Work in progress

One Picture is Worth a ~~Thousand Words~~ Couple Dozen Connectives

Iliano Cervesato **iliano@itd.nrl.navy.mil**

ITT Industries, inc @ NRL Washington, DC

http://theory.stanford.edu/~iliano

Joint work with Cathy Meadows

How this work came about

Analysis of GDOI group protocol

- Requirements expressed in NPATRL
  - Novel group properties
  - Medium size specifications
    - Dozen operators
  - Lots of fine-tuning
- Difficult to read and share specs.
- Informal use of fault trees
  - Intuitive visualization medium
  - Became favored language
- Formal relation with NPATRL

Security Requirements

Describe what a protocol should do

- Verified by
  - Model checking
  - Mathematical proof
  - Pattern-matching (in some cases)
- Expressed
  - Informally
  - Semi-formally
  - Formal language
- Adequate for toy protocols

  BUT, do not scale to real protocols

Example: Kerberos 5

[CSFW'02]

Theorem 1. *For* C : client, T : TGS, $C, T \neq$ I, S : server, k_C : dbK C, k_T : dbK T, $AKey$: shK C T, *and* n_2 : nonce, *if the beginning state of a finite trace does not contain* $\mathsf{I}(k_C)$, $\mathsf{I}(k_T)$, *or any fact* F *with* $\rho_{k_T}(F; AKey, C) > 0$ *or* $\rho_{AKey}(F; C) > 0$, *and at some point in the trace* T *fires rule* $\alpha_{4.1}$, *consuming the fact* $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$, *then earlier in the trace, some* K : KAS *fired rule* $\alpha_{2.1}$, *existentially generating* $AKey$ *and producing the fact* $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k'})$ *for some* n : nonce *and* k' : dbK C. *Also, after* K *fired this rule and before* T *fired the rule in the hypothesis,* C *fired rule* $\alpha_{3.1}$ *to create the fact* $\mathsf{N}(X, \{C\}_{AKey}, C, S', n')$ *for some* X : msg, S' : server, *and* n' : nonce.

- Semi-formal
  - But very precise
- Bulky and unintuitive
  - Requires several readings to grasp

Example: GDOI

[CCS'01]

$$
\begin{aligned}
& \mathsf{learn}(P, (), (K_G), \_) \\
\Rightarrow\ & \diamondminus \mathsf{learn}(P, (), (K_G), \_) \wedge \diamondminus (\mathsf{gcks\_createkey}(GCKS, (), (K_G), \_) \\
& \qquad \wedge \diamondminus \mathsf{gcks\_createkey}(GCKS, (), (K'_G), \_)) \\
\vee\ & \diamondminus \mathsf{gcks\_losegroupkey}(GCKS, (), (K_G), \_) \\
\vee\ & \diamondminus (\ \mathsf{gcks\_sendpushkey}(GCKS, (), (K_G, K'_G), N) \\
& \qquad \wedge \diamondminus \mathsf{gcks\_sendpullkey}(GCKS, M_{\mathsf{d}}, (N_{GM}, K''_G, K_{GM}), \_)) \\
& \wedge \neg \diamondminus (\ \mathsf{gcks\_sendpushkey}(GCKS, (), (K_G, K'_G), N) \wedge \diamondminus \mathsf{gcks\_cancel}(GCKS, M_{\mathsf{d}}, (N_{GM}), \_)) \\
\vee\ & \diamondminus \mathsf{gcks\_sendpullkey}(GCKS, M_{\mathsf{d}}, (N_{GM}, K_G, K_{GM}), \_) \\
\vee\ & \diamondminus \mathsf{gcks\_losepairwisekey}(GCKS, (), (M, K_{GM}), \_) \\
& \wedge \diamondminus \mathsf{gcks\_sendpullkey}(GCKS, M, (\_, K_G, K_{GM}), \_)
\end{aligned}
$$

- Formal
  - NPATRL protocol spec. language
- Ok for a computer
- Bulky and unintuitive for humans
  - About 20 operators

Example: Authentication [Lowe, CSFW'97]

Definition 1 (Aliveness). *We say that a protocol guarantees to an initiator A* aliveness *of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.*

Definition 2 (Weak Agreement). *We say that a protocol guarantees to an initiator A* weak agreement *with another agent B if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.*

Definition 3 (Non-Injective Agreement). *We say that a protocol guarantees to an initiator A* non-injective agreement *with a responder B on a set of data items $\mathbf{t}$ (where $\mathbf{t}$ is a set of free variables appearing in the protocol description) if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to the variables in $\mathbf{t}$.*

- Informal
  - Made precise as CSP expressions
- Simple, but …
  - … many very similar definitions

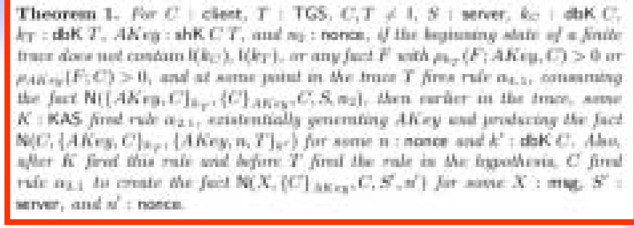
The Problem

- Desired properties are difficult to
  - Phrase & get right
  - Explain & understand
  - Modify & keep right
- Examples
  - Endless back and forth on GDOI
    - Are specs. right now?
  - K5 properties read over and over

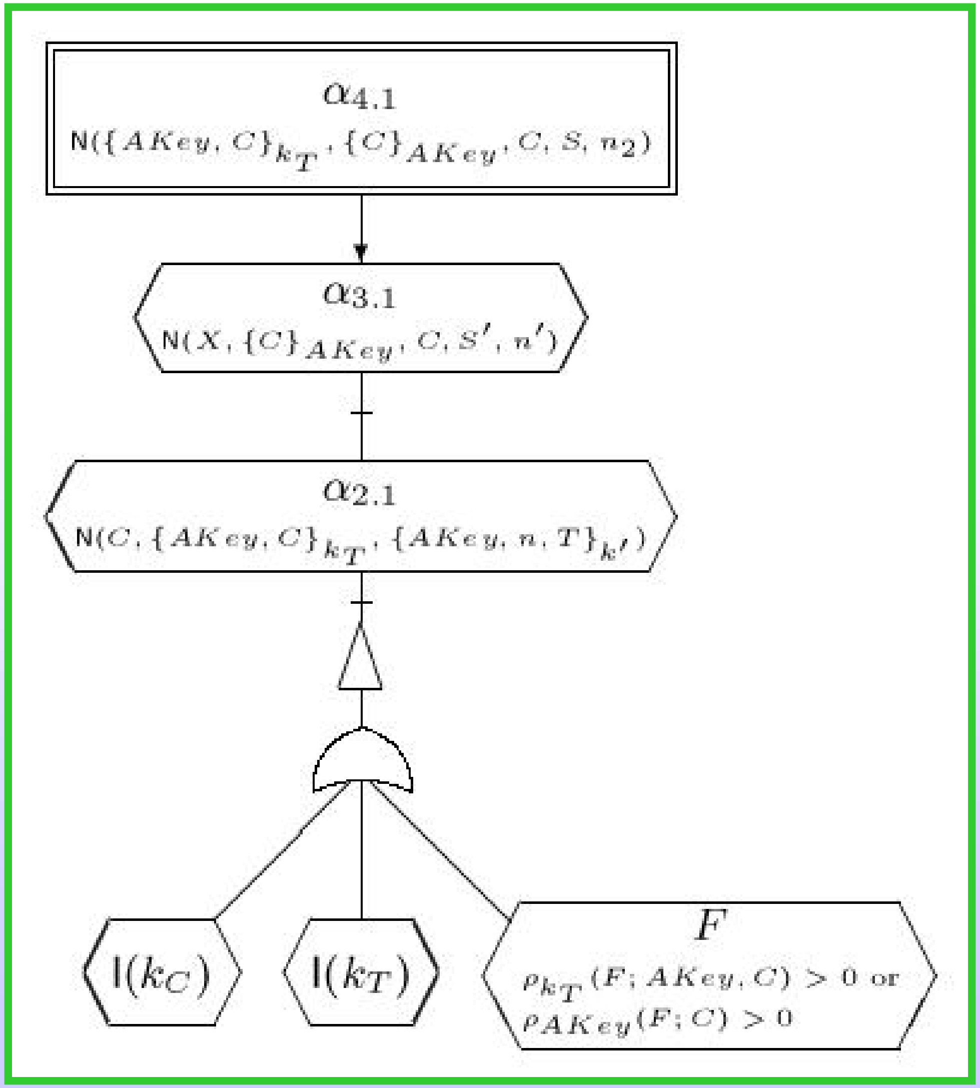
Dealing with Textual Complexity

- HCI response: graphical presentation
- Our approach: Dependence Trees
  - Re-interpretation of fault trees
  - 2D representation of NPATRL
  - Intuitive for medium size specs.

Example: Kerberos 5



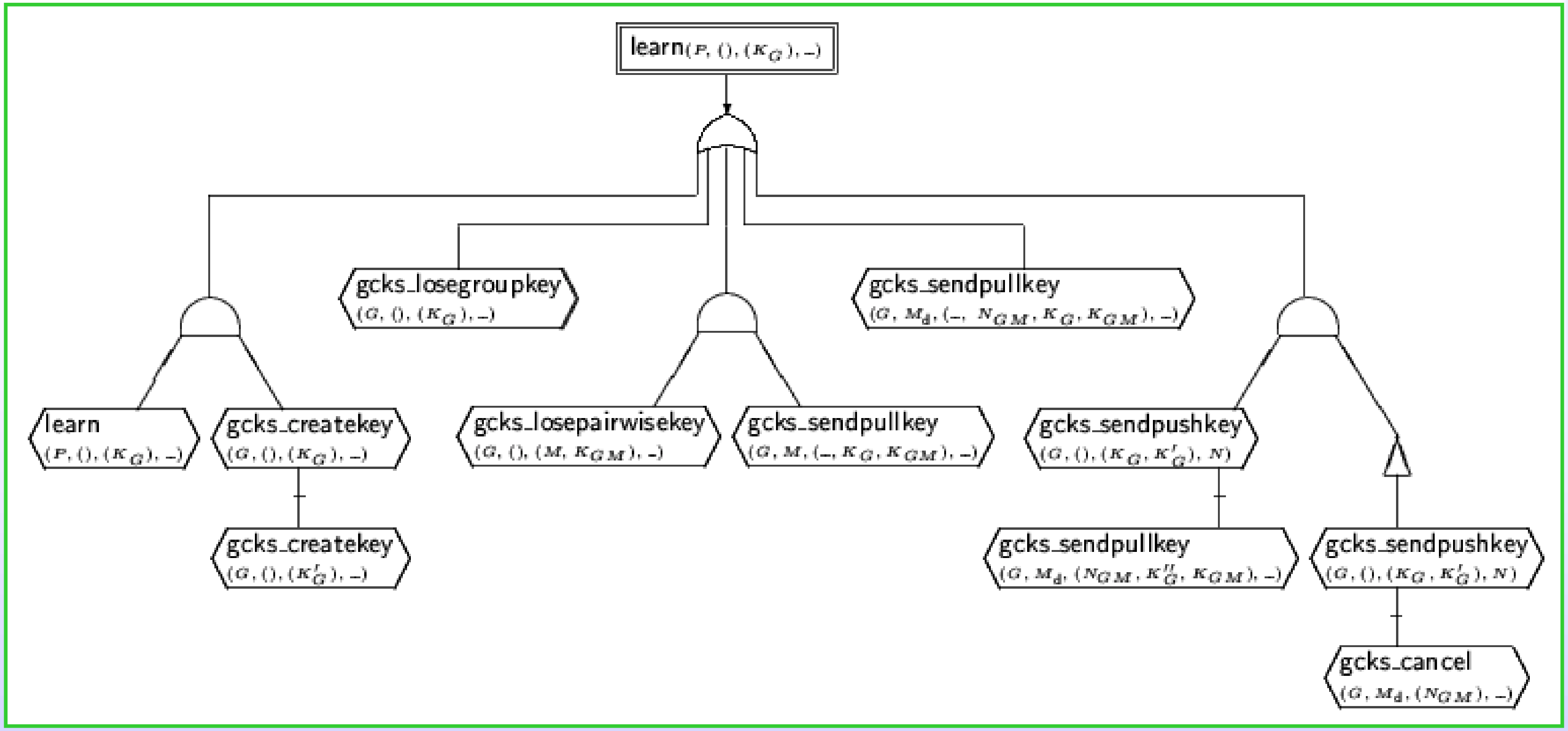
Theorem 1. *For* C : client, T : TGS, $C, T \neq I$, S : server, k_C : dbK C, k_T : dbK T, $AKey$: shK C T, *and* n_2 : nonce, *if the beginning state of a finite trace does not contain* $I(k_C)$, $I(k_T)$, *or any fact* F *with* $\rho_{k_T}(F; AKey, C) > 0$ *or* $\rho_{AKey}(F; C) > 0$, *and at some point in the trace* T *fires rule* $\alpha_{4.1}$, *consuming the fact* $N(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$, *then earlier in the trace, some* K : KAS *fired rule* $\alpha_{2.1}$, *existentially generating* $AKey$ *and producing the fact* $N(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k'})$ *for some* n : nonce *and* k' : dbK C. *Also, after* K *fired this rule and before* T *fired the rule in the hypothesis,* C *fired rule* $\alpha_{3.1}$ *to create the fact* $N(X, \{C\}_{AKey}, C, S', n')$ *for some* X : msg, S' : server, *and* n' : nonce.



- Excises the gist of the theorem
- Highlights dependencies
- Fairly intuitive
  - … in a minute …

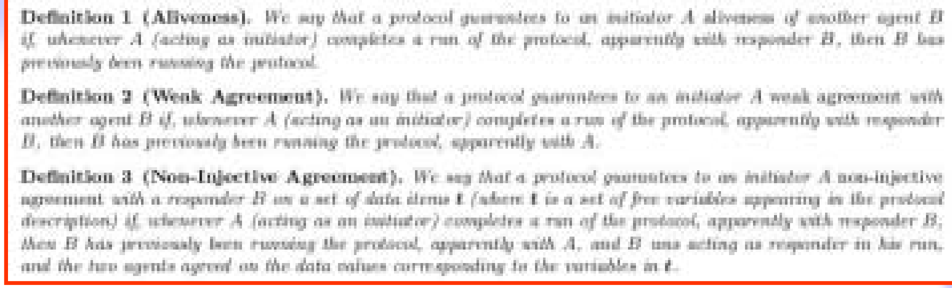
Example: GDOI

$$
\begin{array}{l}
\mathsf{learn}(P, (), (K_G), \_) \\
\rightarrow \quad \ominus\mathsf{learn}(P, (), (K_G), \_) \wedge \ominus(\mathsf{gcks\_createkey}(GCKS, (), (K_G), \_) \\
\qquad \wedge \ominus\mathsf{gcks\_createkey}(GCKS, (), (K'_G), \_)) \\
\vee \ominus\mathsf{gcks\_losegroupkey}(GCKS, (), (K_G), \_) \\
\vee \quad \ominus(\ \ \mathsf{gcks\_sendpushkey}(GCKS, (), (K_G, K'_G), N) \\
\qquad \wedge \ominus\mathsf{gcks\_sendpullkey}(GCKS, M_a, (N_{GM}, K''_G, K_{GM}), \_)) \\
\quad \wedge \neg\ominus(\ \ \mathsf{gcks\_sendpushkey}(GCKS, (), (K_G, K'_G), N) \wedge \ominus\mathsf{gcks\_cancel}(GCKS, M_a, (N_{GM}), \_)) \\
\vee \ominus\mathsf{gcks\_sendpullkey}(GCKS, M_a, (N_{GM}, K_G, K_{GM}), \_) \\
\vee \quad \ominus\mathsf{gcks\_losepairwisekey}(GCKS, (), (M, K_{GM}), \_) \\
\quad \wedge \ominus\mathsf{gcks\_sendpullkey}(GCKS, M, (\_, K_G, K_{GM}), \_)
\end{array}
$$



- Isomorphic to NPATRL specifications
- Much more intuitive
  - ... in a minute ...

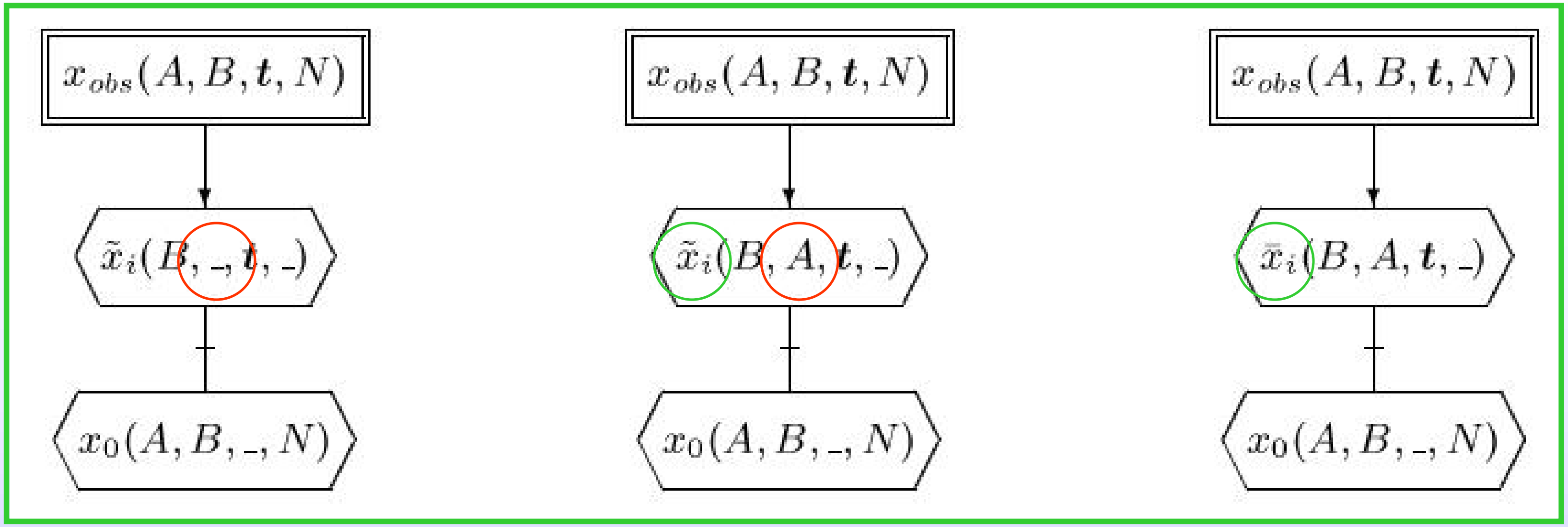
Example: Authentication



Definition 1 (Aliveness). *We say that a protocol guarantees to an initiator A aliveness of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.*

Definition 2 (Weak Agreement). *We say that a protocol guarantees to an initiator A weak agreement with another agent B if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.*

Definition 3 (Non-Injective Agreement). *We say that a protocol guarantees to an initiator A non-injective agreement with a responder B on a set of data items $\mathbf{t}$ (where $\mathbf{t}$ is a set of free variables appearing in the protocol description) if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to the variables in $\mathbf{t}$.*



- Formalize definitions
- Easy to compare …
  - … and remember …

Rest of this Talk

- Logic for protocol specs
  - NPATRL Logic
  - NRL Protocol Analyzer fragment
  - Model checking
- Precedence trees
  - Fault trees
  - NPATRL semantics
- Analysis of an example
- Future Work

NPATRL

- Formal language for protocol requirements
  - Simple temporal logic
- Designed for NRL Protocol Analyzer
  - Simplify input of protocol specs
    - Sequences of events that should not occur
  - Applies beyond NPA
- Used for many protocols
  - SET, GDOI, …

NPATRL Logic

- Events

initiator_accept_key(A, (B,S), (K_{AB},n_A), N)

name | actuator | other agents | terms | round

- Classical connectives: $\wedge$, $\vee$, $\neg$, ...
- "Previously": #

initiator_accept_key(A, (B,S), (K_{AB},n_A), N)

$\Rightarrow$ # server_sent_key(S, (A,B), (K_{AB}), _)

NPA Fragment

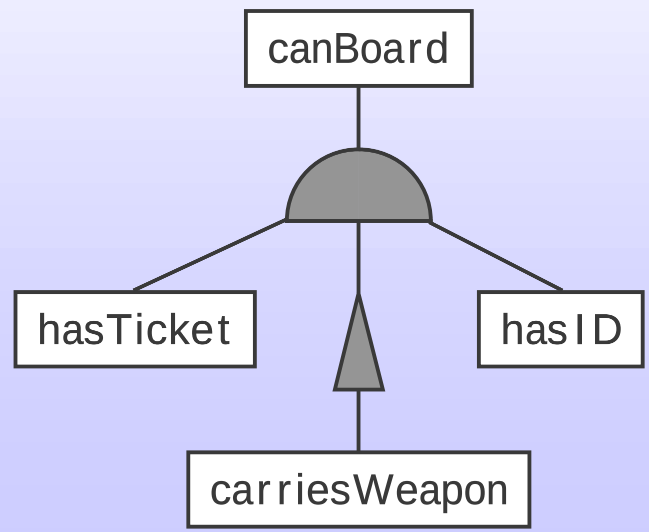
NPA uses a small fragment of NPATRL

$$R ::= a \Rightarrow F$$
$$F ::= E \mid \neg E \mid F_1 \wedge F_2 \mid F_1 \vee F_2$$
$$E ::= \#a \mid \#(a \wedge F)$$

- Efficient model checking

Fault Trees

- Safety analysis of system design
  - Root is a failure situation
    - Extended to behavior descriptions
  - Inner nodes are conditions enabling fault
    - Events
    - Combinators (logical gates)
- Example
  - *A passenger needs a ticket <u>and</u> a photo ID <u>to</u> board a plane, but should <u>not</u> carry a weapon*



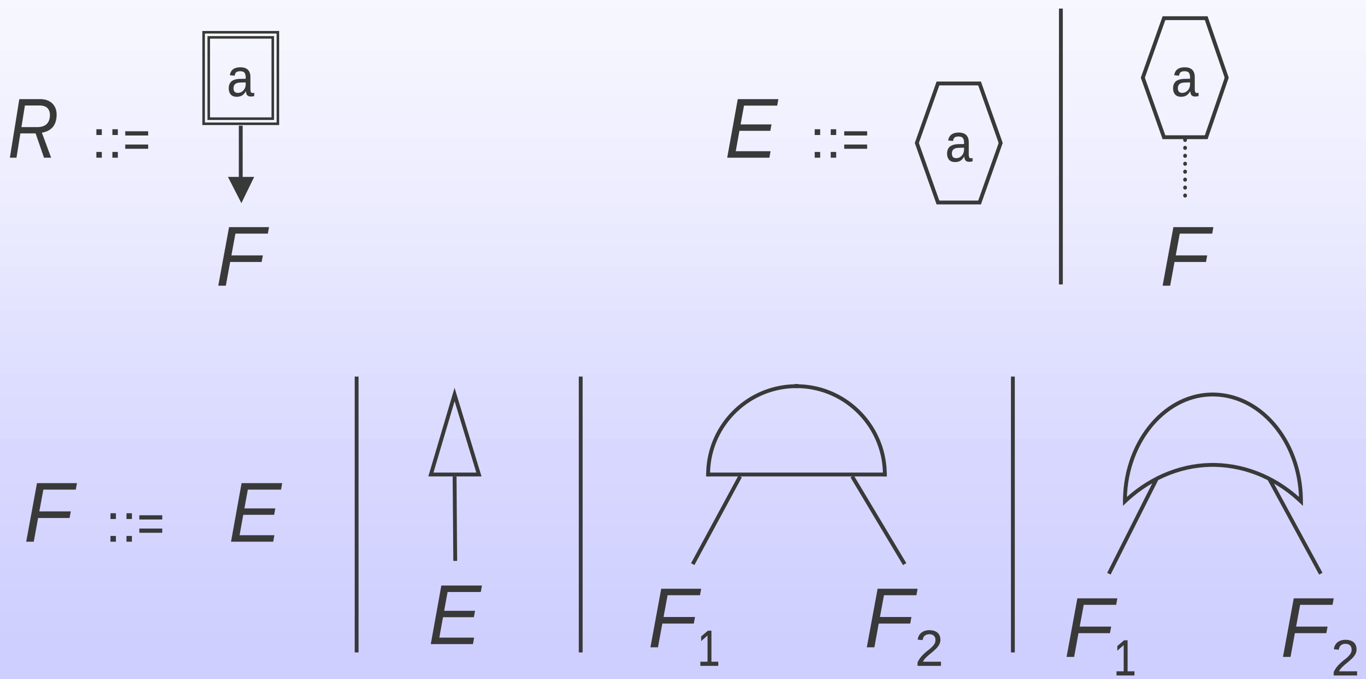
Precedence Trees

$R ::= a \Rightarrow F$

$F ::= E \mid \neg E \mid F_1 \wedge F_2 \mid F_1 \vee F_2$

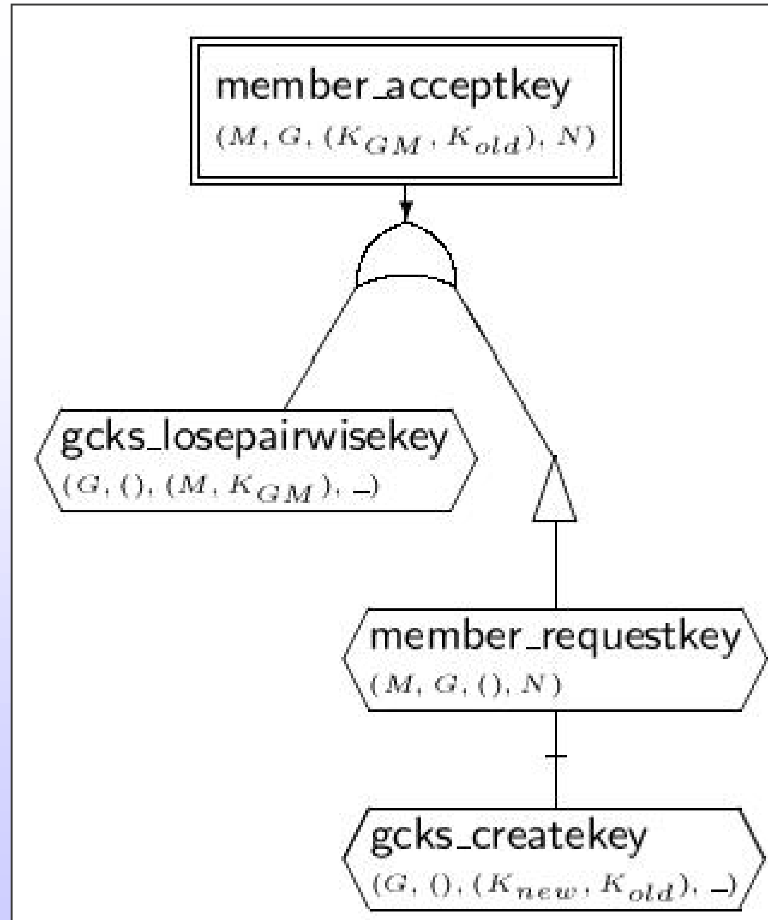
$E ::= \#a \mid \#(a \wedge F)$

- Fault tree representation of $\mathrm{NPATRL}_{\mathrm{NPA}}$
  - Isomorphism



“Recency Freshness” in GDOI

if a member accepts a key from the controller in a protocol run, no newer key should have been distributed prior to the member's request



member_accept_key(M,G,(K_{GM},K_{old}),N)
$\Rightarrow$
gcks_loseparwisekey(G,(),(M,K_{GM}),_)
$\vee$ $\neg$(# (member_requestkey(M,G,(),N)
$\wedge$ #gcks_createkey(G,(),K_{new},K_{old}),_)))

“Sequential Freshness” in GDOI

if a member accepts a key from the group controller in a protocol run, then it should not have previously accepted a later key

member_accept_key(M,G,(K_{GM},K_{old}),_)

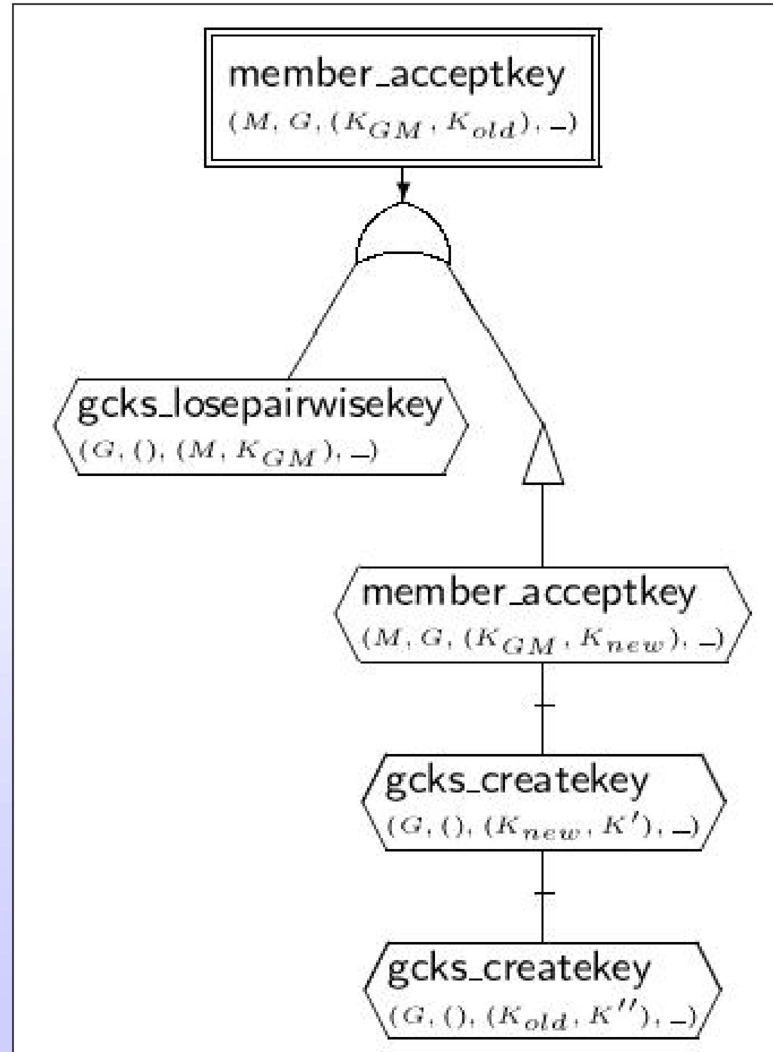
$\Rightarrow$

\# gcks_loseparwisekey(G,(),(M,K_{GM}),_)

$\vee$ $\neg$(# (member_acceptkey(M,G,(K_{GM},K_{new}),_)

$\wedge$ #(gcks_createkey(G,(),K_{new},K’),_)

$\wedge$ #gcks_createkey(G,(),K_{old},K”),_))))



Conclusions

- Explored tree representation of protocol reqs.
  - Promising initial results
  - Complex requirements now intuitive
- Precedence trees
  - Draw from fault trees research
  - Specialized to NPATRL and NPA
  - NPATRL semantics
  - Better understanding of NPATRL
- Papers
  - "A Fault-Tree Representation of NPATRL Security Requirements", with Cathy Meadows
    - WITS'03
    - TCS (long version, submitted)

Future Work – Theory

- What properties can be expressed?
  - All of safety?
  - Liveness?
- Graphical equivalence of requirements?
- Expressive power
  - Recursive trees?
  - More complex quantifier patterns?
- Graphical gist of theorems
  - Useful classes?
  - Proofs?

Future Work – Practice

- Gain further experience
  - Can they be used for other requirements?
- Scaling up
  - When are trees so big they are non-intuitive?
    - Existing requirements?
  - Modularity
- Interaction with fault tree community
  - Broader applications of dependence trees?
  - Tools we can use?
    - NPATRL <-> dependence trees