보안 공학

2022학년도 2학기
경상국립대학교 컴퓨터과학부

2022.11.01

김지윤

Overview

- **AVISPA**

✓ 개요

✓ HLPSL

01

AVISPA 개요

AVISPA 개요 (1/7)

- AVISPA
- Automated Validation of Internet Security Protocols and Applications
- 인터넷 보안 프로토콜 및 어플리케이션 보안 분석 자동화 도구

- 특징
- **Push-Button** Security Protocol Analyzer
- AVISPA 전용 언어인 **HLPSL**을 통한 **모듈화와 다양한 표현**이 가능
- HLPSL (High-Level Protocol Specification Language)
- 최신 자동화 분석 기술을 구현한 **4개의 Back-End**를 통합

- 장점
- 공격을 발견하여 대상 프로토콜의 취약점 분석
- 추상화 기반의 검증 기법 지원
- 유한 혹은 무한 세션 지원

AVISPA 개요 (2/7)

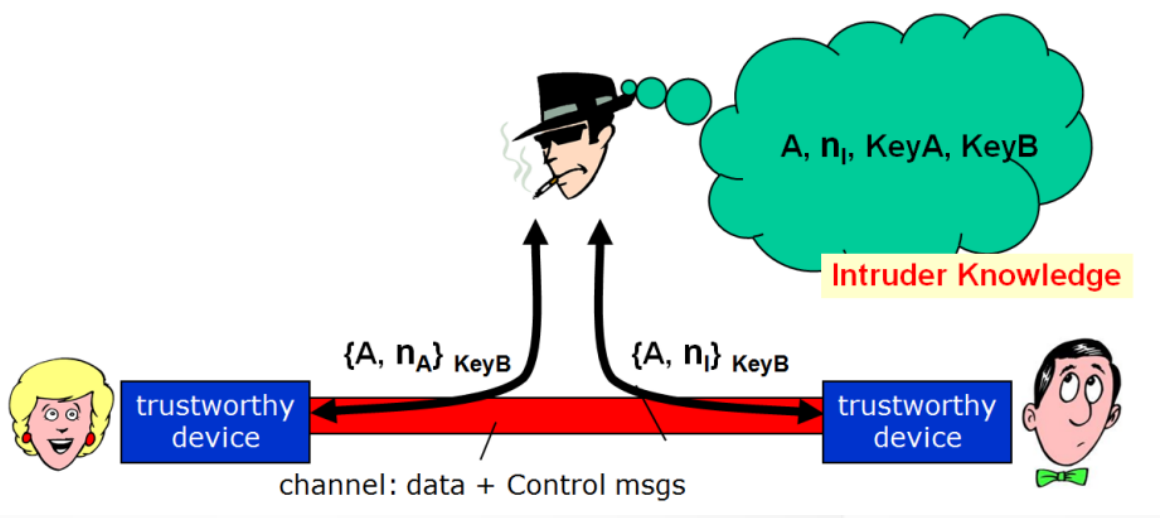
- HLPSL (High-Level Protocol Specification Language)

- 역할 (Role) 기반의 언어
- 대칭키 암호, 공개키 암호, Diffie-Hellman 키 교환, 해시함수 등의 중요한 암호학적 요소의 표현을 지원
- 두 가지 보안속성 (인증, 기밀) 검증 가능
- 공격자 (Intruder)는 통신이 발생하는 채널에 의해서 모델링
- 공격자는 Dolev-Yao의 공격자 모델을 준수
- HLPSL2IF에 의해 Intermediate Format (IF)로 번역되고, If는 4개의 Back-End의 입력으로 사용

AVISPA 개요 (3/7)

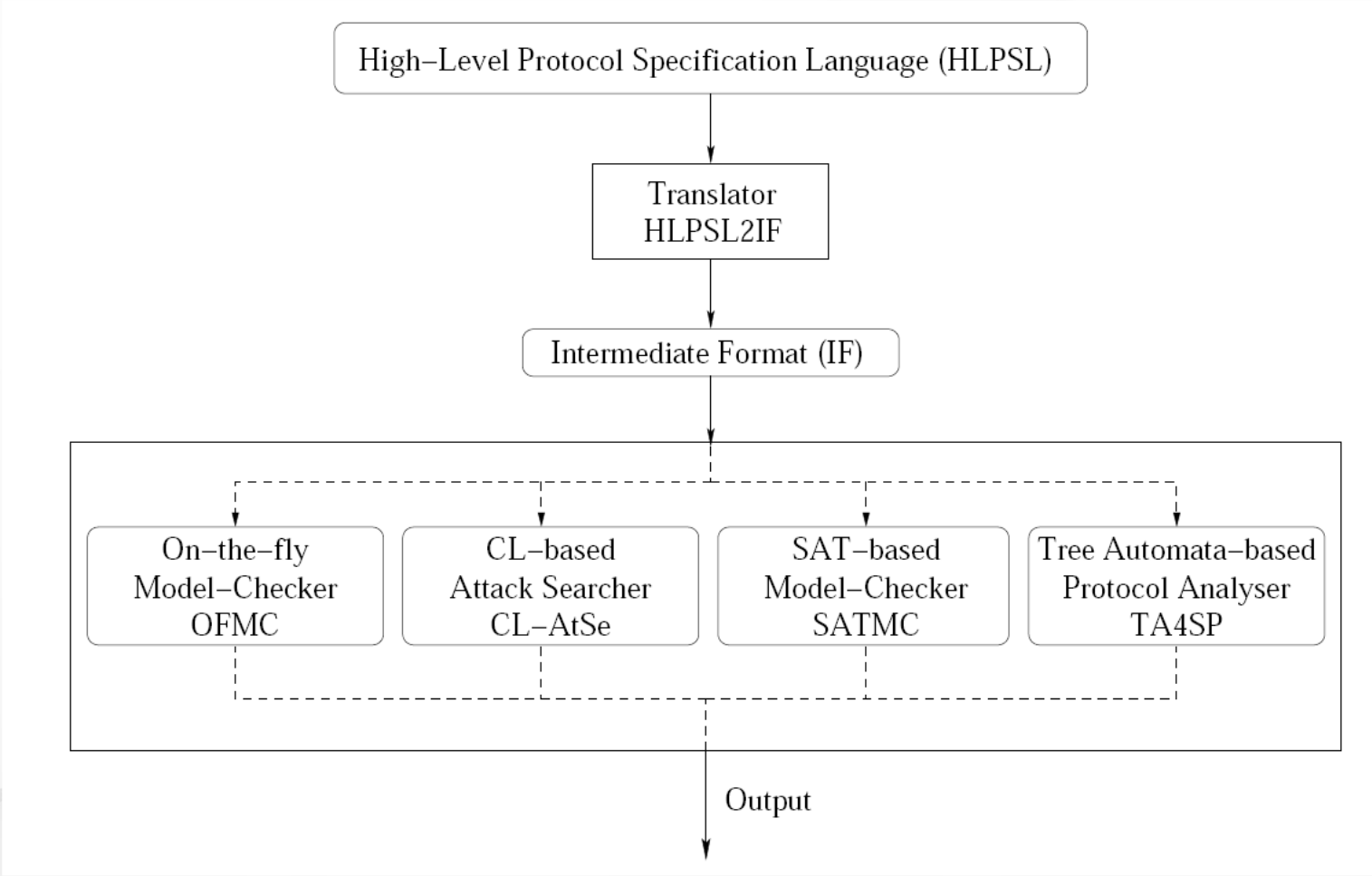
- Dolev-Yao 공격자 모델

- 보안 프로토콜의 정형화된 보안 분석을 위하여 제안된 표준 공격자 모델



- ✓ 프로토콜 … 전달
- ✓ 공격자는 …
- ✓ 암호화된 … 공격자는 … 가능
- ✓ 공격자는 … 능
- ✓ 공격자는 …
- ✓ 도청만을 수행하는 공격자는 수동 공격자로 간주
- ✓ 일반적으로 Alice-Bob 표기법에서 Eve의 E로 표기

AVISPA 개요 (4/7)


High–Level Protocol Specification Language (HLPSL)
Translator
HLPSL2IF
Intermediate Format (IF)
On–the–fly
Model–Checker
OFMC
CL–based
Attack Searcher
CL–AtSe
SAT–based
Model–Checker
SATMC
Tree Automata–based
Protocol Analyser
TA4SP
Output

AVISPA 개요 (5/7)

- 주로 활용되는 Back-End 2가지

- On-the-fly Model-Checker (OFMC)
  - ✓ ETH Zurich의 David Basin 팀에 의해서 개발
  - ✓ Lazy & Demand-driven 검색방식을 채택함공격자는 모든 메시지를 접근, 수정 등이 가능

- Constraint-Logic-based Attack Searcher (CL-AtSe)
  - ✓ INRIA-Lorraine의 M. Rusinowitch 팀 (France)에 의해 개발
  - ✓ 유한 개수의 세션을 지원하고 XOR 관련 연산 지원
  - ✓ 모듈화되어 있고, 자유로운 확장을 위해 개방된 구조

AVISPA 개요 (6/7)

- 활용 범위나 표현법의 차이로 거의 활용되지 않는 Back-End

- The SAT-based Model-Checker (SATMC)
  - ✓ The University of Genova의 A. Armando팀 (Italy)에 의해 개발
  - ✓ Typed된 모델을 지원하고, 유한의 세션을 고려
  - ✓ 입력된 문제를 일련의 SAT solver 호출로 축약하여 분석

- TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols)
  - ✓ INRIA/PAREO 의 Y. Boichut 팀 (France)에 의해 개발
  - ✓ 추상화 기법을 적용하여 무한 세션 모델의 검증을 지원

AVISPA 개요 (7/7)

- AVISPA 팀은 IETF에서 개발되었거나 개발중인 보안 프로토콜의 HLPSL 모델화를 진행하여 Tutorial로 제공
- 33개의 프로토콜에서 112개의 보안문제를 발견

- AVISPA가 널리 사용되는 이유
- 위와 같이 다양한 표준 혹은 표준안의 보안 프로토콜을 예시로 제공하여 추후 보안 프로토콜의 검증을 위해 매우 유용
- 보고에 의하면 대부분의 프로토콜은 수초에 검증이 완료
- 4개의 Back-End가 상호 간의 약점을 보완하여 신뢰성이 높음

02

HLPSL

HLPSL (1/11)

- HLPSL은 역할 (Role) 기반의 언어임
- 즉, 프로토콜이 HLPSL로 변환되면, 프로토콜은 세 개의 역할로 구성됨

- Basic Role (기본 역할): 프로토콜 참여자의 동작을 모델링 함

- Composed Role (복합 역할): 하나의 세션을 모델링 하고, 세션을 구성하는 Basic Role들을 실제로 활성화 함

- Environment Role (환경 역할): C로 예를 들면 main 함수에 해당하는 Role로서 프로토콜의 목표, 가정, 공격자 모델, 세션이 실행되는 방식 등을 정의함

- **HLPSL 역할 실행 순서:**
  Environment Role → Composed Role → Basic Role

HLPSL (2/11)

환경 역할

```
role environment()
def=

  const a, b, s        : agent,
        kas, kbs, kis : symmetric_key

  intruder_knowledge = {a, b, s, kis}

  composition

       session(a,b,s,kas,kbs)
    /\ session(a,i,s,kas,kis)
    /\ session(i,b,s,kis,kbs)

end role
```

복합 역할

```
role session(A,B,S     : agent,
             Kas, Kbs : symmetric_key) def=

local SA, RA, SB, RB SS, RS: channel (dy)

composition
    alice (A, B, S, Kas, SA, RA)
/\  bob   (B, A, S, Kbs, SB, RB)
/\  server(S, A, B, Kas, Kbs, SS, RS)

end role
```

기본 역할

```
role alice(A,B,S : agent,
           Kas : symmetric_key,
           SND, RCV : channel (dy))
  played_by A def=
    local State: nat, Kab: symmetric_key
    init  State := 0
    transition
      ...
  end role
```

HLPSL (3/11)

- HLPSL 변수의 타입

- ✓ Agent: 프로토콜 참여자 (예: Alice, Bob)
- ✓ public_key: 공개키
- ✓ symmetric key: 대칭키
- ✓ nat: 자연수
- ✓ message: 일반적인 메시지를 표현할 때 사용됨
- ✓ text: 해석되지 않은 비트 문자열 (nonce와 같은)
- ✓ hash_func: 해쉬함수
- ✓ bool: 불리언 변수
- ✓ set: 타입화된 변수들의 비정렬된 집합
  예) KeyMap: (agent.public_key) set
  init KeyMap := {a.ka, b.kb, i.ki}

HLPSL (4/11)

- HLPSL 기호

✓ . : concatenation 연산 (예: A||B를 A.B로 표현)
✓ := : 할당 연산자 (예: A := 3)
✓ =, < : 비교 연산자 (예: A = 3)
✓ not, /\ : Not, And 논리연산자
✓ inv : 공개키의 개인키를 의미 (예: inv(PKa)는 PKa의 개인키)
✓ { }_ : 암호화 혹은 전자서명 연산 (예: {M}_K 혹은 {M}_inv(K))
✓ exp: 지수승 표현 (예: exp(g, x)는 gx)
✓ xor: 배타적 논리합 표현
✓ owns: 어떤 변수의 소유권을 표시,
소유권이 표시되면 소유자만이 해당 변수의 값을 변경

HLPSL (5/11)

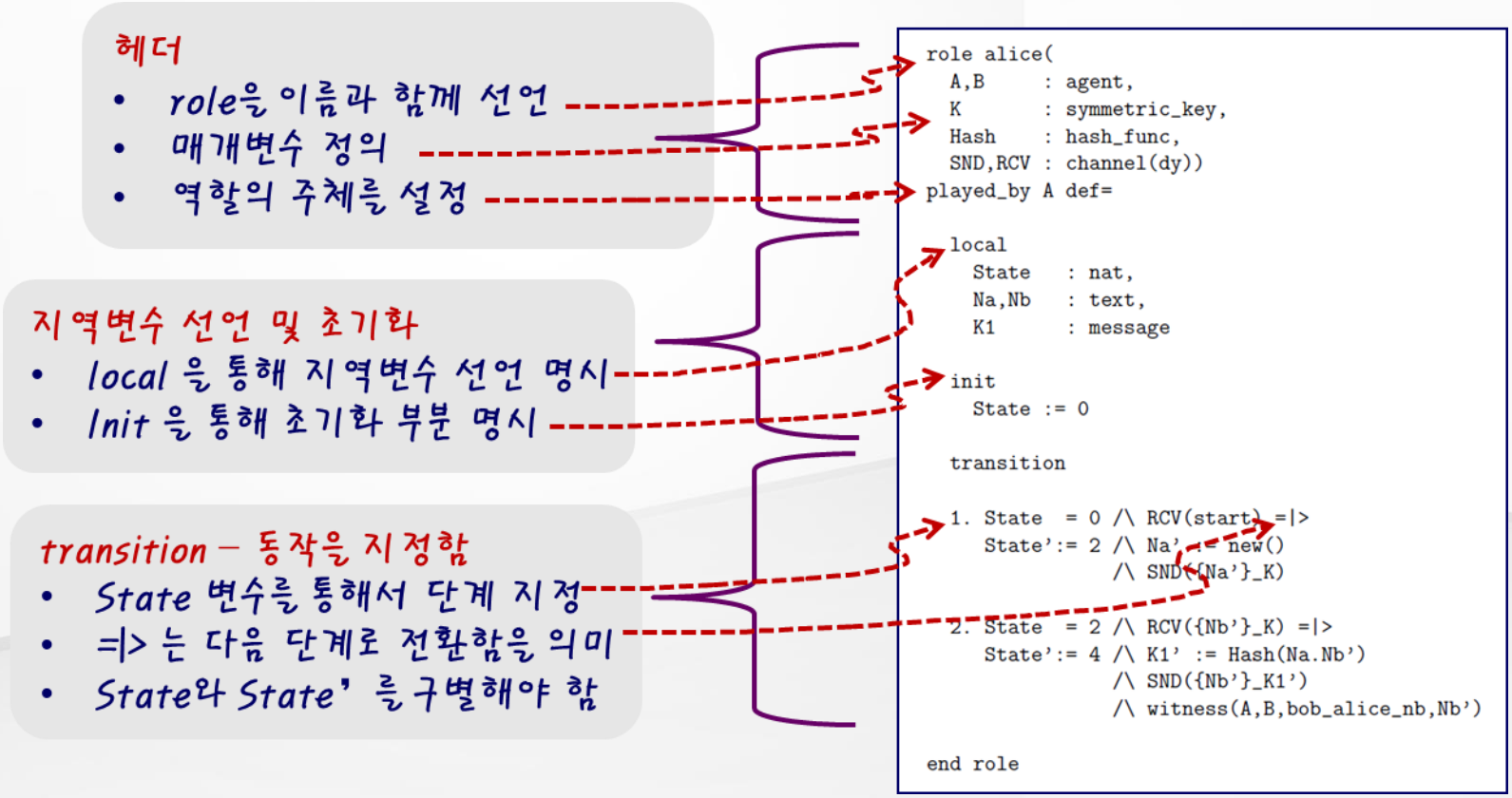
- HLPSL 예제

가정: K는 A와 B의 공유비밀키

(1) A -> B: {Na}_K
(2) B -> A: {Nb}_K
(3) A -> B: {Nb}_K1, where K1=Hash(Na.Nb)

HLPSL (6/11)

- Basic Role (기본 역할)
- Basic Role은 참여자를 모델링하는 HLPSL의 가장 기본적인 도구
- 아래 그림처럼 매개변수를 갖는 함수형태로 표현되고,
  헤더, 지역변수 선언 및 초기화, transition, tail로 구성



HLPSL (7/11)

- Basic Role (기본 역할)
- **헤더**의 매개변수 부분에는 역할이 동작하기 위해 필요한 참여자를 명시
- 역할의 주체는 명시된 참여자 중의 하나를 played_by 명령어를 통해서 지정
- 참여자 사이의 통신을 할 통신 채널을 channel을 명시
- **지역변수**에서는 해당 역할에서만 활용할 인자를 선언하고, 필요에 따라 초기화

```
role alice(
  A,B      : agent,
  K        : symmetric_key,
  Hash     : hash_func,
  SND,RCV  : channel(dy))
played_by A def=

  local
    State    : nat,
    Na,Nb    : text,
    K1       : message

  init
    State := 0
```

HLPSL (8/11)

- Basic Role (기본 역할)
- **Transition**을 통해 실제 프로토콜 동작을 명세
- secret 명령어는 대상 값에 대한 기밀 검증 명령어
- witness와 request (혹은 wrequest)는 인증 검증 명령어
- witness와 request (혹은 wrequest)는 쌍으로 활용
- 인증에 대한 근거를 (즉 메시지) 받으면
  그것을 검증하는 측에서 검증 후에 request 호출하고,
  인증에 대한 근거를 보낸 측에서는 보낸 후 바로 witness를 호출

```
transition

1. State  = 0 /\ RCV(start) =|>
   State'= 2 /\ Na' := new()
             /\ SND({Na'}_K)

2. State  = 2 /\ RCV({Nb'}_K) =|>
   State'= 4 /\ K1' := Hash(Na.Nb')
             /\ SND({Nb'}_K1')
             /\ witness(A,B,bob_alice_nb,Nb')
```

```
transition

1. State  = 1 /\ RCV({Na'}_K) =|>
   State'= 3 /\ Nb' := new()
             /\ SND({Nb'}_K)
             /\ K1'= Hash(Na'.Nb')
             /\ secret(K1',k1,{A,B})

2. State  = 3 /\ RCV({Nb}_K1) =|>
   State'= 5 /\ request(B,A,bob_alice_nb,Nb)
```

HLPSL (9/11)

- Basic Role (기본 역할)

```
role alice(
  A,B     : agent,
  K       : symmetric_key,
  Hash    : hash_func,
  SND,RCV : channel(dy))
played_by A def=

  local
    State   : nat,
    Na,Nb   : text,
    K1      : message

  init
    State := 0

  transition

  1. State  = 0 /\ RCV(start) =|>
     State':= 2 /\ Na' := new()
                /\ SND({Na'}_K)

  2. State  = 2 /\ RCV({Nb'}_K) =|>
     State':= 4 /\ K1' := Hash(Na.Nb')
                /\ SND({Nb'}_K1')
                /\ witness(A,B,bob_alice_nb,Nb')

end role
```

```
role bob(
    A,B     : agent,
    K       : symmetric_key,
    Hash    : hash_func,
    SND,RCV : channel(dy))
played_by B def=

  local


    State   : nat,
    Nb,Na   : text,
    K1      : message

  init
    State := 1

  transition

  1. State  = 1 /\ RCV({Na'}_K) =|>
     State':= 3 /\ Nb' := new()
                /\ SND({Nb'}_K)
                /\ K1':= Hash(Na'.Nb')
                /\ secret(K1',k1,{A,B})

  2. State  = 3 /\ RCV({Nb}_K1) =|>
     State':= 5 /\ request(B,A,bob_alice_nb,Nb)

end role
```

가정: K는 A와 B의 공유비밀키

(1) A -> B: {Na}_K
(2) B -> A: {Nb}_K
(3) A -> B: {Nb}_K1,
where K1=Hash(Na.Nb)

HLPSL (10/11)

- Composed Role (복합 역할) 과 Environment Role (환경 역할)

- Composed Role (복합 역할)
- Basic Role들로 구성된 하나의 세션을 표현
- Basic Role과 구조는 유사하지만 played_by와 transition이 없고 composition이 존재
- composition에서 Basic Role을 실제 호출
- 일반적으로 Composed Role에서 통신채널을 선언

- Environment Role (환경 역할)
- main에 해당하는 역할로 agent와 암호화키, 함수등을 위한 변수를 선언
- 기밀성과 인증을 검증하기 위해 protocol_id 변수를 선언
- 공격자의 지식 범위를 설정
- composition 에서 session이 호출되는 방식을 설정

HLPSL (10/11)

- Composed Role (복합 역할) 과 Environment Role (환경 역할)

- Composed Role (복합 역할)
- 기밀성과 인증을 검증하기 위해 protoco
- 공격자의 지식 범위를 설정
- composition 에서 session이 호출되는 방식을 설정

```
role session(
    A,B  : agent,
    K    : symmetric_key,
    Hash : hash_func)
def=

  local SA, SB, RA, RB : channel (dy)

  composition

     alice(A,B,K,Hash,SA,RA)
  /\ bob  (A,B,K,Hash,SB,RB)

end role
```

```
role environment()
def=

  const a, b, s         : agent,
        ka, kb, ki      : symmetric_key,
        alice_bob_na, k: protocol_id

  intruder_knowledge = {a, b, s, ki}

  composition

     session(a,s,b,ka,kb)
  /\ session(a,s,i,ka,ki)
  /\ session(i,s,b,ki,kb)

end role

goal
  secrecy_of k
  authentication_on alice_bob_na
end goal

environment()
```


감사합니다.