

Formal Specification of Sectrace

A Protocol to Set up Security Associations in IPSec Networks

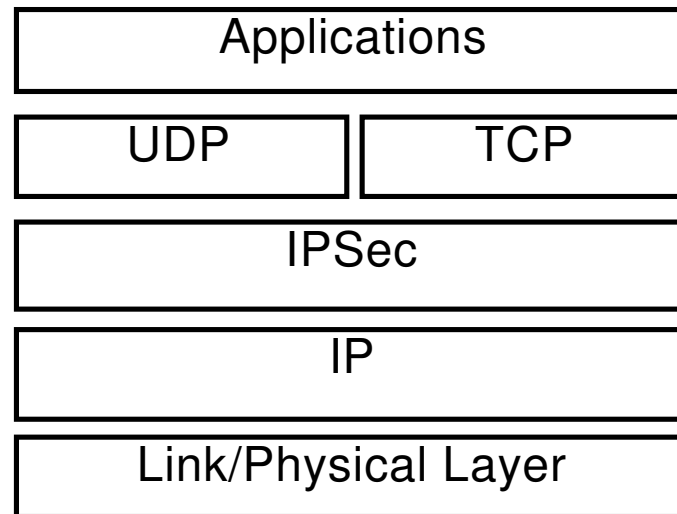
Mark-Oliver Stehr
University of Illinois at Urbana-Champaign

Joint Work with
Ambarish Sridharanarayanan

Introduction

- IPSec: Security Architecture for the Internet Protocol (RFC 2401)
- IPSec provides Authentication and Encryption for IP Packets
- Implemented between Network and Transport Layer

IPSec Protocol Stack:

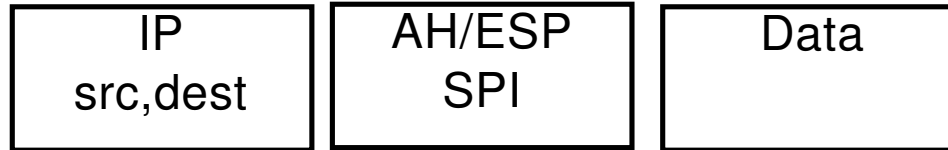


Simple IPSec Packet:



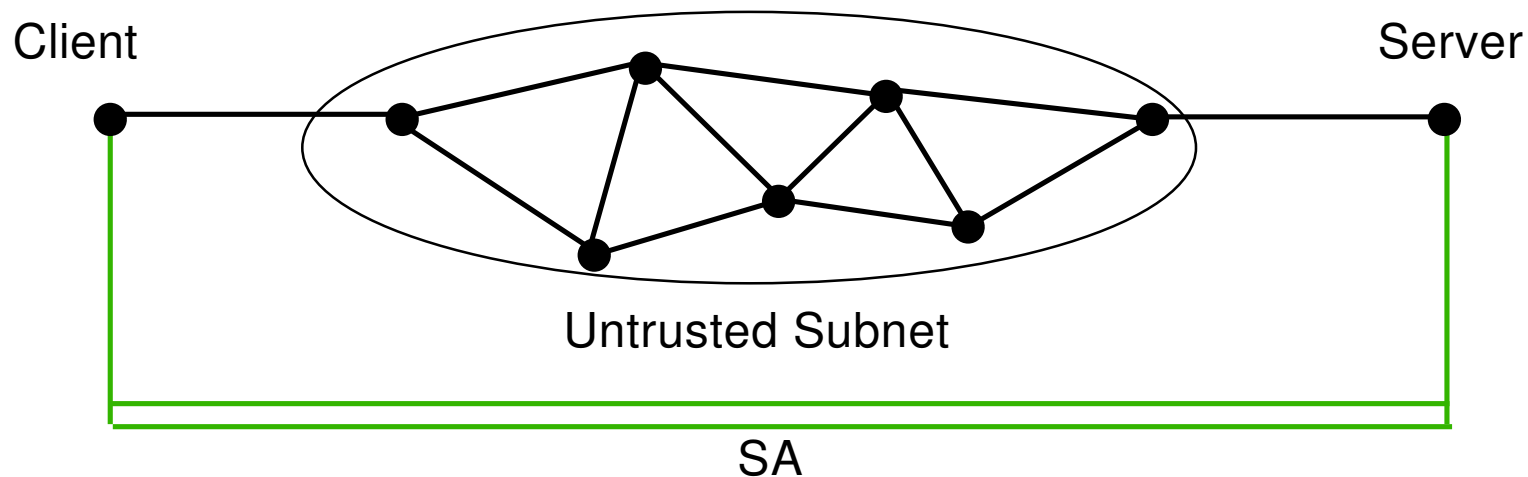
Security Associations

Simple IPsec Packet:



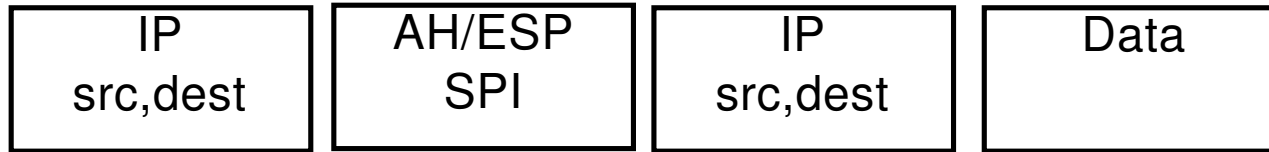
- Security Associations (SAs) are shared parameters between nodes (e.g. secret key)
- Security Parameter Index (together with dest) determines which SA to use

Simple Scenario:

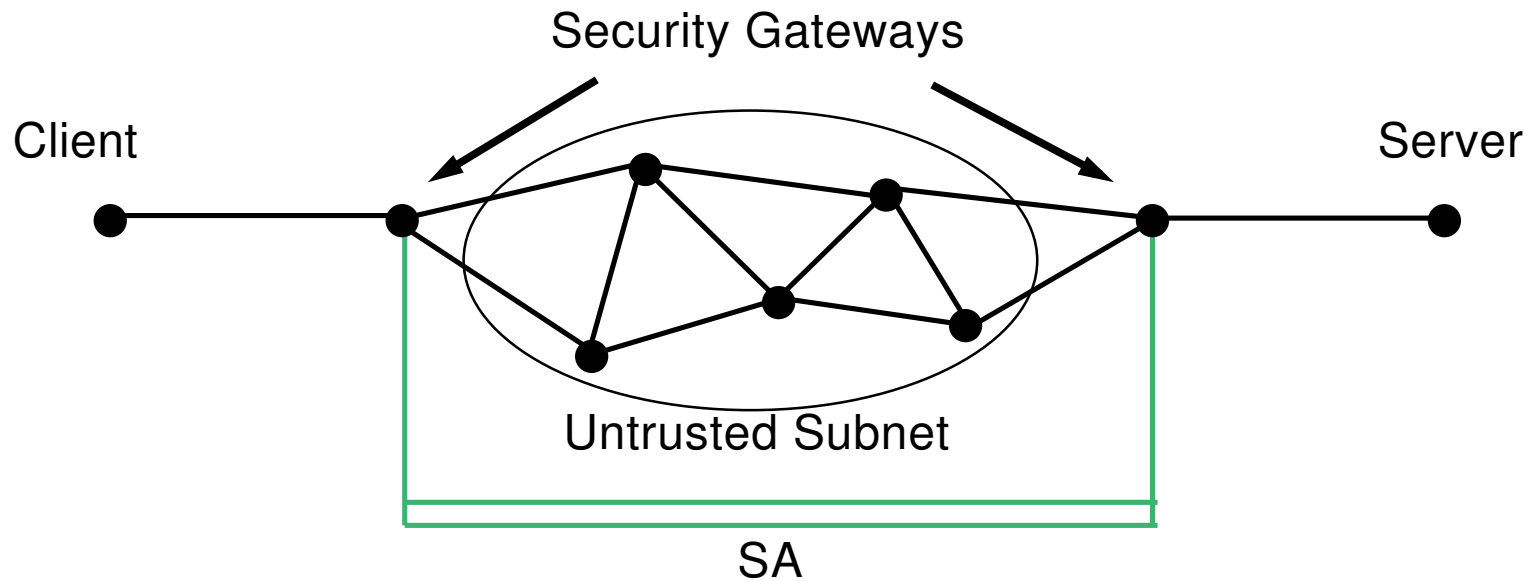


Tunneling

Tunnel Mode IPsec Packet:

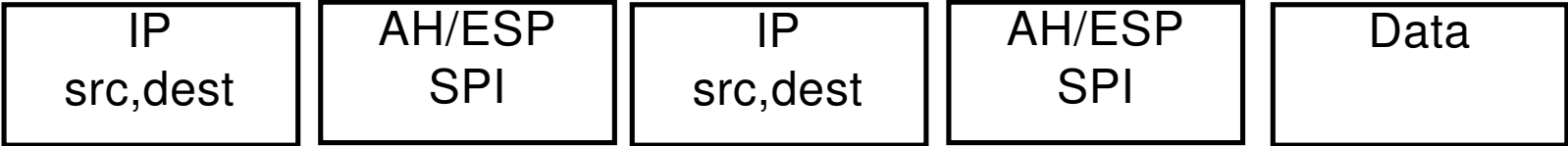


Tunnel Mode Scenario:

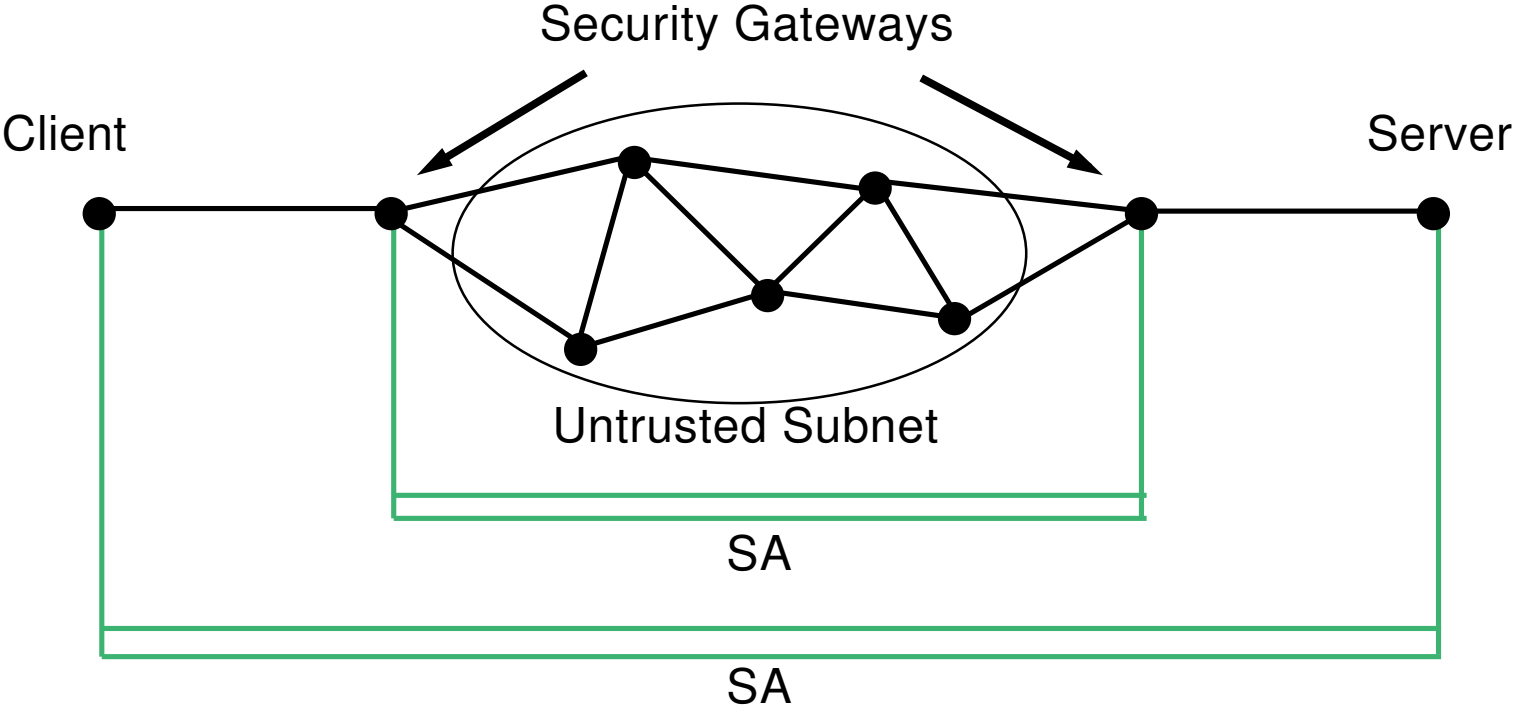


Nested SAs

Nested IPsec Packet:



Nested SA Scenario:



Sectrace

- Reference: Secure Traceroute (Sectrace), Draft, December 2002, by Carl A. Gunter, Alwyn Goodloe, and Michael McDougall
- IPSec assumes existence of Security Associations (SAs)
- Local Security Policy Database determines how SAs are used
- Sectrace sets up Security Associations and modifies the Security Policy Databases correspondingly
- Security Associations are set up by negotiating parameters (secret key, etc.) using Public Key Infrastructure (PKI)
- SA from A to B can be set up if A is trusted by B, i.e. the root certificate authority of A is among those trusted by B

The Maude Methodology

- Formal modelling using Maude within rewriting logic and its membership equational sublogic
- Key questions:
 1. Does the formal model adequately capture intended model ?
 2. Does the formal model have the desired properties ?
- Light-weight techniques:
 - Execution
 - State space exploration
 - Model checking
- Heavy-weight Techniques:
 - Informal mathematical proofs
 - Rigorous formal proofs

Formal Spec of Sectrace

- Important Qualities:
 - Small representational distance
(minimize gap between formal model and reality)
 - Abstract from all nonessential aspects
(to reduce explosion of complexity)
 - Simplicity, mathematical elegance
(enables better understanding)
 - Modularity: IP layer, IPSec layer, sectrace layer, ...
(increase maintainability)
 - Executability
(allows use of light-weight formal methods)
- Development cycle:
 - Several earlier versions of formal specification
 - Debugging/Validation using light-weight methods
 - Interaction with the authors of Sectrace draft specification

Formal Spec of Sectrace

"The client initiates the protocol by sending an RReq toward a server S.
When it does so it sets the RReqTmt timeout and sets OReq to be RReq."

```
r1 [sectrace-start]:  
  sectrace-start(node,client,server) =>  
  sectrace-rreq(node,client,server,sMessageList(rreq(client,server)),eMessageList)  
  ipsec-send(node,ip(client,server,rreq(client,server))) .
```

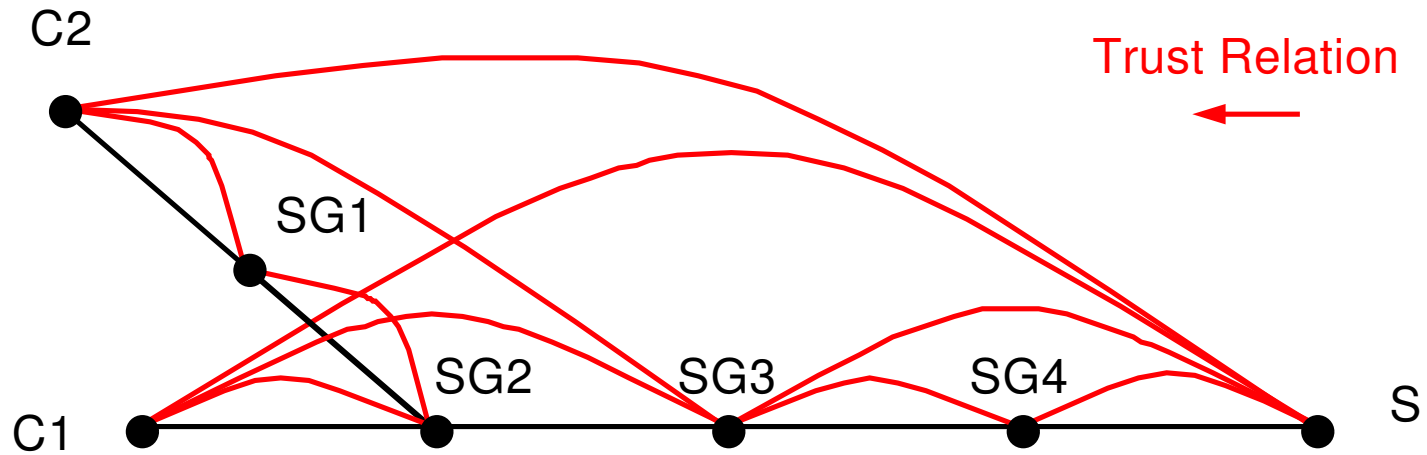
Formal Spec of Sectrace

"If the client receives an RRep message, then it first checks whether it matches the outstanding request OReq. If it does not, then the RRep is ignored. ... It inspects the collection of roots in the message and the RRep list to choose an initiator the sectrace SA selection protocol. The client creates an SAREq with the chosen initiator (SAREq.dst) and uses the source of the RRep as the requested responder (SAREq.resp = RRep.src). This SAREq is entered as the outstanding request. The SAREq and RRep messages are added to the list RRepLst of prior RRep messages."

```
cr1 [sectrace-rrep-3]:
  sectrace-rrep(node, client, server, sMessageList(rreq(client, server)), rreplist)
  rootinfo(node, myroot, mytrustedroots)
  ipsec-delivered(node, rinterface, attrs, sabundle, message) =>
  sectrace-rrep3(node, client, server,
    sMessageList(sareq(client, server, initiator, responder)),
    (rreplist sMessageList(message)))
  rootinfo(node, myroot, mytrustedroots)
  if not(contains(rreplist, message)) /\
    ip(responder, dest, rrep(client, server, root, trustedroots, done)) := message /\
    done == false /\
    initiator := select(client, myroot, (rreplist sMessageList(message)), responder) /\
    initiator /= client .

cr1 [sectrace-rrep-3']:
  sectrace-rrep3(node, client, server,
    sMessageList(sareq(client, server, initiator, responder)),
    (rreplist sMessageList(message)))
  rootinfo(node, myroot, mytrustedroots) =>
  sectrace-sareq(node, client, server,
    sMessageList(sareq(client, server, initiator, responder)),
    (rreplist sMessageList(message)))
  rootinfo(node, myroot, mytrustedroots)
  ipsec-send(node, message')
  if message' := ip(client, initiator, sareq(client, server, initiator, responder)) .
```

Typical Scenario



Representation of Network

eq network =

```
shost(node("C1")) shost(node("C2")) ...
```

```
sgw(node("SG1")) sgw(node("SG2")) ...
```

```
sectraced(node("C1")) sectraced(node("C2")) ...
```

```
subnet(sAddrSet(addr("C1a")) sAddrSet(addr("SG2b")))
subnet(sAddrSet(addr("C2a")) sAddrSet(addr("SG1a"))) ...
```

```
interfaces(node("C1"),sAddrSet(addr("C1a"))) interfaces(node("C2"),sAddrSet(addr("C2a"))) ...
```

```
routetab(node("C1"),
  sRouteList(route(addr("C1a"), addr("C1a"), addr("C1a")))
  sRouteList(route(addr("C2a"), addr("C1a"), addr("SG2b"))) ...)
routetab(node("C2"),
  sRouteList(route(addr("C1a"), addr("C2a"), addr("SG1a")))
  sRouteList(route(addr("C2a"), addr("C2a"), addr("C2a"))) ...)
```

```
rootinfo(node("C1"), addr("CAC1"), sAddrSet(addr("CAC1")))
rootinfo(node("C2"), addr("CAC2"), sAddrSet(addr("CAC2"))) ...
```

```
sadb(node("C1"),eSASet)
sadb(node("C2"),eSASet)
```

```
...
```

```
spdb(node("C1"),eSPList,
  sSPList(sp(isinitiation,eSAList)) sSPList(sp(isresponse,eSAList)))
spdb(node("C2"),eSPList,
  sSPList(sp(isinitiation,eSAList)) sSPList(sp(isresponse,eSAList)))
```

```
...
```

Execution Plan

```
op start : -> State .  
op next : -> State .  
op terminated : -> State .
```

```
r1 start =>  
  sectrace-start(node("C1"), addr("C1a"), addr("Sa")) .
```

```
r1 sectrace-terminated(node("C1"), addr("C1a"), addr("Sa"))  
=> next .
```

```
r1 next =>  
  sectrace-start(node("C2"), addr("C2a"), addr("Sa")) .
```

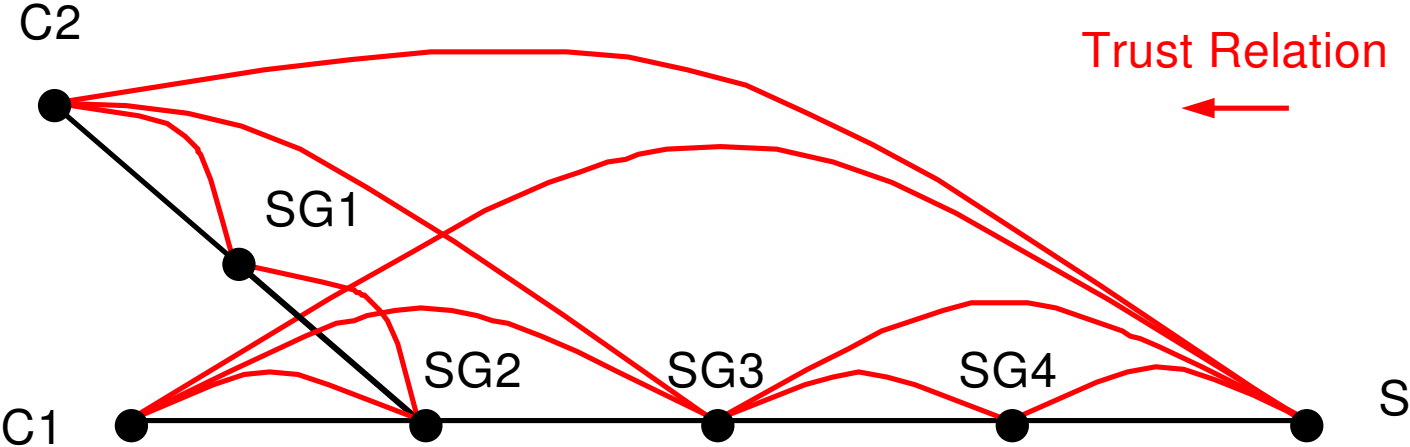
```
r1 sectrace-terminated(node("C2"), addr("C2a"), addr("Sa"))  
=> terminated .
```

Execution (Default Strategy)

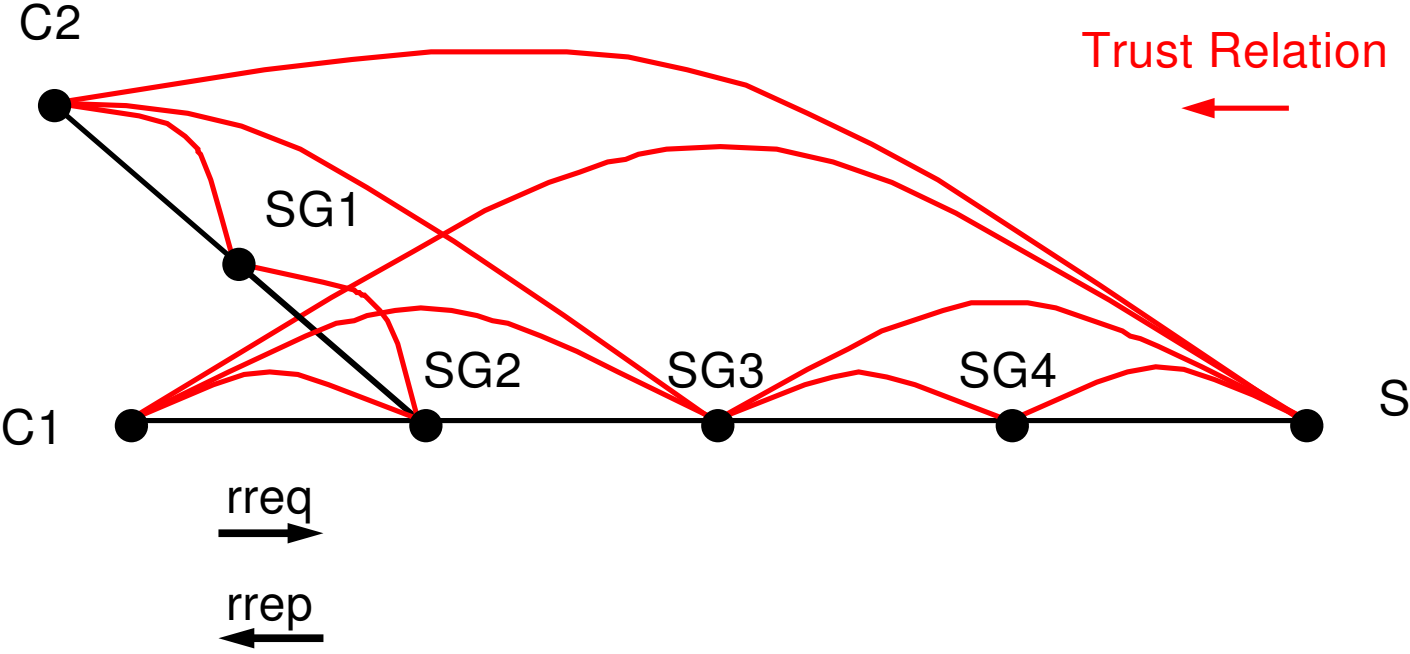
```
rew network start .
rewrites: 270753 in 160ms cpu (160ms real) (1692206 rewrites/second)
result State:
terminated
...
sadb(node("C1"), sSASet(sa(addr("C1a"), addr("SG2b"), 0))
                sSASet(sa(addr("C1a"), addr("SG3a"), 0)))
sadb(node("C2"), sSASet(sa(addr("C2a"), addr("SG1a"), 0))
                sSASet(sa(addr("C2a"), addr("SG3a"), 0)))
sadb(node("S"), sSASet(sa(addr("SG4a"), addr("Sa"), 0)))
sadb(node("SG1"), sSASet(sa(addr("C2a"), addr("SG1a"), 0))
                sSASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG2"), sSASet(sa(addr("C1a"), addr("SG2b"), 0))
                sSASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG3"), sSASet(sa(addr("C1a"), addr("SG3a"), 0))
                sSASet(sa(addr("C2a"), addr("SG3a"), 0))
                sSASet(sa(addr("SG3a"), addr("SG4a"), 0)))
sadb(node("SG4"), sSASet(sa(addr("SG3a"), addr("SG4a"), 0))
                sSASet(sa(addr("SG4a"), addr("Sa"), 0)))
...
spdb(node("C1"),
      eSPList,
      sSPList(sp(towards(addr("Sa")),
                sSAList(sa(addr("C1a"), addr("SG3a"), 0)) sSAList(sa(addr("C1a"), addr("SG2b"), 0))))
      sSPList(sp(isinitiation, eSAList))
      sSPList(sp(isresponse, eSAList)))
spdb(node("SG2"),
      sSPList(sp(towards(addr("Sa")), sSAList(sa(addr("C1a"), addr("SG2b"), 0))))
      sSPList(sp(towards(addr("Sa")), sSAList(sa(addr("SG1a"), addr("SG2a"), 0))))
      sSPList(sp(isinitiation, eSAList))
      sSPList(sp(isresponse, eSAList)))
...

```

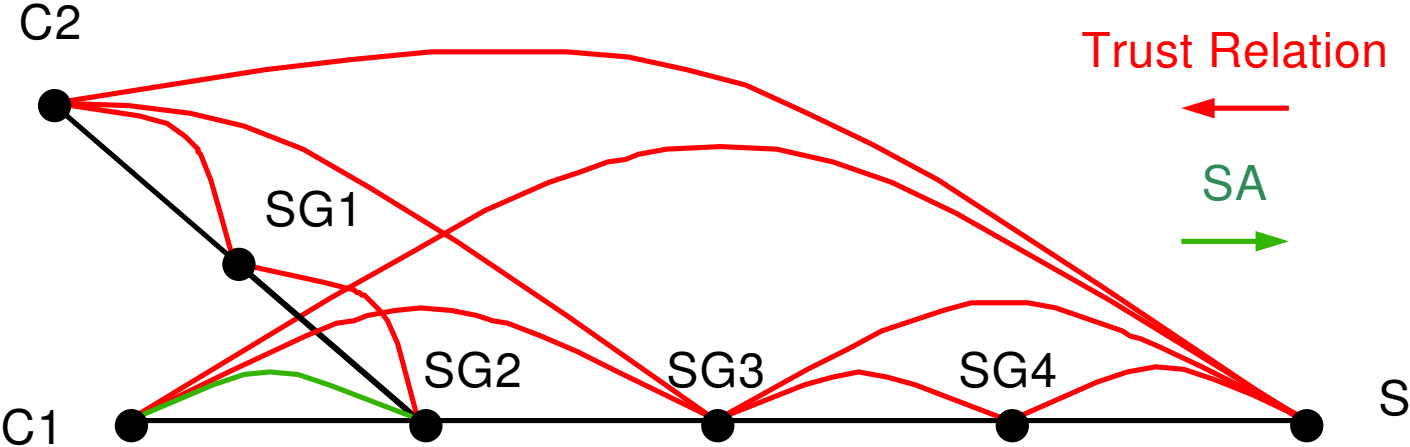
Execution (Default Strategy)



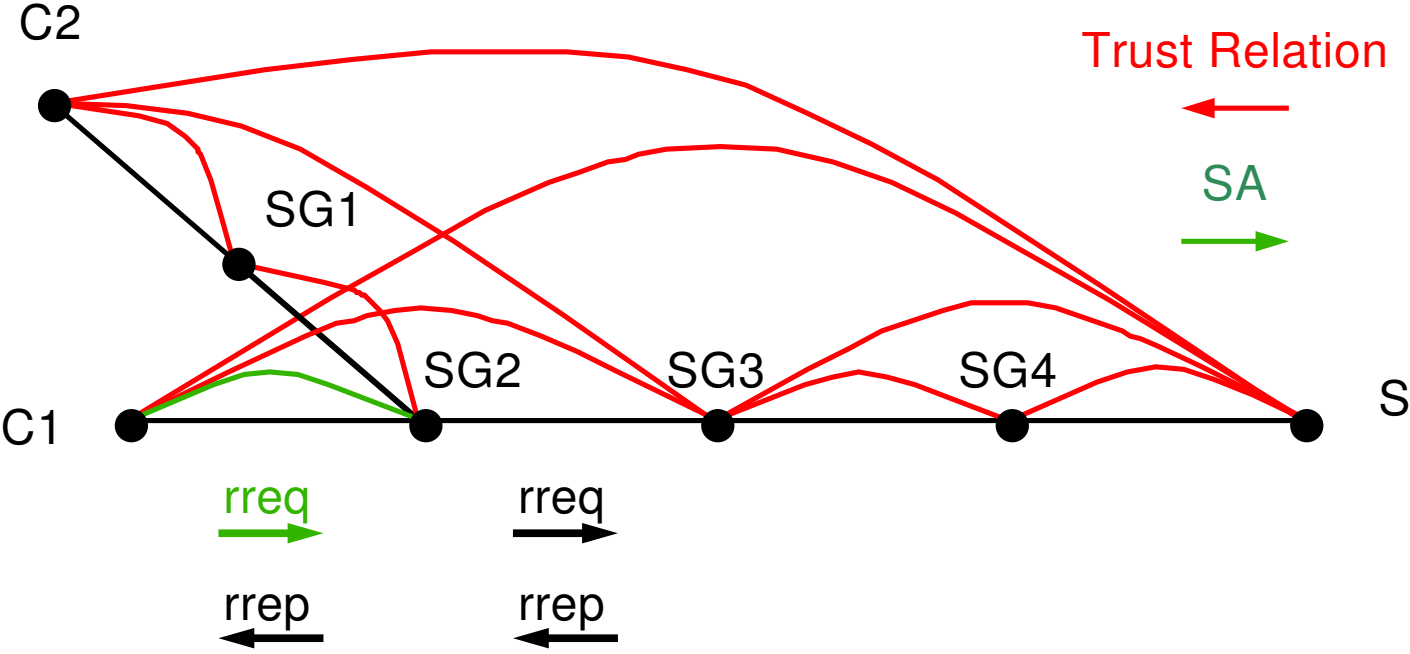
Execution (Default Strategy)



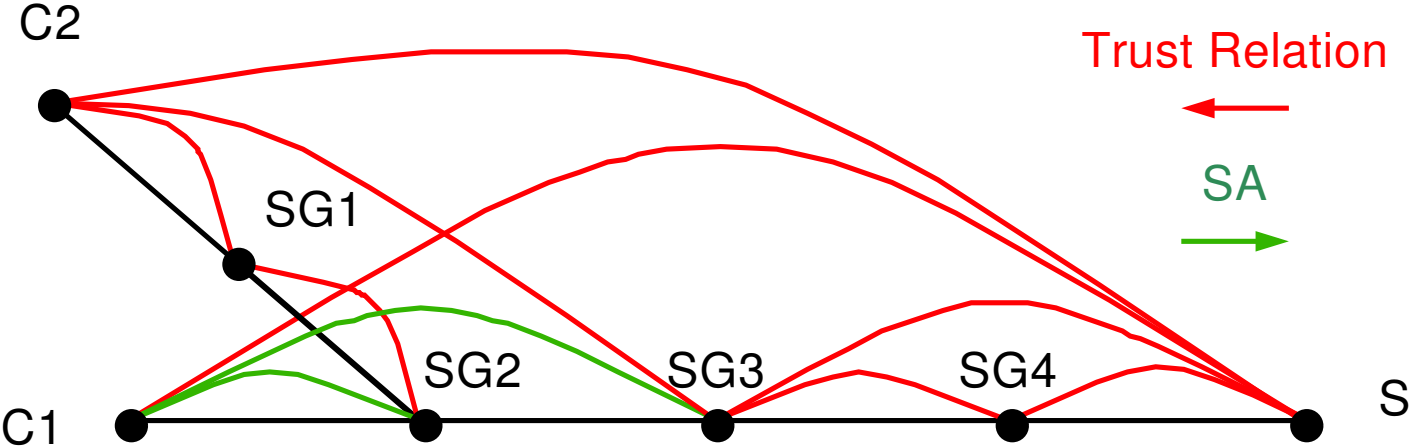
Execution (Default Strategy)



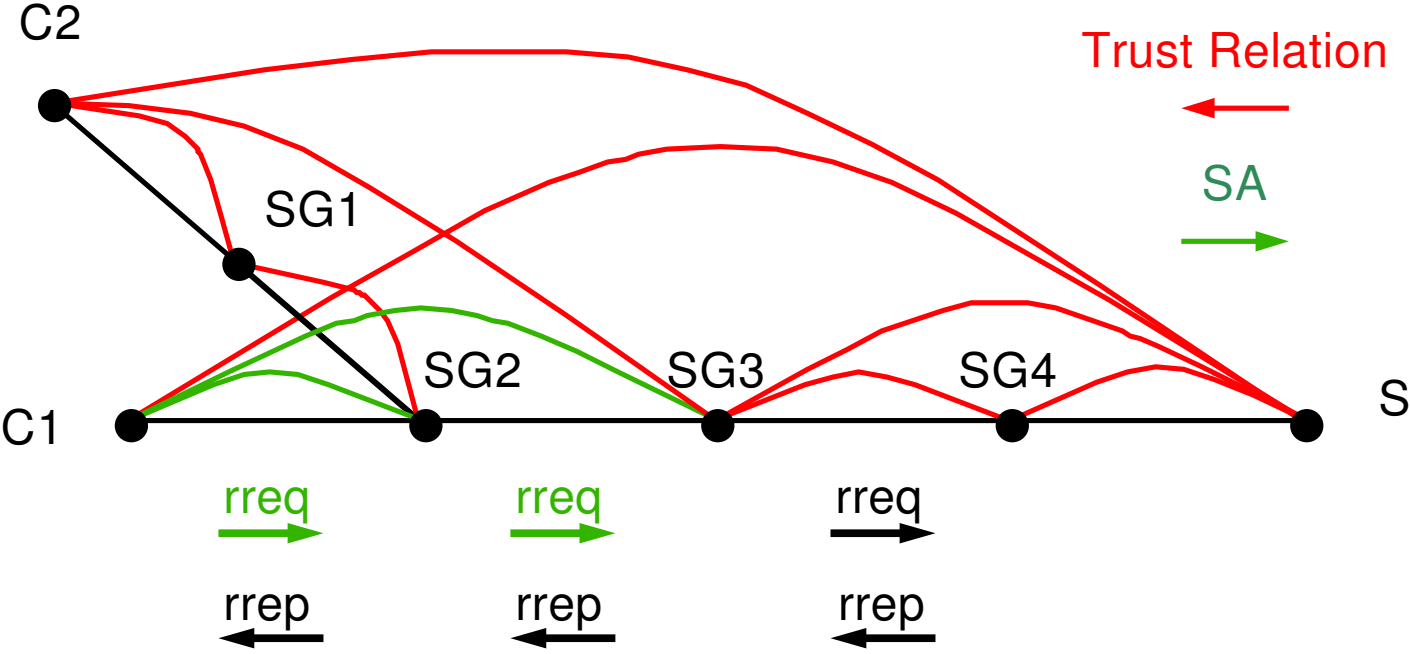
Execution (Default Strategy)



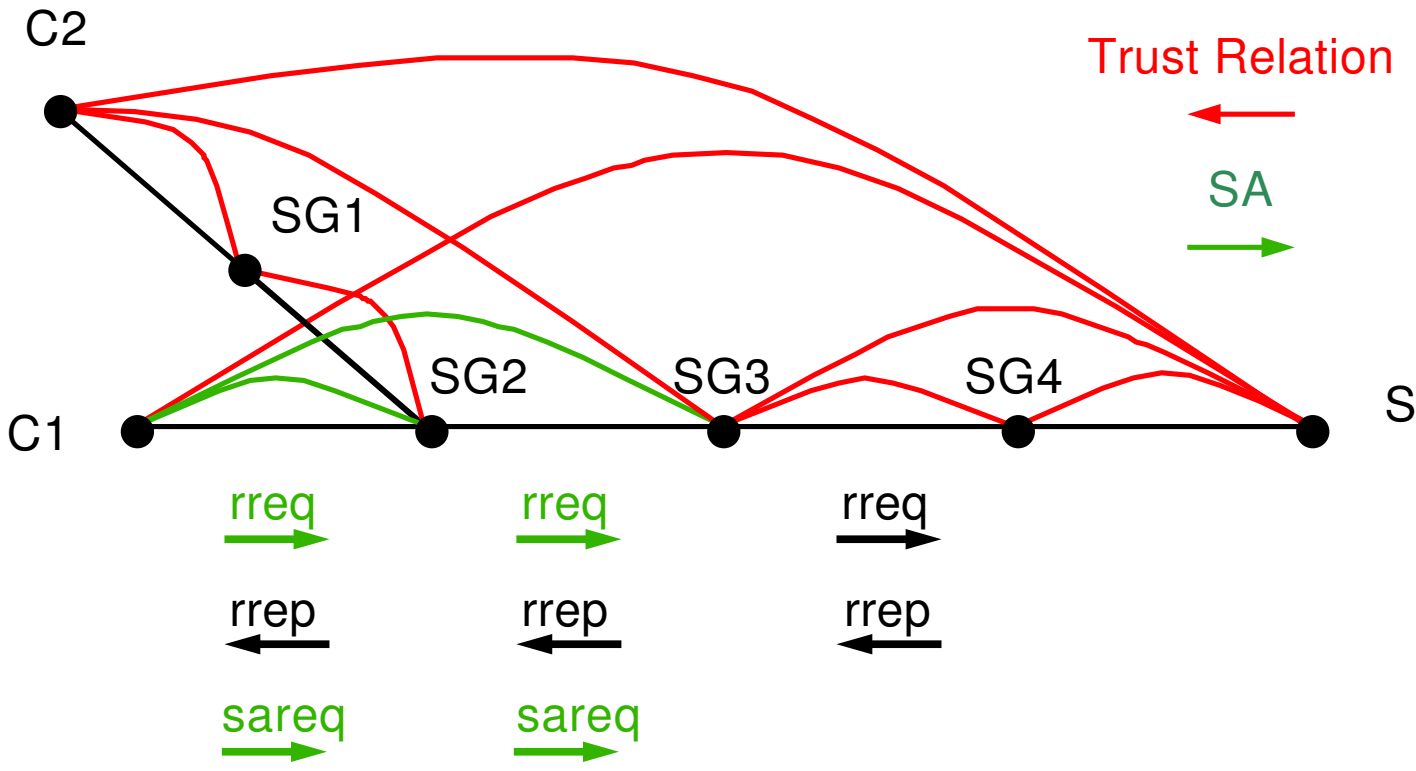
Execution (Default Strategy)



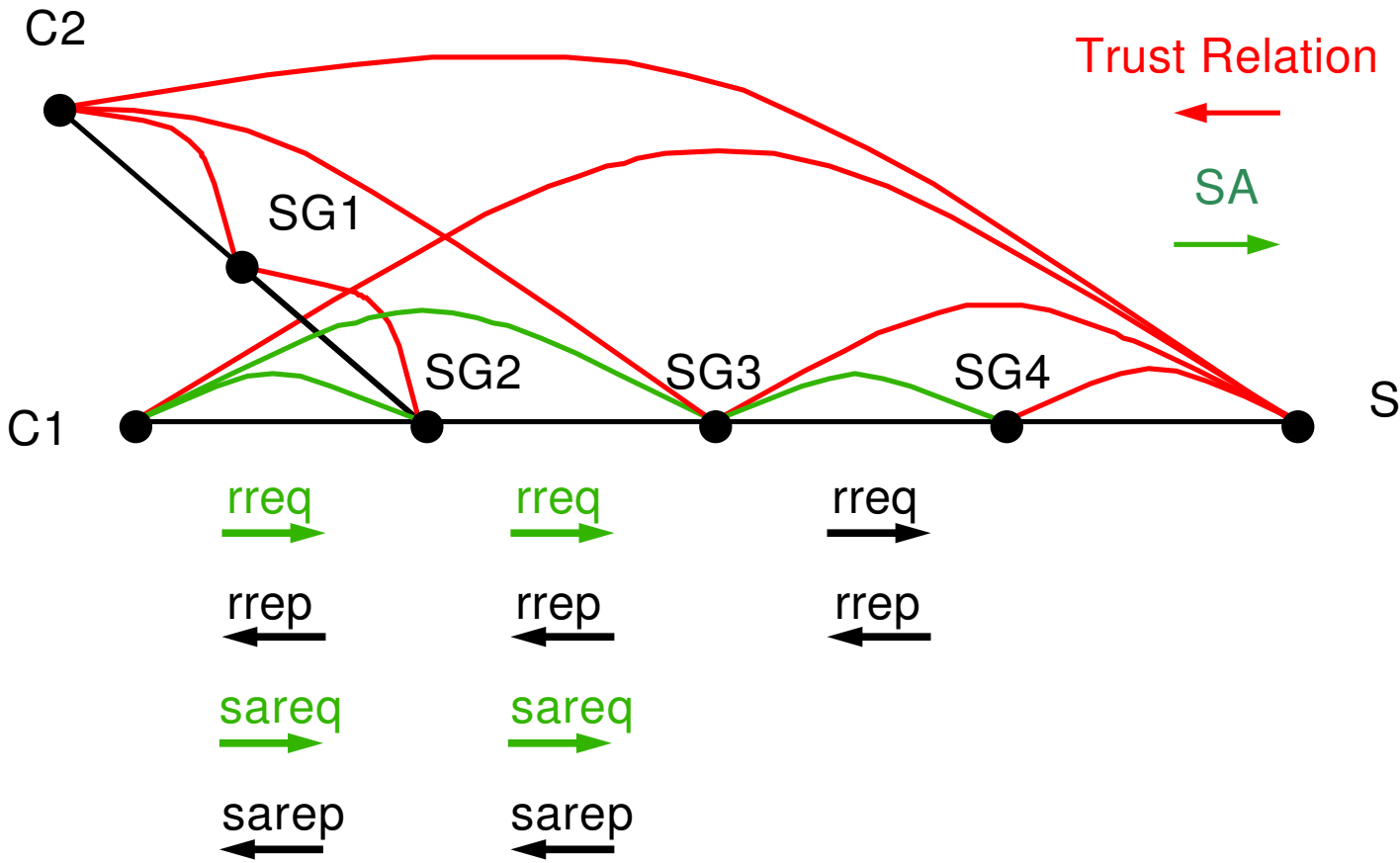
Execution (Default Strategy)



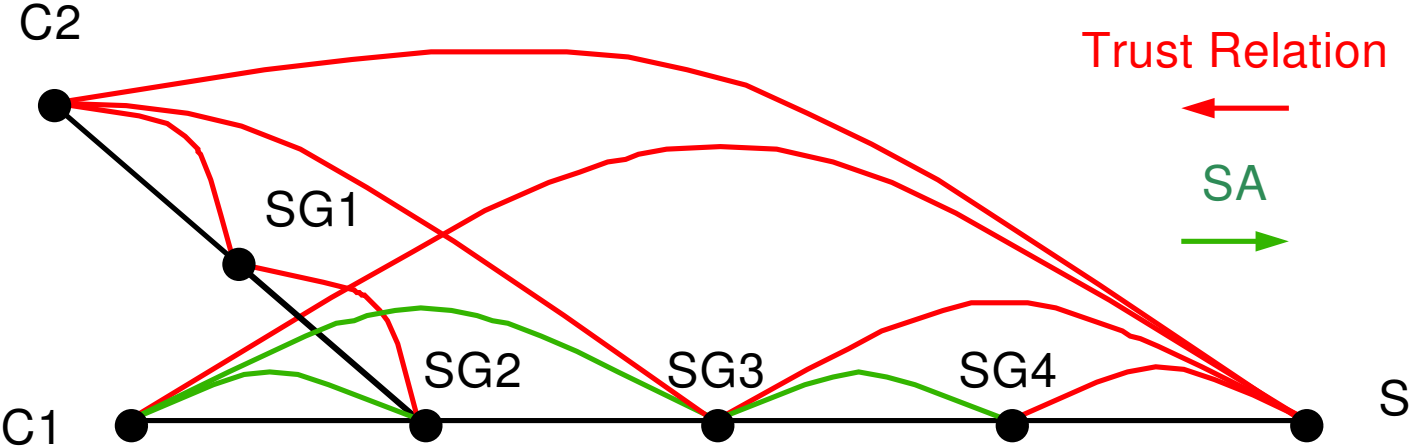
Execution (Default Strategy)



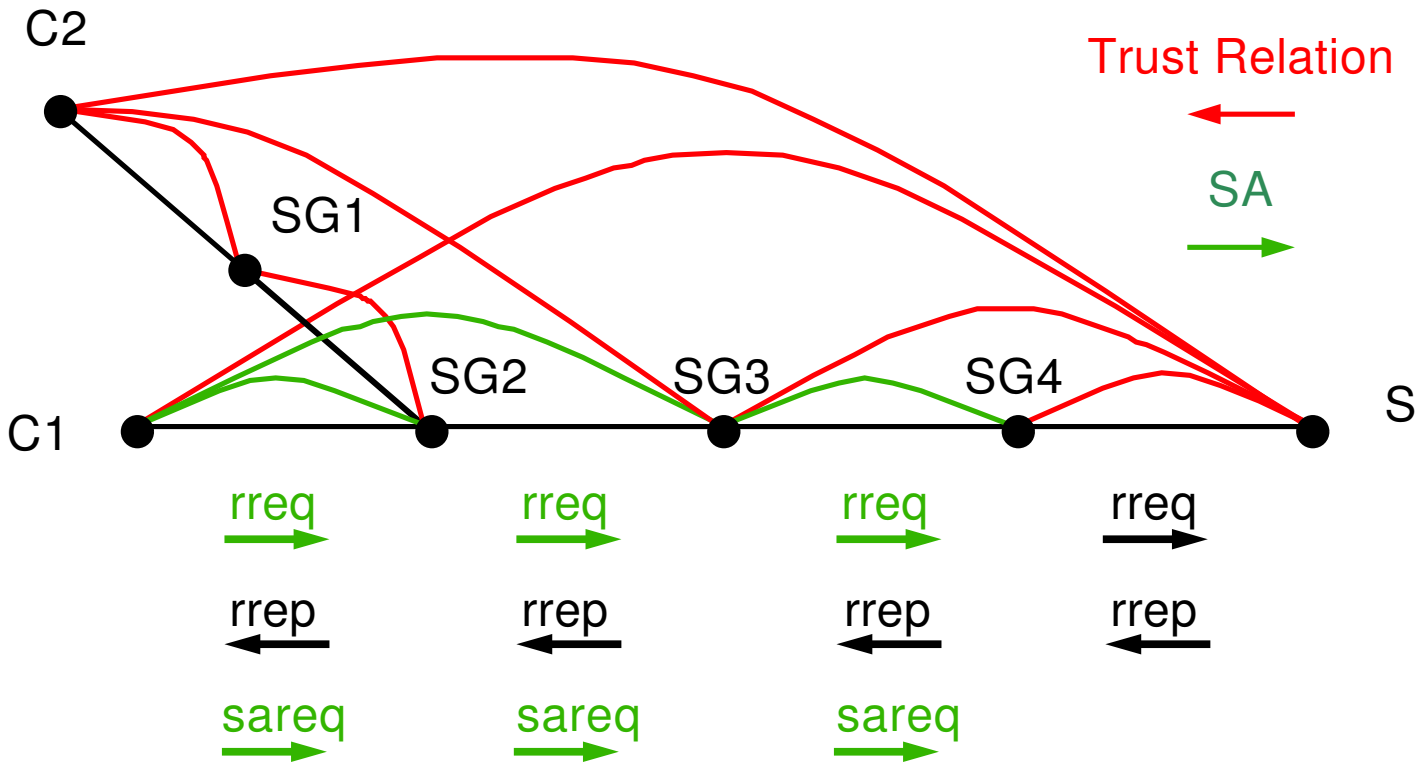
Execution (Default Strategy)



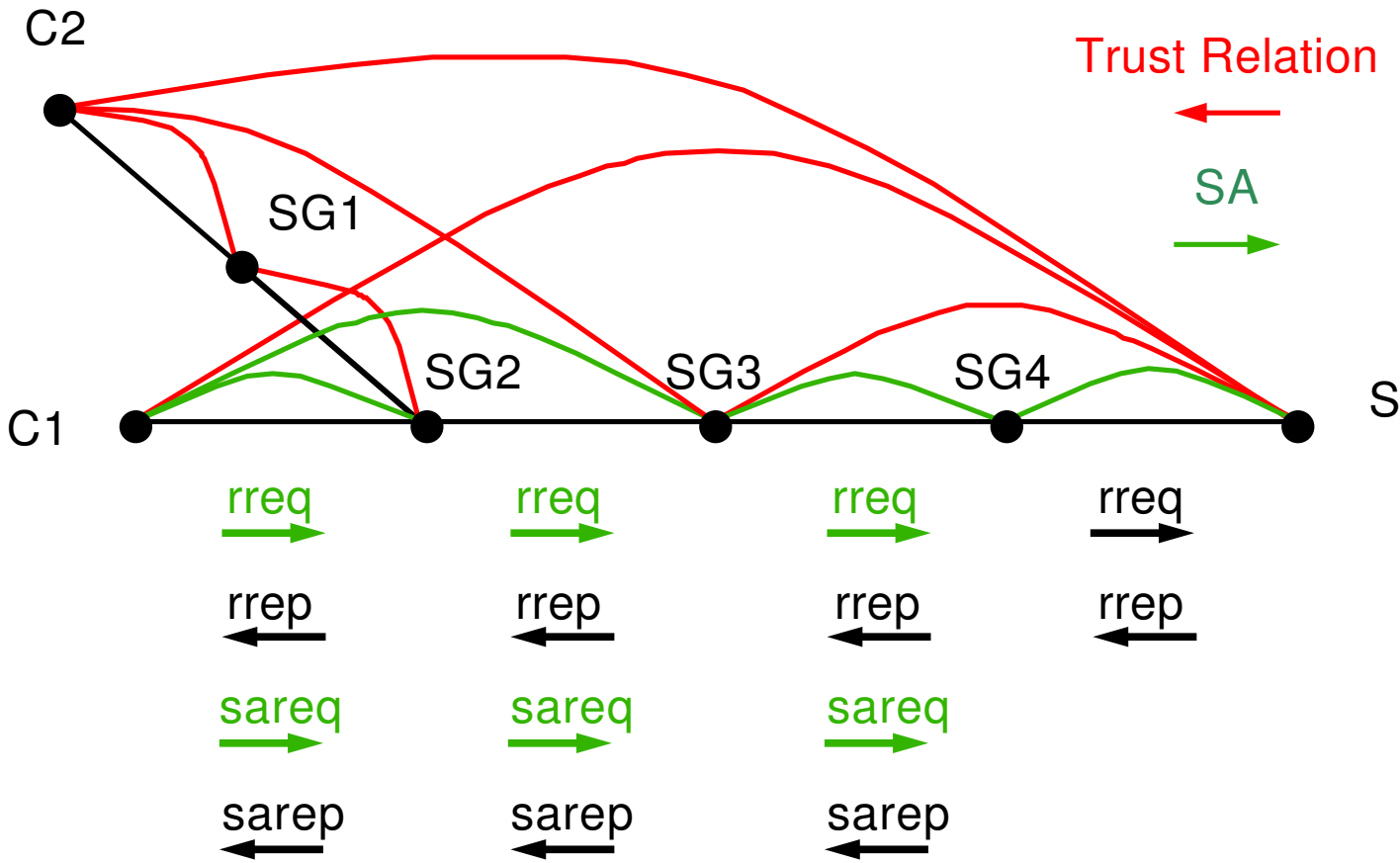
Execution (Default Strategy)



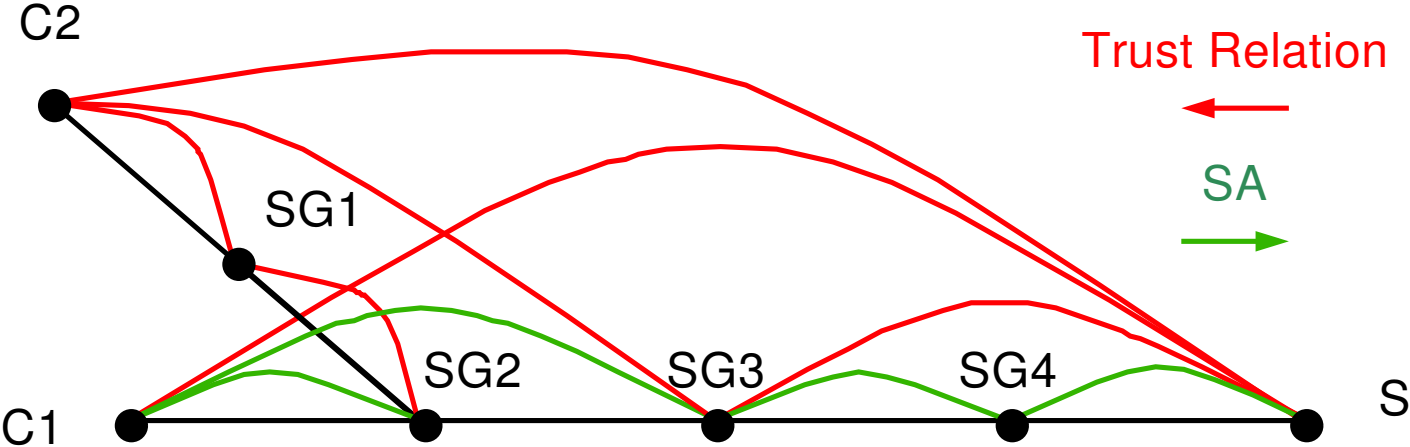
Execution (Default Strategy)



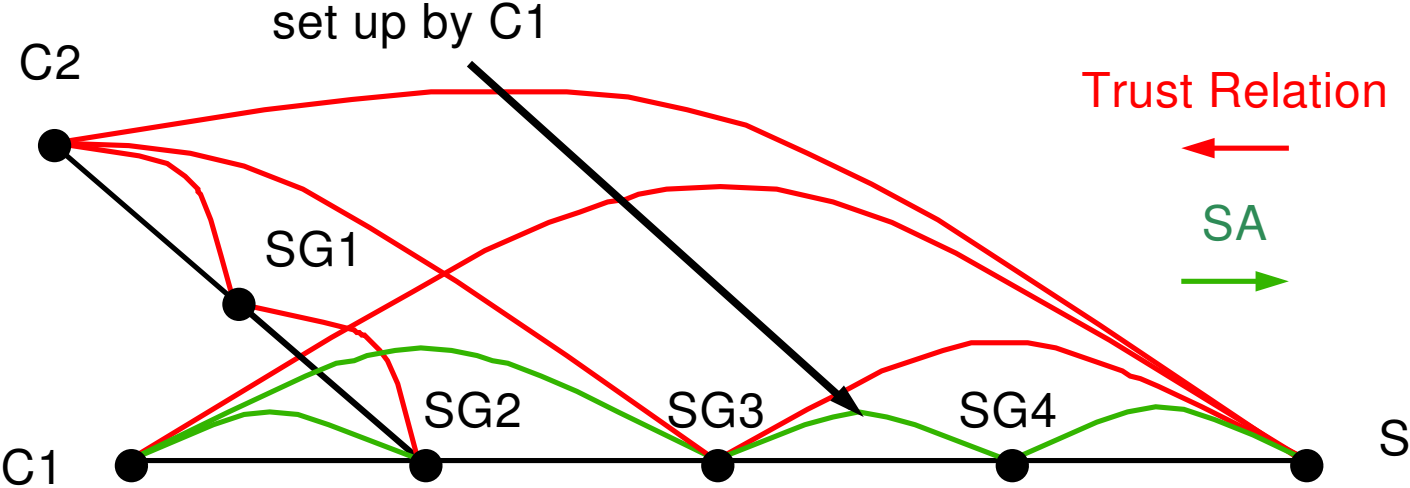
Execution (Default Strategy)



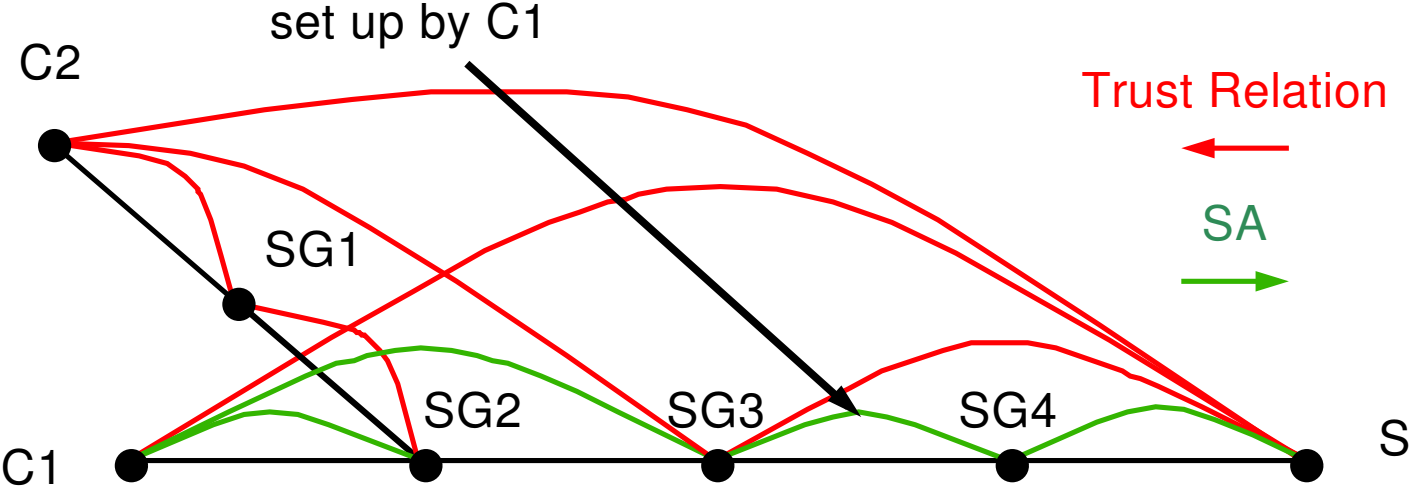
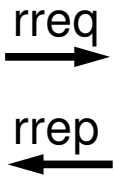
Execution (Default Strategy)



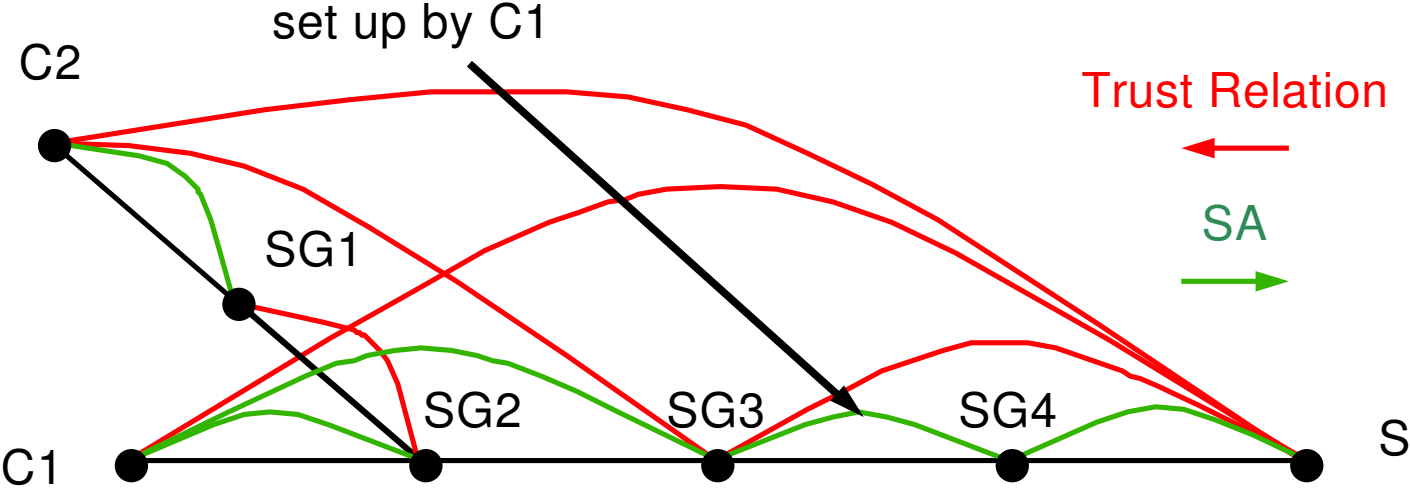
Execution (Default Strategy)



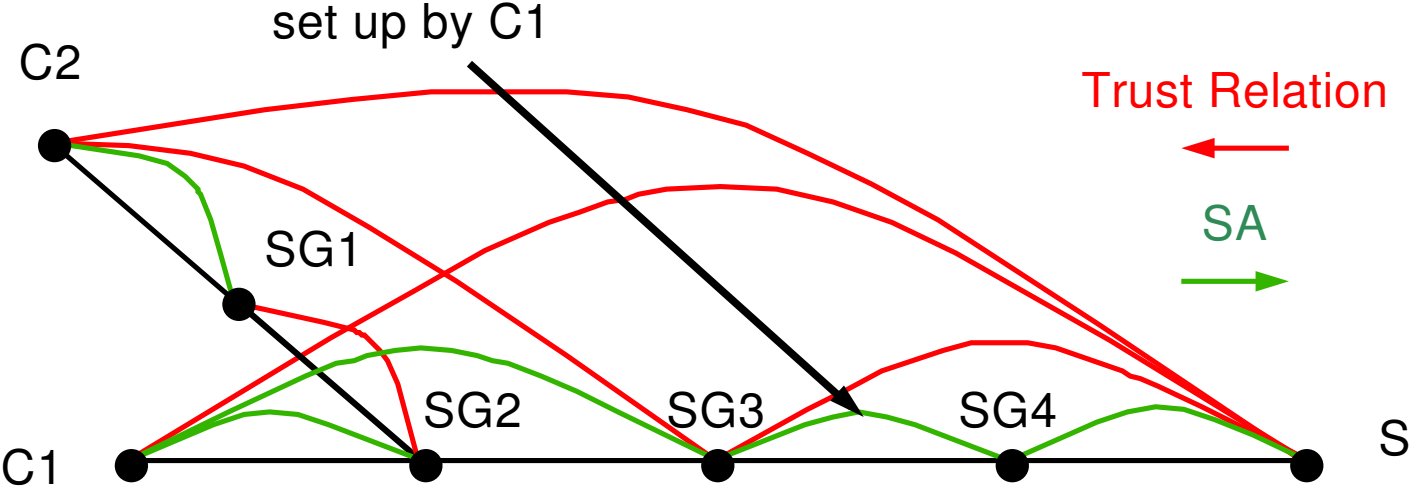
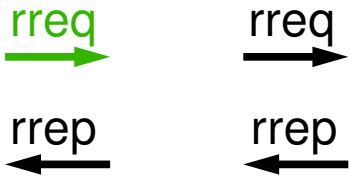
Execution (Default Strategy)



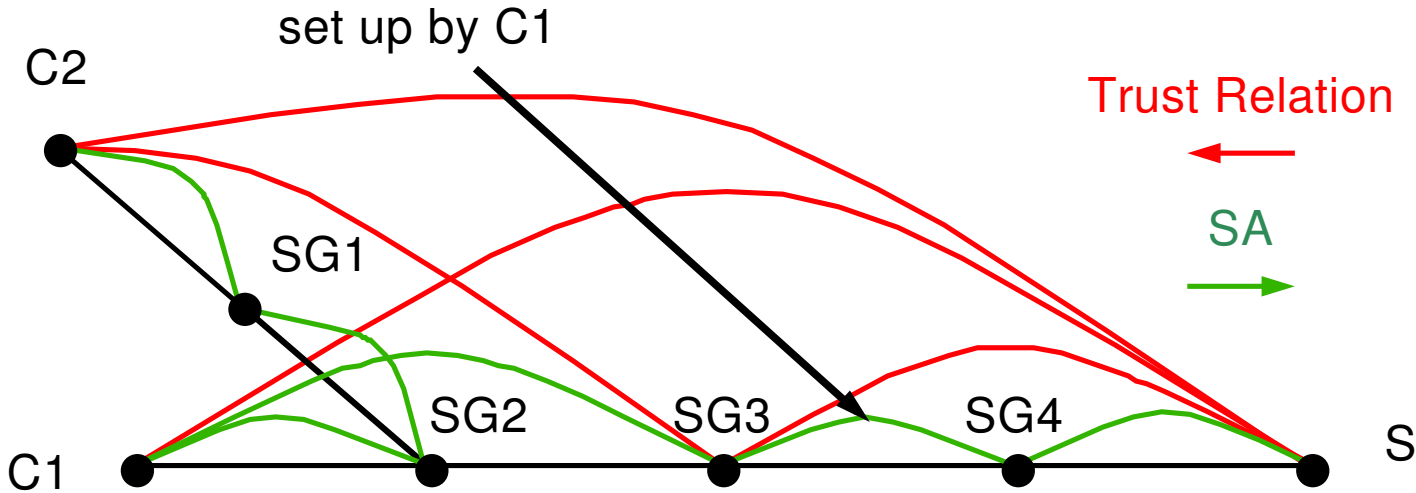
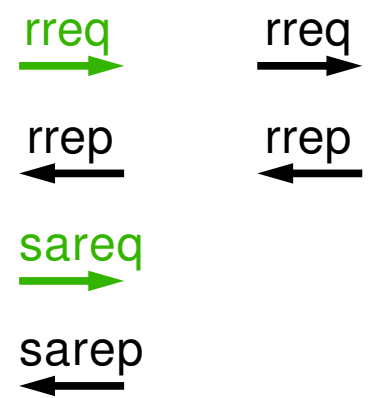
Execution (Default Strategy)



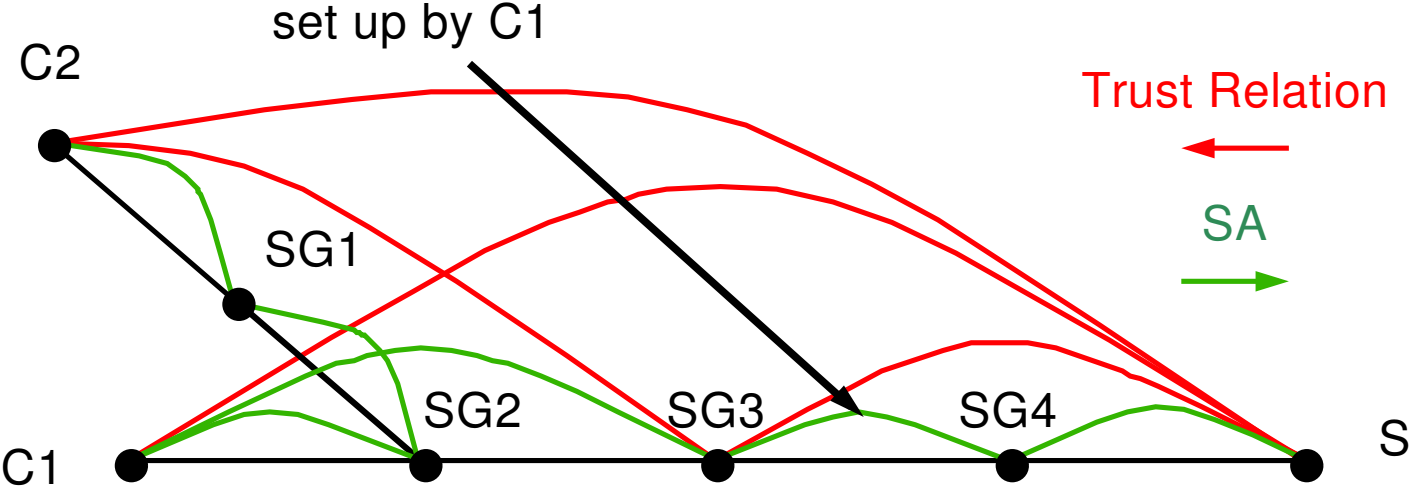
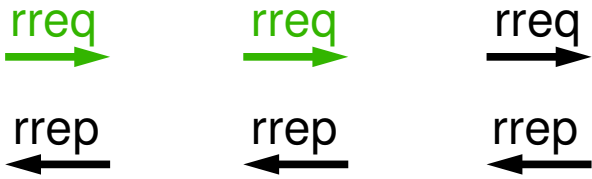
Execution (Default Strategy)



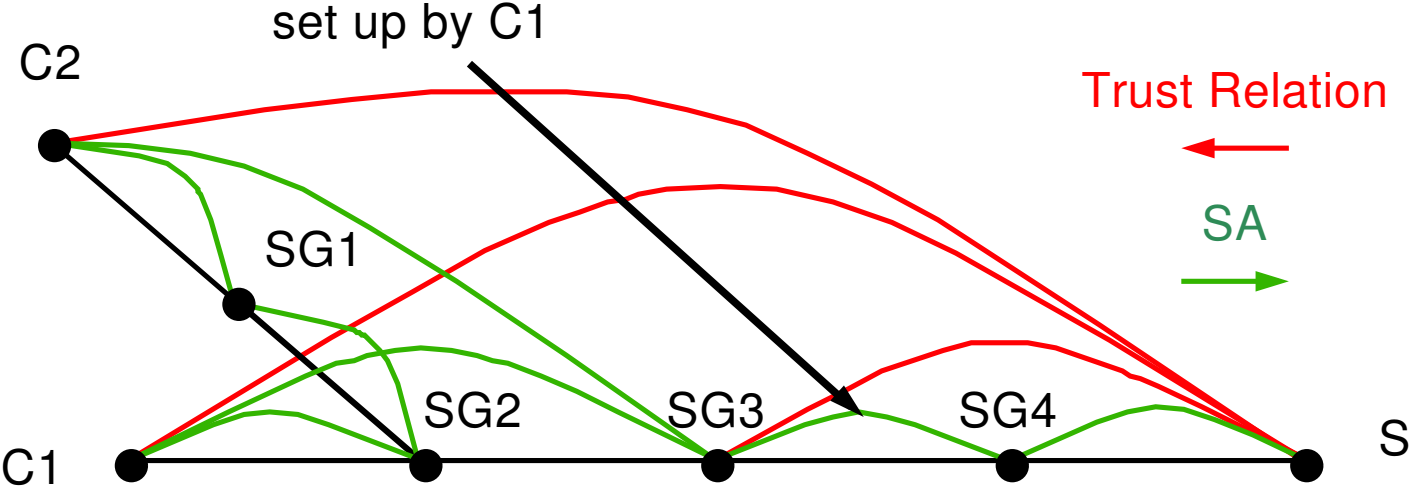
Execution (Default Strategy)



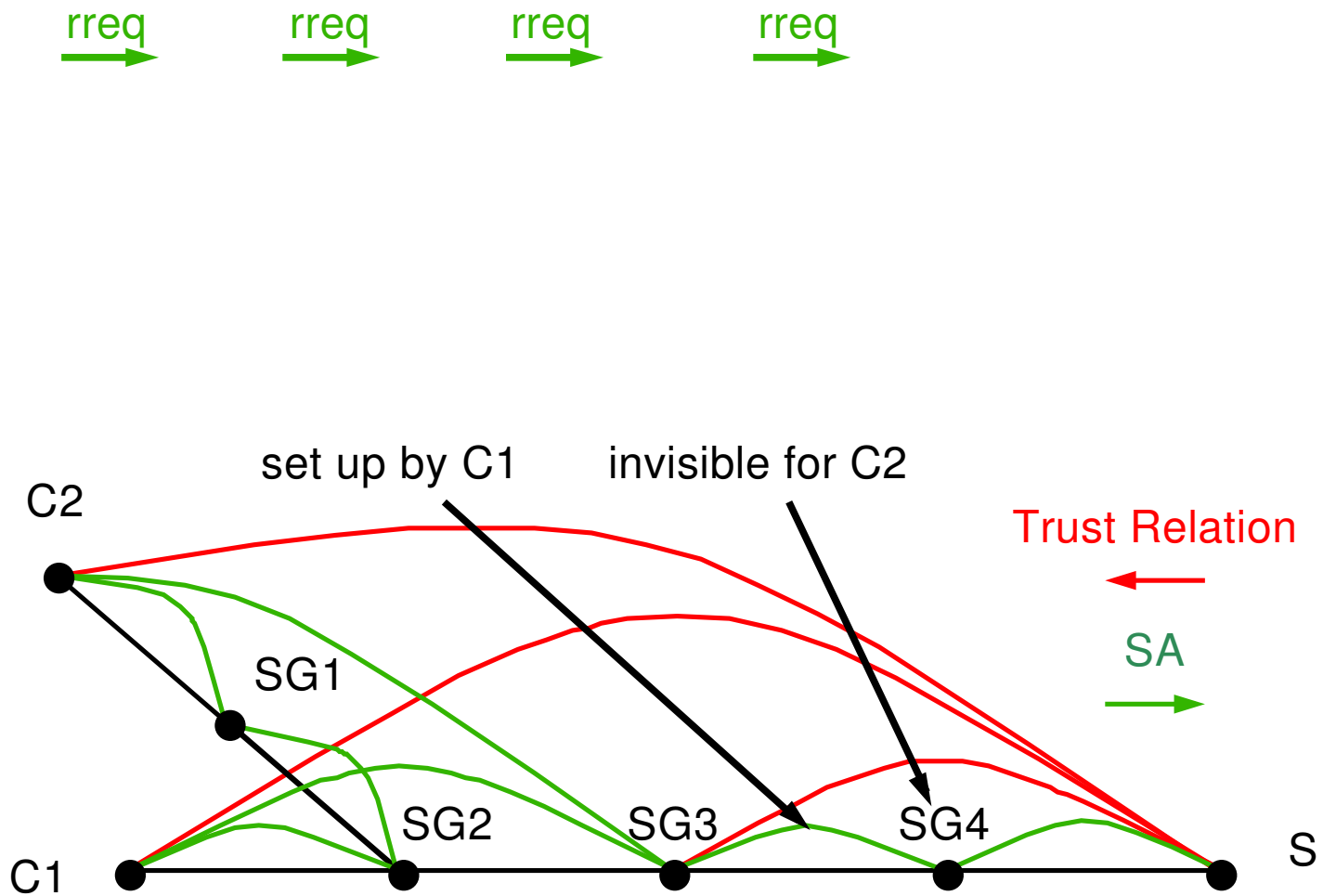
Execution (Default Strategy)



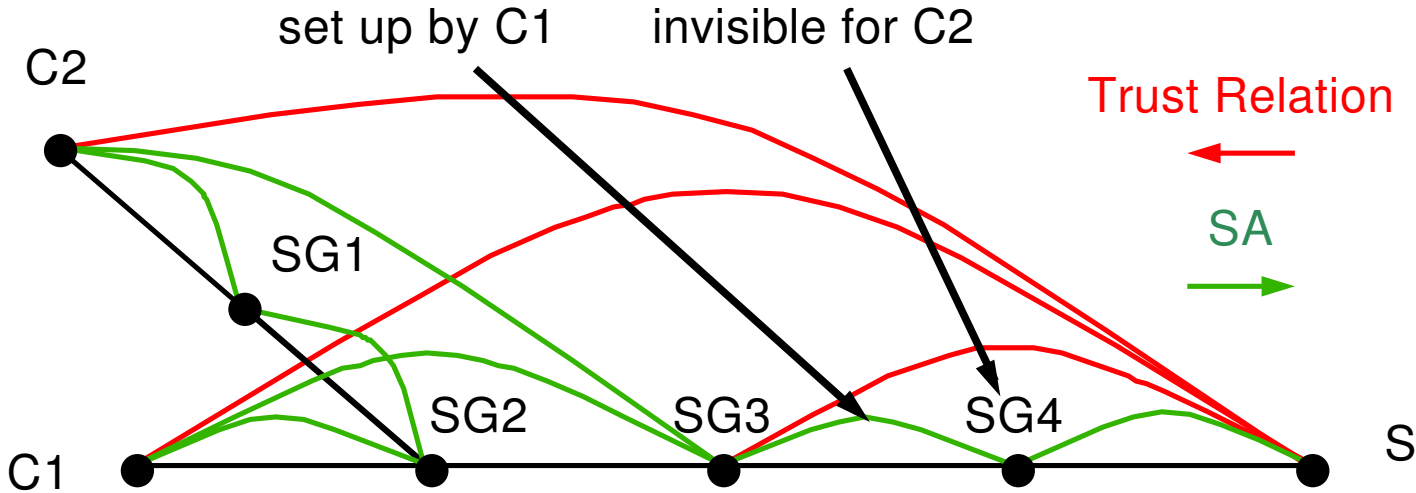
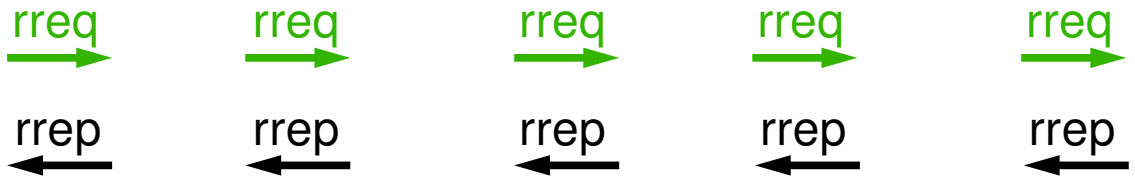
Execution (Default Strategy)



Execution (Default Strategy)

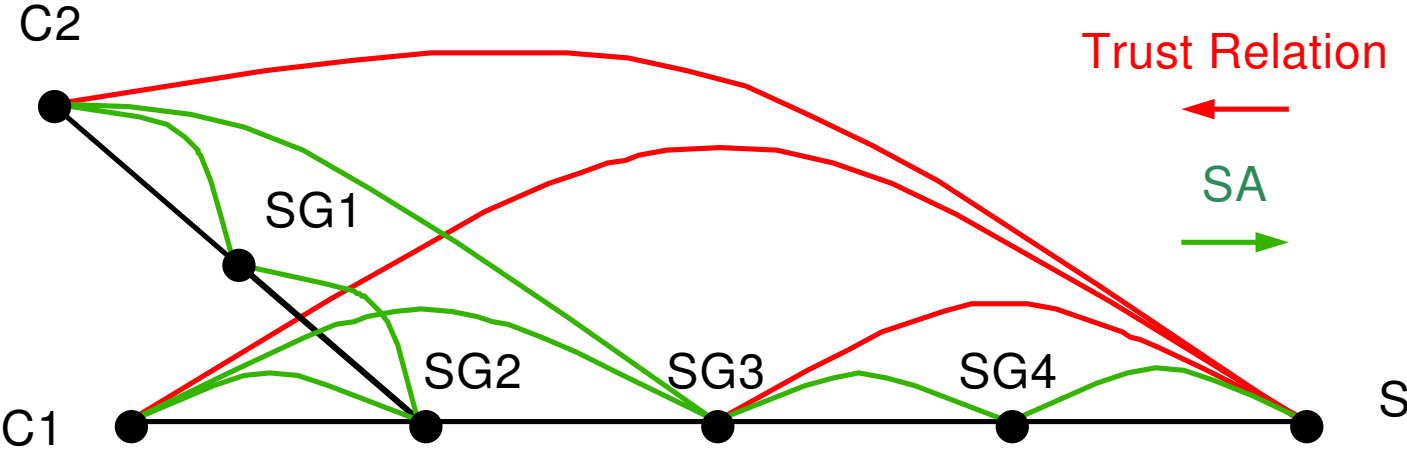


Execution (Default Strategy)



Execution (Default Strategy)

Sectrace terminates successfully
and has set up shortest SA



State Space Exploration

```
search start =>! state:State .
```

```
Solution 1 (state 304)
```

```
states: 305 rewrites: 434933 in 370ms cpu (370ms real) (1175494 rewrites/second)
```

```
state:State -->
```

```
terminated
```

```
...
```

```
sadb(node("C1"), sSASet(sa(addr("C1a"), addr("SG2b"), 0))
                sSASet(sa(addr("C1a"), addr("SG3a"), 0)))
sadb(node("C2"), sSASet(sa(addr("C2a"), addr("SG1a"), 0))
                sSASet(sa(addr("C2a"), addr("SG3a"), 0)))
sadb(node("S"), sSASet(sa(addr("SG4a"), addr("Sa"), 0)))
sadb(node("SG1"), sSASet(sa(addr("C2a"), addr("SG1a"), 0))
                sSASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG2"), sSASet(sa(addr("C1a"), addr("SG2b"), 0))
                sSASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG3"), sSASet(sa(addr("C1a"), addr("SG3a"), 0))
                sSASet(sa(addr("C2a"), addr("SG3a"), 0))
                sSASet(sa(addr("SG3a"), addr("SG4a"), 0)))
sadb(node("SG4"), sSASet(sa(addr("SG3a"), addr("SG4a"), 0))
                sSASet(sa(addr("SG4a"), addr("Sa"), 0)))
```

```
...
```

```
spdb(node("C1"),
      eSPList,
      sSPList(sp(towards(addr("Sa")),
                sSAList(sa(addr("C1a"), addr("SG3a"), 0)) sSAList(sa(addr("C1a"), addr("SG2b"), 0))))
      sSPList(sp(isinitiation, eSAList))
      sSPList(sp(isresponse, eSAList)))
```

```
...
```

```
No more solutions.
```

```
states: 305
```

```
rewrites: 434933 in 380ms cpu (1260ms real) (1144560 rewrites/second)
```

Concurrent Execution Plan

```
op start : -> State .  
op terminated : -> State .
```

```
rl start =>  
  sectrace-start(node("C1"), addr("C1a"), addr("Sa"))  
  sectrace-start(node("C2"), addr("C2a"), addr("Sa")) .  
  
rl sectrace-terminated(node("C1"), addr("C1a"), addr("Sa"))  
  sectrace-terminated(node("C2"), addr("C2a"), addr("Sa")) =>  
  terminated .
```

State Space Exploration

```
search start =>! state:State .
```

```
Solution 1 (state 139379) state:State -->
```

```
... terminated ...
```

```
Solution 2 (state 139423) state:State -->
```

```
... terminated ...
```

```
...
```

```
Solution 9 (state 155255) state:State -->
```

```
...
```

```
sectrace-terminated(node("C2"), addr("C2a"), addr("Sa"))
```

```
ipsec-received(node("S"), addr("Sa"), eAttrSet, sSAList(sa(addr("SG3a"), addr("Sa"), 0)),
```

```
ip(addr("C1a"), addr("SG4a"), sareq(addr("C1a"), addr("Sa"), addr("SG4a"), addr("Sa"))))
```

```
sectrace-sareq(node("C1"), addr("C1a"), addr("Sa"), ...) ...
```

```
Solution 10 (state 155543) state:State -->
```

```
...
```

```
sectrace-terminated(node("C2"), addr("C2a"), addr("Sa"))
```

```
ipsec-received(node("S"), addr("Sa"), eAttrSet, sSAList(sa(addr("SG3a"), addr("Sa"), 0)),
```

```
ip(addr("C1a"), addr("SG4a"), sareq(addr("C1a"), addr("Sa"), addr("SG4a"), addr("Sa"))))
```

```
sectrace-sareq(node("C1"), addr("C1a"), addr("Sa"), ...) ...
```

```
...
```

```
Solution 17 (state 165315) state:State -->
```

```
... terminated ...
```

```
...
```

```
Solution 24 (state 165783) state:State -->
```

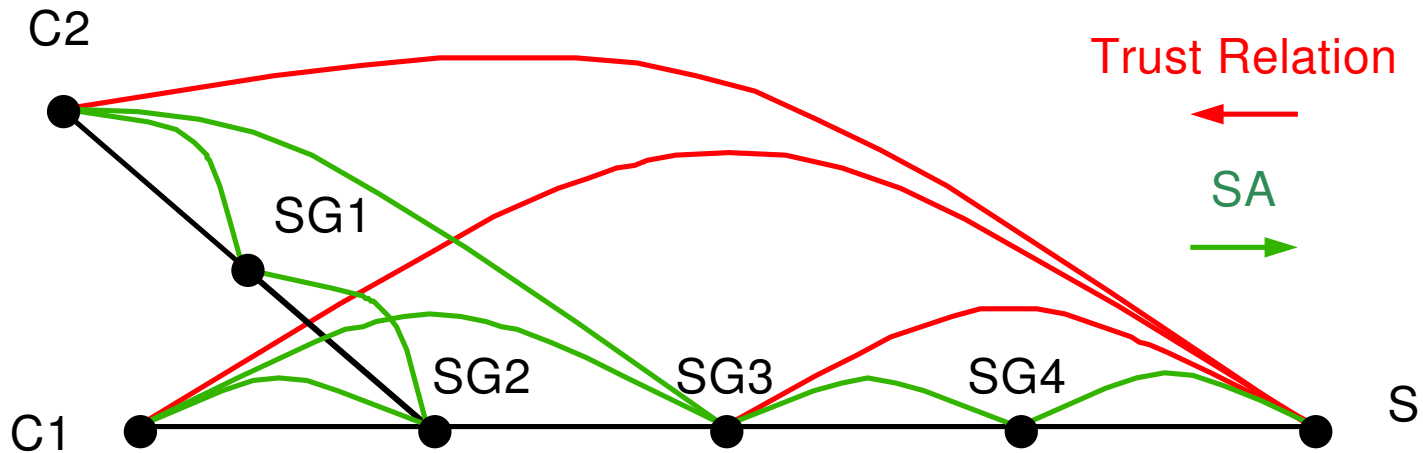
```
... terminated ...
```

```
No more solutions.
```

```
states: 185271
```

```
rewrites: 556616699 in 3311100ms cpu (3342860ms real) (168106 rewrites/second)
```

Type 1 Solution

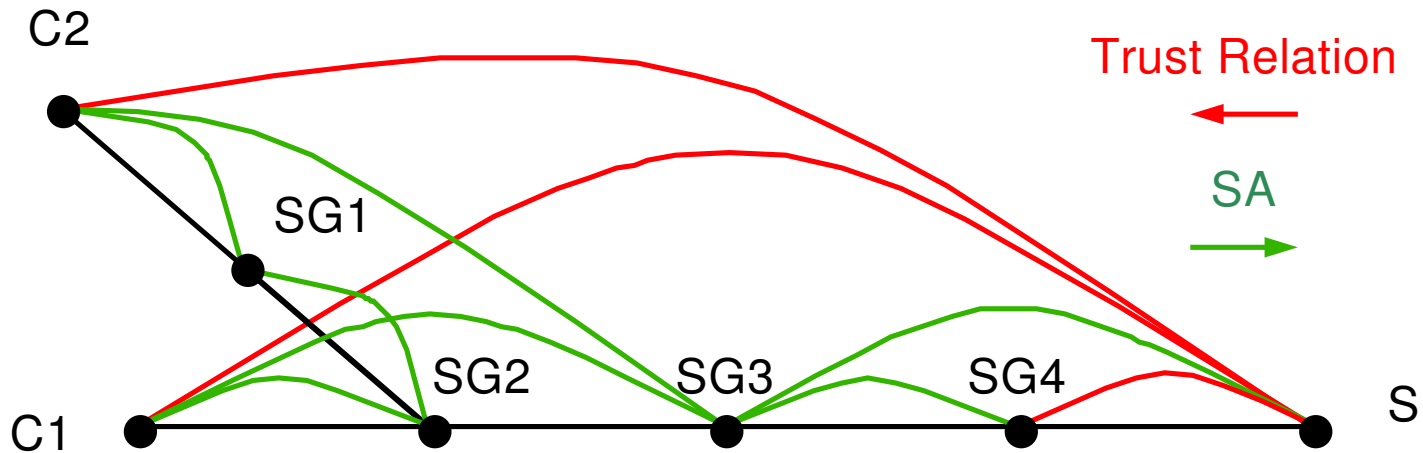


terminated

```
sadb(node("C1"),  sSASet(sa(addr("C1a"),  addr("SG2b"),  0))  sSASet(sa(addr("C1a"),  addr("SG3a"),  0)))
sadb(node("C2"),  sSASet(sa(addr("C2a"),  addr("SG1a"),  0))  sSASet(sa(addr("C2a"),  addr("SG3a"),  0)))
sadb(node("SG1"), sSASet(sa(addr("C2a"),  addr("SG1a"),  0))  sSASet(sa(addr("SG1a"),  addr("SG2a"),  0)))
sadb(node("SG2"), sSASet(sa(addr("C1a"),  addr("SG2b"),  0))  sSASet(sa(addr("SG1a"),  addr("SG2a"),  0)))
sadb(node("SG3"), sSASet(sa(addr("C1a"),  addr("SG3a"),  0))  sSASet(sa(addr("C2a"),  addr("SG3a"),  0))
sSASet(sa(addr("SG3a"),  addr("SG4a"),  0)))
sadb(node("SG4"), sSASet(sa(addr("SG3a"),  addr("SG4a"),  0))  sSASet(sa(addr("SG4a"),  addr("Sa"),  0)))
sadb(node("S"),   sSASet(sa(addr("SG4a"),  addr("Sa"),  0)))
```

Both clients terminate successfully
and have set up shortest SA

Type 2 Solution

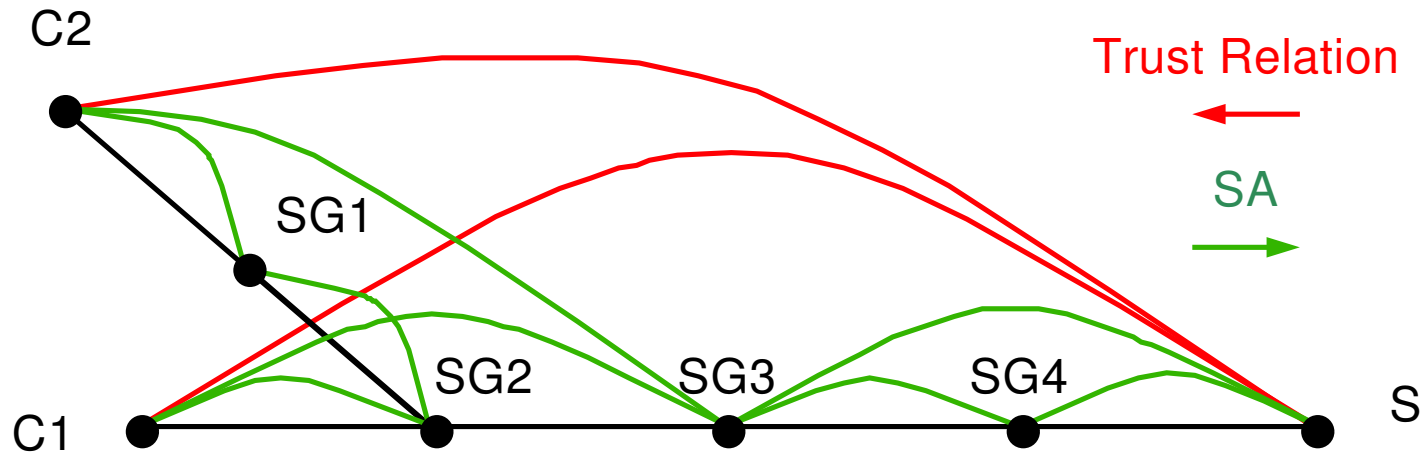


terminated

```
sadb(node("C1"),  sSASet(sa(addr("C1a"),  addr("SG2b"),  0))  sSASet(sa(addr("C1a"),  addr("SG3a"),  0)))
sadb(node("C2"),  sSASet(sa(addr("C2a"),  addr("SG1a"),  0))  sSASet(sa(addr("C2a"),  addr("SG3a"),  0)))
sadb(node("SG1"), sSASet(sa(addr("C2a"),  addr("SG1a"),  0))  sSASet(sa(addr("SG1a"),  addr("SG2a"),  0)))
sadb(node("SG2"), sSASet(sa(addr("C1a"),  addr("SG2b"),  0))  sSASet(sa(addr("SG1a"),  addr("SG2a"),  0)))
sadb(node("SG3"), sSASet(sa(addr("C1a"),  addr("SG3a"),  0))  sSASet(sa(addr("C2a"),  addr("SG3a"),  0))
sSASet(sa(addr("SG3a"),  addr("SG4a"),  0)) sSASet(sa(addr("SG3a"),  addr("Sa"),  0)))
sadb(node("SG4"), sSASet(sa(addr("SG3a"),  addr("SG4a"),  0)))
sadb(node("S"),   sSASet(sa(addr("SG3a"),  addr("Sa"),  0)))
```

Both clients terminate successfully
but SA is not the shortest one

Type 3 Solution

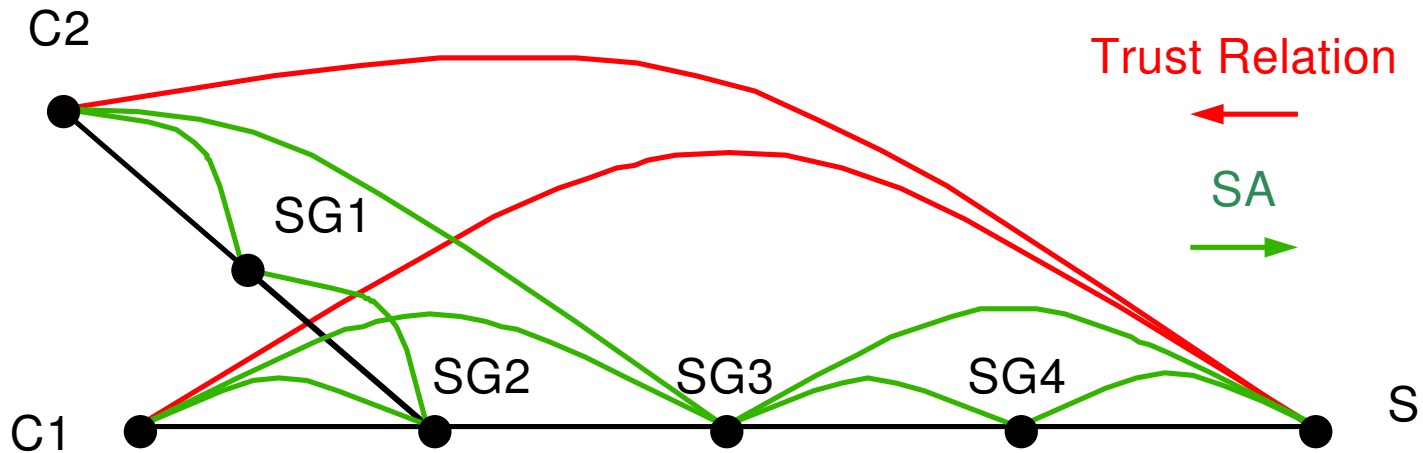


terminated

```
sadb(node("C1"), ssASet(sa(addr("C1a"), addr("SG2b"), 0)) sASet(sa(addr("C1a"), addr("SG3a"), 0)))
sadb(node("C2"), ssASet(sa(addr("C2a"), addr("SG1a"), 0)) sASet(sa(addr("C2a"), addr("SG3a"), 0)))
sadb(node("S"), ssASet(sa(addr("SG4a"), addr("Sa"), 0)) sASet(sa(addr("SG3a"), addr("Sa"), 0)))
sadb(node("SG1"), ssASet(sa(addr("C2a"), addr("SG1a"), 0)) sASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG2"), ssASet(sa(addr("C1a"), addr("SG2b"), 0)) sASet(sa(addr("SG1a"), addr("SG2a"), 0)))
sadb(node("SG3"), ssASet(sa(addr("C2a"), addr("SG3a"), 0)) sASet(sa(addr("SG3a"), addr("SG4a"), 0)))
sadb(node("SG4"), ssASet(sa(addr("C1a"), addr("SG3a"), 0)) sASet(sa(addr("SG3a"), addr("Sa"), 0)))
sadb(node("SG4"), ssASet(sa(addr("SG3a"), addr("SG4a"), 0)) sASet(sa(addr("SG4a"), addr("Sa"), 0)))
```

Both clients terminate successfully,
but one SA is redundant

Type 4 Solution



```
sectrace-terminated(node("C2"), addr("C2a"), addr("Sa"))
```

```
ipsec-received(node("S"), addr("Sa"), eAttrSet,  
  sSAList(sa(addr("SG3a"), addr("Sa"), 0)),  
  ip(addr("C1a"), addr("SG4a"), sareq(addr("C1a"), addr("Sa"), addr("SG4a"), addr("Sa"))))
```

```
sectrace-sareq(node("C1"), addr("C1a"), addr("Sa"), ...) ...
```

One client does not terminate,
because sareq misguided by security policy

Model Checking

```
op soln-type3 : -> Prop .

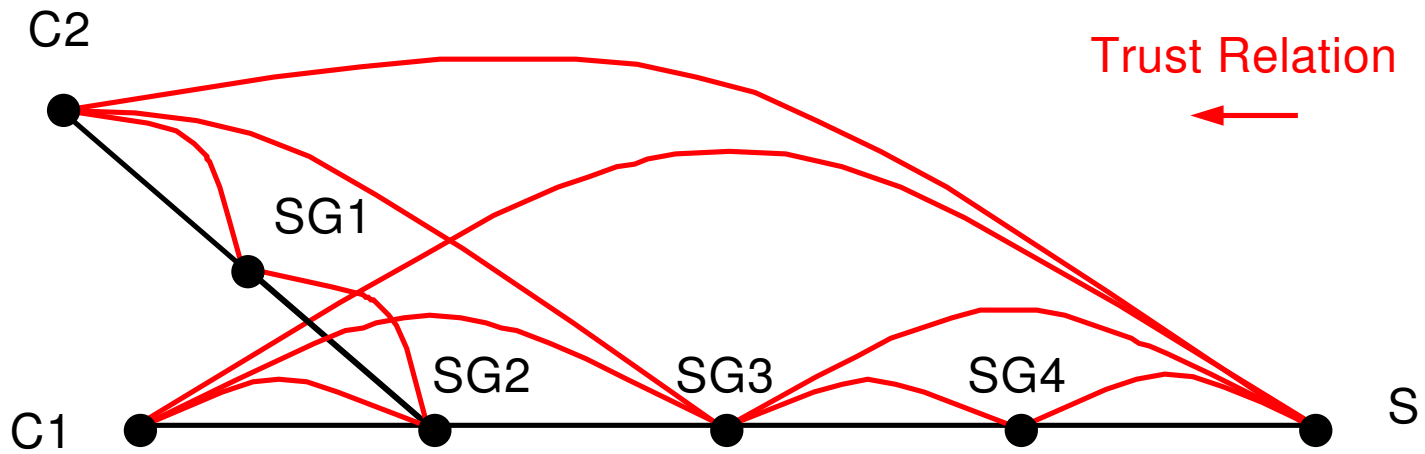
eq (terminated
    sadb(node("S"),
        sSASet(sa(addr("SG4a"), addr("Sa"), 0))
        sSASet(sa(addr("SG3a"), addr("Sa"), 0))) state:State |= soln-type3) = true .

red (network start) |= ~ <> soln-type3 .

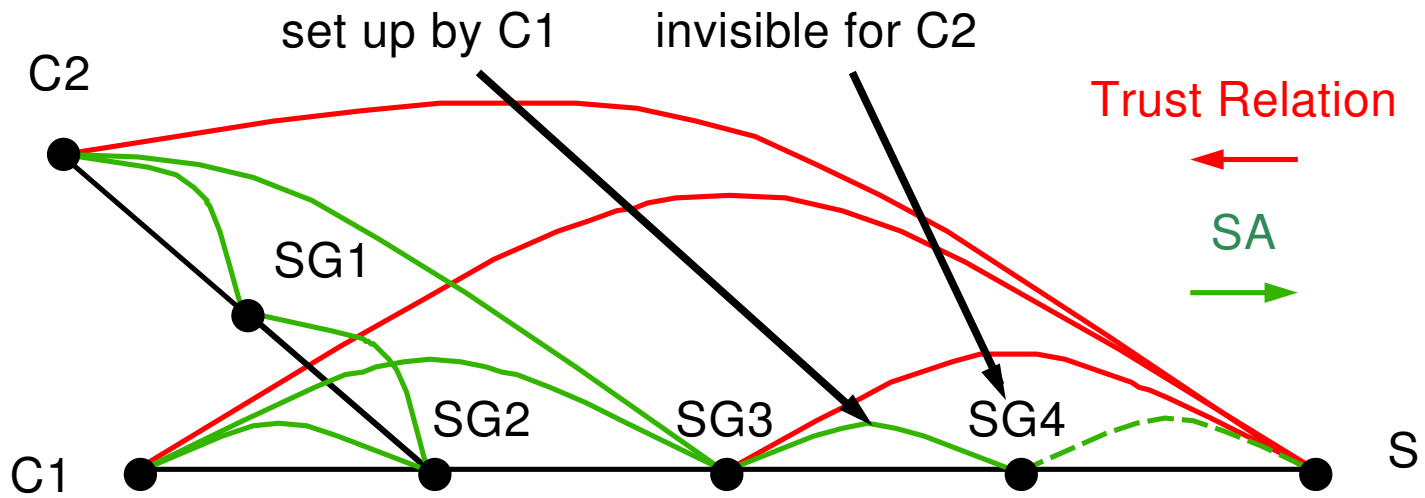
rewrites: 5994428 in 5120ms cpu (5160ms real) (1170786 rewrites/second)

result ModelCheckResult:
  counterexample({...,'unlabelled}
    {...,'sectrace-start}
    {...,'ipsec-send}
    ...
    {... sadb(node("S"), sSASet(sa(addr("SG3a"), addr("Sa"), 0))
      sSASet(sa(addr("SG4a"), addr("Sa"), 0))),'stop}, ...)
```

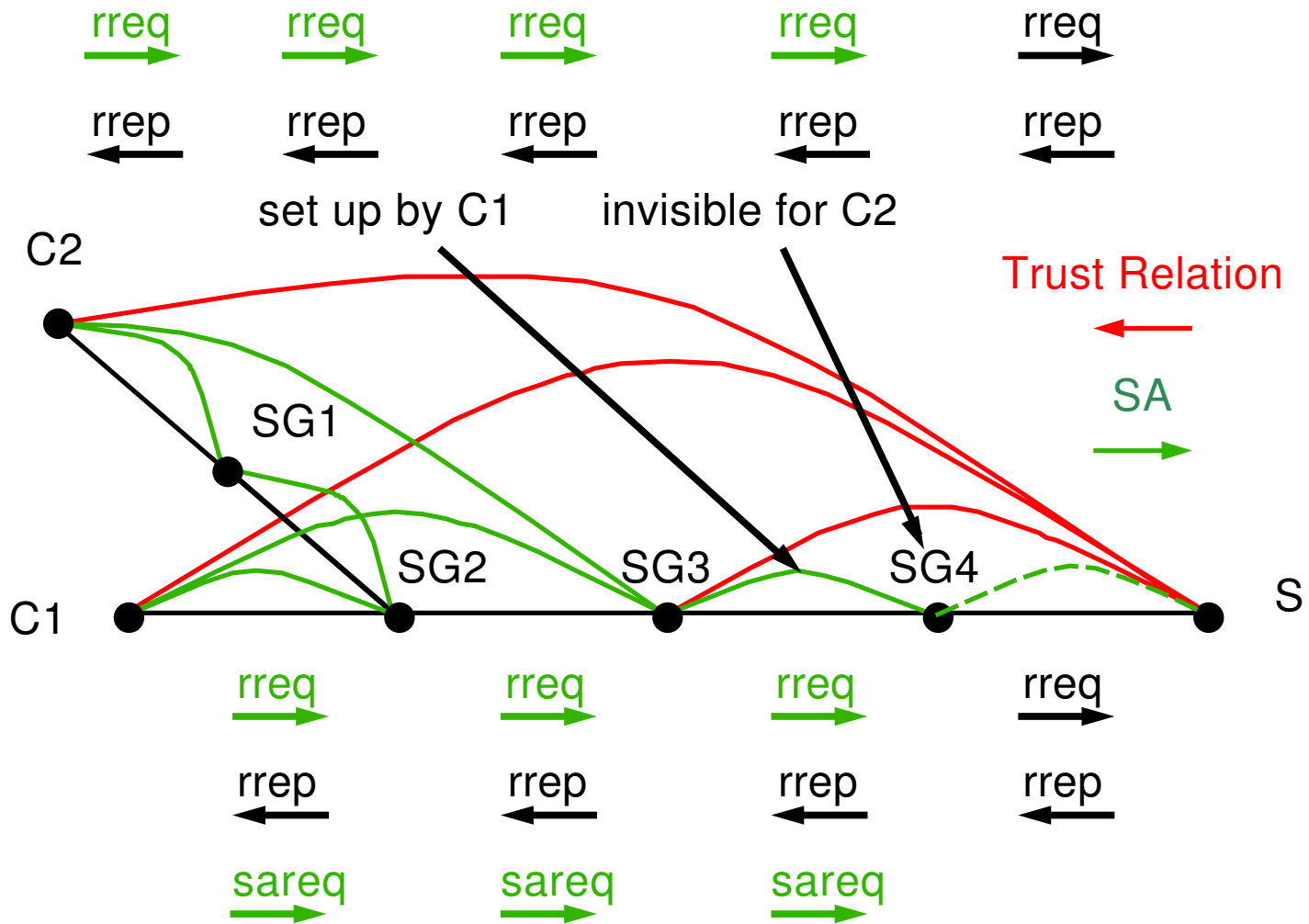
Type 3 Trace



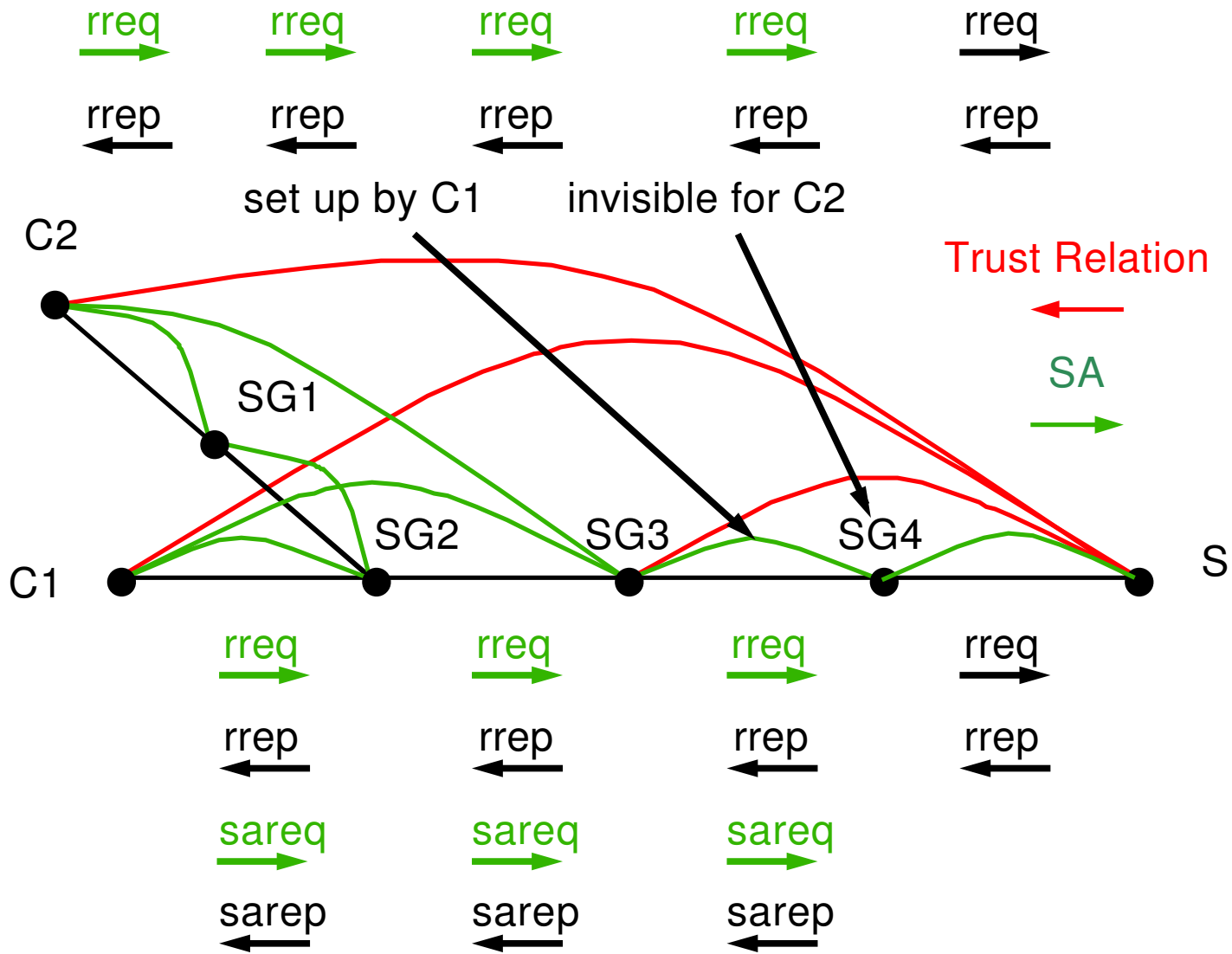
Type 3 Trace



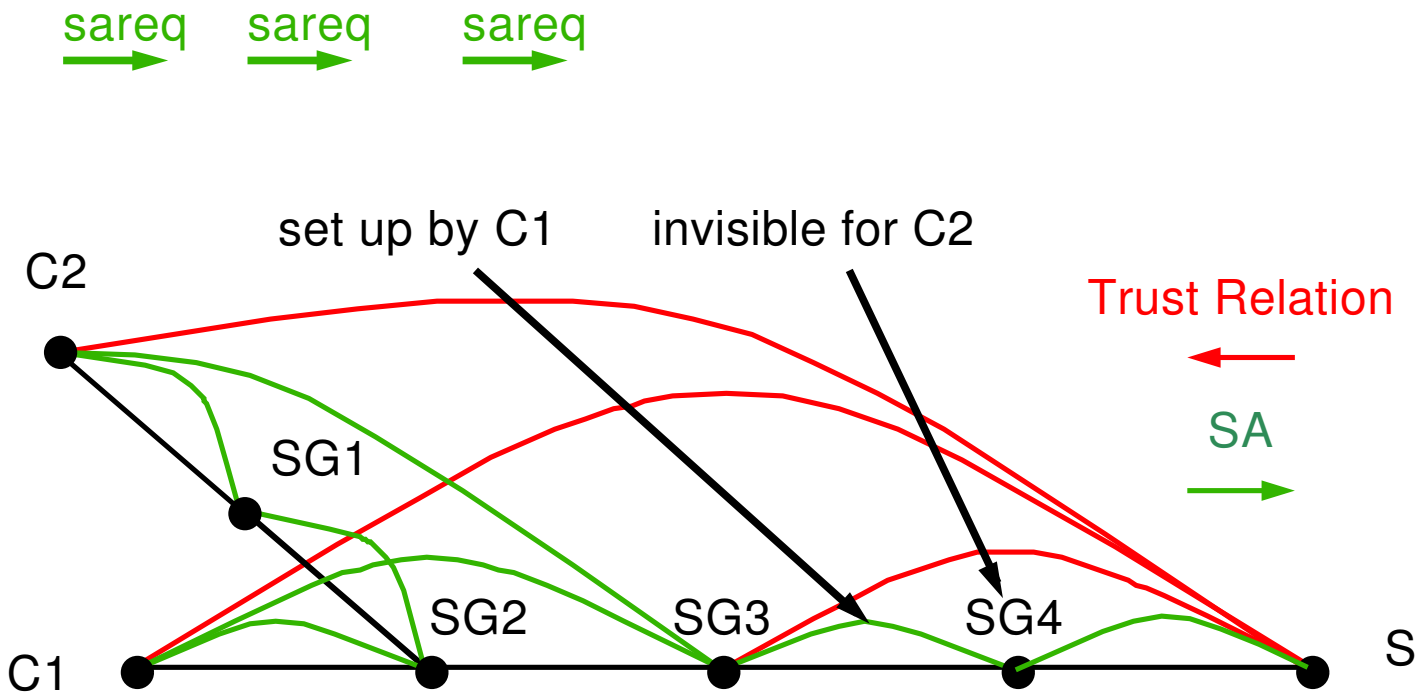
Type 3 Trace



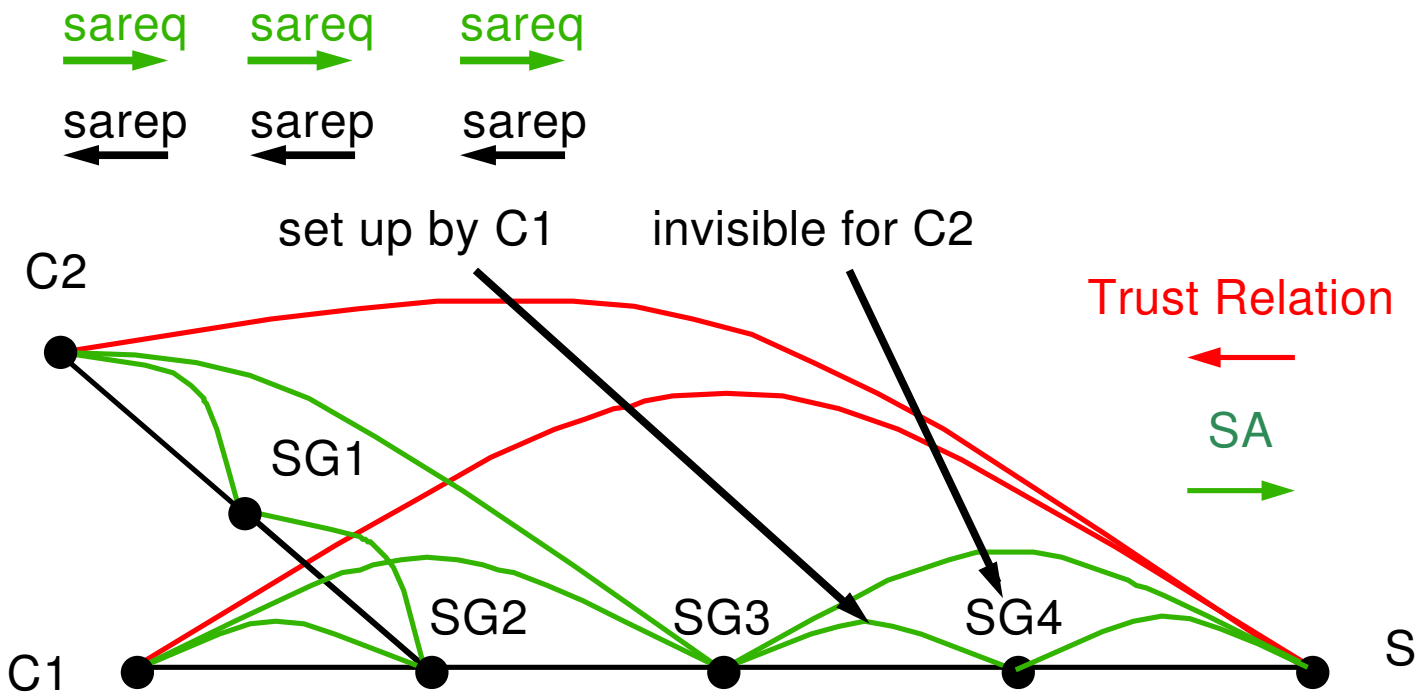
Type 3 Trace



Type 3 Trace

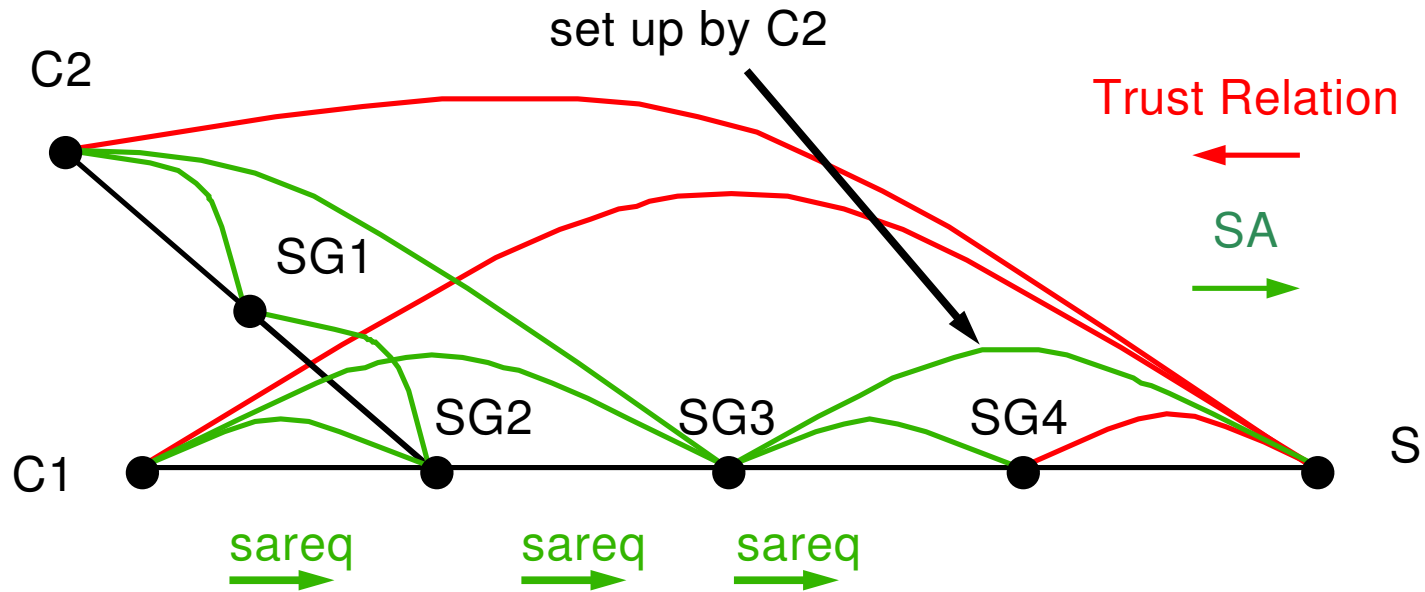


Type 3 Trace



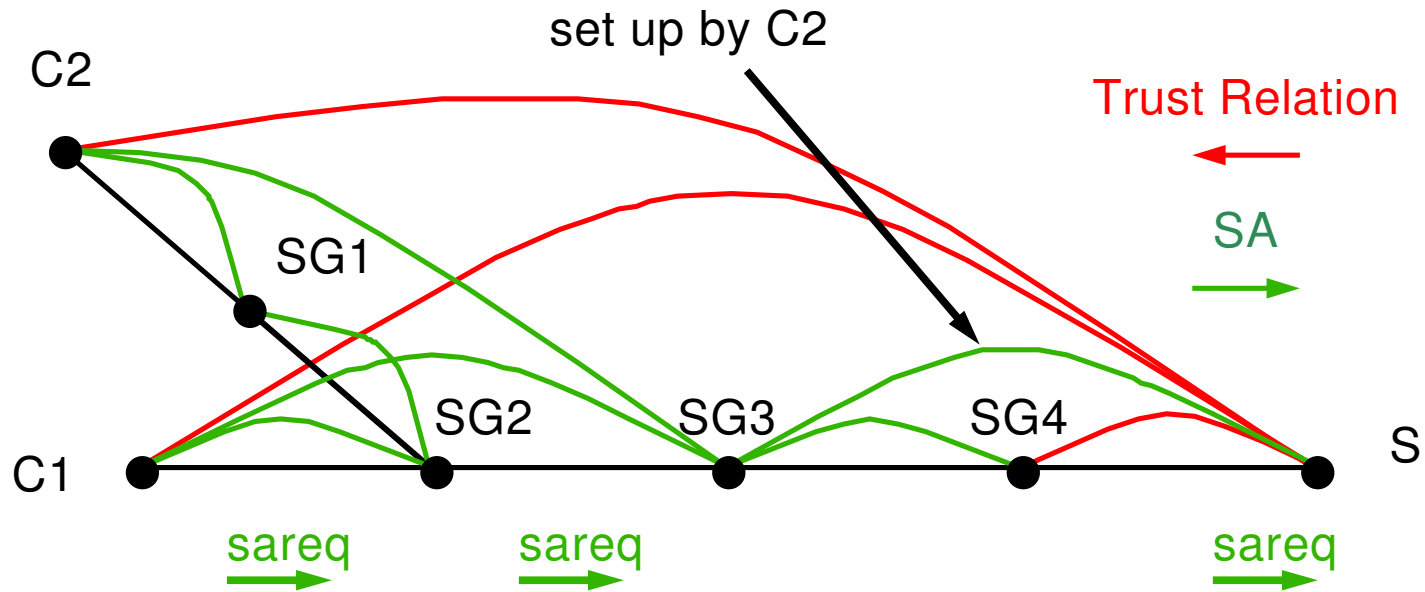
Both clients terminate successfully,
but one SA is redundant

Type 4 Trace



```
ipsec-send(node("SG3"),  
  ip(addr("C1a"), addr("SG4a"), sareq(addr("C1a"), addr("Sa"), addr("SG4a"), addr("Sa"))))
```

Type 4 Trace



```
ipsec-received(node("S"), addr("Sa"), eAttrSet,  
  ssAList(sa(addr("SG3a"), addr("Sa"), 0)),  
  ip(addr("C1a"), addr("SG4a"), sareq(addr("C1a"), addr("Sa"), addr("SG4a"), addr("Sa"))))
```

Conclusion & Future Work

- Recall key questions:
 - Does the formal model adequately capture intended model ?
After several revisions using light-weight methods for validation we are confident that this is the case, with the exception that failures/retries are not formalized yet.
 - Does the formal model have the desired properties ?
Our analysis show that the protocol successfully sets up security associations in a various scenarios.
 - Sequential exeuction: unique solution as expected
 - Concurrent execution: additional solutions due to interferece
Type 2 and 3 solutions are not globally optimal, but these effects of interference have been expected.
Type 4 solution was surprising, but the retry mechanism ensures that the protocol succeeds in this case as well.

Conclusion & Future Work

- Gaps filled in the formal specification:
 - Choice of security policy patterns and entries
(we introduced three patterns: initiation, response, towards(server))
 - Protocol used to negotiate SAs and to modify SP databases
(currently modelled in the most abstract way)
 - API used between sectrace and IPSec/packetfilter
(sectrace must be aware of discarded rrep packets)
- Future Work:
 - Model/handle network errors/timeouts (intentionally omitted so far)
 - Formalize key properties of the protocol