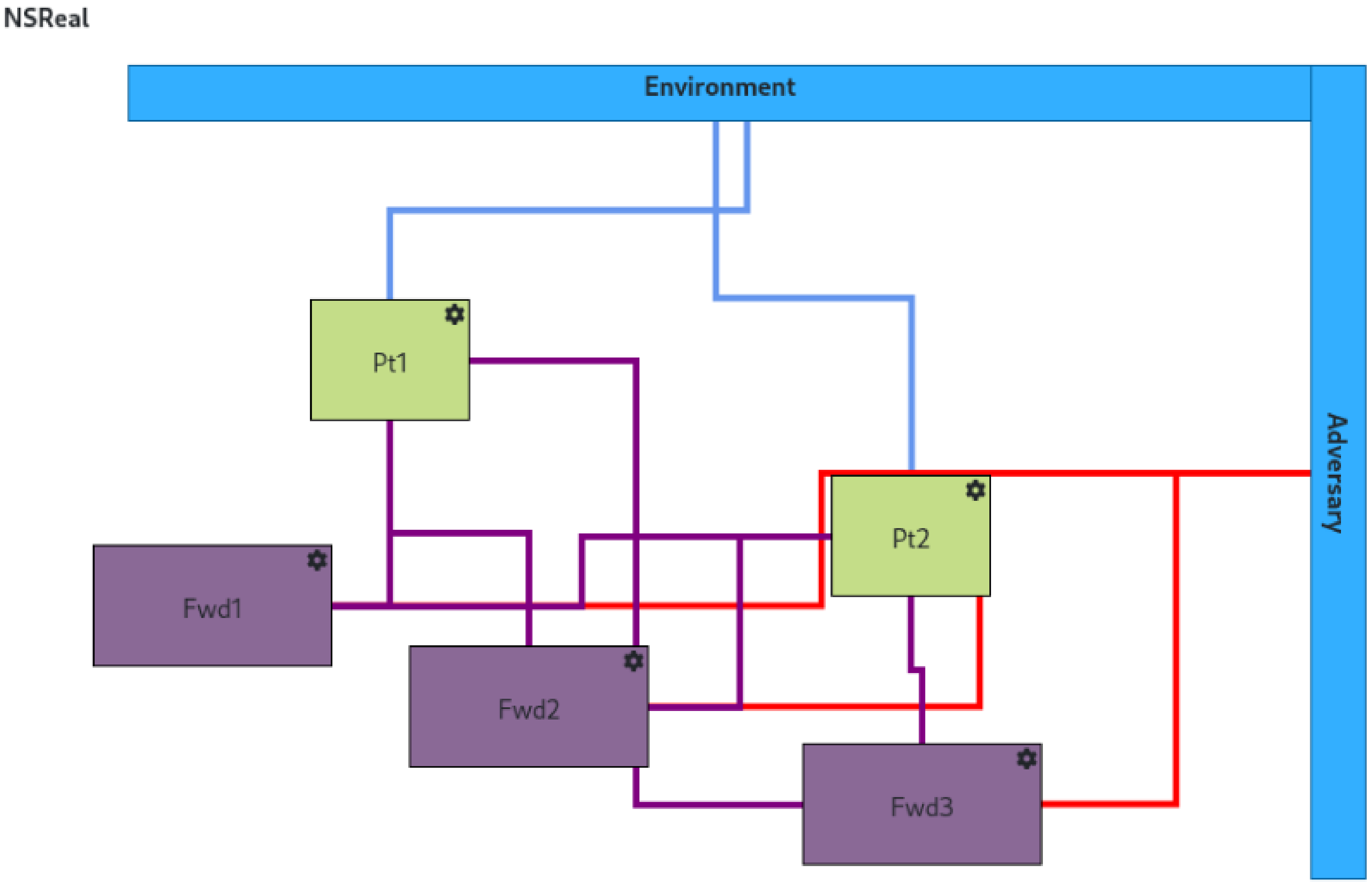
```
op [opaque smt_opaque] epdp_cipher_univ : (ctxt_t, univ) epdp.
axiom valid_epdp_cipher_univ : valid_epdp epdp_cipher_univ.
hint simplify valid_epdp_cipher_univ.

op [opaque smt_opaque]
  epdp_plain_pair_plain : (ptxt_t * ptxt_t, ptxt_t) epdp.
axiom valid_epdp_plain_pair_plain : valid_epdp epdp_plain_pair_pair.
hint simplify valid valid_epdp_plain_pair_plain.
```

EasyCrypt Support Theory for Needham-Schroeder

- NeedhamSchroeder.ec

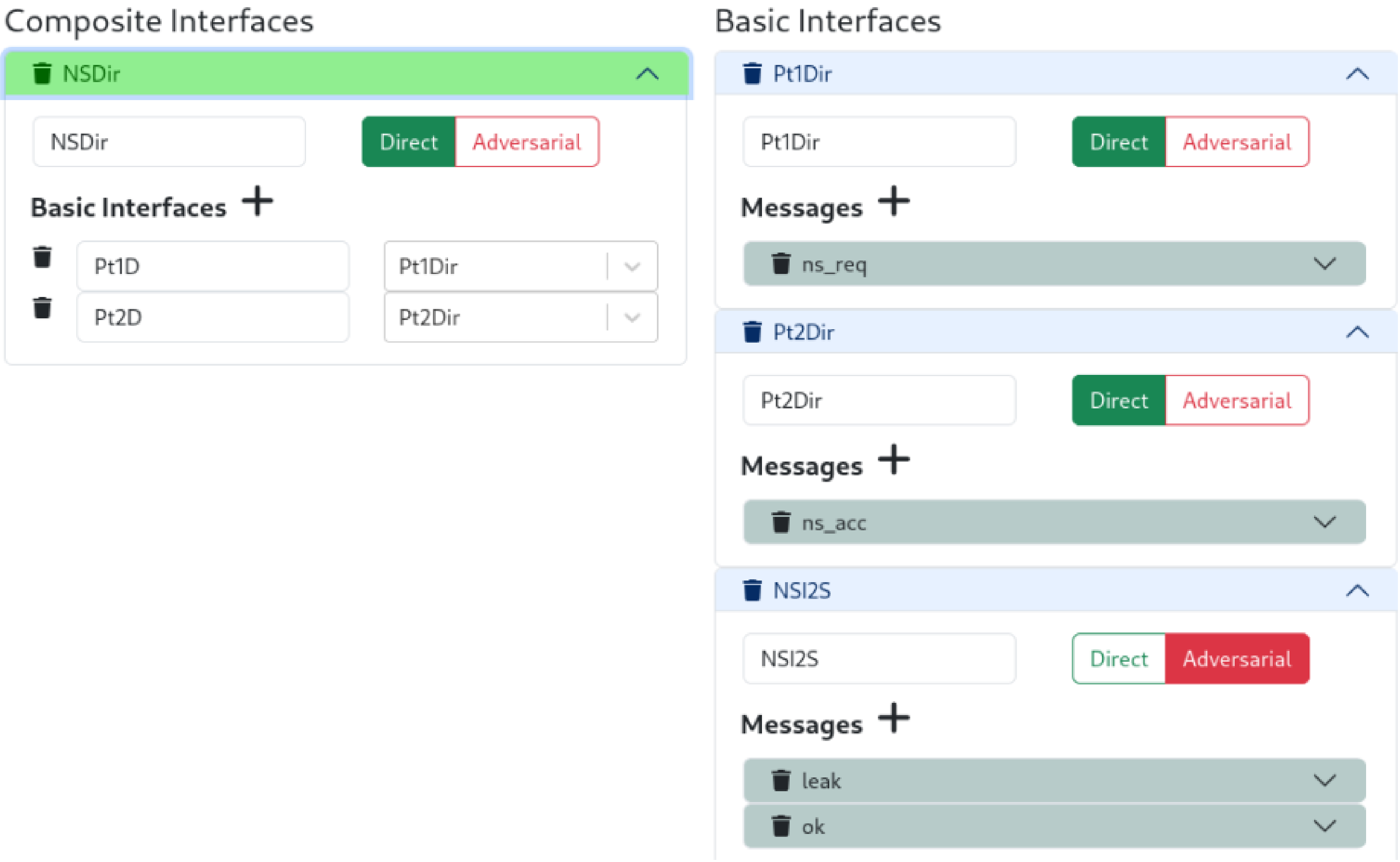
```
require import Distr Int PKE PKE_EPDP UCBasicTypes.

op nonce : int distr = drange 0 18446744073709551616. (* 2^64 *)

const pk_a, pk_b : pk_t.
const sk_a, sk_b : sk_t.

axiom gp_pk_sk_a : gen_pair pk_a sk_a.
axiom gp_pk_sk_b : gen_pair pk_b sk_b.
hint simplify gp_pk_sk_a, gp_pk_sk_b.

op [opaque smt_opaque] epdp_port_port_cipher_univ :
  (port * port * ctxt_t, univ) epdp =
  epdp_tuple3_univ epdp_port_univ epdp_port_univ epdp_cipher_univ.
lemma valid_epdp_port_port_cipher_univ :
  valid_epdp epdp_port_port_cipher_univ
  by rewrite /epdp_port_port_cipher_univ.
```

The Real World for Needham-Schroeder



RIVERSIDE RESEARCH

Interface Definitions



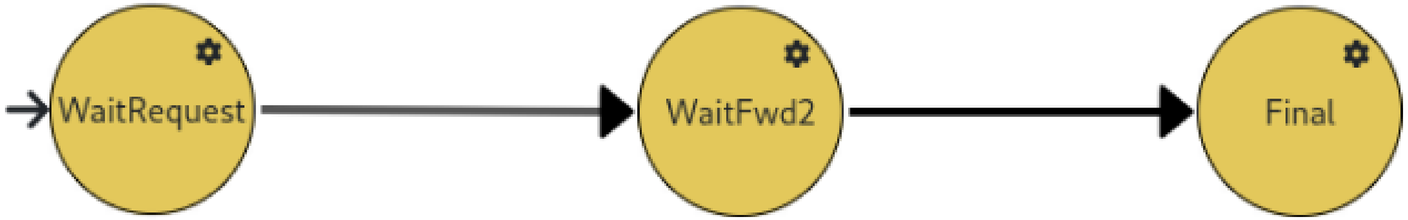
in pt1@ns_req (pt2 : port)

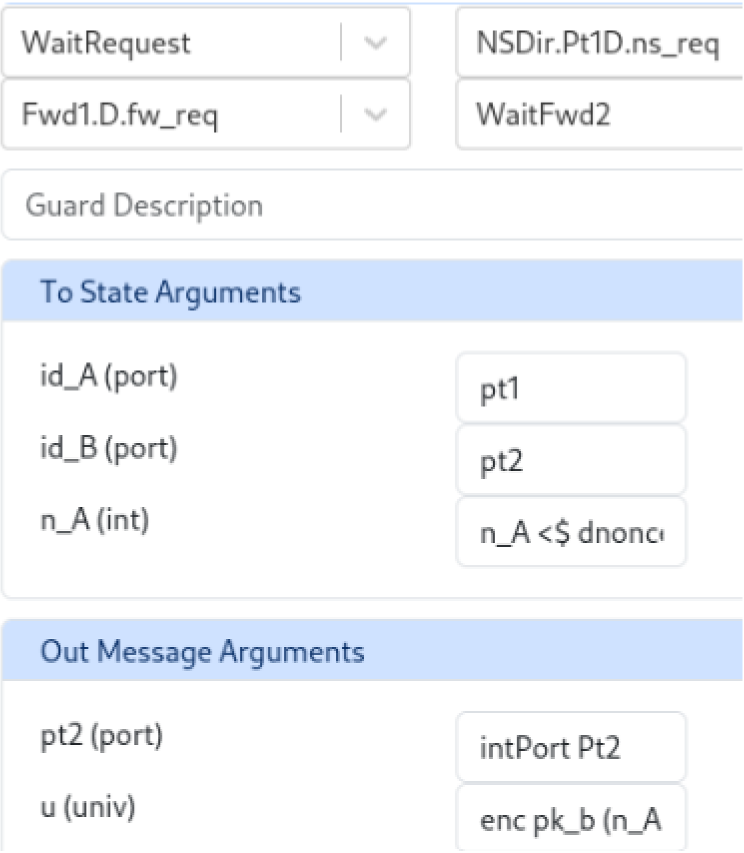
out ns_acc (pt1 : port)@pt2

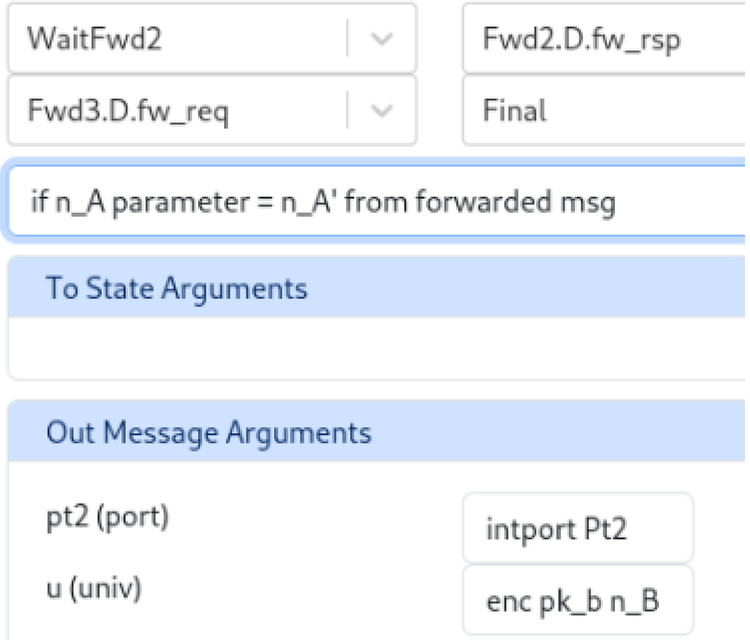
out leak (pt1 : port, pt2 : port)

in ok

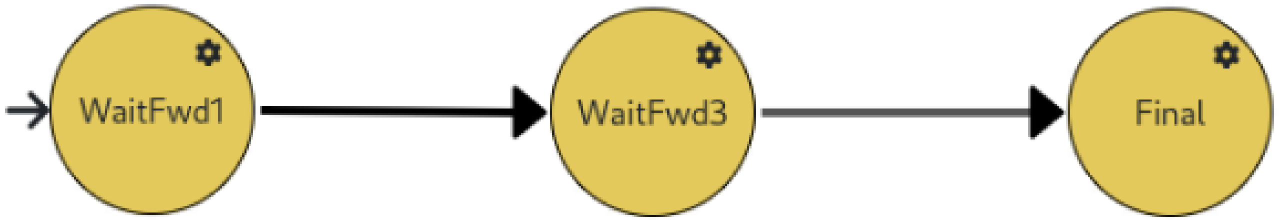
Party Pt1 State Machine

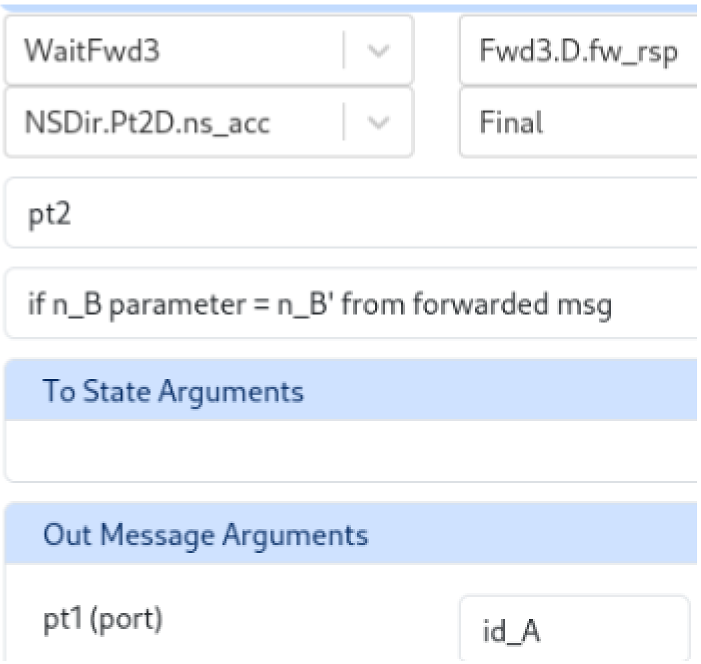






Party Pt2 State Machine



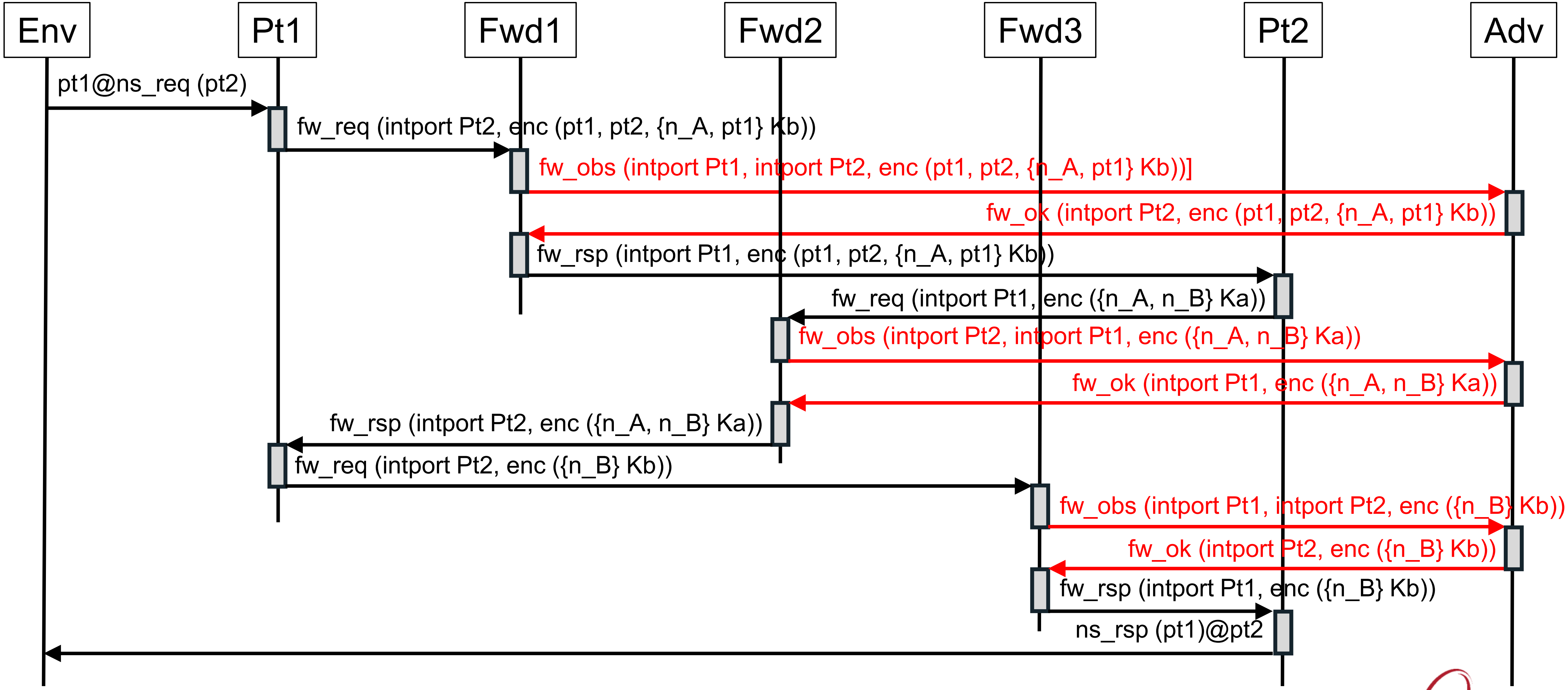


RIVERSIDE RESEARCH

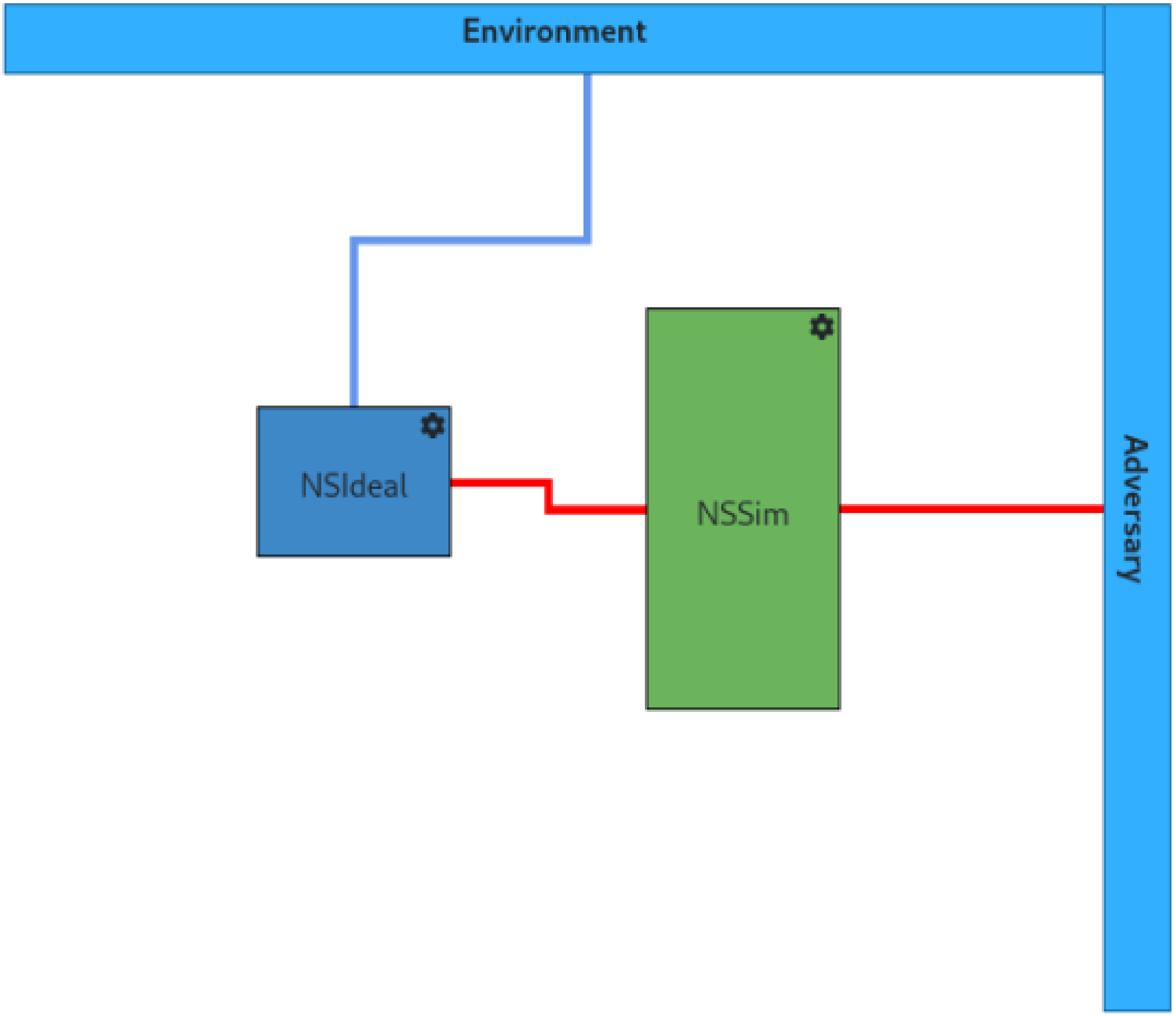
Sample DSL code from Party Pt2

```
state WaitFwd3 (id_A : port, id_B : port, n_A : int, n_B : int) {
  match message with
  | Fwd3.D.fw_rsp (_, u) => {
      match epdp_cipher_univ.`dec u with
      | Some ciphertext => {
          match epdp_int_plain.`dec (dec sk_b ciphertext) with
          | Some n_B’ => {
              if (n_B’ <> n_B) { fail. }
              else {
                send NSDir.Pt2D.ns_acc (id_A)@id_B
                and transition Final.
              }
            }
          | None => { fail. }
          end
        }
      | None => { fail. }
      end
    }
  | * => { fail. }
  end
}
```

Real World Sequence Diagram

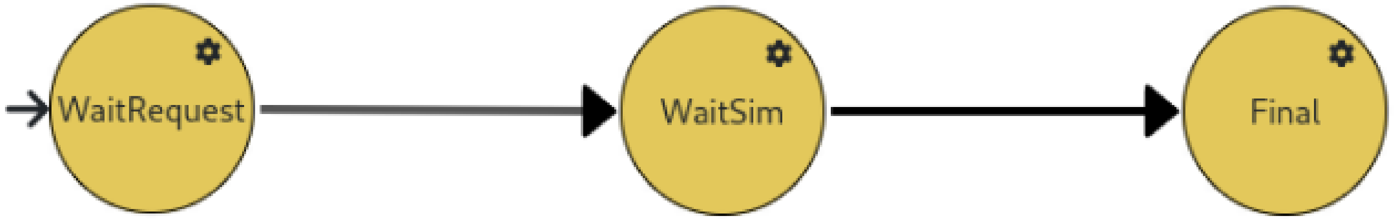


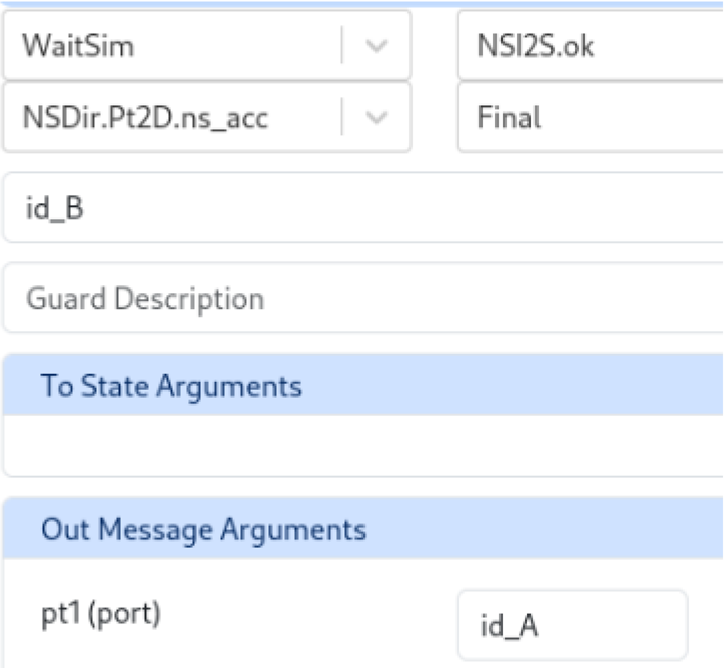
The Ideal World for Needham-Schroeder



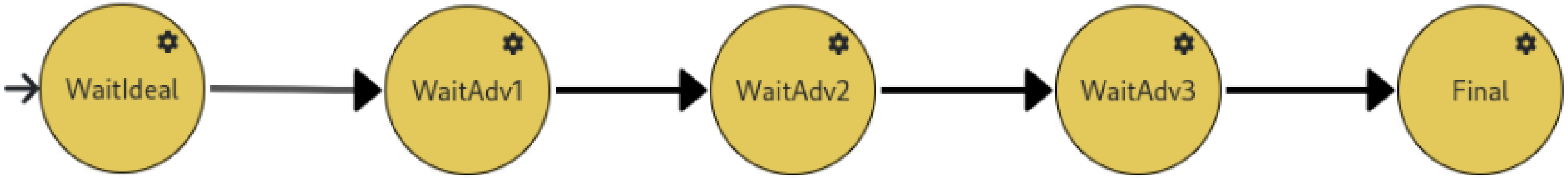
RIVERSIDE RESEARCH

NSIDEAL STATE MACHINE





NSSim State Machine



WaitIdeal	NSI2S.leak
Fwd1.FwdAdv.fw_obs	WaitAdv1

Guard Description

To State Arguments

id_A (port)	pt1
id_B (port)	pt2
n_A (int)	n_A <$ dnonc

Out Message Arguments

pt1 (port)	intport Pt1
pt2 (port)	intport Pt2
u (univ)	enc pk_b (n_A

WaitAdv1	Fwd1.FwdAdv.fw_ok
Fwd2.FwdAdv.fw_obs	WaitAdv2

if id_A param = id_A' from msg and id_B param

To State Arguments

n_A (int)	n_A
n_B (int)	n_B <$ dnonc

Out Message Arguments

pt1 (port)	intport Pt2
pt2 (port)	intport Pt1
u (univ)	enc pk_a (n_A,

WaitAdv2	Fwd2.FwdAdv.fw_ok
Fwd3.FwdAdv.fw_obs	WaitAdv3

if n_A param = n_A' from msg and id_B param

To State Arguments

n_B (int)	n_A

Out Message Arguments

pt1 (port)	intport Pt1
pt2 (port)	intport Pt2
u (univ)	enc pk_b n_B

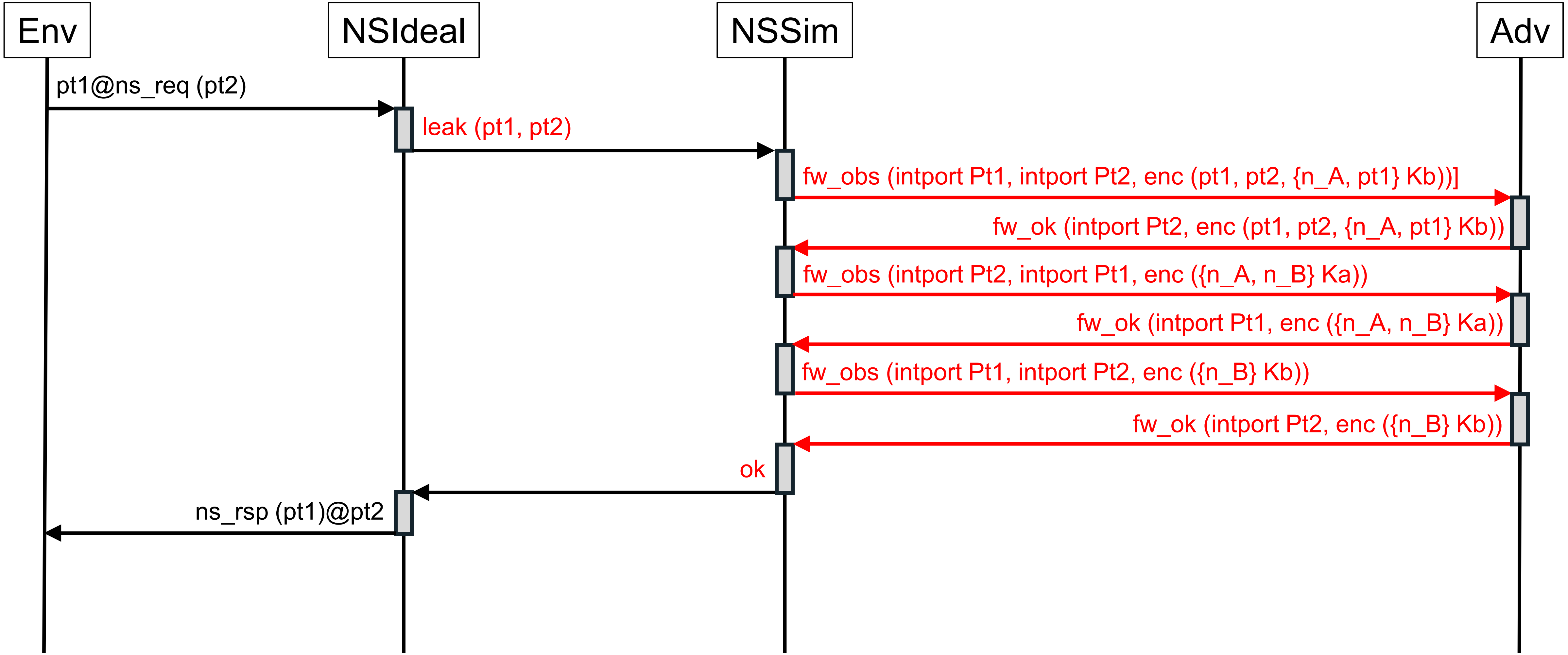
WaitAdv3	Fwd3.FwdAdv.fw_ok
NSI2S.ok	Final

if n_B param = n_B' from msg

To State Arguments

Out Message Arguments

IDEAL WORLD SEQUENCE DIAGRAM



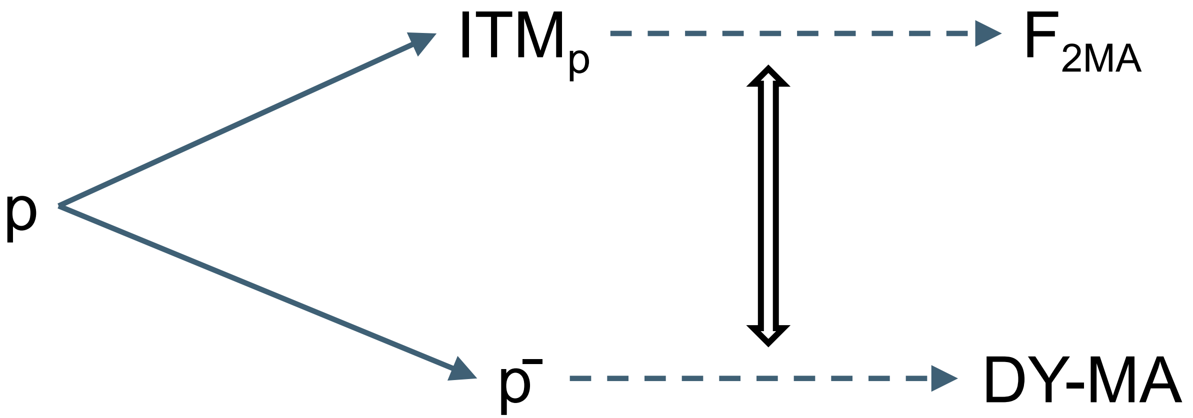
Discussion

Analysis of the EasyUC Model

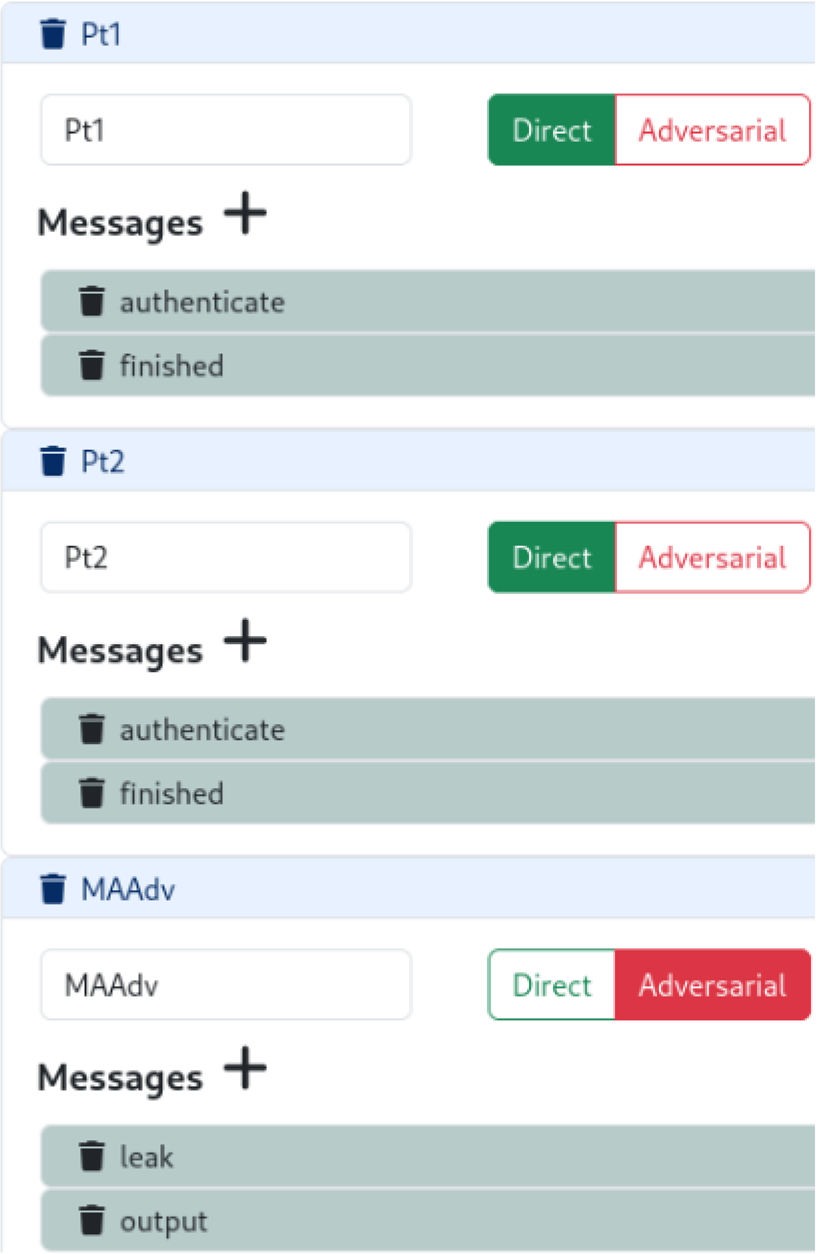
- What does the ideal functionality guarantee?
  - B authenticates A
  - A authenticates B
  - What definition of authentication is this?
- How would A and B carry on a conversation?
  - Maybe they should swap nonces…

Symbolic Analysis of UC Models

- Canetti and Herzog, "Universally Composable Symbolic Security Analysis," 2011
  - The Dolev-Yao model for symbolic encryption
    - Message algebra, symbolic protocols, adversary and executions
  - *Simple protocols* with 2 principals
    - Symbolic and UC semantics in terms of constructions
  - Symbolic analysis of UC mutual authentication
    - Theorem 14. A simple protocol p UC-realizes F_{2MA} iff the corresponding Dolev-Yao model $\bar{p}$ satisfies the symbolic criterion DY-MA
      - Informally, if A completes a session with B, then B has started a session with A



F_{2MA} in EasyUC



in pt1@authenticate (pt2 : port)

out finished@pt1

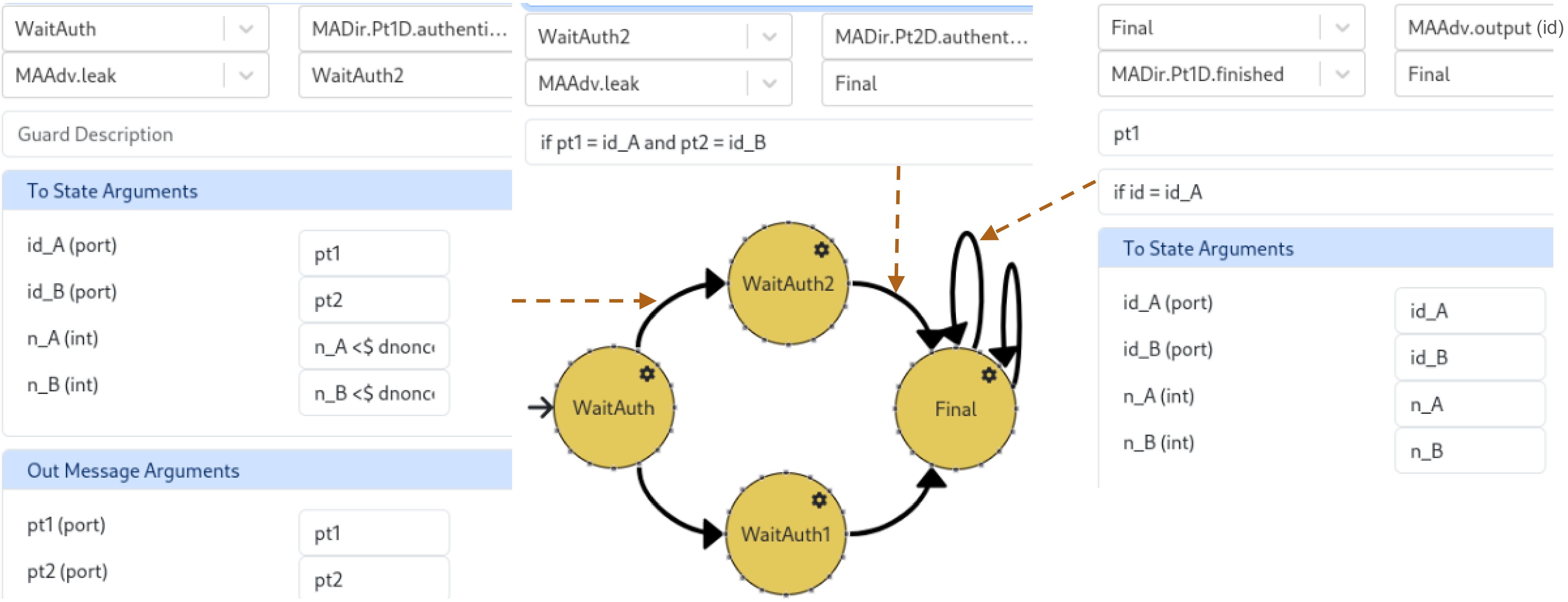
in pt2@authenticate (pt1 : port)

out finished@pt2

out leak (pt1 : port, pt2 : port))

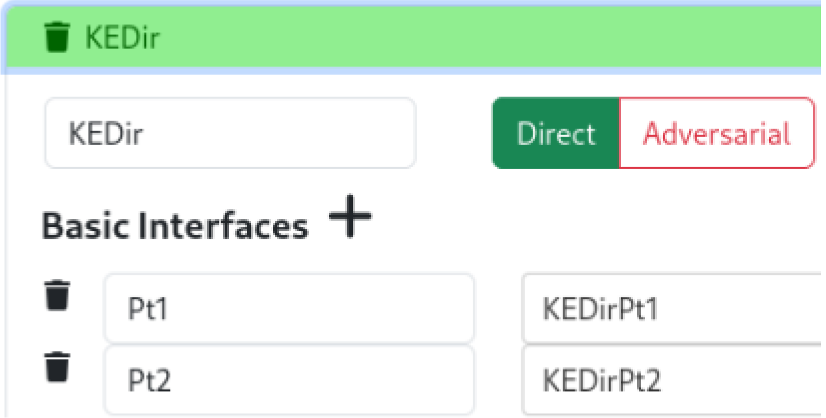
in finished (pt : port)

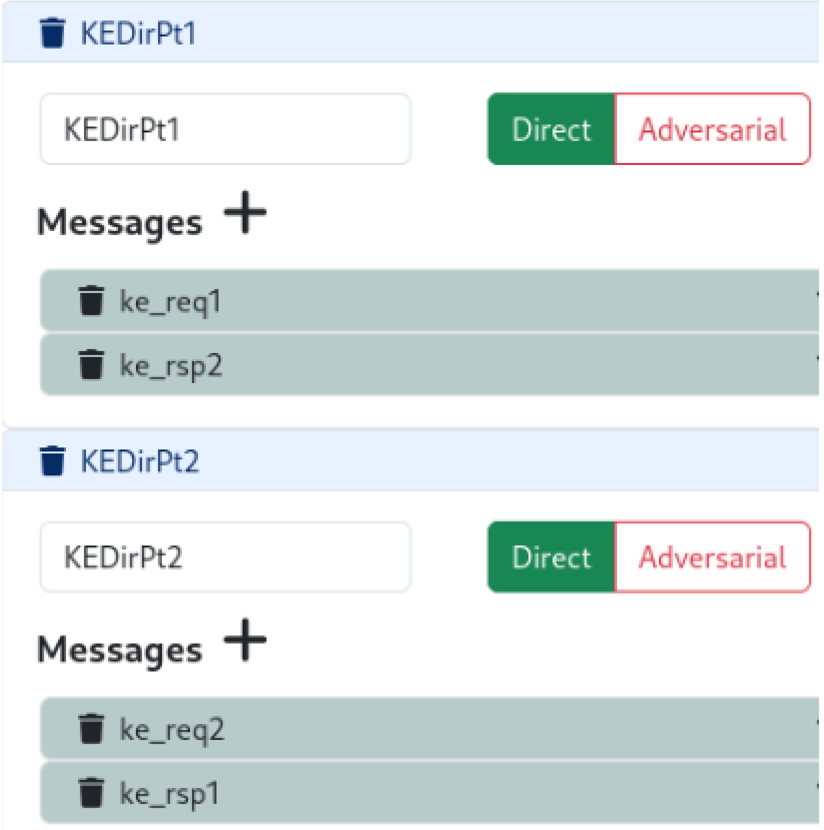
F_{2MA} in EasyUC, cont



RIVERSIDE RESEARCH

F_{KE} IN EASYUC





in pt1@ke_req1 (pt2 : port)

out ke_rsp2 (k : Key)@pt1

in pt2@ke_req2 (pt1 : port)

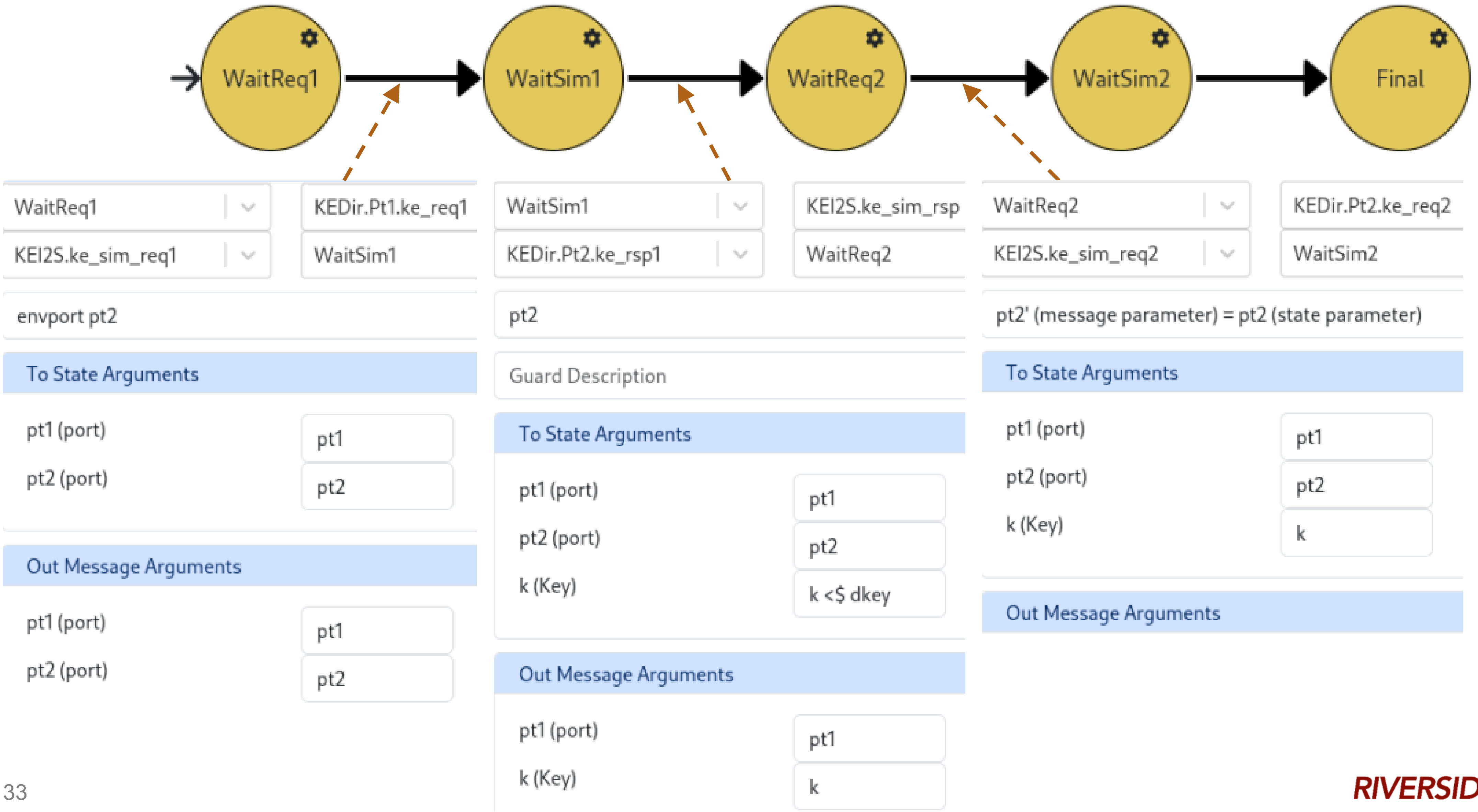
out ke_rsp1 (k : Key)@pt2

out ke_sim_req1 (pt1 : port, pt2 : port))

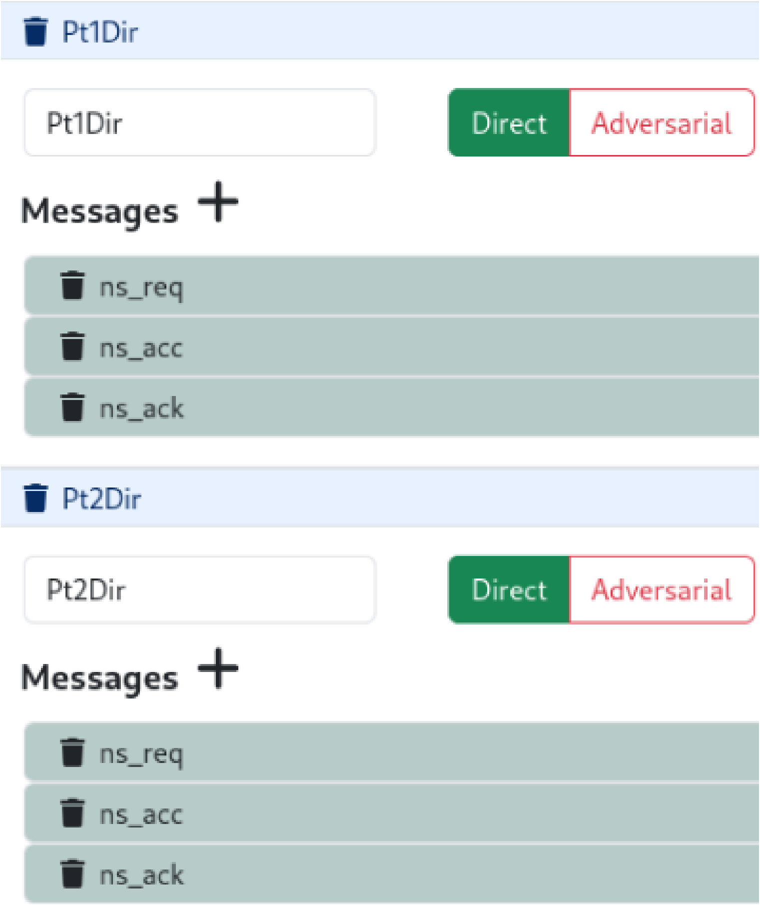
in ke_sim_rsp

out ke_sim_req2

F_{KE} IN EASYUC, CONT.

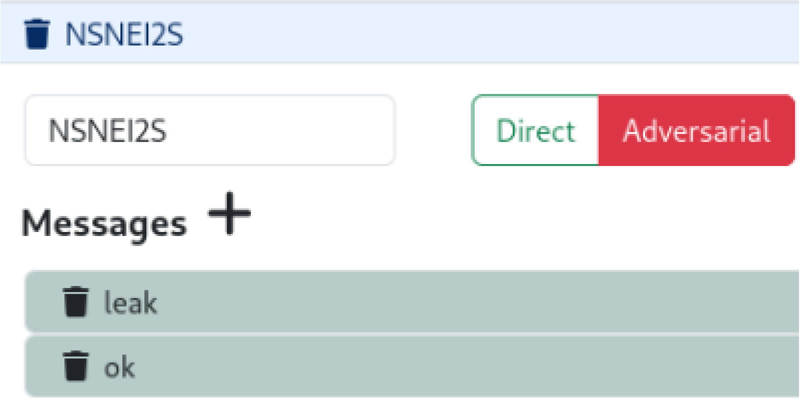


Revised Interfaces for Needham-Schroeder



in pt1@ns_req (pt2 : port)
out ns_acc (n_A : int, n_B : int)@pt1
in pt1@ns_ack

out ns_req (pt1 : port)@pt2
in pt2@ns_acc)
out @ns_ack (n_A : int, n_B : int)

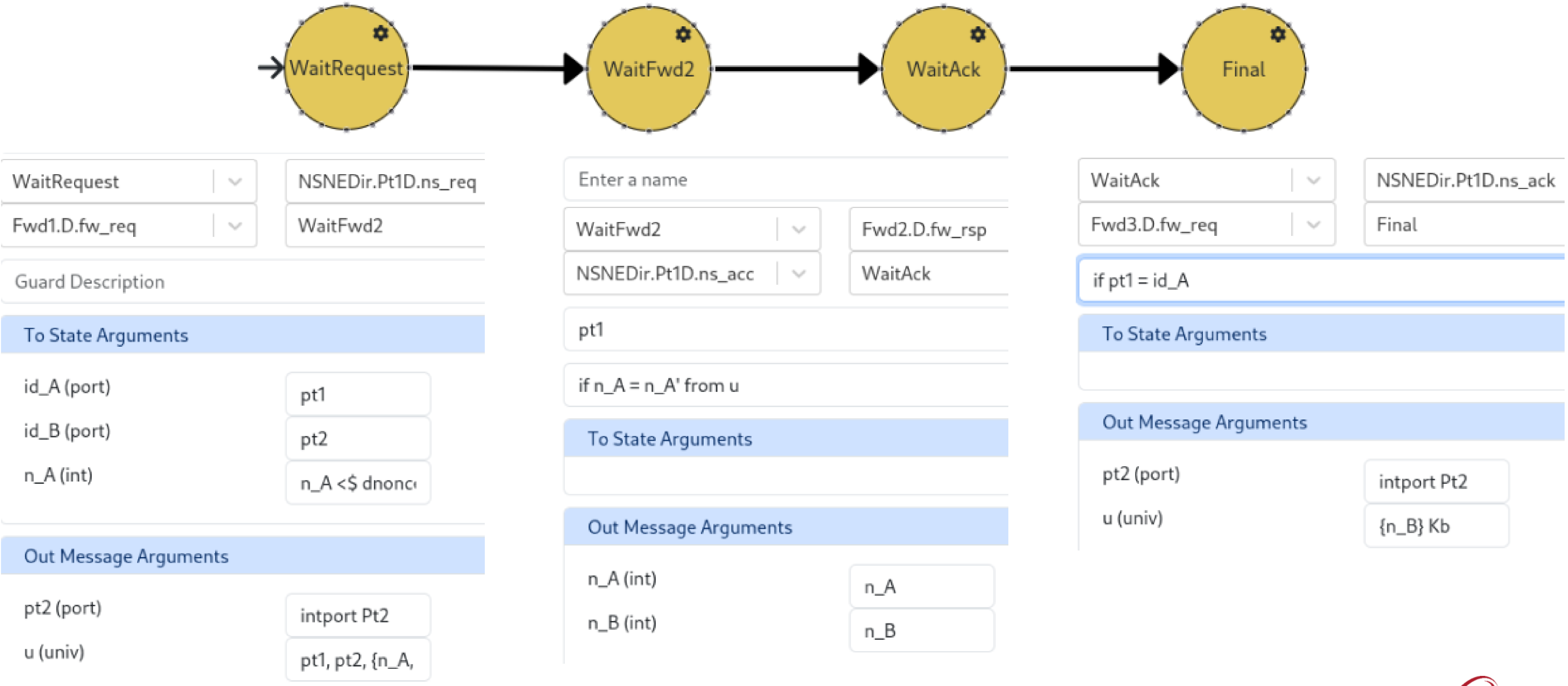


out leak (pt1 : port, pt2 : port)
in ok)

Revised Ideal Functionality for Needham-Schroeder

REVISED REAL FUNCTIONALITY FOR NEEDHAM-SCHROEDER

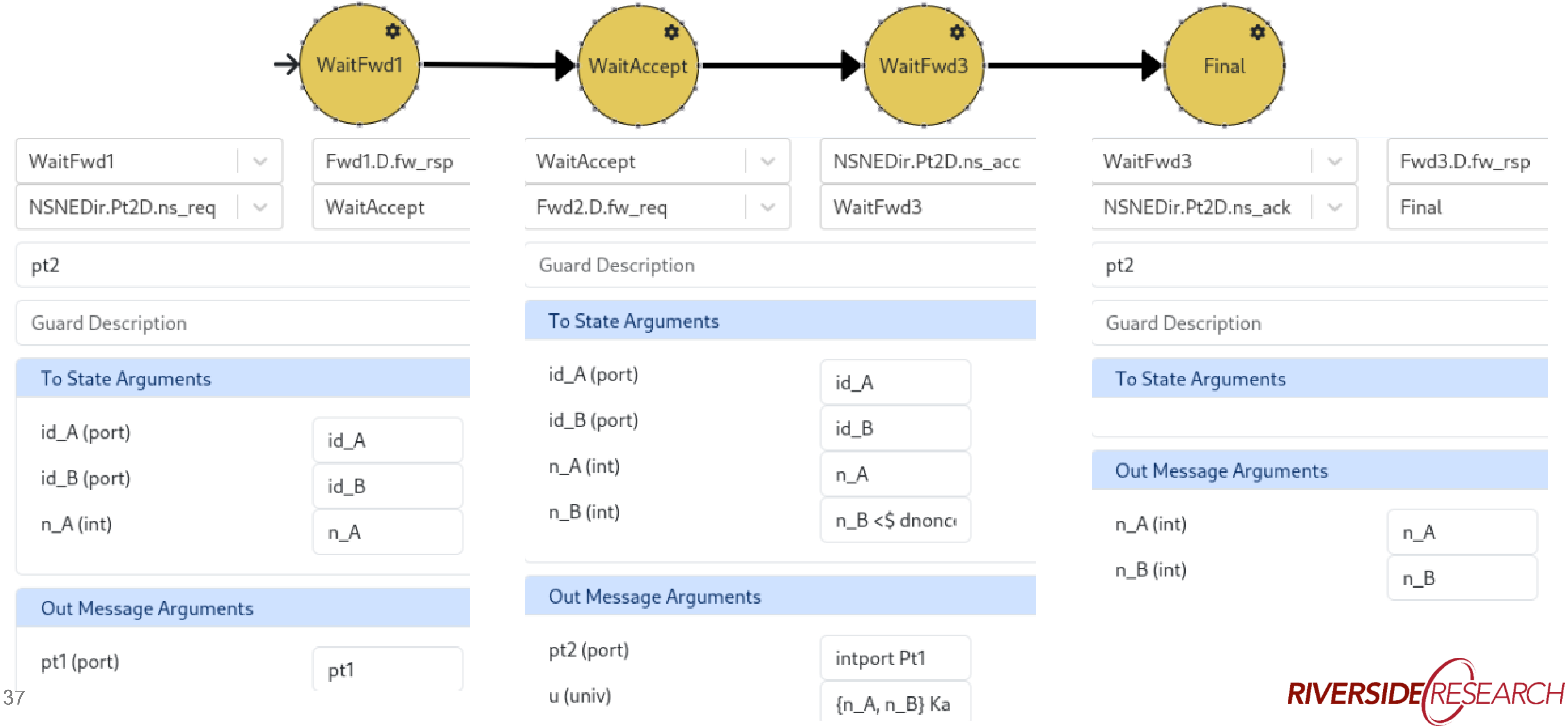
- Party Pt1 state machine



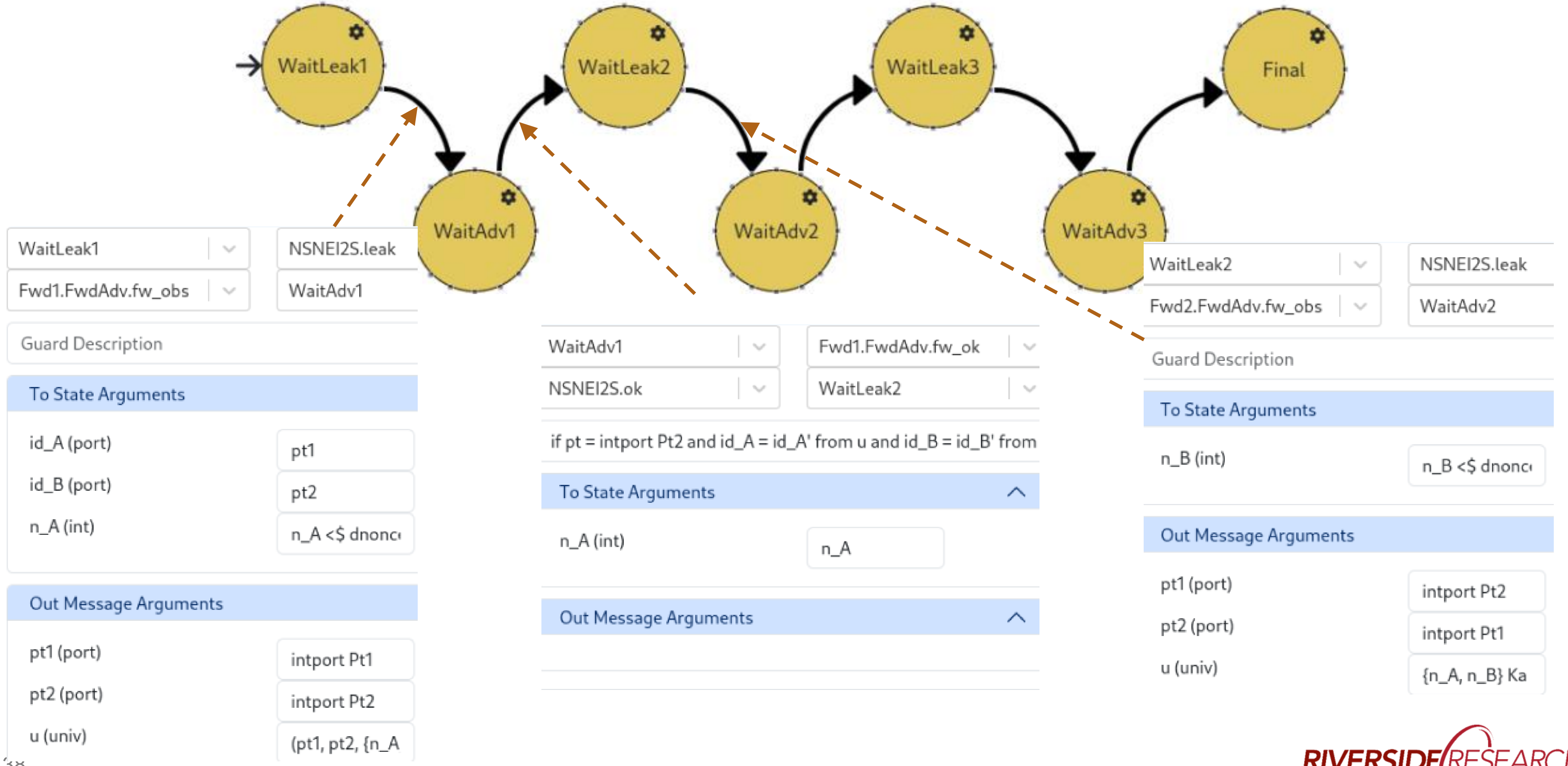
RIVERSIDE RESEARCH

REVISED REAL FUNCTIONALITY FOR NEEDHAM-SCHROEDER

- Party Pt2 state machine



REVISED SIMULATOR FOR NEEDHAM-SCHROEDER



RIVERSIDE RESEARCH

Modeling the MITM Attack

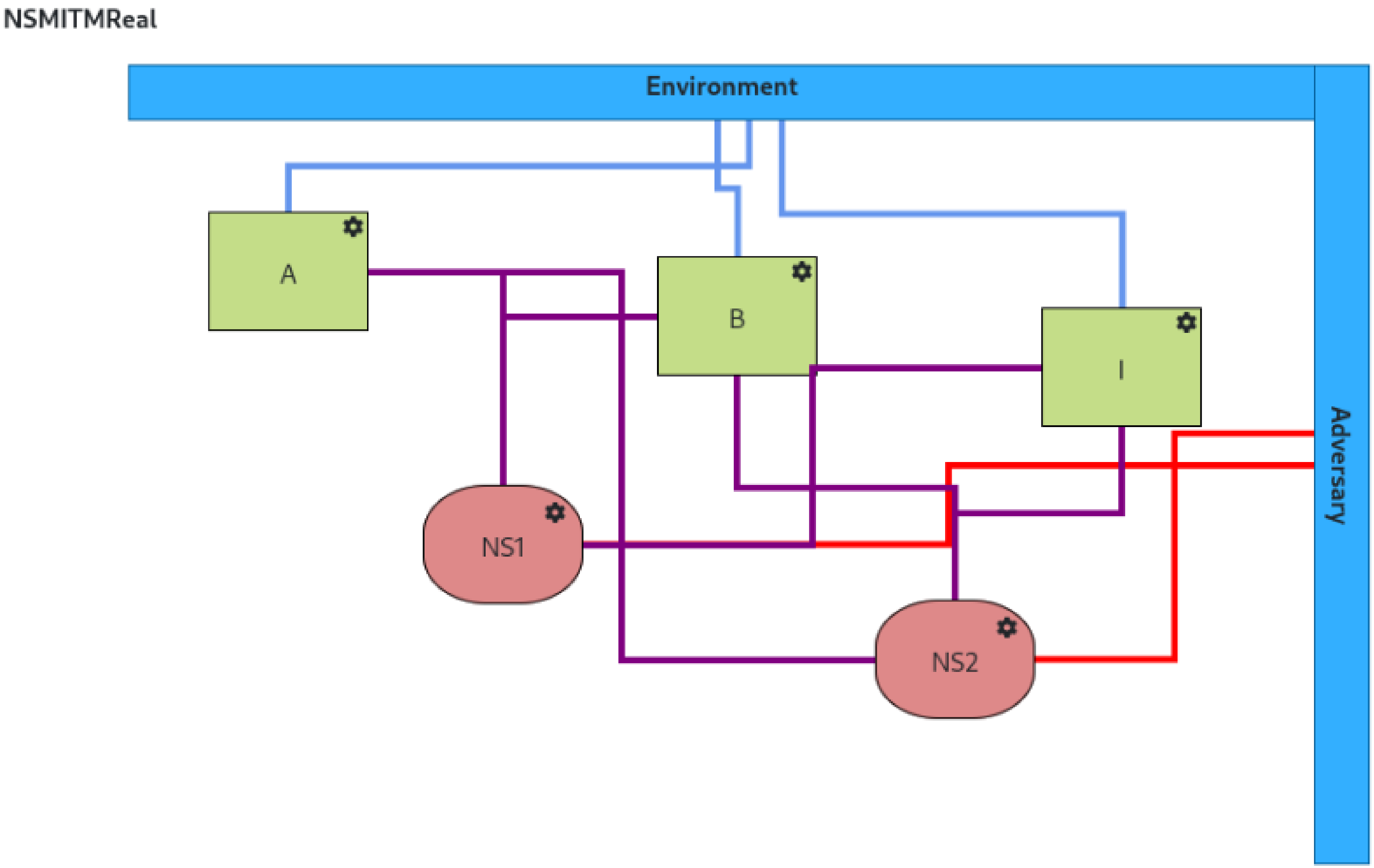
APPROACH

- In NSReal, replace hard-coded keys with keys provided by **`init_PKE`** messages:

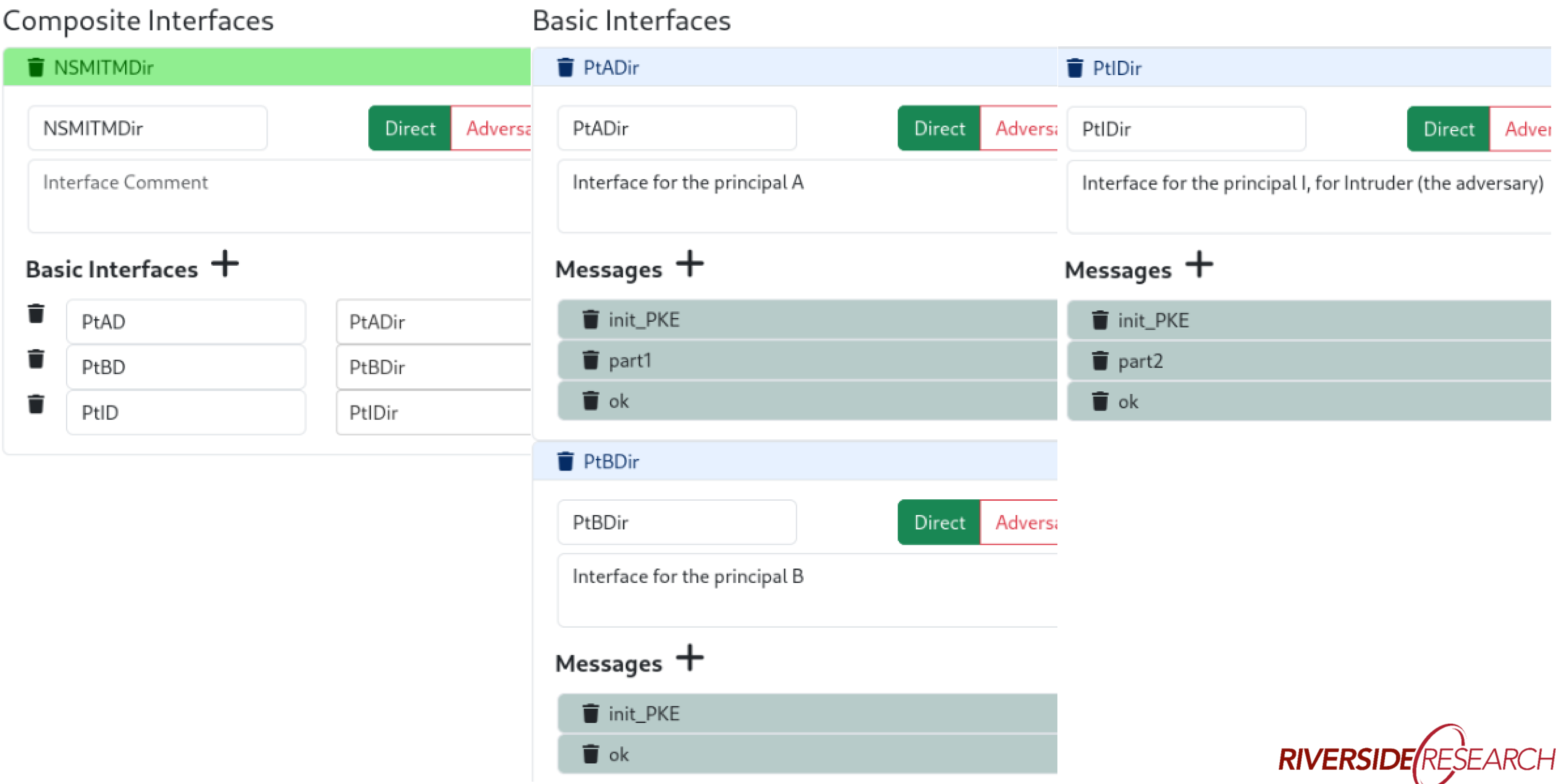
```
in  pt1@init_PKE (sk_a : sk_t, pk_table : (port pk_t) fmap)
out init_PKE_resp@pt1
```

  - NSIdeal will ignore this message
- Create a higher-level model containing two instances of Needham-Schroeder and 3 parties:
  - Party A is initiator of NS1; Party B is responder of NS2; Party I is responder of NS1 and initiator of NS2
- Option 1 (chosen)
  - NS1 and NS2 behave normally but party I has shared its key with the adversary
  - The adversary intercepts and modifies network traffic through forwarders (as already allowed)
- Option 2
  - Modify Needham-Schroeder to allow for corruption of a party
    - I.e., the adversary tells it what to send at each step
  - No need for adversary to modify any network traffic
- The ideal functionality and simulator are vestigial; just enough to satisfy the type checker

MITM Attack Real World



MITM Attack Interfaces



Party A State Diagram

Initialize
NSMITMDir.PtAD.ini...
NS1.Pt1D.initPKE
WaitNS1
Guard Description

To State Arguments

respPort (port) pt

Out Message Arguments

sk_a (sk_t) sk_a
pk_table ((port, pk_t) fmap) pk_table

WaitNS1
NS1.Pt1D.initPKE_resp
NSMITMDir.PtAD.ok
Part1
respPort
Guard Description

To State Arguments

Out Message Arguments

Part1
NSMITMDir.PtAD.pa...
NS1.Pt1D.ns_req
Final
Guard Description

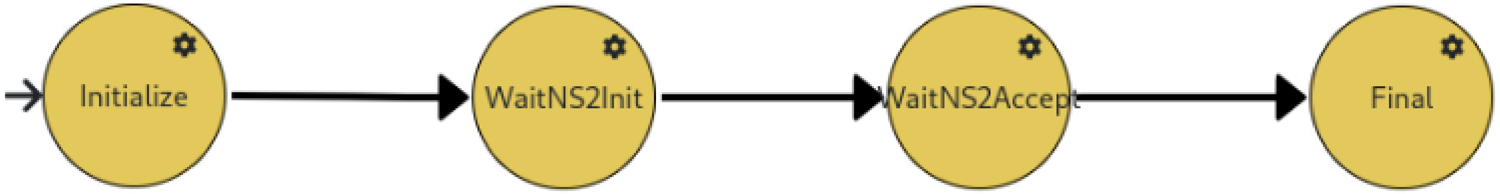
To State Arguments

Out Message Arguments

pt2 (port) intport I

RIVERSIDE RESEARCH

Party B State Diagram



Initialize
NS2.Pt2D.initPKE
NSMITMDir.PtBD.ini…
WaitNS2Init

Guard Description

To State Arguments

respPort (port) | pt

Out Message Arguments

sk_b (sk_t) | sk_B
pk_table ((port, pk_t) fmap) | pk_table

WaitNS2Init
NSMITMDir.PtBD.ok
NS2.Pt2D.initPKE_r…
WaitNS2Accept

respPort

Guard Description

To State Arguments

respPort (port) | respPort

Out Message Arguments

WaitNS2Accept
NSMITMDir.PtBD.ok
NS2.Pt2D.ns_acc
Final

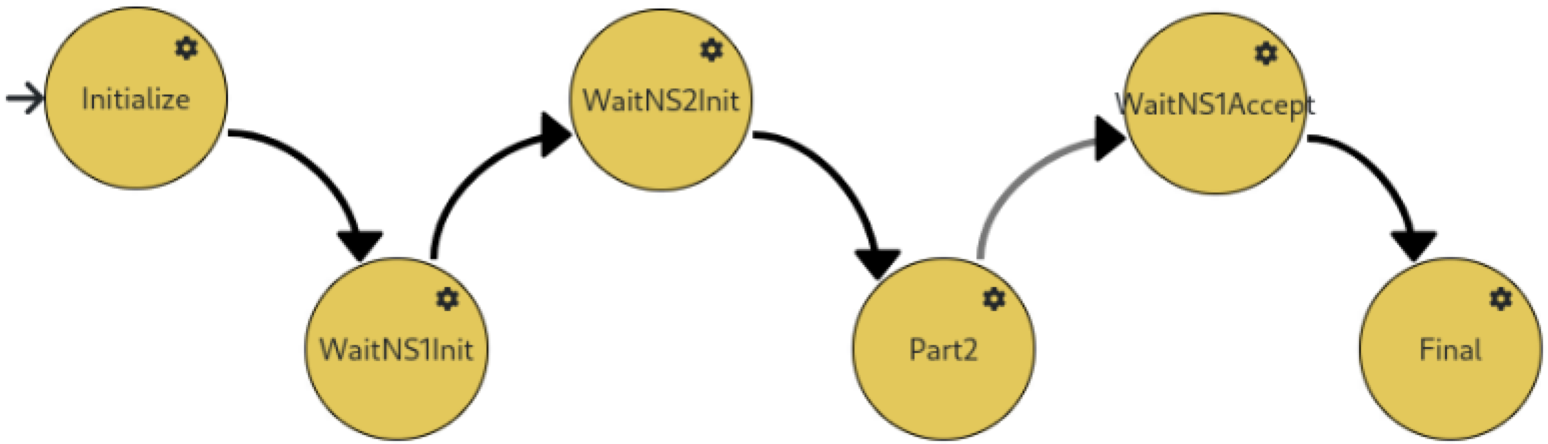
respPort

Guard Description

To State Arguments

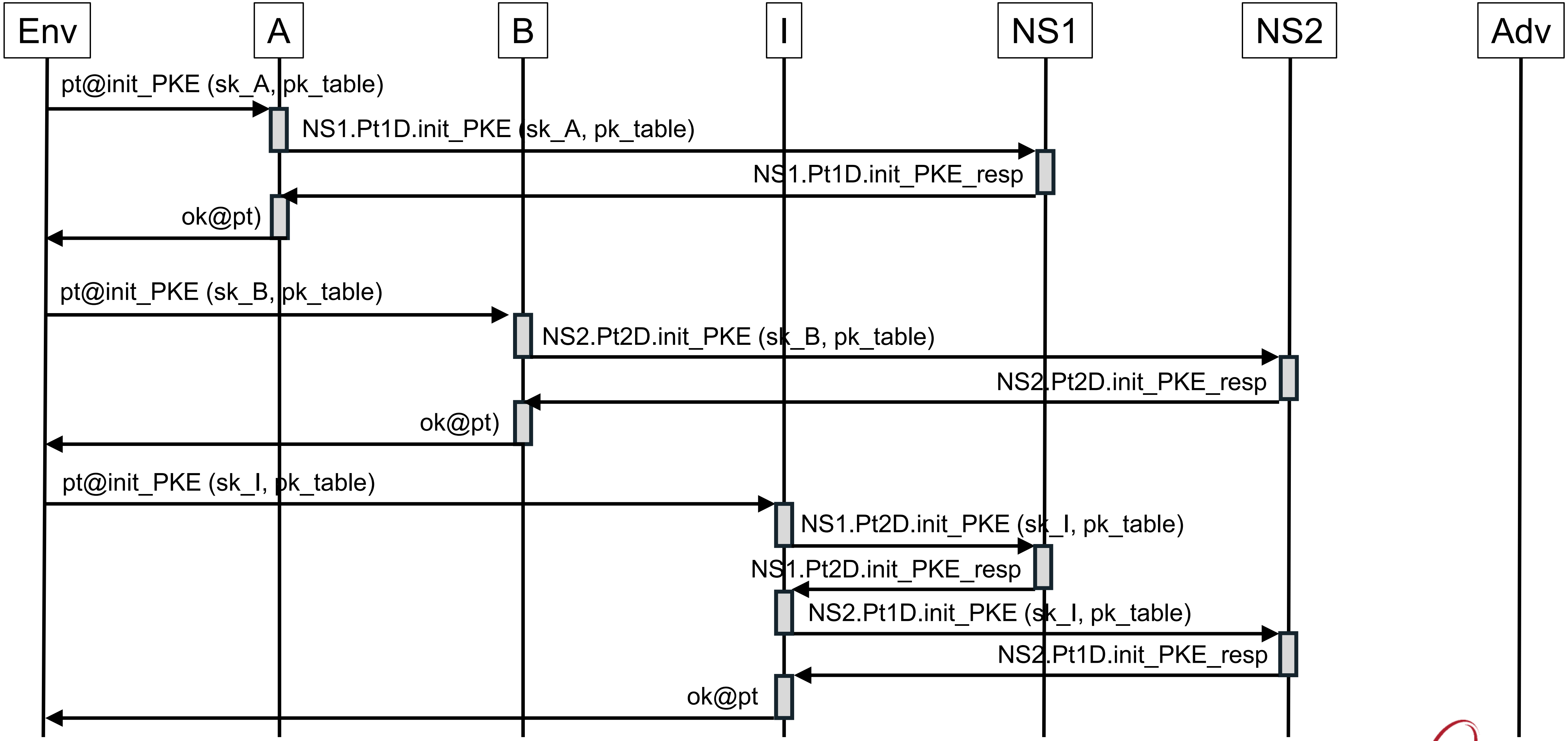
Out Message Arguments

Party I State Diagram

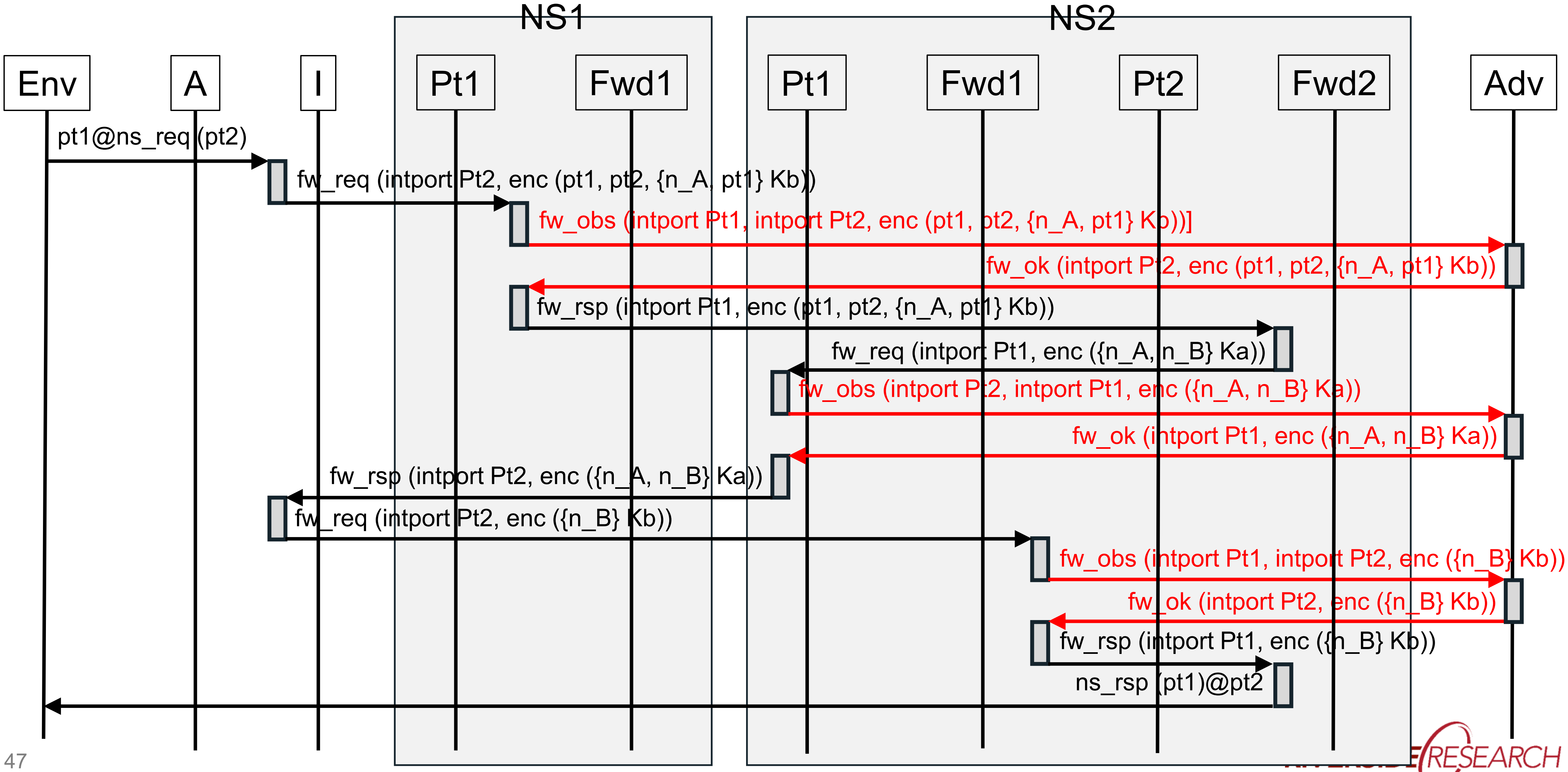


From State	In Message	Out Message	To State
Initialize	pt@NSMITMDir.PtID.initPKE (sk_I, pk_table)	NS1.Pt2D.initPKE (sk_I, pk_table)	WaitNS1Init
WaitNS1Init	NS1.Pt2D.initPKE_resp	NS2.Pt1D.initPKE (sk_I, pk_table)	WaitNS2Init
WaitNS2Init	NS2.Pt1D.initPKE_resp	NSMITMDir.PtId.ok	Part2
Part2	NSMITMDir.PtId.part2	NS2.Pt1D.ns_req (intport B)	WaitNS1Accept
WaitNS1Accept	NS1.Pt2D.ns_acc (_)	NSMITMDir.PtID.ok	Final

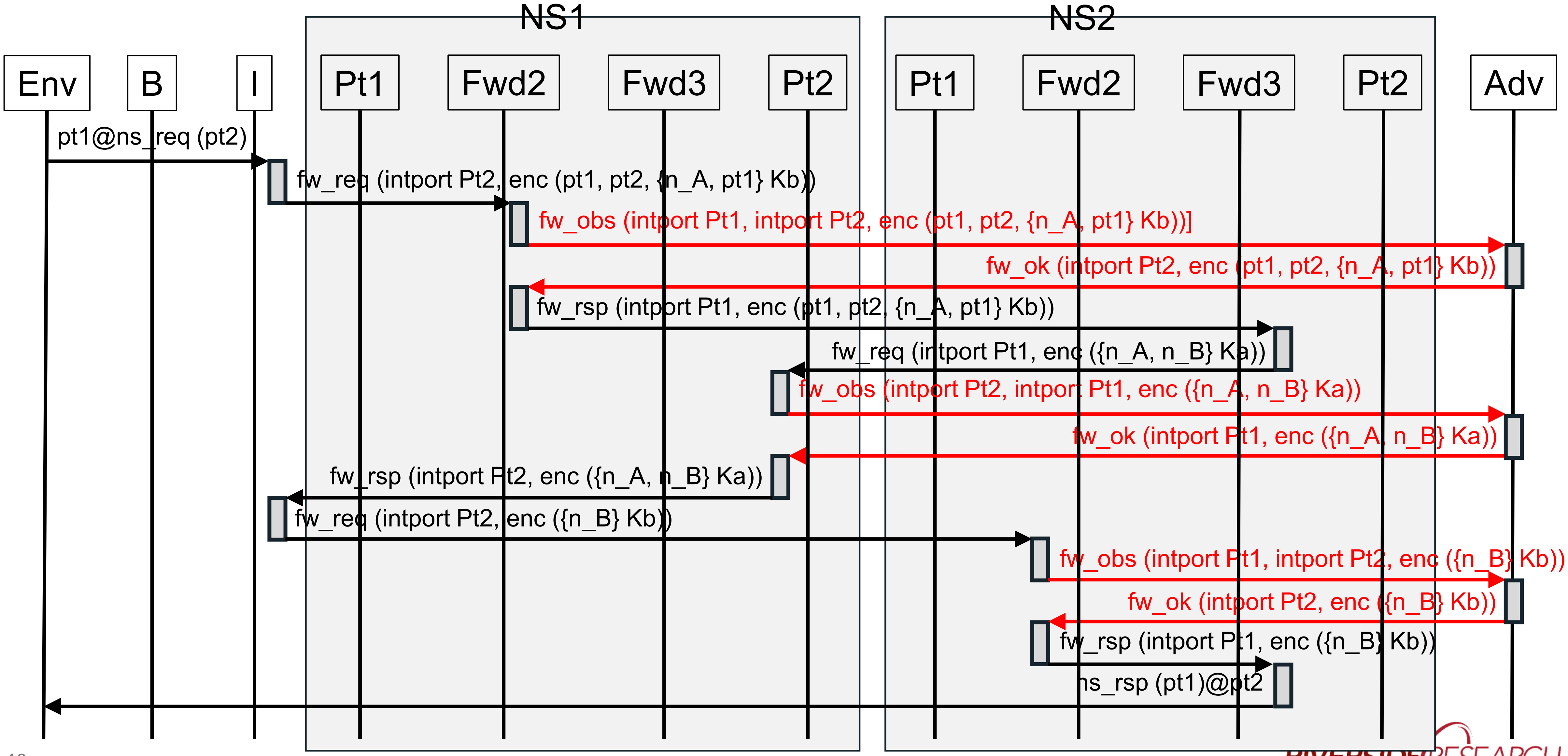
SEQUENCE DIAGRAM OF EXPLOIT (INITIALIZATION)



Sequence Diagram of Exploit, cont



SEQUENCE DIAGRAM OF EXPLOIT, CONT



RIVERSIDE RESEARCH

References

- Ross Anderson and Roger Needham, “Programming Satan’s Computer,” in van Leeuwen, J. (eds) Computer Science Today, Lecture Notes in Computer Science, vol 1000, 1995. Springer, Berlin, Heidelberg.
- Roger M. Needham and Michael D. Schroeder, “Using Encryption for Authentication in Large Networks of Computers,” Communications of the ACM, Volume 21, Issue 12, 1978.
- Michael Burrows, Martín Abadi and Roger Needham, “A Logic of Authentication", in Proceedings of the Royal Society of London A v 426, pp 233-271, 1989.
- Gavin Lowe, “An attack on the Needham-Schroeder public-key authentication protocol,” Information Processing Letters, Volume 56, Issue 3, 10 November 1995, Pages 131-133.
- Gavin Lowe, “Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR,” in Margaria, T., Steffen, B. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 1996. Lecture Notes in Computer Science, vol 1055. Springer, Berlin, Heidelberg.
- Ran Canetti and Jonathan Herzog, “Universally Composable Symbolic Security Analysis,” *Journal of Cryptology*, 24(1):83–147, 2011.

RIVERSIDE RESEARCH