CSP, Casper and Security Verification

고려대학교 정형기법 연구실

김 일 곤

2003. 8. 20

Security

- Security
  : Confidentiality, Authentication, Integrity and Availability

- Research
  1. Security System
     Access Control Model, Information Flow Model
- 2. Security Protocol
  SSH, Kerberos, RADIUS and etc

Formal Approach

- Theorem Proving
  - BAN, GNY, SVO logic
- Model Checking
  - FDR(CSP), SPIN, SMV, CADP
- Type Theory
  - PCC(Proof Carrying Code)
- Global Computing
  - spi calculus

Process Algebra and CSP

- Process Algebra
  - a formal description technique to complex computer systems, especially those involving communication, concurrently executing components
- CSP(Communicating Sequential Processes)
  - It is a process specification language designed by C.A.R. Hoare, at the University of Oxford during the 1980s
  - A formal notation in which the computations of concurrent processes communicating by channel can be concisely described and modelled.

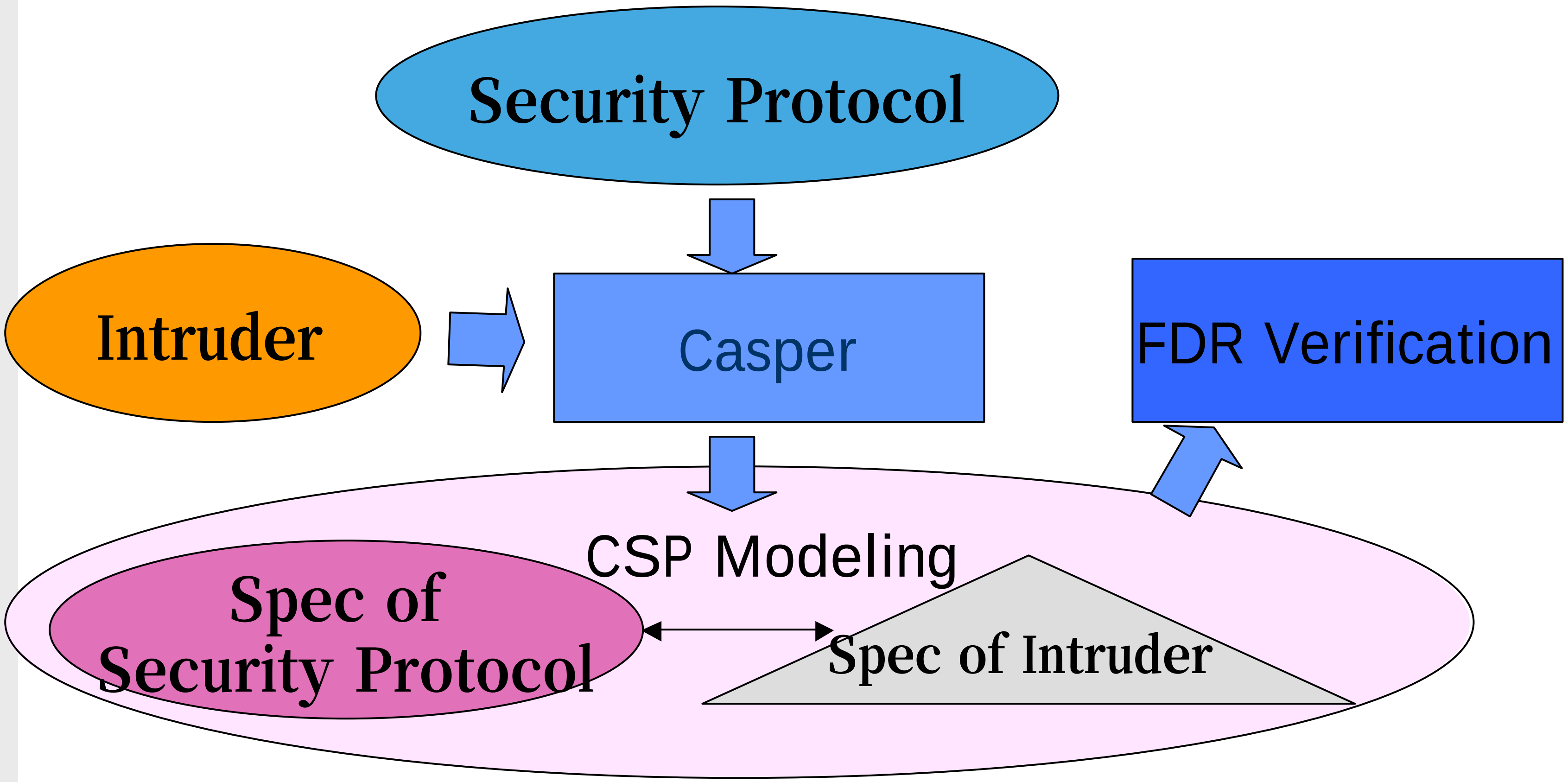
Casper

- Casper(A Compiler for the Analysis of Security Protocols)

- CSP description of the system is
  1. very time-consuming
  2. only possible for people well practiced in CSP
  2. even the experts will often make mistakes that prove hard to spot

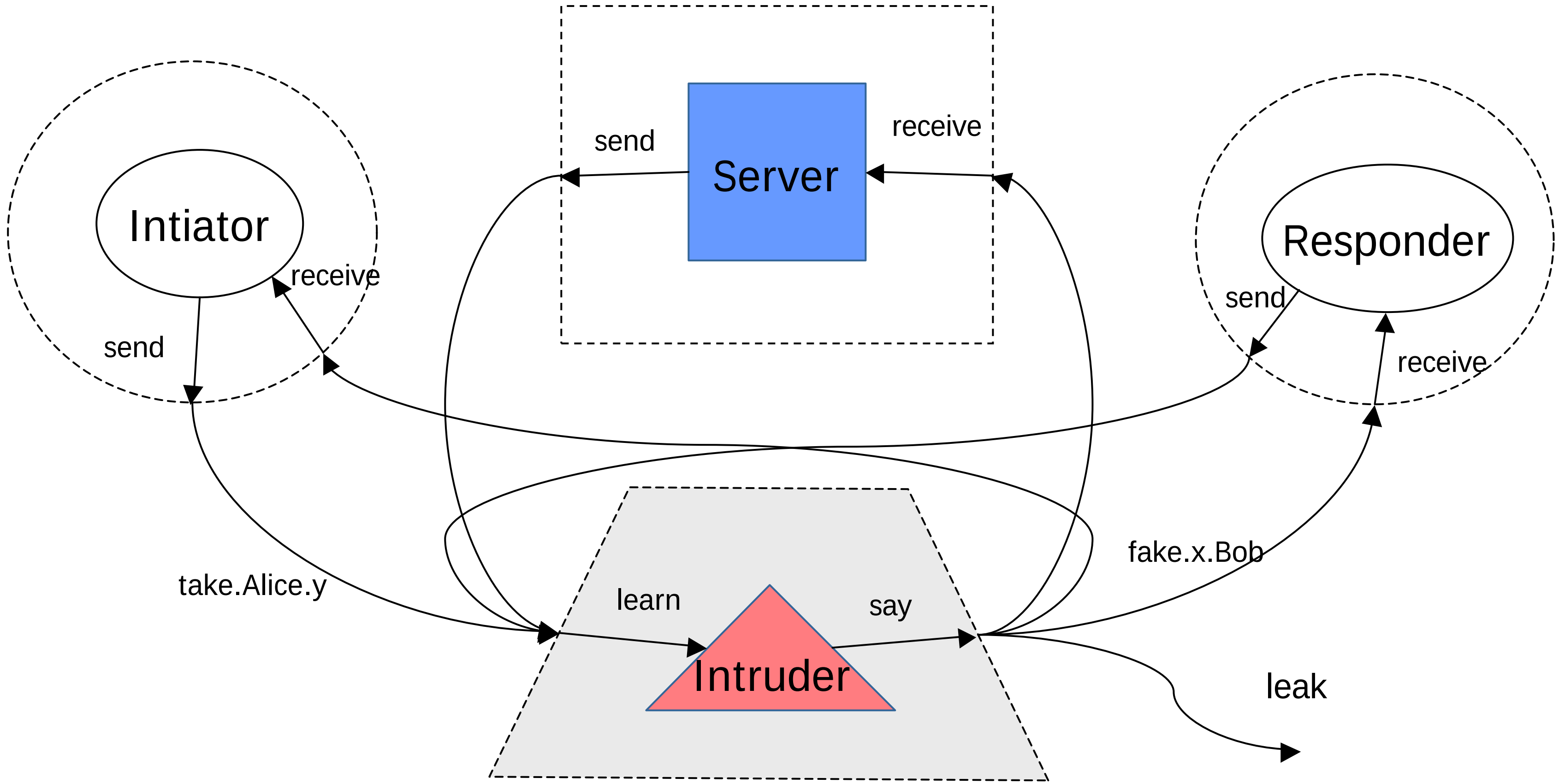
※ Casper simplifies this process.

Casper and CSP/FDR



CSP Modeling

Initiator ||| Responder ||| Server ||| Intruder



Refinement Checking

- 1. Trace Refinement: Safety

$$P \sqsubseteq_T Q \equiv traces(Q) \subseteq traces(P)$$

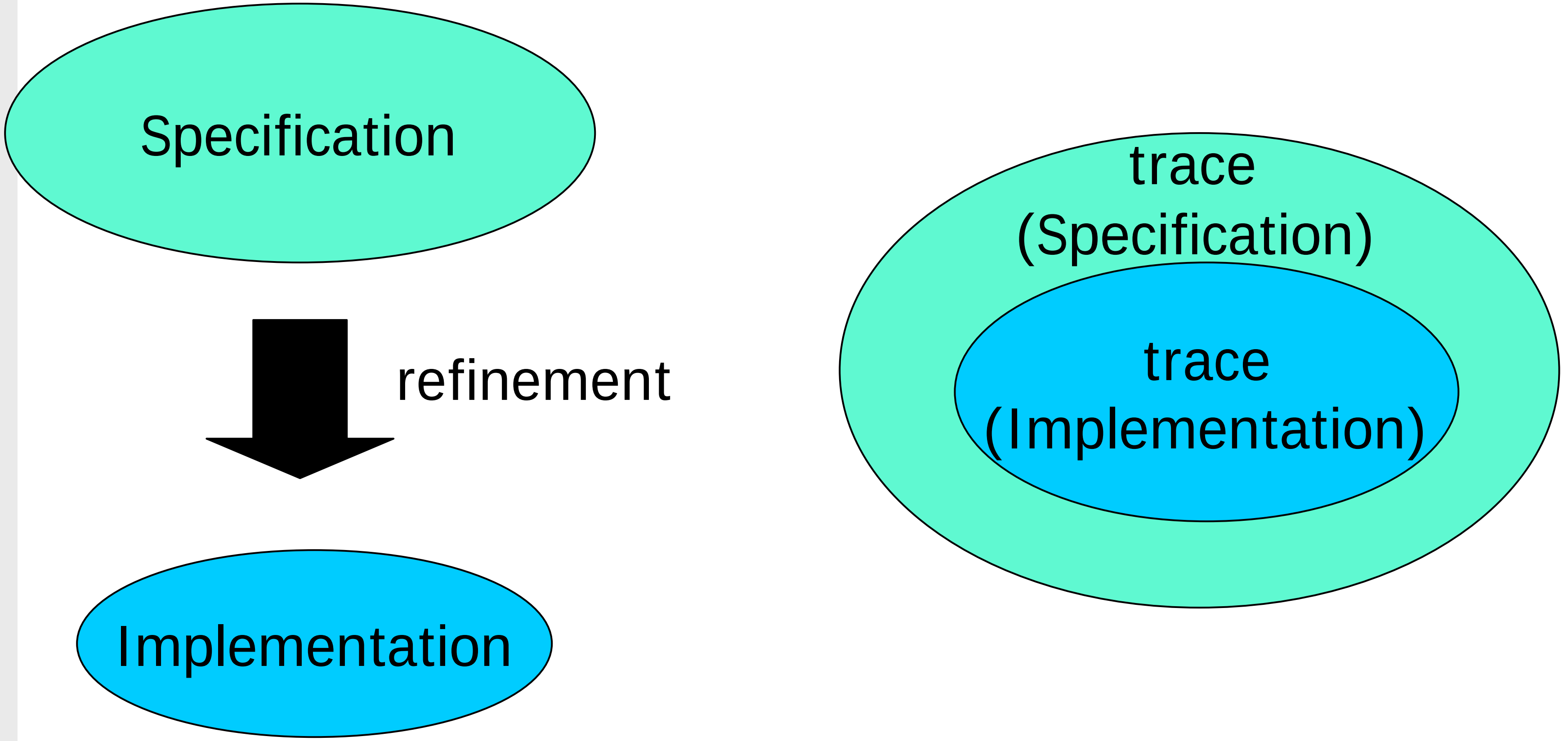
- 2. Failures Refinement : Deadlock

$$P \sqsubseteq_F Q \equiv traces(P) \supseteq traces(Q) \wedge failures(P) \supseteq failures(Q)$$

- 3. Failures - Divergences Refinement : Liveness

$$P \sqsubseteq_{FD} Q \equiv failures(Q) \subseteq failures(P) \wedge divergences(Q) \subseteq divergences(P)$$

Refinement Checking



Traces of a Process

- A trace of a process is a finite sequence of events, representing the behaviour of the process up to a certain point in time. Trace set is written traces(P)

traces(coin -> STOP) ={<>, <coin>}

CLOCK = tick -> CLOCK

traces(CLOCK) = {<>,<tick>,<tick,tick>,<tick,tick,tick>, … }

= {tick}*

Examples :

a -> b -> STOP $\sqsubseteq_T$ a -> STOP

A -> b -> STOP $\sqsubseteq_T$ STOP

Secrecy and Authentication

- They are both safety properties: *a certain bad thing should not happen*

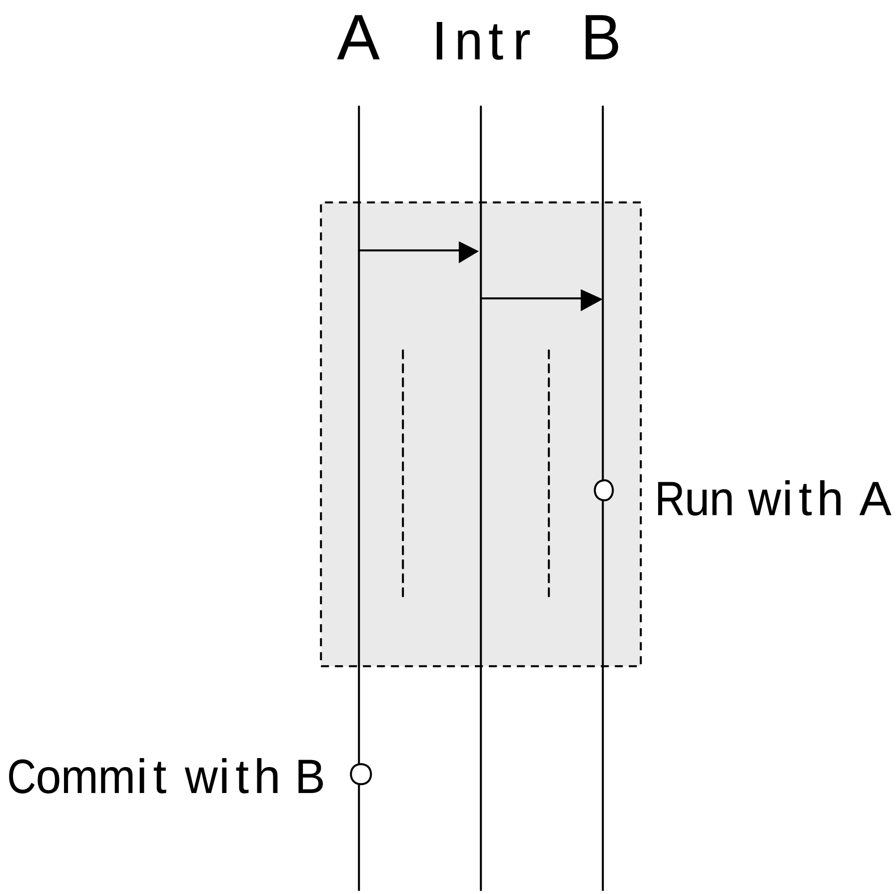
- Secrecy:

  Information m has not become known to the intruder

- Authentication:

  The matching of these two events guarantees the identities of A and B

Authentication Property



Example: The Yahalom Protocol

- The protocol

Message 1 $a \text{ -> } b : a.n_a$

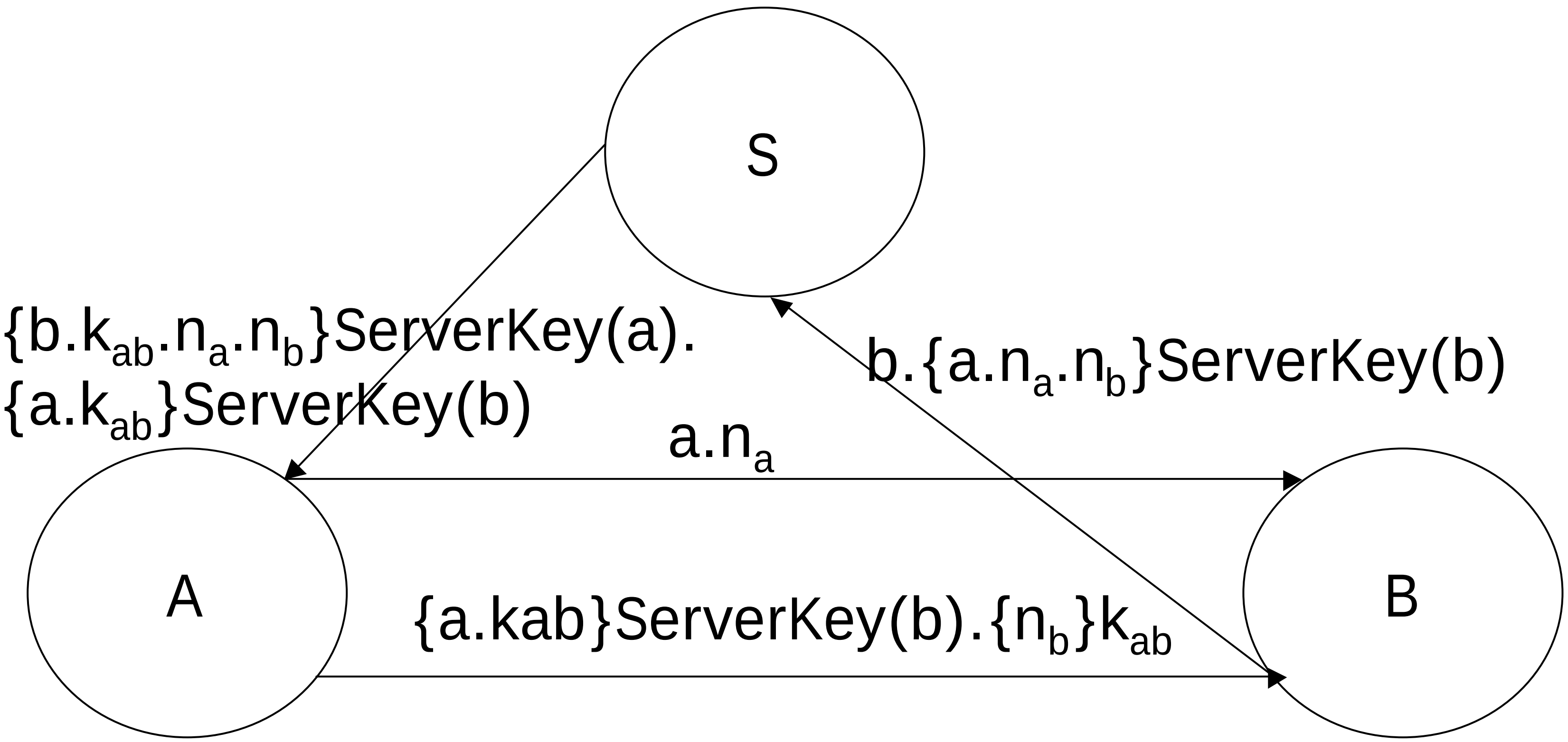
Message 2 $b \text{ -> } s : b.\{a.n_a.n_b\}_{ServerKey(b)}$

Message 3 $s \text{ -> } a : \{b.k_{ab}.n_a.n_b\}_{ServerKey(a)}$

$\{a.k_{ab}\}_{ServerKey(b)}$

Message 4 $a \text{ -> } b : \{a.k_{ab}\}_{ServerKey(b)}.\{n_b\}_{kab}$

Yahalom Protocol



#Free Variables

A, B : Agent

S : Server

na, nb : Nonce

SKey : Agent -> ServerKey

kab : SessionKey

InverseKeys = (SKey, SKey), (kab,kab)

#Processes

INITIATOR(A,na) knows SKey(A)

RESPONDER(B,S,nb) knows SKey(B)

SERVER(S,kab) knows SKey

#Protocol description

0.   -> A : B
1. A -> B : na
2. B -> S : {A, na, nb}{SKey(B)}
3a. S -> A : {B, kab, na, nb}{SKey(A)}
3b. S -> A : {A, kab}{SKey(B)} % enc
4a. A -> B : enc % {A, kab}{SKey(B)}
4b. A -> B : {nb}{kab}

#Specification

Agreement(B, A, [na])
Agreement(A, B, [na])

#Actual variables

Alice, Bob, Mallory : Agent

Sam : Server

Na, Nb : Nonce

Kab : SessionKey

InverseKeys = (Kab, Kab)

#Inline functions

- symbolic SKey

#System

INITIATOR(Alice, Na)

RESPONDER(Bob, Sam, Nb)

SERVER(Sam, Kab)

Authentication

- The CSP approach is based on inserting signals:
  - Running.a.b (in a's protocol)
    - Agent a is executing a protocol run apparently with b
  - Commit.b.a (in b's protocol)
    - Agent b has completed a protocol run apparently with a
- Authentication is achieved if Running.a.b always precedes Commit.b.a in the traces of the system
  - Weaker or stronger forms of authentication can be achieved by variations of the parameters of these signals and the constraints on them

Authentication in the Yahalom Protocol

- The Yahalom Protocol aims at providing authentication of both parties : authentication of the initiator to the responder, and viceversa
- We will analyze the two authentication properties separately
- This requires two separate enhancements of the protocol

Yahalom: authentication of initiator

- CSP description of the two parties - Enhanced

Initiator'$(a,n_a) =$

env?b: Agent

→ send.a.b.a.n_a

→ [] (receive.J.a$\{b.\ k_{ab}.n_a.n_b\}_{ServerKey(a)}$.m

k_{ab} ε Key → signal.Running_Initiator.a.b.$n_a.n_b.k_{ab}$

n_b ε Nonce → send.a.b.m.$\{n_b\}_{k_{ab}}$

m ε T → Session(a,b,k_{ab},n_a,n_b))

Responder'$(b,n_b) =$

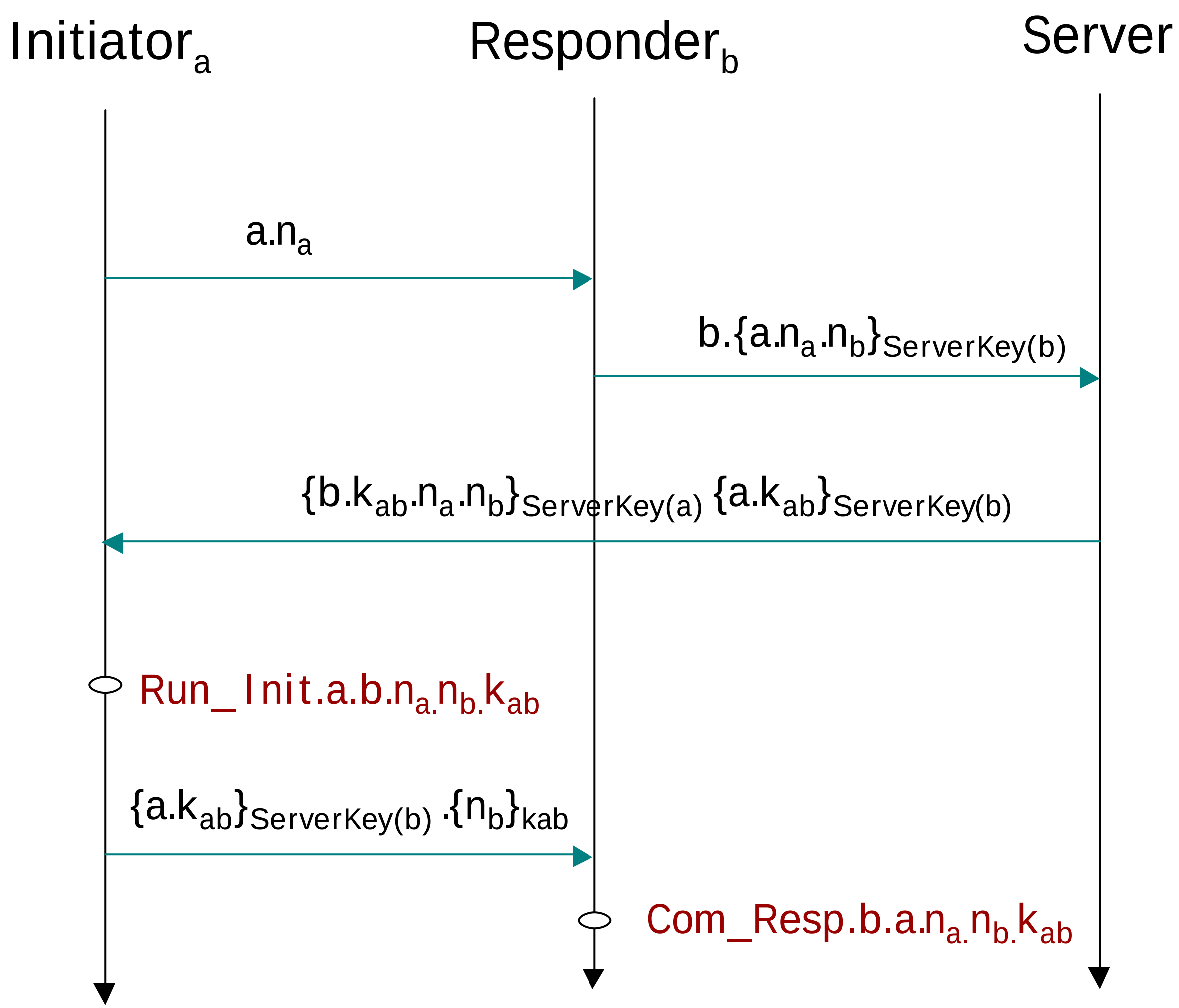
[] (receive.a.b.a.n_a → send.b.J.b .$\{a.n_a.n_b\}_{ServerKey(b)}$

k_{ab} ε Key → receive.a.b.$\{a.\ k_{ab}\}_{ServerKey(b)}$.$\{n_b\}_{k_{ab}}$

n_b ε Nonce → signal. Commit_Responder.b.a.$n_a.n_b.k_{ab}$

m ε T → Session(b,a,k_{ab},n_a,n_b))

Yahalom: authentication of initiator



Yahalom: authentication of initiator

- The property to be verified:

  signal. Running_Initiator.a.b.$n_{a.}n_{b.}k_{ab}$

  precedes

  signal.Commit_Responder.b.a.$n_{a.}n_{b.}k_{ab}$

  in all the Traces(System)

- Again, this property can be verified automatically by checking the traces

Yahalom: authentication of initiator

Specification

AuthenticateINITIATORToRESPONDERAgreement_na(A) =

```
signal.Running2.INITIATOR_role.A?B?na ->
signal.Commit2.RESPONDER_role.B.A.na -> STOP
```

System

```
SYSTEM_0 =
(AGENT_Alice
  [| inter(Alpha_Alice, union(Alpha_Bob, Alpha_Sam)) |]
(AGENT_Bob
  [| inter(Alpha_Bob, Alpha_Sam) |]
AGENT_Sam))
SYSTEM = SYSTEM_0 [| {|comm, fake, intercept|} |] INTRUDER
```

Verification

Assert Specification [T= System

Yahalom: authentication of responder

- CSP description of the two parties - Enhanced

Initiator'$(a,n_a) =$

env?b: Agent

→ $send.a.b.a.n_a$

→ [] $(receive.J.a\{b.\ k_{ab}.n_a.n_b\}_{ServerKey(a)}\ .m$

k_{ab} ε Key → $send.a.b.m.\{n_b\}_{kab}$

n_b ε Nonce → $signal.Commit\_Initiator.a.b.n_{a.}n_{b.}k_{ab}$

m ε T → $Session(a,b,k_{ab},n_a,n_b)\ )$

Responder'$(b,n_b) =$

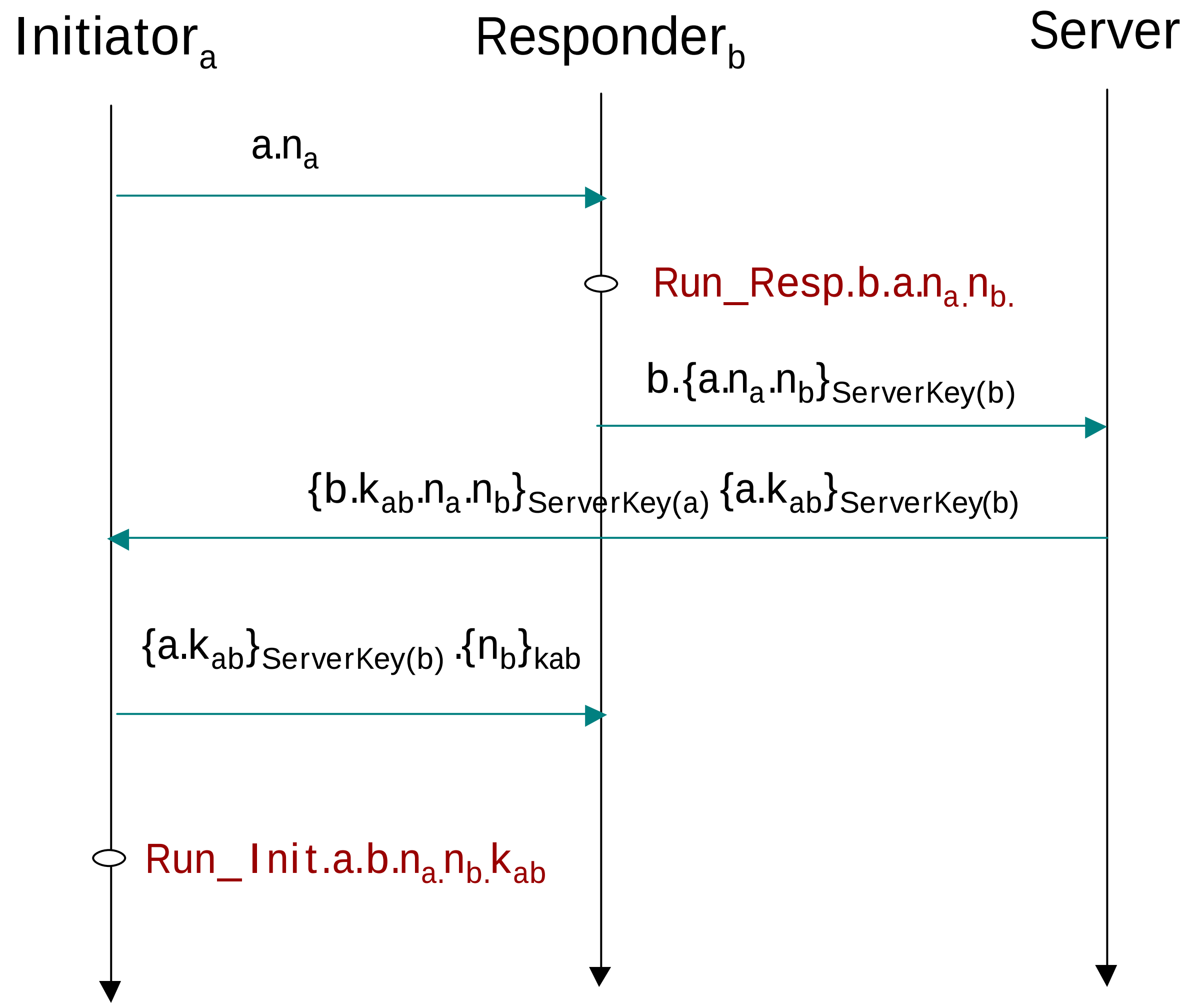
[] $(receive.a.b.a.n_a$ → $send.b.J.b\ .\{a.n_a.n_b\}_{ServerKey(b)}$

k_{ab} ε Key → $signal.\ Running\_Responder.b.a.n_{a.}n_b$

n_b ε Nonce → $receive.a.b.\{a.\ k_{ab}\}_{ServerKey(b)}\ .\{n_b\}_{kab}$

m ε T → $Session(b,a,k_{ab},n_a,n_b)\ )$

Yahalom: authentication of responder



Yahalom: authentication of responder

- The property to be verified:

$$signal.\ Running\_Responder.b.a.n_{a.}n_b$$

precedes

$$signal.Commit\_Initiator.a.b.n_{a.}n_{b.}k_{ab}$$

in all the Traces(System)

- Again, this property can be verified automatically by checking the traces

Yahalom: authentication of responder

Specification

```
AuthenticateRESPONDERToINITIATORAgreement_na(B) =
    signal.Running1.RESPONDER_role.B?A?na ->
    signal.Commit1.INITIATOR_role.A.B.na -> STOP
```

System

```
  SYSTEM_0 =
  (AGENT_Alice
    [| inter(Alpha_Alice, union(Alpha_Bob, Alpha_Sam)) |]
  (AGENT_Bob
    [| inter(Alpha_Bob, Alpha_Sam) |]
  AGENT_Sam))
SYSTEM = SYSTEM_0 [| {|comm, fake, intercept|} |] INTRUDER
```

Verification

```
Assert Specification [T= System
```