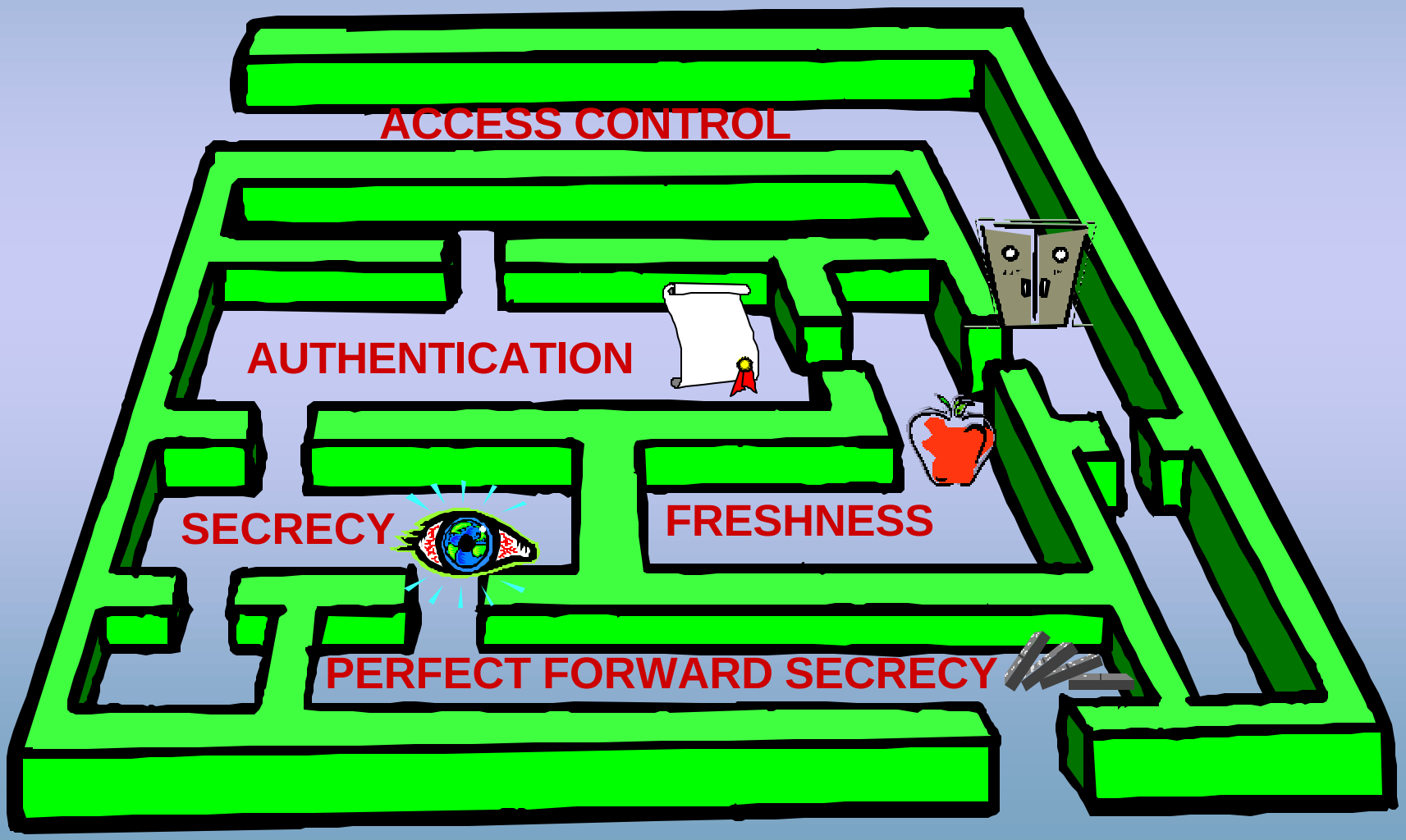
FORMALIZING SECURITY REQUIREMENTS FOR GDOI

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

GOAL OF THIS WORK

- Provide set of formal requirements for GDOI that can provide
  - **Goals for a formal analysis**
  - **Guidance and understanding for writers/implementers/users of GDOI**
- Original idea
  - **Write formal spec for GDOI requirements using NPATRL requirements language for NRL Protocol Analyzer**
  - **Translate into precise natural language specification**
  - **Didn't quite work that way, but has prompted research into**
    - Developing NPATRL into a full-scale logic
    - Developing more readable notation for NPATRL

A MAZE OF REQUIREMENTS



NPATRL

- NRL-Protocol-Analyzer-Temporal-Requirements-Language
  - **Pronounced 'N Patrol'**
- Requirements characterized in terms of event statements
- 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., **Ø**, **Ù**, **=>**
- 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, X), [Key], N?) =>
♦send(server, (user(A, honest), user(B,honest), [Key], N?)

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

Challenges In Developing Requirements for Group Protocols

- In pairwise protocols, have notion of a *session*
  - **Secrecy means keys not learned by parties not involved in the 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?

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**
- Mix and match statements to get requirement of your choice
- Calculus is still under development

GDOI EVENTS OF INTEREST

learn(P,(),(K,G),N)

stealgroupkey(env,(),(G,K),N)

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

gcks_sendpushkey(GCKS,(),(K,G,K1),N)

gcks_admit(GCKS,M,(G,I),N)

gcks_sendpullkey(GCKS,M,(G,I,K,PK),N)

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

gcks_expire(GCKS,(),(G,K),N)

gcks_expel(GCKS,M,(G,I),N)

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

member_requestkey(M,(GCKS,Nonce,PK),N)

member_acceptpushkey(N,GCKS,(G,K,PK),N)

BASE CASES

Group key K is stolen:

BC1(K,N) = stealgroupkey(env,(),(G,K),N?)

Group key sent via a push message while dishonest member in group:

BC2a(K,G) =

(gcks_sendpushkey(GCKS,(),(K,G,K 1),N) &
gcks_admit(GCKS,member(U,dishonest),(G,I),N?)) &
not(gcks_sendpushkey(GCKS,(),(K,G,K 1),N) &
gcks_expel(GCKS,member(U,dishonest),(G,I),N?)))

Group key distributed to dishonest member via a pull protocol

BC2b(K,G) =

gcks_sendpullkey(GCKS,member(U,dishonest),(G,I),N?)

BASE CASES (cont.)

Pairwise key stolen and same key used to distribute group key

BC3a(K,N) =

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

gcks_sendpullkey(GCKS,M,(K,Nonce,PK),N?))

Key distributed with pairwise key after pairwise key stolen

BC3b(K,G) =

(gcks_sendpullkey(GCKS,M,(K,Nonce,PK),N?) &

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

SOME SECRECY SPECS

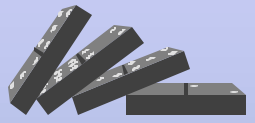
WEAK SECRECY

learn(P,(),(K ,G),N?) =>

◊ BC1(K',G) or ◊ BC2b(K',G) or ◊ BC3a(K',G)

Once a group key is stolen, or a dishonest member joins the group, or a pairwise key that is ever used to distribute a group key is stolen, then intruder learns all keys

STRONG SECRECY (with PFS)

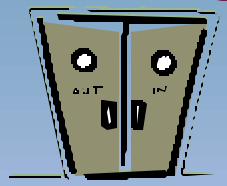


learn(P,(),(K ,G),N?) =>

◊ BC1(K,G) or ◊ BC2a(K,G) or ◊ BC2b(K,G) or ◊ BC3b(K ,G)

The intruder can learn a key K for a group G if and only if that key is stolen, of the key is distributed while a dishonest member is in the group, or distributed using a pairwise key that had already been stolen

FORWARD AND BACKWARD ACCESS CONTROL



- Intruder knowledge vs. member knowledge
  - **FAC and BAC specified in terms of keys learned by members**
  - **NPA model only concerned with intruder knowledge**
  - **Solution: recast requirements in terms of intruder learning key from dishonest member in group**
- How do FAC and BAC fit in with requirements concerning key compromise such as FPS?
  - **Need to pay careful attention to interaction between requirements**

BACKWARDS INFERENCE AND FORWARD ACCESS CONTROL

BI(K,G) =

(learn(P,(),K1,G) & ◇(gcks_makecurrent(GCKS,(),(G,K1),N?) &
gcks_makecurrent(GCKS,(),(G,K),N?)))

A requirement:

learn(P,(),(K,G),N?) =>

◇ BC2a(K,G) or ◇ BC2b(K,G)
or ◇ BI(K,G)

learn(P,(),(K,G),N?)

expand ...

◇[**learn(P,(),(K1,G),N?)** **&** ◇[**gcks_makecurrent(GCKS,(),(G,K),N?)**

expand ... ◇ **gcks_makecurrent(GCKS,(),(G,K),N?)**]]

BC2a(K1,G) **or** **BC2b(K1,G)**

K1 distributed while dishonest member in group

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
  - **Current**
    - In GDOI, there is only one current KEK at a time, and it is the most recent one
  - **Most recent known to a principal**
    - There may be the other keys that are more recent, but this is the most recent one this principal knows about
- In some cases, can only ensure that key principal accepts is most recent known to it, not that it is current

CURRENCY FRESHNESS FOR PULL PROTOCOL

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

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


(member_requestkey(M,(GCKS,Nonce,PK),N) and

not(
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 previous 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

SEQUENTIAL FRESHNESS FOR PUSH DATAGRAM

Member_acceptpushkey(M,GCKS,(G,K,K2),N?) =>

learn(P,(),K2,N?) or

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

◇− (gcks_makecurrent(GCKS,(),(G,K1),N?) &

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

No such thing as currency freshness for push datagram

However, given the intended applications of GDOI, this is probably not a problem

SOME RESULTS OF SPECIFYING REQUIREMENTS

- 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**

CURRENT PLANS AND FUTURE WORK

- Go over requirements with GDOI designers for accuracy and completeness
  - **Already did this for earlier versions**
- Develop NPATRL into a logic that can provide a formal justification for or "mix and match" approach
  - **Use ideas from logic programming**
- Develop "user-friendly" presentation for NPATRL language
  - **Investigating use of fault trees**

CURRENCY FRESHNESS FOR PULL PROTOCOL