On the Significance of Formal Methods in the Development of High-Assurance Secure Systems

Mark-Oliver Stehr
University of Illinois at Urbana-Champaign
Overview

• The Maude Methodology

• Role of Formal Methods in Real-World Case Studies:
  • PLAN: Packet Language for Active Networks
  • Sectrace: Protocol to Configure Security Associations/Policies
  • Spread: Group Communication Middleware
  • Cliques: Toolkit for Secure Group Communication

• Conclusion
• Outlook
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
PLAN

- PLAN = Packet Language for Active Networks
- set of packets executing in the network
- functional language with side effects
  - usual functional data types
  - network specific types
- computational/communication resource limited
- primitives for sending packets
  - remotely executed function call
- service packages for interacting with nodes
Active Networks

- What are active Networks?
  - Wired, wireless or hybrid networks supporting active packets
  - Active packets carry code that
    - executes on routers
    - gather information
    - modify node state
    - configure router
- Executing active packets allows for
  - optimizations and adaptation to changing requirements
  - extensions of current protocols
A PLAN Program

(\text{LetRec}\ ["goback" = \text{Lam}(["k","route"): (TKey,(TList TAddr))]) \ldots ]

(\text{LetRec}
\["find" = \text{Lam}(["dest","previous","k"): (TAddr,TAddr,TKey)])
(\text{If} (\text{Exists} ((\text{String} ""),"k"{}{0}))
   \text{Then Dummy}
\text{Else ((Put ((\text{String} ""),"k"{}{0}, "previous"{0}, (\text{Int} 200)))):
   (\text{If} (\text{ThisHostIs} "dest"{0})
      \text{Then ("goback"{0} ("k"{}{0},Nil))}
      \text{Else (}
      (\text{Let} ["neighbors" = (\text{GetNeighbors empty-exl})]
      (\text{Let} ["srcdev" = (\text{GetSrcDev empty-exl})]
      (\text{Let} ["childrb" = \ldots ] *** divide up rb
      (\text{Let} ["sendchild" = \text{Lam} [ ("n","u") : ((TPair TAddr TAddr),TUnit) ]
         (\text{OnNeighbor}
            (\text{Chunk} ("find"{0}, ("dest"{0},(\text{Snd} "n"{0}),"k"{0}))),
            (\text{Fst} "n"{0}), *** neighbor dev
            "childrb"{0}, *** resource bound
            (\text{Snd} "n"{0}))) *** out dev]
         (\text{Foldr} ("sendchild"{0},"neighbors"{0},\text{Dummy})))))))
\]

(\text{"find"}{0} ((\text{Addr find-dest}), (\text{GetSource empty-exl}),
   (\text{GenerateKey empty-exl}))))).
Example Network

0 - 1 - 2 - 5
  |   |
  a   b   d
  |   |
  4 - 3 - 1
  |   |
  c   e
Example Network

element-topology =

  FreshKey(10)
  Node(loc("l0"), (addr("i0"), addr("a0")),
       ((addr("a0") >> addr("a1"))),
       ((addr("a1") via (addr("a0") >> addr("a1")) ),
        (addr("b1") via (addr("a0") >> addr("a1")) ),
        ...
      ))
  Data(loc("l0"), empty-dil)
  Node(loc("l1"), (addr("a1"), addr("b1"), addr("c1")),
        ((addr("a1") >> addr("a0"))),
        ((addr("a0") via (addr("a1") >> addr("a0")) ),
         (addr("b1") via (addr("b1") >> addr("b2")) ),
         (addr("c1") via (addr("c1") >> addr("c3")) ),
         ((addr("a0") via (addr("a1") >> addr("a0")) ),
          (addr("b2") via (addr("b1") >> addr("b2")) ),
          ...
        ))
  Data(loc("l1"), empty-dil)
  ...
  ...

...
Packet Emission Rule

crl Node(l,devs,nbrs,rt) 
    Process(l, orign, ardev, ssn, rb, 
        RedState(cx, (OnNeighbor ((Chunk (val,vall)), 
            (Addr dest),(Int int),(Addr dev)))))
=>
    Node(l,devs,nbrs,rt) 
    Process(l, orign, ardev, ssn, (rb - int), 
        RedState(cx, Dummy)) 
    Packet(dest, dest, orign, ssn, (int - 1), NoRoute, val, vall)

if connection(devs,nbrs,(dev >> dest)) and 
    (rb >= int) and (int > 0) .
Packet Delivery Rule

crl Node(l,devs,nbrs,rt)
    Packet(dest, fdest, orig, ssn, rb, rf, val, vall)
    =>
    Node(l,devs,nbrs,rt)
    Process(l, orig, dest, ssn, rb,
        RedState(?,(val vall)))

if (dest == fdest) and contains(devs,dest).
Symbolic Execution

rew example-topology
    Process(loc("l0"), addr("i0"), addr("i0"), 1, 100,
        RedState(? , find-prog(addr("e4")))) .
result Configuration:
    FreshKey(11)
    Data(loc("l0"), DataItem("", 10, Addr addr("i0"), 200))
    Data(loc("l1"), DataItem("", 10, Addr addr("a0"), 200))
    Data(loc("l2"), DataItem("", 10, Addr addr("b1"), 200))
    Data(loc("l3"), DataItem("", 10, Addr addr("c1"), 200))
    Data(loc("l4"), DataItem("", 10, Addr addr("e3"), 200))
    Data(loc("l5"), DataItem("", 10, Addr addr("d3"), 200)) ...
    Process(loc("l0"), addr("i0"), addr("a0"), 1, 4,
        RedState(? , Print (Cons (Addr addr("a1"),
            Cons (Addr addr("c3"),
                Cons (Addr addr("e4"),Nil)))))))
Graphical Representation

\[ \text{find(e4)} \]

![Graphical diagram with nodes and edges labeled with letters a, b, c, d, and e.](image-url)
PLAN Conclusion

• We have employed a general technique to specify the operational semantics of programming languages.
• We have filled gaps and resolved ambiguities of the original specification:
  • issues of names and binding, environments in packets
  • exception handling mechanism
  • side-effects in iterators
  • modelling concurrency/distribution
• We have specified a more general language xPLAN that represents a family of programming languages for mobile computation based on remote function calls.
• We have proved termination:
  If a PLAN program is injected into the network, then all associated processes terminate (assuming fairness).
Sectrace

- 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:**

```
Link/Physical Layer
  IP
  IPSec
  UDP/TCP
  UDP
  TCP

  Applications
```

**Simple IPSec Packet:**

```
MAC   IP   AH/ESP   UDP/TCP   Data
```
Security Associations

Simple IPSec Packet:

- IP
  src, dest
- AH/ESP
  SPI
- Data

- 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:

| IP src,dest | AH/ESP SPI | IP src,dest | Data |

Tunnel Mode Scenario:

Client

Untrusted Subnet

Security Gateways

Server

SA
Nested SAs

Nested IPSec Packet:

<table>
<thead>
<tr>
<th>IP</th>
<th>AH/ESP</th>
<th>IP</th>
<th>AH/ESP</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>src,dest</td>
<td>SPI</td>
<td>src,dest</td>
<td>SPI</td>
<td></td>
</tr>
</tbody>
</table>

Nested SA Scenario:

Security Gateways

Client

Untrusted Subnet

Server

SA

SA
Sectrace

- 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

- Reference: Secure Traceroute (Sectrace), Draft, December 2002, by Carl A. Gunter, Alwyn Goodloe, and Michael McDougall
"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 using 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."

\[
\text{crl sectrace-rreq(node,client,server,sMessageList(rreq(client,server)),rreplist)}
\]

\[
\text{rootinfo(node,myroot,mytrustedroots)
ipsec-delivered(node,rinterface,attrs,sabundle,message)} \Rightarrow
\text{sectrace-rrep3(node,client,server,
  sMessageList(sareq(client,server,initiator,responder)),
  (rreplist sMessageList(message)))}
\]

\[
\text{rootinfo(node,myroot,mytrustedroots)
if not(contains(rreplist,message)) \land
  ip(responder,dest,rrep(client,server,root,trustedroots,done)) := message \land
  initiator := select(client,myroot,(rreplist sMessageList(message)),responder) \land
  initiator /= client.}
\]

\[
\text{crl sectrace-rrep3(node,client,server,}
  sMessageList(sareq(client,server,initiator,responder)),
  (rreplist sMessageList(message)))
\]

\[
\text{rootinfo(node,myroot,mytrustedroots) =>
sectrace-sareq(node,client,server,
  sMessageList(sareq(client,server,initiator,responder)),
  (rreplist sMessageList(message)))}
\]

\[
\text{rootinfo(node,myroot,mytrustedroots)
ipsec-send(node,message')
if message' := ip(client,initiator,sareq(client,server,initiator,responder)).
\]
Typical Scenario

Trust Relation

C2

SG1

SG2

SG3

SG4

S

C1
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("C1a")))
    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.

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

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

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

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

spdb(node("C1"),
  eSPList,
  sPList(sp(towards(addr("Sa")),
    sAList(sa(addr("C1a"), addr("SG2b"), 0)) sSet(sa(addr("C1a"), addr("SG3a"), 0))
    sSet(sp(isinitiation, eAList))
    sSet(sp(isresponse, eAList)))
spdb(node("SG2"),
  sPList(sp(towards(addr("Sa")), sAlist(sa(addr("C1a"), addr("SG2b"), 0))))
  sPList(sp(towards(addr("Sa")), sAlist(sa(addr("SG1a"), addr("SG2a"), 0))))
  sSet(sp(isinitiation, eAList))
  sSet(sp(isresponse, eAList)))
...
Execution (Default Strategy)

Trust Relation
Execution (Default Strategy)

Trust Relation

C1 -> SG1, SG2, SG3, SG4 -> S

rreq

rrep
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)

set up by C1
Execution (Default Strategy)

\[ \text{C1} \rightarrow \text{SG2} \rightarrow \text{SG3} \rightarrow \text{SG4} \rightarrow \text{S} \]

Trust Relation

set up by C1

SA
Execution (Default Strategy)

set up by C1

Trust Relation

SA
Execution (Default Strategy)

C1

SG2

SG3

SG4

Trust Relation

SA

rreq

rreq

rrep

rrep

set up by C1

C2

SG1

SG2

SG3

SG4

S

C1
Execution (Default Strategy)

\[ \begin{align*}
&\text{rreq} \quad \text{rreq} \\
&\text{rrep} \quad \text{rrep} \\
&\text{sareq} \\
&\text{sarep}
\end{align*} \]

\[ \begin{align*}
C1 & \quad \text{set up by } C1 \\
C2 \\
S
\end{align*} \]
Execution (Default Strategy)

C1
SSG2 SGG3 SGG4
C2
SG1

Trust Relation
rreq rreq rrep
rrep rrep rrep

set up by C1

SG1 SG2 SG3 SG4
S
Execution (Default Strategy)

The diagram illustrates a trust relation set up by C1. The nodes C1, C2, SG1, SG2, SG3, SG4, and S are connected to show the execution flow according to the default strategy. The trust relation and SA (anonymity) are indicated by arrows.
Execution (Default Strategy)

set up by C1
invisible for C2

Trust Relation

SA
Execution (Default Strategy)

- rreq  rreq  rreq  rreq  rreq
- rrep  rrep  rrep  rrep  rrep

*set up by C1*

*invisible for C2*

**Trust Relation**

**SA**
Execution (Default Strategy)

Sectrace terminates successfully
and has set up shortest SA
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("SG3a"), 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 331100ms cpu (3342860ms real) (168106 rewrites/second)
Sectrace Conclusion

- Gaps filled in the formal specification:
  - Choice of security policy patterns and entries
  - Protocol used to negotiate SAs and to modify SP databases
  - API used between sectrace and IPSec
- Formal Specification has led to better understanding of the protocol
- Formal Analysis revealed unexpected behaviour
- Currently several variations of the protocol are being explored on the basis of the formal model
Group Communication: Spread

• Evolved from Transis and Totem
• Developed by Yair Amir and Jonathan Stanton
  (Johns Hopkins University)
• Allows Network Partitioning and Merging
• Supports different Ordering Guarantees:
  • Unordered, FIFO Order, Causal Order, Total Order
• Supports different Reliability Guarantees:
  • Unreliable, Reliable, Safe Delivery
• Implementations exist for various systems
  (UNIX and Windows)
Configurations & Groups
Configurations & Groups
Configurations & Groups
Configurations & Groups
Configurations & Groups
Configurations & Groups
Partial Order of Configurations
Partial Order of Configurations

repeated
split

repeated
merge
Partial Order of Configurations

concurrent
split & merge
Partial Order of Configurations

asymmetric behavior
Spread API Overview

Messages

ops reliable fifo causal agreed safe : -> Mode.

op msg-data : Agent Group Data Mode -> Message.
op msg-trans : Group -> Message.
op msg-join : Agent Group AgentList Membership -> Message.
op msg-leave : Agent Group AgentList Membership -> Message.
op msg-network : Group Membership AgentList AgentList -> Message.

Join

op sp-join-req : Agent Group -> Configuration.
op sp-join-ack : Agent -> Configuration.
op sp-join-err : Agent JoinError -> Configuration.

Leave

op sp-leave-req : Agent Group -> Configuration.
op sp-leave-ack : Agent -> Configuration.
op sp-leave-err : Agent LeaveError -> Configuration.

Multicast

op sp-multicast-req : Agent Group Data Mode -> Configuration.
op sp-multicast-ack : Agent -> Configuration.
op sp-multicast-err : Agent MulticastError -> Configuration.

Receive

op sp-receive-req : Agent -> Configuration.
op sp-receive-ack : Agent Message -> Configuration.
op sp-receive-err : Agent ReceiveError -> Configuration.

Miscellaneous

op sp-connect-req : Agent -> Configuration.
op sp-connect-ack : Agent -> Configuration.
op sp-connect-err : Agent ConnectError -> Configuration.

op sp-disconnect-req : Agent -> Configuration.
op sp-disconnect-ack : Agent -> Configuration.
op sp-disconnect-err : Agent DisconnectError -> Configuration.
Delivering Safe Message

crl operational(proc)
  localconf(proc,conf)
  delivered(proc,delivered)
  knownseqs(proc,conf,knownseqs)
  received(proc,sMessage Set(message) received)
  causalorder(conf,constraints)
  totalorder(conf,events)
=>
  operational(proc)
  localconf(proc,conf)
  knownseqs(proc,conf,(knownseqs knownseqs(message)))
  received(proc,received)
  \textbf{delivered}(proc,(delivered sMessage List(message)))
  causalorder(conf,constraints)
  totalorder(conf,(events sEvent List(event(src(message),seq(message))))

if deliverablesafe(proc,conf,received,delivered,message,constraints,events) and
  mode(message) == safe and isacked(message) .
Typical Scenario
Initial Configuration

network(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c"))))

operational(proc("a")) operational(proc("b")) operational(proc("c"))

reachable(proc("a"), sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")))
reachable(proc("b"), sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")))
reachable(proc("c"), sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")))

sent(proc("a"), eMessageSet)
sent(proc("b"), eMessageSet)
sent(proc("c"), eMessageSet)

received(proc("a"), eMessageSet)
received(proc("b"), eMessageSet)
received(proc("c"), eMessageSet)

delivered(proc("a"), eMessageList)
delivered(proc("b"), eMessageList)
delivered(proc("c"), eMessageList)

localconf(proc("a"), regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0))
localconf(proc("b"), regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0))
localconf(proc("c"), regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0))

causaloader(regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0), eConstraintSet)
totalorder(regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0), eEventList)

multicast-req(proc("a"), safe, data(""))
multicast-req(proc("c"), agreed, data(""))
...

network(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")))

operational(proc("a")) operational(proc("b")) operational(proc("c"))

reachable(proc("a"), sProcSet(proc("a")))
reachable(proc("b"), sProcSet(proc("b")) sProcSet(proc("c")))
reachable(proc("c"), sProcSet(proc("b")) sProcSet(proc("c")))

sent(proc("a"), eMessageSet)
sent(proc("b"), eMessageSet)
sent(proc("c"), eMessageSet)

received(proc("a"), eMessageSet)
received(proc("b"), eMessageSet)
received(proc("c"), eMessageSet)

delivered(proc("a"),
sMessageList(msg-data(proc("c"), agreed, regconf(sProcSet(proc("a")) sProcSet(proc("b")) sProcSet(proc("c")), 0), 1, eNatSet, data(""), false))

localconf(proc("a"), regconf(sProcSet(proc("a")), 2))
localconf(proc("b"), regconf(sProcSet(proc("b")) sProcSet(proc("c")), 4))
localconf(proc("c"), regconf(sProcSet(proc("b")) sProcSet(proc("c")), 4))
Spread Conclusion

- Abstract specification of group communication layer
- Spread is one possible implementation among others
- Use of abstract specification:
  - Verification of Spread itself
  - Verification of layers and applications on top of Spread
- Challenge: Capture best-effort principle formally
  - Idea: Explicit representation of all delivery constraints
  - Deliver as much as possible under the given constraints
Group Key Agreement: Cliques

- Objective:
  - Agree on shared secret in a peer group
  - Update shared secret if membership changes

- Existing Group Key Agreement Protocols:
  - Centralized Key Distribution
  - Burmester-Desmedt, 1994
  - Steer-Strawczynski-Diffie-Wiener, 1988
  - Steiner-Tsudik-Waidner (Group Diffie-Hellman), 1996
  - Kim-Perrig-Tsudik (Tree-Based Group Diffie-Hellman), 2000

- Cliques-Toolkit:
  - Implements all the above protocols
  - API independent of protocol and underlying GCS
Join Protocol

group $M_1 \ldots M_n$ with group key: $\alpha^{N_1 \times N_2 \times \cdots \times N_{n-1} \times N_n}$ for $i$ in $[1 \ldots n]$

each member $M_i$ knows: $\alpha^{N_1 \times N_2 \times \cdots \times N_{n-1} \times N_n}$ for $i$ in $[1 \ldots n]$

$M_n$ (current controller) $\rightarrow M_{n+1}$ (new controller)

new keyshare

$\alpha^{N_1 \times N_2 \times \cdots \times N_{n-1} \times N_{n+1}}$ for $i$ in $[1 \ldots n+1]$

each member $M_i$ knows: $\alpha^{N_1 \times N_2 \times \cdots \times N_{n-1} \times N_{n+1}}$ for $i$ in $[1 \ldots n+1]$

each member $M_i$ factors in its keyshare to obtain group key:

$\alpha^{N_1 \times N_2 \times \cdots \times N_{n-1} \times N_n}$ for $i$ in $[1 \ldots n+1]$
Cyclic Groups

sort PartialKey.
op idPartialKey : -> PartialKey.
op _*_ : PartialKey PartialKey -> PartialKey
   [assoc comm id: idPartialKey].
op inv : PartialKey -> PartialKey.
eq inv(x) * x = idPartialKey.

op alpha : -> PartialKey.

sort KeyShare.
op idKeyShare : -> KeyShare.
op _*_ : KeyShare KeyShare -> KeyShare
   [assoc comm id: idKeyShare].
op inv : KeyShare -> KeyShare.
eq inv(y) * y = idKeyShare.

op _^_ : PartialKey KeyShare -> PartialKey.
eq x ^ idKeyShare = x.
eq (x ^ y) ^ z = x ^ (y * z).

op random : Nat -> KeyShare.

group of (partial) keys
(group secrets)
group generator
represented as constant
group of key contributions
(key shares)

exponential normal form
alpha ^ N_1 * N_2 * ... * N_{n-1} * N_n

random keyshares
represented as constant
 Cliques API Overview

**Single Member Join**

- `op clq-proc-join-req : Agent Group Context Agent → State .`
- `op clq-join-req : Agent Group Token → State .`

**Leave Operation**

- `op clq-leave-req : Agent Group Context AgentList → State .`
- `op clq-refresh-key-req : Agent Group Context Token → State .`
- `op clq-refresh-key-ack : Agent Group Context → State .`

**Merge Operation**

- `op clq-update-key-first-req : Agent Group Context AgentList → State .`
- `op clq-update-key-first-ack : Agent Group Context Token → State .`
- `op clq-update-key-intermediate-req : Agent Group Context Token → State .`
- `op clq-update-key-intermediate-ack : Agent Group Context Token → State .`
- `op clq-update-key-last-req : Agent Group Context Token → State .`
- `op clq-update-key-last-ack : Agent Group Context Token → State .`
- `op clq-factor-out-req : Agent Group Context Token → State .`
- `op clq-merge-req : Agent Group Context Agent Token AgentSet → State .`
- `op clq-merge-ack : Agent Group Context AgentSet → State .`
- `op clq-merge-ack : Agent Group Context AgentSet Token → State .`

**Other Operations**

- `op clq-first-user-req : Agent Group → State .`
- `op clq-first-user-ack : Agent Group Context → State .`
- `op clq-new-user-req : Agent Group → State .`
Symbolic Execution

rew fresh(0)
  start(agent("a"), group("G"))
  join(agent("b"), ..., group("G"))
  join(agent("c"), ..., group("G"))
  join(agent("d"), ..., group("G"))
  leave(sAgentList(agent("d")), ..., group("G")) .

rewrites: 3508 in 20ms cpu (175400 rewrites/second)
result State: fresh(7)
ready(agent("a"), group("G"),
  context(random(0), alpha ^ random(0)*random(2)*random(5)*random(6), ...)

ready(agent("b"), group("G"),
  context(random(2), alpha ^ random(0)*random(2)*random(5)*random(6), ...)

ready(agent("c"), group("G"),
  context(random(6), alpha ^ random(0)*random(2)*random(5)*random(6), ...)

ready(agent("d"), group("G"),
  context(random(0), alpha ^ random(0)*random(2)*random(5)*random(6), ...)
State Space Exploration

search fresh(0)
  start(agent("a"), group("G"))
  merge(sAgentList(agent("b")) sAgentList(agent("c")), ..., group("G"))
  =>!
state:State.

Solution 1 (state 260645)
states: 260646 rewrites: 27859114 in 524600ms cpu (53105 rewrites/second)

state:State --> fresh(6)
ready(agent("a"), group("G"),
  context(random(0), alpha ^ random(0) * random(2) * random(4), ...)
ready(agent("b"), group("G"),
  context(random(2), alpha ^ random(0) * random(2) * random(4), ...)
ready(agent("c"), group("G"),
  context(random(4), alpha ^ random(0) * random(2) * random(4), ...)
No more solutions.
states: 260646 rewrites: 27859114 in 524600ms cpu (53105 rewrites/second)
Modelchecking

op done : -> Prop.
ceq fresh(var)
    ready(agent("a"), group("G"), contexta, memberlista)
    ready(agent("b"), group("G"), contextb, memberlistb)
    ready(agent("c"), group("G"), contextc, memberlistc)
    |=
    done = true
if groupsecret(contexta) == groupsecret(contextb) /
    groupsecret(contextb) == groupsecret(contextc).

op initial : -> State.
eq initial = fresh(0)
    start(agent("a"), group("G"))
    merge(sAgentList(agent("b")) sAgentList(agent("c")), ..., group("G"))

red modelCheck(initial, <> done).
rewrites: 27753469 in 176690ms cpu (157074 rewrites/second)
result Bool: true
Cliques Conclusion

- Abstract specification of Cliques toolkit
- focussing on Group Diffie-Hellman protocol

- Use of abstract specification:
  - Verification of Cliques itself
  - Verification of layers and applications on top of Cliques

- Challenges:
  - Capture generic character of API
  - Abstract Specification of Cryptographic Primitives
  - Formal model obtained directly from the C code
General Conclusion

- Formal Methods at various stages of the development process:
  - PLAN: Language Design, Properties of Programs
  - Sectrace: Formal Prototypes, Exploring the Design Space
  - Spread, Cliques: Post-Implementation, Abstract Specification
- Formal specification of languages/protocols/middleware is worth the effort because critical and basis for many applications
- Provides unambiguous documentation/clarification of the behavior of implementations in all conceivable circumstances
- High degree of data-driven concurrency and nondeterminism makes rewriting logic a natural choice as a specification language
- Light-weight methods can lead to important results and insights with a minimum amount of effort
- Heavy-weight methods are applied after a mature state and a clear understanding has been reached
Outlook

- Composable Services

- Composable Interaction Patterns