Università degli Studi di Catania
Corso di Laurea in Informatica

Casimiro Greco

Analysis of Attacks to Multi-Protocols

mWSF05

The 2005 miniWorkshop on Security Frameworks

Catania, 16 Dicembre 2005

INTRODUCTION

Many formal methods in last years have been developed

These analysis supposed that protocols run in isolation

MAIN QUESTIONS

Is it realistic to assume that one protocol is the only protocol on the network?

MAIN QUESTIONS

Can protocols interact ?

Could this interaction be used by an intruder?

Use different key structures

Not always a good idea:

1) Key distribution/management problems

2) Limited Resources (smartcards)

MULTI-PROTOCOL ATTACK

An attack that involves more than one protocol sharing network and key structures is called multi-protocol attack

The intruder uses messages from different instances of different protocols

CHOSEN PROTOCOL ATTACK

Kelsey, Schneier, Wagner

Given a correct security protocol, there exists another correct security protocol, such that their composition is incorrect.

- **Chosen Protocol**
- **Target Protocol**

TARGET PROTOCOL

Lowe's Version of Needham-Schroeder

Message 1. $A \rightarrow S : A.S.B$

Message 2. $S \rightarrow A : S.A.\{PK(B),B\}_{PKS(S)}$

Message 3. $A \rightarrow B : A.B.\{N_a.A\}_{PK(B)}$

Message 4. $B \rightarrow S : B.S.A$

Message 5. $S \rightarrow B : S.B.\{PK(A),A\}_{PKS(S)}$

Message 6. $B \rightarrow A : B.A.\{N_a.N_b.B\}_{PK(A)}$

Message 7. $A \rightarrow B : A.B.\{N_b\}_{PK(B)}$

CHOSEN PROTOCOL

$$\text{Message i.}\quad B \rightarrow A : B.A.\{M.N_b.B\}_{PK(A)}$$

$$\text{Message ii.}\quad A \rightarrow B : A.B.\{N_b.B\}_{PKS(A)}$$

<u>**Message I) has the same structure of 6)**</u>

CHOSEN PROTOCOL ATTACK

target **chosen**

Message 3. $E_A \rightarrow B : A.B.\{N_a.A\}_{PK(B)}$

Message 6. $B \rightarrow E_A : B.A.\{N_a.N_b.B\}_{PK(A)}$

Message i. $E_B \rightarrow A : B.A.\{M{=}N_a.N_b.B\}_{PK(A)}$

Message ii. $A \rightarrow E_B : A.B.\{N_b.B\}_{PKS(A)}$

Message 7. $E_A \rightarrow B : A.B.\{N_b\}_{PK(B)}$

B authenticates E_A as A

MULTI-PROTOCOL GUESSING ATTACKS

Guessing Attack: attack where an attacker guesses a poorly chosen secret and then seeks to verify that guess using other information.

Multi-Protocol Guessing Attack: information comes from messages of other protocols

ASSUMPTIONS

1)The passwords being guessed have low entropy

2)The verification of a guess does not need on-line interaction with other parties

Failed attempts are undetectable

EKE (Encrypted Key Exchange)

$$\begin{array}{ll} \text{Msg 1.} & a \rightarrow b : \{pk_a\}_{passwd(a,b)} \\ \text{Msg 2.} & b \rightarrow a : \{\{k\}_{pk_a}\}_{passwd(a,b)} \\ \text{Msg 3.} & a \rightarrow s : \{n_a\}_k \\ \text{Msg 4.} & s \rightarrow a : \{(n_a, n_b)\}_k \\ \text{Msg 5.} & a \rightarrow s : \{n_b\}_k \end{array}$$

GONG

$$\begin{array}{ll} \text{Msg 1.} & a \rightarrow b : \{\cancel{t,}\, n\}_{k1} \\ \text{Msg 2.} & b \rightarrow a : \{f(n)\}_{passwd(a)} \end{array}$$

ATTACK

EKE (Encrypted Key Exchange)

Msg 1. a → b: $\{pk_a\}_{passwd(a,b)}$

Msg 2. b → a: $\{\{k\}_{pk_a}\}_{passwd(a,b)}$

Msg 3. a → s: $\{n_a\}_k$

Msg 4. s → a: $\{(n_a,n_b)\}_k$

Msg 5. a → s: $\{n_b\}_k$

GONG

Msg i. a → b: $\{n\}_{k1}$

Msg ii. b → a: $\{f(n)\}_{passwd(a)}$

1. Guess *passwd(a,b)* from Msg 1 to obtain *pk*a

2. Guess *passwd(a)* from Msg ii. to obtain *f(n)*

3. Learn *n* from *f(n)*, encode it with *pk*a

4. Compare this value with Msg i

If the values coincide, attack takes place.

LITERATURE SAYS...

For all correct protocols, there exists a protocol attack such that the composition contains a security flaw.

.

.

.

What happens with composition of actual protocols?

.

.

.

If all the protocols have been designed according to the guidelines, executing them in parallel will not introduce any new attack.

INTRODUCING THE ANALYSIS

Analysis and tests of Multi-protocol attacks by Cas Cremers

Analyzed two and three concurrent protocols from Clark and Jacob library, SPORE library and works of Boyd and Mathuria

Analyzed 30 different protocols

PROPERTIES

Protocols have been tested on three properties:

1) **Secrecy**

2) **Agreement**

3) **Synchronisation**

READ SEMANTIC

Most agents cannot verify the values received

When an agent expects e.g. a Nonce and accepts:

1) Only constants of type Nonce →**No Type Flaws**

2) Any simple constants →**Basic Type Flaws**

3) Any terms →**Full Type Flaws**

RESULTS

No Type Flaws allowed:

38 Multi-protocols attacks

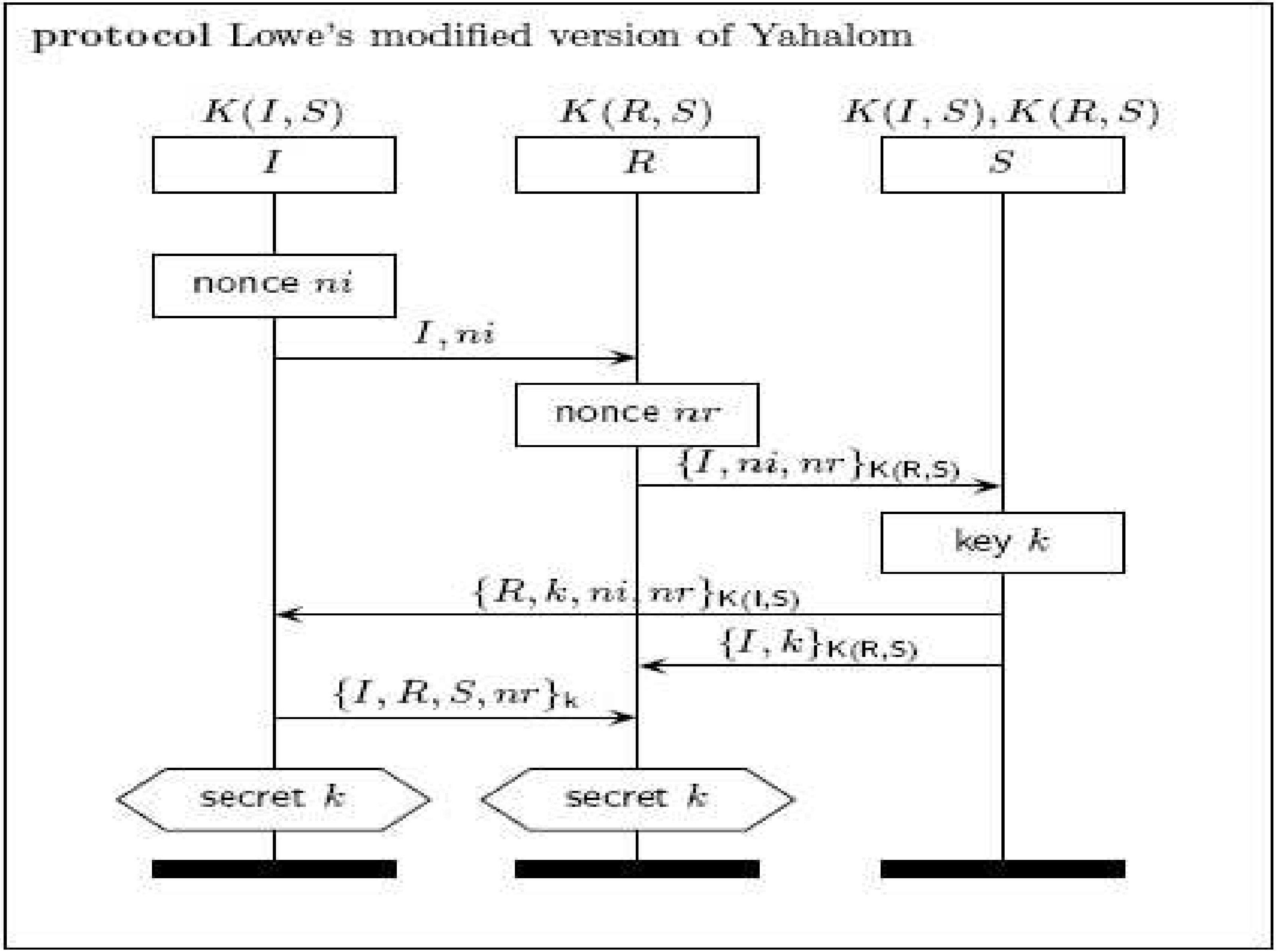
Basic Type Flaws allowed:

41 Multi-protocols attacks

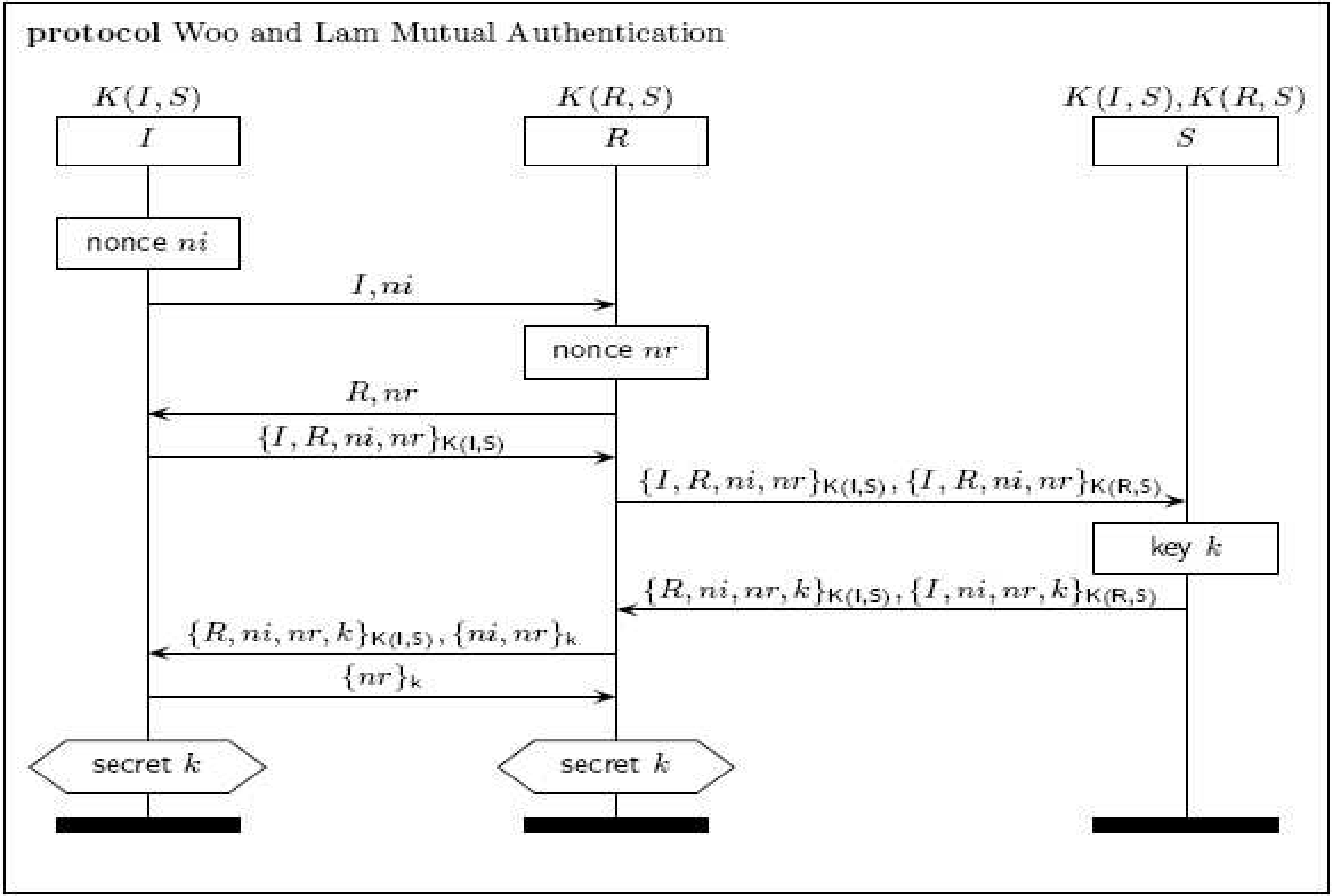
Full Type Flaws allowed:

83 Multi-protocols attacks

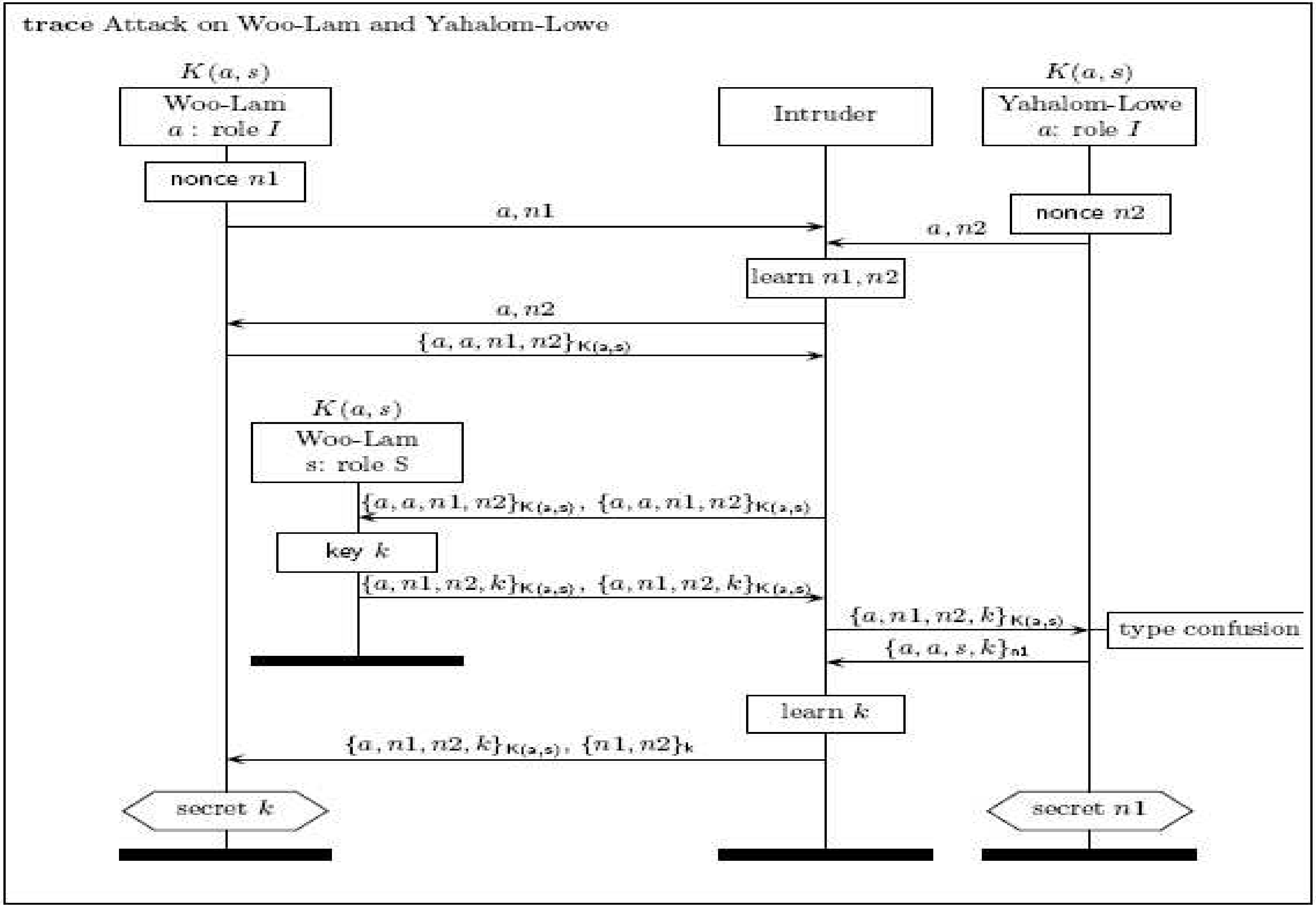
YAHALOM-LOWE



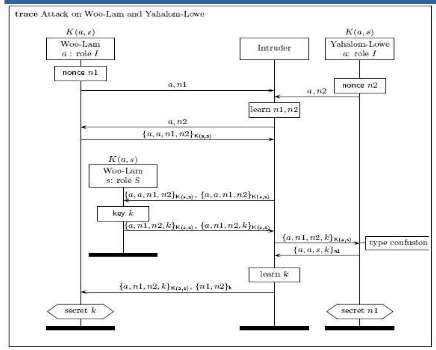
WOO-LAM

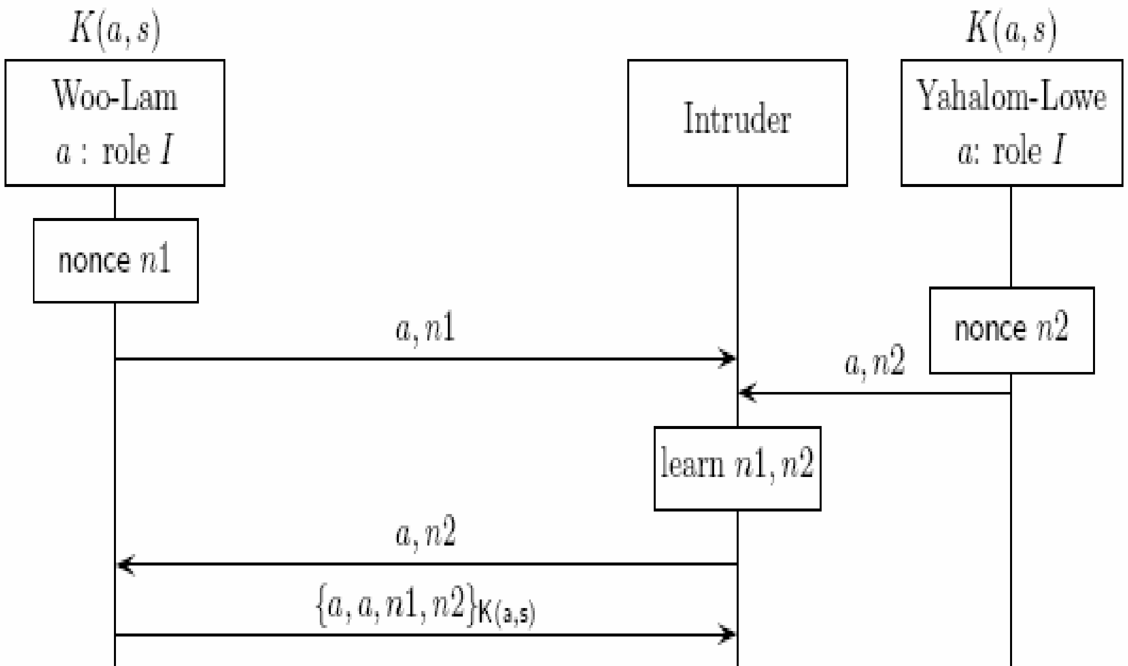

protocol Woo and Lam Mutual Authentication
K(I, S)
I
K(R, S)
R
K(I, S), K(R, S)
S
nonce ni
I, ni
nonce nr
R, nr
{I, R, ni, nr}K(I,S)
{I, R, ni, nr}K(I,S), {I, R, ni, nr}K(R,S)
key k
{R, ni, nr, k}K(I,S), {I, ni, nr, k}K(R,S)
{R, ni, nr, k}K(I,S), {ni, nr}k
{nr}k
secret k
secret k

WOO-LAM & YAHALOM-LOWE



WOO-LAM & YAHALOM-LOWE

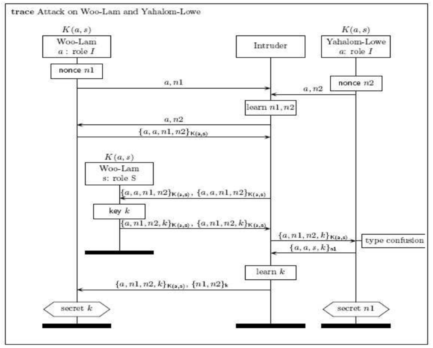


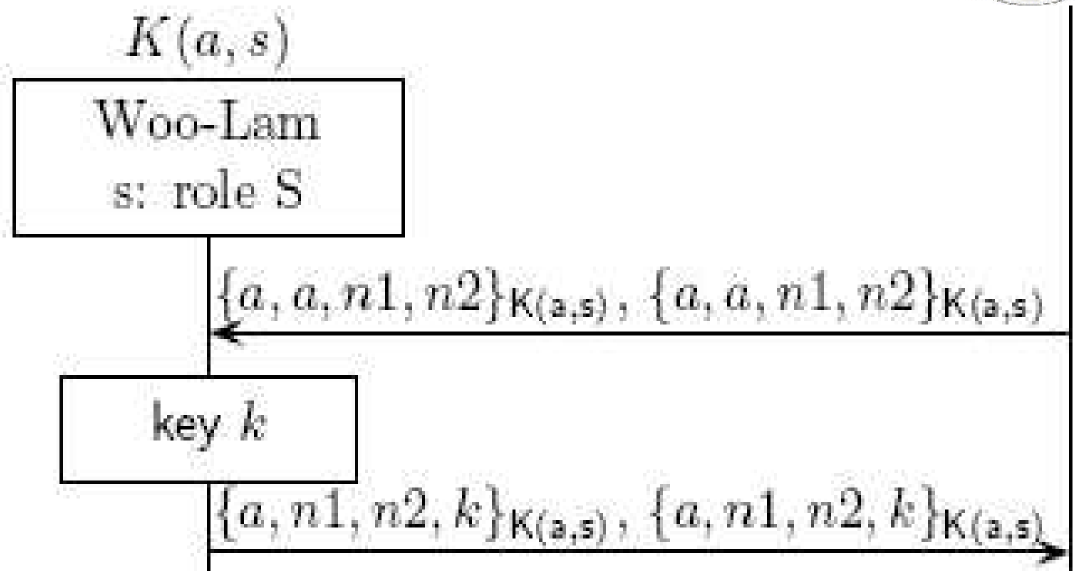


An agent *a* starts the Woo-Lam protocol in the *I* role and sends a fresh nonce *n1*. The agent starts a Yahalom-Lowe session in parallel, in the *I* role. *a* creates and sends *n2*. The two nonces are intercepted by the intruder.

The intruder sends the nonce *n2* to *a* in the Woo-Lam protocol, as if it was sent by a Woo-Lam responder role. The agent responds with a server request with the names of both the agents and the nonces.

WOO-LAM & YAHALOM-LOWE

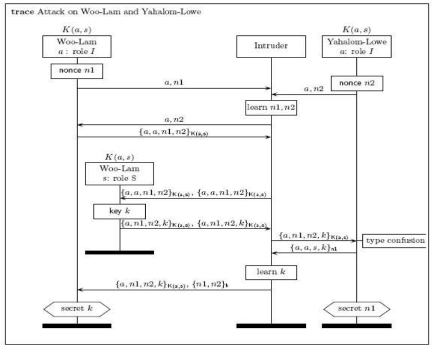


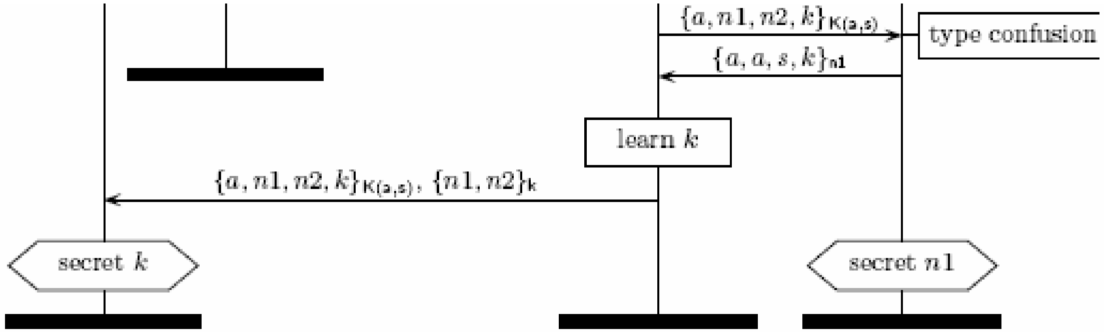


The last message sent from the agent is intercepted by the intruder, concatenated with itself and sent to the Woo-Lam server S

The server generates a fresh session key *k* and sends back two identical messages. One of this is redirected to the Yahalom-Love *I* role.

WOO-LAM & YAHALOM-LOWE





I is waiting a message in the form $\{a,Key,n2,Nonce\}_{K(a,s)}$ *but…*

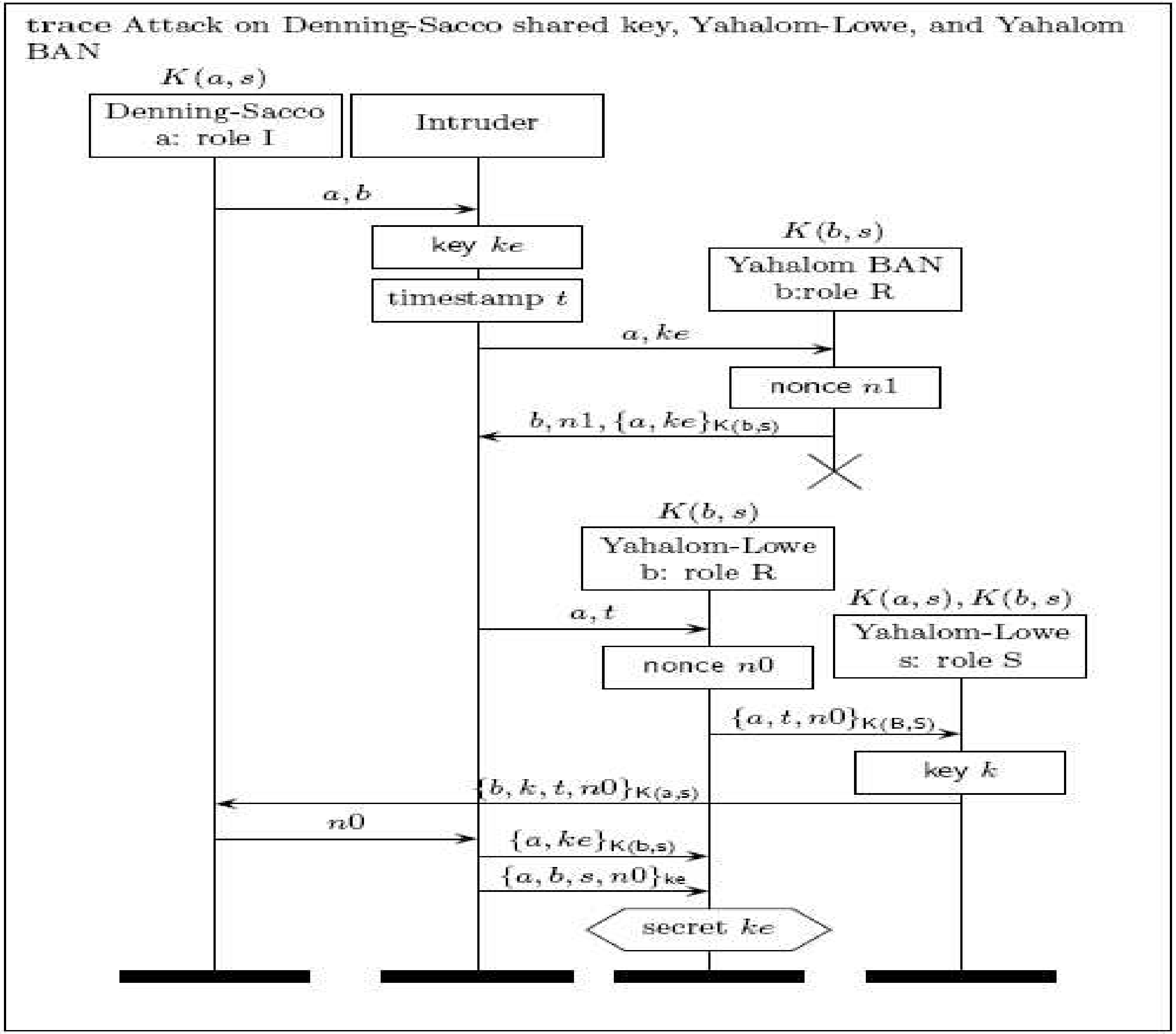
the agent can't distinguish the difference between a Session Key and a Nonce (Basic Type Flaw)

The intruder knows *n1*. He can decrypt the message and learn the session key *k*

so he accepts $\{a,n1,n2,k\}_{K(a,s)}$ and sends $\{a,a,s,k\}_{n1}$ encoding the session key *k* with *n1*

sending the message $\{a,n1,n2,k\}_{K(a,s)}$, $\{n1,n2\}_K$ the intruder completes the attack

EN PASSANT…


trace Attack on Denning-Sacco shared key, Yahalom-Lowe, and Yahalom BAN
K(a, s)
Denning-Sacco
a: role I
Intruder
a, b
key ke
timestamp t
K(b, s)
Yahalom BAN
b:role R
a, ke
nonce n1
b, n1, {a, ke}K(b,s)
K(b, s)
Yahalom-Lowe
b: role R
a, t
nonce n0
K(a, s), K(b, s)
Yahalom-Lowe
s: role S
{a, t, n0}K(B,S)
key k
{b, k, t, n0}K(a,s)
n0
{a, ke}K(b,s)
{a, b, s, n0}ke
secret ke

SCYTHER TOOL

Developed at the ECSS group of the Technical University of Eindhoven as a part of the PhD research performed by Cas Cremers

Scyther rev. 1410

SCYTHER'S TECHNIQUES

Security protocols are analyzed with two different techniques:

1) Finite state model checker

2) Backward symbolic state search

Use the Arachne engine, based on the Athena method. It supports tickets and type flaws

COMMAND-LINE SWITCHES

Usage: scyther [switches] [-|FILE] [-o FILE]

--help
print help and exit

--version
print version information and exit. It shows the Subversion revision number, and whether or not Scyther was built with debugging support.

Input/Output

-

If the filename is set to '-', input is read from stdin.

-o, --output=FILE

output file (default is stdout)

-e, --empty

do not generate output

-P, --proof

generate proof output in ASCII

-C, --class

Do not instantiate variables that the intruder can instantiate at will, and leaves them visibly as variables.

-S, --summary

show summary on stdout instead of stderr

Algorithms and checks

-**M, --modelchecker**
use ModelChecker

-a, --arachne
use ArachneEngine

-m, --match=[int]
MatchingMethod (default is 0)
0: Typed matching

1: Allow for basic typeflaws

2: Detect all typeflaw attacks

COMMAND-LINE SWITCHES

Pruning of the searches (Bounds)

-p, --prune=[int]

Pruning method (default is 2)

0: Explore all traces.

1: Do not explore traces which have more than one security violation.

2: Once an attack is found, only scan shorter traces.

-l, --max-length=[int]

prune traces longer than [int] events

-r, --max-runs=[int]

create at most [int] runs. If [int] is zero, the Arachne method can perform an unbounded search.

For the Modelchecker, this means that the number of runs is either this number, or the number of runs defined in the input file, whichever is smaller.

For the Arachne (theorem proving) method, this is simply the maximum number of runs involved in the proof.

--max-attacks=[int]

stop exploring the state space after finding [int] attacks.

INPUT: Hwang modified version of Neumann Stubblebine

```
usertype Server, SessionKey, TimeStamp, TicketKey;
usertype ExpiredTimeStamp;

secret k: Function;

const a, b, e: Agent;
const s: Server;
const Fresh: Function;
const ne: Nonce;
const kee: SessionKey;
untrusted e;
```

INPUT: Hwang modified version of Neumann Stubblebine

```
protocol neustub-Hwang(I,R,S)
{
  role I
  {
    const Ni,Mi: Nonce;
    var Nr,Mr: Nonce;
    var T: Ticket;
    var Tb: TimeStamp;
    var Kir: SessionKey;

    send_1(I,R, I, Ni);
    read_3(S,I, { R,Ni,Kir,Tb}k(I,S), T, Nr);
    send_4(I,R,T,{Nr}Kir);
    send_5(I,R,Mi,T);
    read_6(R,I,Mr,{Mr}Kir);
    send_7(I,R,{Mr}Kir);

    claim_I1(I,Secret, Kir);
    claim_I2(I,Niagree);
    claim_I3(I,Nisynch);
    claim_I4(I,Empty,(Fresh,Kir));
  }
```

INPUT: Hwang modified version of Neumann Stubblebine

```
role R
  {
    var Ni,Mi: Nonce;
    const Nr,Mr: Nonce;
    var Kir: SessionKey;
    const Tb: TimeStamp;
    var T: Ticket;

    read_1(I,R, I, Ni);
    send_2(R,S, R, {I, Ni, Tb, Nr}k(R,S));
    read_4(I,R,{I,Kir,Tb}k(R,S),{Nr}Kir);
    read_5(I,R,Mi,T);
    send_6(R,I,Mr,{Mr}Kir);
    read_7(I,R,{Mr}Kir);

    claim_R1(R,Secret, Kir);
    claim_R2(R,Niagree);
    claim_R3(R,Nisynch);
    claim_R4(R,Empty,(Fresh,Kir));
  }
```

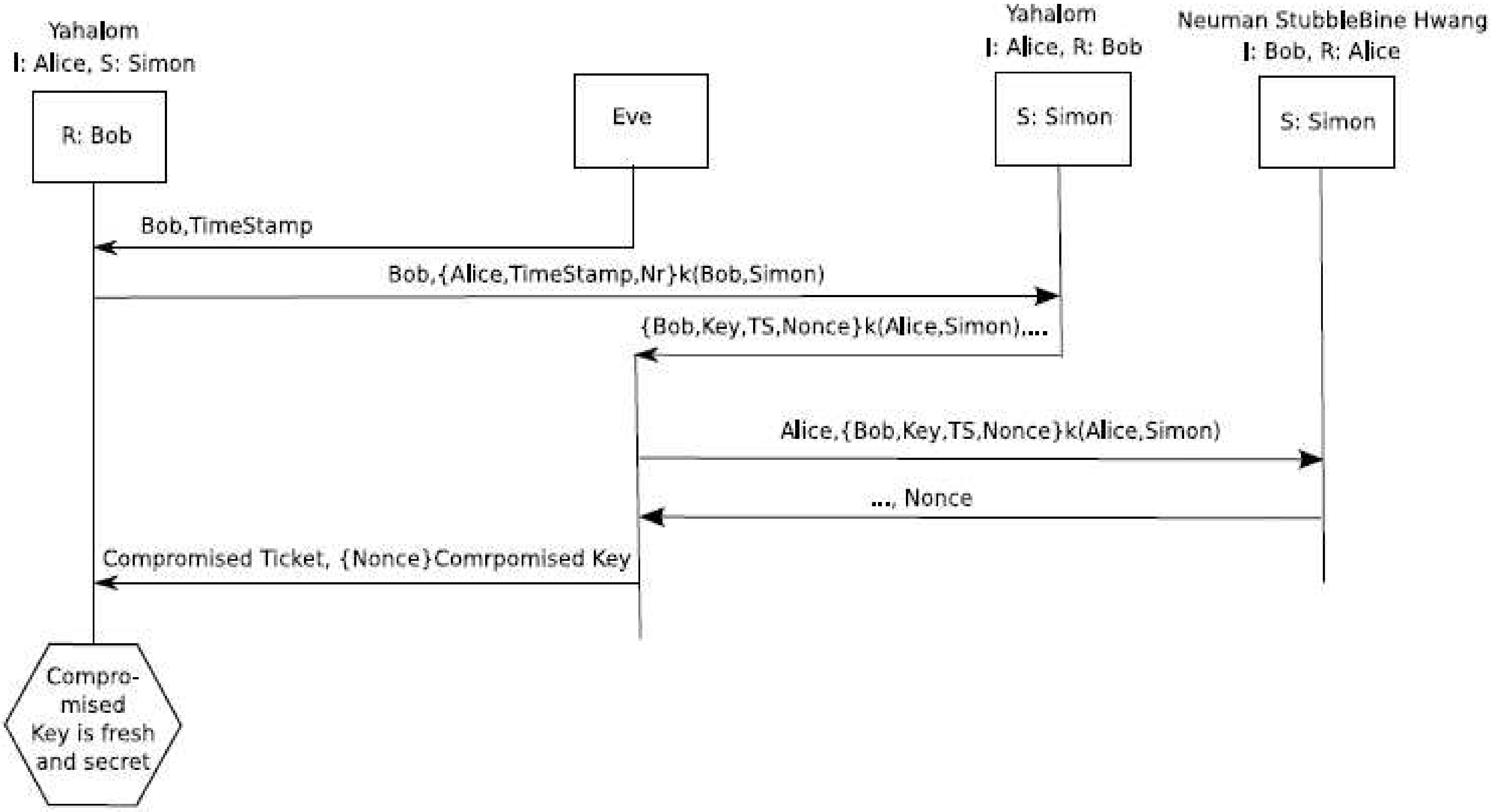
INPUT: Hwang modified version of Neumann Stubblebine

```
role S
{
   var Ni, Nr: Nonce;
   const Kir: SessionKey;
   var Tb: TimeStamp;
   read_2(R,S, R, {I,Ni,Tb,Nr}k(R,S));
   send_3(S,I, { R, Ni, Kir, Tb}k(I,S), { I,Kir,Tb}k(R,S),Nr );
}
```

INPUT:Yahalom

```
secret k : Function;
usertype SessionKey;
const Fresh: Function;
protocol yahalom(I,R,S)
{
    role I
    {
        const Ni: Nonce;
        var Nr: Nonce;
        var T: Ticket;
        var Kir: SessionKey;
        send_1(I,R, I,Ni);
        read_3(S,I,
{R,Kir,Ni,Nr}k(I,S), T );
        send_4(I,R, T, {Nr}Kir );
        claim_I1(I, Secret,Kir);
        claim_I2(I, Nisynch);
        claim_I3(I, Empty,
(Fresh,Kir));
    }
role R
    {
        const Nr: Nonce;
        var Ni: Nonce;
        var T: Ticket;
        var Kir: SessionKey;
        read_1(I,R, I,Ni);
        send_2(R,S, R, {I,Ni,Nr}k(R,S) );
        read_4(I,R, {I,Kir}k(R,S) , {Nr}Kir
);
        claim_R1(R, Secret,Kir);
        claim_R2(R, Nisynch);
        claim_R3(R, Empty, (Fresh,Kir));
    }
    role S
    {
        const Kir: SessionKey;
        var Ni,Nr: Nonce;
        read_2(R,S, R, {I,Ni,Nr}k(R,S) );
        send_3(S,I, {R,Kir,Ni,Nr}k(I,S),
{I,Kir}k(R,S) );
    }
}
```

Using --latex

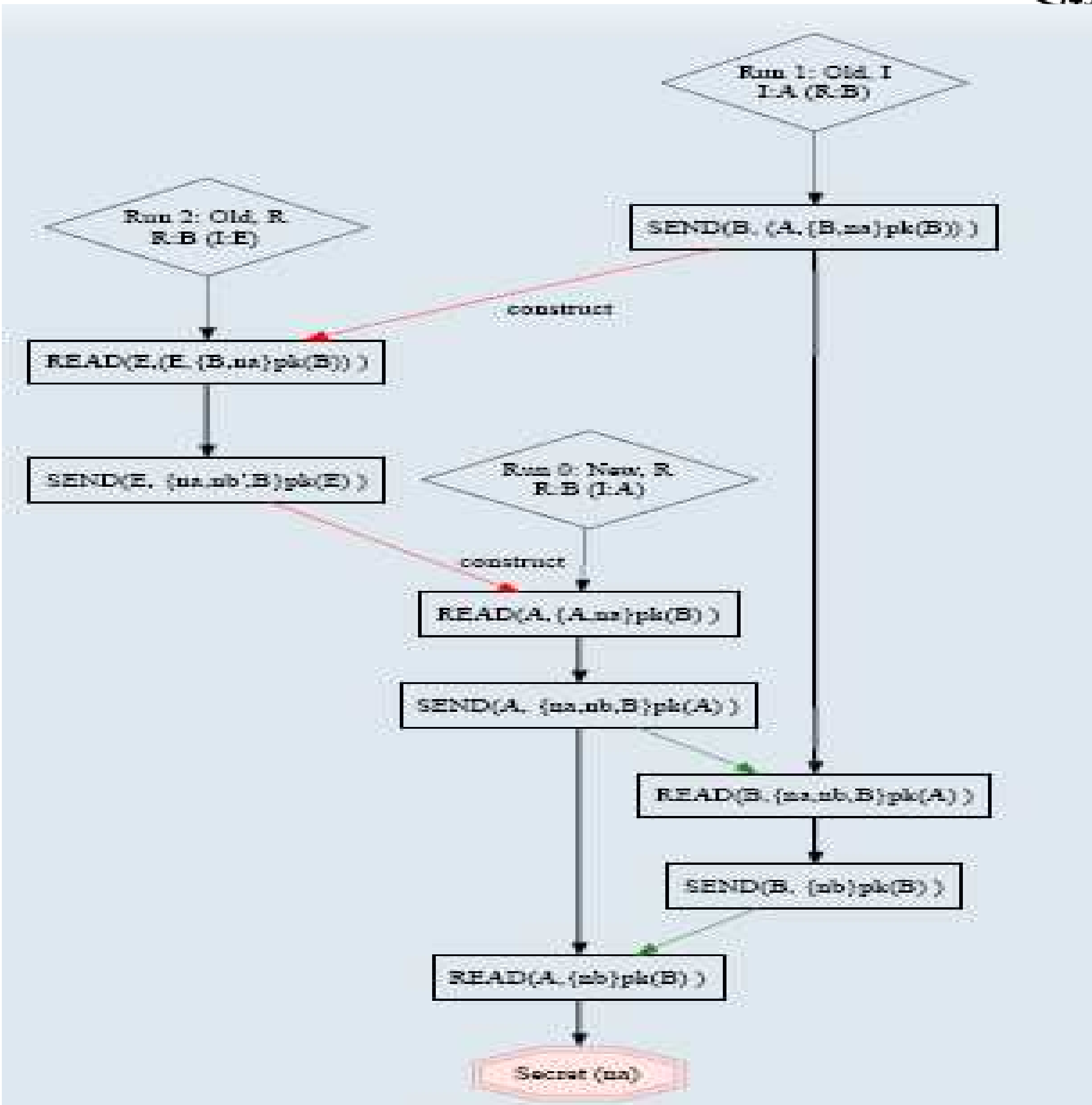

Yahalom
I: Alice, S: Simon
R: Bob
Eve
Yahalom
I: Alice, R: Bob
S: Simon
Neuman StubbleBine Hwang
I: Bob, R: Alice
S: Simon
Bob,TimeStamp
Bob,{Alice,TimeStamp,Nr}k(Bob,Simon)
{Bob,Key,TS,Nonce}k(Alice,Simon),...
Alice,{Bob,Key,TS,Nonce}k(Alice,Simon)
..., Nonce
Compromised Ticket, {Nonce}Comrpomised Key
Compro-
mised
Key is fresh
and secret

GRAPHVIZ OUTPUT

[casimiro@localhost Scyther]$ cat yahalom.spdl neumannstub-hwang.spdl |scyther -

```
digraph semiState1 {
        label = "[Id 1] Protocol neustub-Hwang, role R, claim type Nisynch";
                r0i0 [shape=box,label="READ_1(Alice,(Alice,Ni#1) )"];
                s0 [label="Run 0: neustub-Hwang, R\nR:Bob (I:Alice, S:Simon)",
shape=diamond];
                s0 -> r0i0;
                r0i1 [shape=box,label="SEND_2(Simon,
(Bob,{Alice,Ni#1,Tb#0,Nr#0}k(Bob,Simon)) )"];
                r0i0 -> r0i1 [style="bold", weight="10.0"];
                r0i2
[shape=box,label="READ_4(Alice,({Alice,Kir#2,Tb#0}k(Bob,Simon),{Nr#0}Kir#2) )"];
                r0i1 -> r0i2 [style="bold", weight="10.0"];
                r0i3 [shape=box,label="READ_5(Alice,(Nonce#0,Ticket#0) )"];
                r0i2 -> r0i3 [style="bold", weight="10.0"];
                r0i4 [shape=box,label="SEND_6(Alice, (Mr#0,{Mr#0}Kir#2) )"];
                r0i3 -> r0i4 [style="bold", weight="10.0"];
                r0i5 [shape=box,label="READ_7(Alice,{Mr#0}Kir#2 )"];
                r0i4 -> r0i5 [style="bold", weight="10.0"];
                r0i6 [shape=box,label="CLAIM_R1( Secret, Kir#2 )"];
                r0i5 -> r0i6 [style="bold", weight="10.0"];
                r0i7 [shape=box,label="CLAIM_R2( Niagree, * )"];
                r0i6 -> r0i7 [style="bold", weight="10.0"];
                r0i8
[style=filled,fillcolor=mistyrose,color=salmon,shape=doubleoctagon,label="CLAIM_R3(
Nisynch, * )"];
                r0i7 -> r0i8 [style="bold", weight="10.0"];
                r1i0 [shape=box,label="SEND_1(Bob, (Alice,Ni#1) )"];
                s1 [label="Run 1: neustub-Hwang, I\nI:Alice (R:Bob, S:Simon)",
shape=diamond];
                s1 -> r1i0;
                r1i1
[shape=box,label="READ_3(Simon,({Bob,Ni#1,Kir#2,T
b#0}k(Alice,Simon),Ticket#1,Nr#0) )"];
                r1i0 -> r1i1 [style="bold", weight="10.0"];
                r1i2 [shape=box,label="SEND_4(Bob,
(Ticket#1,{Nr#0}Kir#2) )"];
                r1i1 -> r1i2 [style="bold", weight="10.0"];
                r2i0
[shape=box,label="READ_2(Bob,(Bob,{Alice,Ni#1,Tb#
0,Nr#0}k(Bob,Simon)) )"];
                s2 [label="Run 2: neustub-Hwang,
S\nS:Simon (I:Alice, R:Bob)", shape=diamond];
                s2 -> r2i0;
                r2i1 [shape=box,label="SEND_3(Alice,
({Bob,Ni#1,Kir#2,Tb#0}k(Alice,Simon),{Alice,Kir#2,Tb#
0}k(Bob,Simon),Nr#0) )"];
                r2i0 -> r2i1 [style="bold", weight="10.0"];
        r1i0 -> r0i0 [color=forestgreen];
        r1i2 -> r0i2 [label="construct",color=red];
        r2i1 -> r1i1 [label="construct",color=red];
        r0i1 -> r2i0 [color=forestgreen];
        { rank = same; r1i0; }          // rank 0
        { rank = same; r0i0; }          // rank 1
        { rank = same; r0i1; }          // rank 2
        { rank = same; r2i0; }          // rank 3
        { rank = same; r2i1; }          // rank 4
        { rank = same; r1i1; }          // rank 5
        { rank = same; r1i2; }          // rank 6
        { rank = same; r0i2; }          // rank 7
        { rank = same; r0i3; }          // rank 8
        { rank = same; r0i4; }          // rank 9
        { rank = same; r0i5; }          // rank 10
        { rank = same; r0i6; }          // rank 11
        { rank = same; r0i7; }          // rank 12
        { rank = same; r0i8; }          // rank 13
```

GRAPHVIZ OUTPUT



ASCII OUTPUT

[casimiro@localhost Scyther]$ cat yahalom.spdl neumannstub-hwang.spdl |scyther --summary -

```
states   281
time     9.000e-02
st/sec   3.122e+03
claim    neustub-Hwang    R        Nisynch_R3       found:   1         failed: 1
claim    neustub-Hwang    R        Niagree_R2       found:   1         failed: 1
claim    neustub-Hwang    R        Secret_R1        found:   1         correct:
complete_proof
claim    neustub-Hwang    I        Nisynch_I3       found:   2         failed: 2
claim    neustub-Hwang    I        Niagree_I2       found:   2         failed: 2
claim    neustub-Hwang    I        Secret_I1        found:   1         correct:
complete_proof
claim    yahalom R        Nisynch_R2       found:   1         failed: 1
claim    yahalom R        Secret_R1        found:   1         correct: complete_proof
claim    yahalom I        Nisynch_I2       found:   1         failed: 1
claim    yahalom I        Secret_I1        found:   1         correct: complete_proof
```

SUMMARY

Protocol name	Multi-protocol attack
Bilateral Key Exchange	yes
Boyd key agreement	yes
Denning-Sacco shared key	yes
Gong (nonce based)	yes
Gong (nonce based, version 2)	yes
ISO ccitt 509 (BAN version)	yes
ISO IEC 11770 2-13	no
Kao-Chow	yes
Kao-Chow (version 2)	yes
Kao-Chow (version 3)	yes
KSL (based on Kerberos)	yes
Needham-Schroeder mutual authentication	yes
Needham-Schroeder-Lowe mutual authentication	yes
Needham-Schroeder symmetric	yes
Needham-Schroeder symmetric (amended)	yes
Otway-Rees	no*
SOPH Secret Out, Public Home	yes
Splice-AS	no*
Splice-AS Hwang and Chen modified	no*
Splice-AS Hwang and Chen modified (Clark Jacob)	yes
TMN	no*
Wide Mouthed Frog (Brutus version)	yes
Woo and Lam pi f (unilateral authentication)	yes
Woo-Lam mutual authentication	yes
Yahalom	yes
Yahalom (BAN)	yes
Yahalom Lowe modified	yes
Yahalom Paulson strengthened	yes

(*) There are attacks for these protocols when running in isolation: multi-protocol attacks do not introduce any new attacks on claims that were correct in isolation.

PREVENTING ATTACKS

Explicitness

If all protocols are **consistently tagged**, multi-protocols attacks cannot occur.

Tagging:

Instead of $\{\ldots\}_K$ add a tag within the encryption.

- $\{$"woo-lam",…$\}_K$
- $\{$"yahalom",…$\}_K$

Not always possible

CONCLUSION

MULTI-PROTOCOLS ATTACK ARE A REAL PROBLEM!!!

We must be cautions with the deployment of “probably correct” protocols.

Ambiguos authentication can easily cause problems and is likely to occurr.

Analyze the interactions of different protocols in the same network is very important.

THANKSGIVING

CAS CREMERS

GIJS HOLLESTELLE