FORMALIZING SECURITY REQUIREMENTS FOR GROUP KEY DISTRIBUTION PROTOCOLS

Catherine Meadows
Code 5543
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil
http://chacs.nrl.navy.mil

WHAT'S LACKING IN FORMAL ANALYSIS OF CRYPTO PROTOCOLS

- Have long history (over 10 years)
- Applied to lots of different types of problems
- Has had some real success
  - **Applied to complex, real-life protocols**
    - SET, IKE, TLS, etc.
  - **Found previously undiscovered problems in some of these**
- But -- lack of impact on "real life" protocols
  - **Few examples to point to of formal analysis affecting fielded product**
- WHY?

OUR PLAN

- Integrate protocol analysis into standards process
- Work closely with standards developers as they draft standard
  - **Give feedback as early in the standardization process as possible**
- Discuss any problems we found as they arose
  - **Allowed us to identify quickly which were real problems and which arose from misunderstanding of protocol**
- Recommend fixes when appropriate

GROUP WE WORKED WITH

- Internet Engineering Task Force (IETF)
  - **Mostly volunteer standards group responsible for internet protocol standards**
  - **Made up of different working groups concentrating on standards for different protocols**
- Internet Research Task Force (IRTF)
  - **Research group attached to IETF**
  - **Works on focussed research problems of interest to IETF**
- Secure Multicast Working Group (SMuG) in IRTF
  - **Devoted to protocols associated with secure multicast**

WHAT I'LL TALK ABOUT TODAY

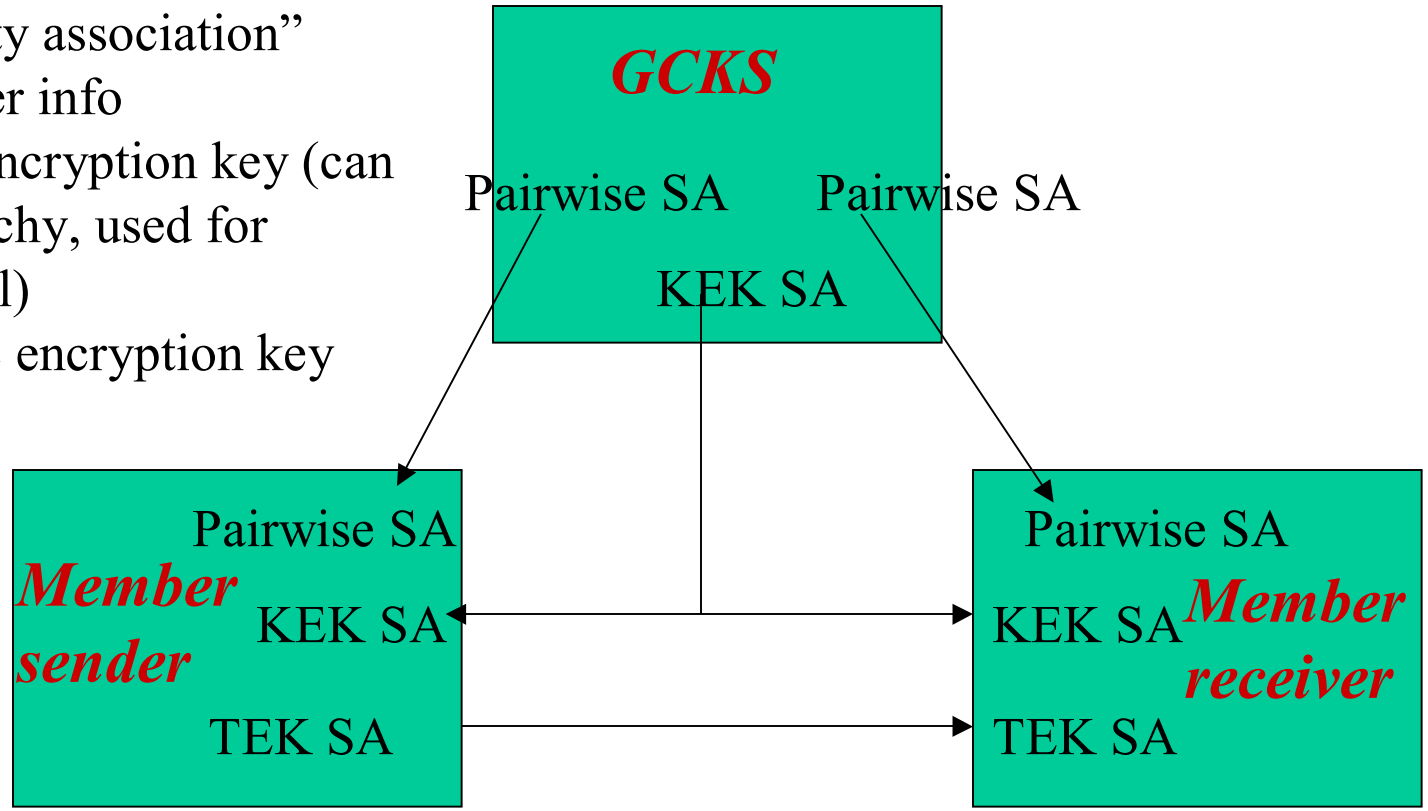
- How we worked with SMuG
- Protocol we worked on, GDOI
- Tool we used, NRL Protocol Analyzer
- Technical challenges we faced
  - **Most of these in formalizing requiremenst**
- The outcome so far

HOW WE WORKED WITH SMUG

- Attended SMuG meetings regularly
  - **Helped to**
    - Get to know SMuG members
    - Learn about background of SMuG protocols
    - Inform SMuG members of our own requirements
- Early on, picked Group Domain of Interpretation (GDOI) protocol as a good candidate
- Used GDOI drafts as basis for formal specifications as they came out
- When found problems or ambiguities, would discuss them with authors
  - **Would often lead to new GDOI drafts**

MULTICAST ARCHITECTURE USED BY GDOI

SA = “security association”
= Key + other info
KEK = key encryption key (can be key hierarchy, used for access control)
TEK = traffic encryption key



GDOI

- Protocol facilitating distribution of group keys by Group Key Distribution Center (GCKS)
  - **Embodies SMuG framework and architecture**
- Based on ISAKMP and IKE
  - **Standards developed for key exchange**
  - **Used to distribute security associations**
- Protocol uses
  - **IKE to distribute pairwise SAs**
  - **Groupkey Pull Protocol initiated by member to distribute KEK SAs to new group member**
  - **Groupkey push Datagram to distribute KEK and TEK SAs to existing group members**

GDOI PROTOCOLS

Groupkey Pull Protocol

```
 Initiator (Member)                      Responder (GCKS)
------------------                      ----------------
HDR*, HASH(1), Ni, ID       -->
                            <--     HDR*, HASH(2), Nr, SA
HDR*, HASH(3) [, KE_I]      -->
   [,CERT] [,POP_I]
                            <--     HDR*, HASH(4), [KE_R,] SEQ,
                                      KD [,CERT] [,POP_R]
```

Hashes are computed as follows:

```
HASH(1) = prf(SKEYID_a, M-ID | Ni | ID)
HASH(2) = prf(SKEYID_a, M-ID | Ni_b | Nr | SA)
HASH(3) = prf(SKEYID_a, M-ID | Ni_b | Nr_b [ | KE_I ] | POP_I)
HASH(4) = prf(SKEYID_a, M-ID | Ni_b | Nr_b [ | KE_R ] | SEQ | KD | POP_R)
```

Groupkey Push Message

```
 Member                                  GCKS or Delegate
------                                  ----------------

                    <---- HDR*, SEQ, SA, KD, [CERT,] SIG
```

"KD" may actually be a data structure such as key hierarchy

THE NRL PROTOCOL ANALYZER

- Formal methods tool for verifying security properties of crypto protocols and finding attacks
- User specifies protocol in terms of communicating state machines communicating by use of a medium controlled by a hostile intruder
- User verifies protocol by
  1. **Proving a set of lemmas to limit size of search space**
  2. **Specifying an insecure state**
  3. **Using NPA to search backwards from that state to see if attack can be found**

NPATRL

- NRL-Protocol-Analyzer-Temporal-Requirements-Language
  - **Pronounced 'N Patrol'**
- Requirements characterized in terms of event statements corresponding to NPA state transitions
- learn events indicate acquisition of information by adversary
- Syntax closely corresponds to NPA language, e.g.,

  receive(A, B, [message], N)
- Add usual logical connectives, e.g., ¬, ∧, =>
- One temporal operator meaning "happens before" 

Example NPATRL Requirement

- If an honest A accepts a key Key for communicating with an honest B, then a server must have generated and sent the key for an honest A and an honest B to use.

accept(user(A, honest), user(B, H), [Key], N?) =>

send(server, (user(A, honest), user(B,H), [Key], N?)

Analysis Using NPA/NPATRL

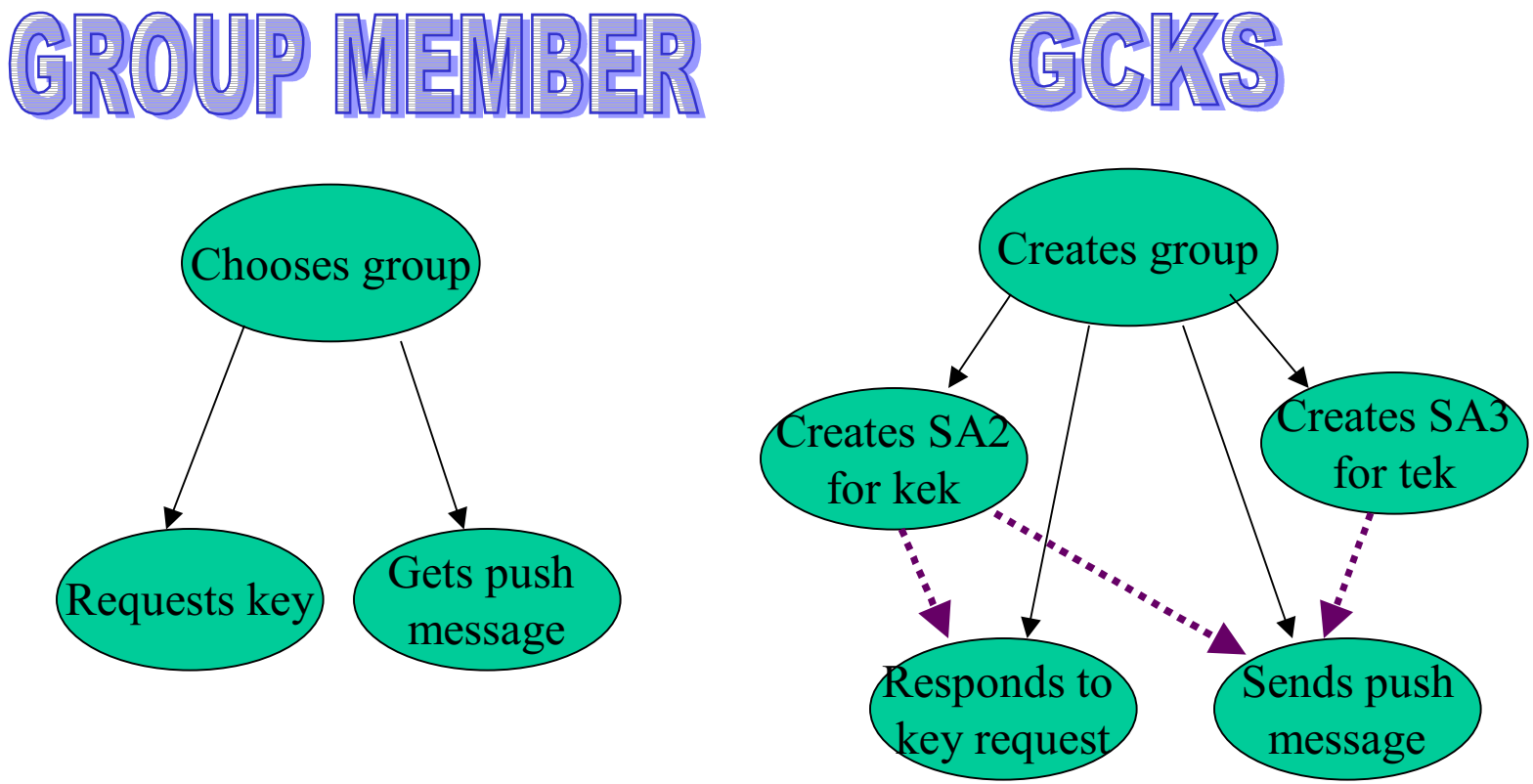
- Map event statements to state transitions in an NRL Protocol Analyzer specification
- Take negation of each NPATRL requirement
  - **Defines a state that should be unreachable iff requirement is satisfied**
- Use NPA to prove goal is unreachable, or

  Use NPA to reach goal, i.e., find attack

NPA SPEC OF GDOI

- Protocol starts with GCKS creating a group and a group key
- At any time after that, a group member may request to join the group by initiating a Groupkey Pull Protocol
  - **GCKS responds by completing protocol**
- At any time after creating a group, GCKS may create new keys and distribute them using Groupkey Push Protocol
- Initial spec took a little under a week to write

STRUCTURE OF SPECIFICATION



HOW SPECIFICATION LIMITED

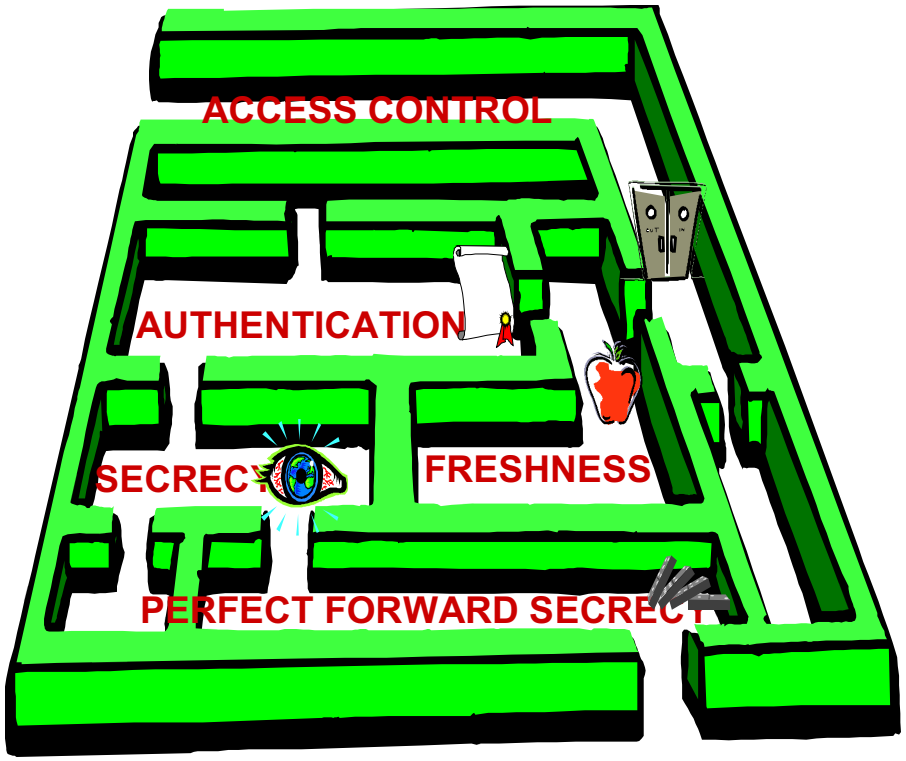
- NPA can't currently handle unbounded data structures such as key hierarchies
  - **Can specify them, but they will send NPA into infinite loop**
  - **Currently investigating appropriate abstractions**
- So --
  - **For the moment did not try to specify key hierarchies, assumed each KEK is a single key**
  - **Assumed that in Groupkey Pull Protocols, one KEK sent**
  - **Assumed three possibilities for Groupkey Push Protocol**
    - One KEK, one TEK, or one KEK and one TEK
- Also, did not include spec of IKE Phase 1

THREE TYPES OF REQUIREMENTS

- Secrecy requirements
  - **Intruder should not learn secrets, except under certain failure conditions**
- Authentication requirements
  - **If A accepts a message as coming from B intended for purpose X, then B should have sent that message to A and intended it for purpose X**
- Freshness requirements
  - **Conditions on recency and/or uniqueness of accepted messages**
- Some models bundle freshness and authentication together

Requirements Challenges

- In pairwise protocols, have notion of a *session*
  - **Secrecy = keys only learned by parties involved in session**
  - **Freshness means key is unique to a session**
- In group protocol session much more open ended
  - **Many keys may be distributed in one session**
  - **Principals may join and leave the group during a session**
    - How should their access to keys be limited?
    - How do secrecy requirements interact with each other?



FRESHNESS ISSUES

- Like secrecy, freshness is more complicated for group protocols
  - **Can no longer tie key to session**
- For GDOI, identified two types of freshness
  - **Recency Freshness**
    - KEK generated most recently (or after a specific time) is the current one
  - **Sequential Freshness**
    - Principal should never accept KEK that is less recent than the one it has
- For Groupkey push protocol, can only ensure that key principal accepts is most recent known to it, not that it is current

RECENCY FRESHNESS FOR PULL PROTOCOL

member_acceptpullkey(N,GCKS,(G,K,PK),N) =>

stealpairwisekey(env,(),(GCKS,M,PK),N?) or

not(⟐ (member_requestkey(M,(GCKS,Nonce,PK),N) and

⟐ gcks_expire(GCKS,(),(G,K),N?)))

if member accepts key K via a pull protocol, then either

1. his pairwise key was stolen, or

2. K should not have expired previously to the request

can't require that key be current at time of receipt, could have expired en route

SEQUENTIAL FRESHNESS FOR PULL PROTOCOL

Member_acceptpullkey(M,GCKS,(G,K,PK),N?) =>

stealpairwisekey(env,(),(GCKS,M,PK),N?) or

not( member_acceptkey(M,GCKS,(G,K1),N?) &

 (gcks_makecurrent(GCKS,(),(G,K1),N?)

&

gcks_makecurrent(GCKS,(),(G,K),N?)))

If member accepts a key K, then either

1. his pairwise key was stolen, or
2. he should not have previously accepted a key that became current later than K

SECRECY REQUIREMENTS FOR GDOI

- Forward access control
  - **Principals should not learn keys distributed after they leave the group**
- Backward access control
  - **Principals should not learn keys that expired before they joined the group**
- Perfect forward secrecy
  - **If pairwise key stolen, only keys distributed with that key after the event should be compromised**
- Other requirements may govern effects of stealing key encryption keys, etc.
- How do these interact with each other?

SOLUTION: DEVELOP CALCULUS OF SECRECY REQUIREMENTS

- Build collection of NPATRL statements of events that can lead to key compromise
  - **Currently restricted to requirements for keks**
  - **Five non-recursive base cases describing**
    - Stealing of pairwise and group keys
    - Group keys sent to dishonest members
  - **Two recursively defined cases addressing generalizations of forward and backward access control**
    - Use NPATRL logic to reduce to non-recursive statements
- Mix and match statements to get requirement of your choice

AN UNEXPECTED DEVELOPMENT

- All requirements could easily be expressed in terms of “fault trees”
  - **Described sequences of events that should or should not lead up to event such as accepting a key, learning a key,etc.**
  - **Can reason about sequences that**
    - Should both happen (AND)
    - One of which should happened (OR)
    - Should not happen (NOT)

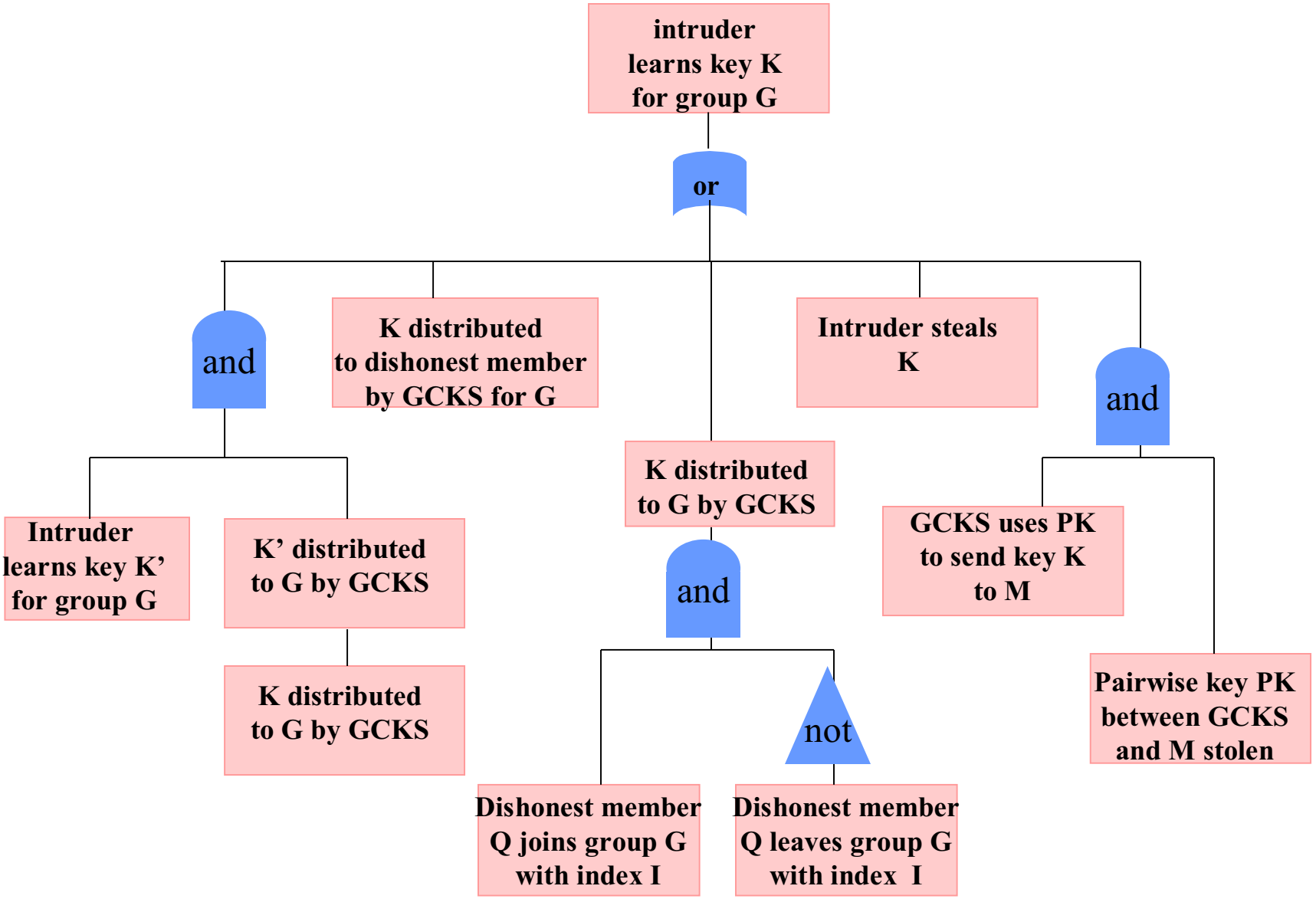


Fig. 4 Forward Access Control Without PFS or Backward Access Control

BUGS FOUND

- Identified bugs in specification, requirements specification and analysis phases
- Bugs found in requirements specification phase:
  - **Improvement to Proof-of-possession option**
    - In old version, principals only signed own nonces
    - Didn't work if pairwise keys compromised
    - Now, principals sign hash of both nonces
  - **Found detail that needed to be added to Groupkey Pull protocol**
    - Did not satisfy sequential freshness unless require that member checks that SEQ number received in last message was greater than SEQ number it may currently hold

RESULTS

- Specified set of protocol requirements for GDOI that could be applied/expanded to other group key distribution protocols
- Identified potential GDOI problems early on, resulting in a better protocol
- Formal analysis credited with speeding up acceptance of GDOI and of the new MSeC (multicast security) working group formed out of SMuG
- Starting to see interest from other parts of IETF in performing or applying formal analyses