Automated Detection of Guessing and Denial of Service Attacks in Security Protocols

Marius Minea

Politehnica University of Timișoara

CMACS seminar, CMU, 18 March 2010
In this talk

**Formalizing attacks** on protocols

*denial of service* by resource exhaustion

*guessing* of low-entropy secrets

**Modeling**

in the AVANTSSAR validation platform

combining rule-based transitions and Horn clauses

**Example attacks**

Joint work with Bogdan Groza  [ISC’09, FC’10, ASIACCS’11]
Part 1: Denial of service by resource exhaustion

Resource exhaustion:
- force victim to consume excessive resources
- with lower costs by attacker

Focus: *computation* resources

Some cryptographic operations are more expensive:
(exponentiation, public-key encryption/decryption, signatures)
Design flaws and solutions

Cost imbalance (usually affects server side)
    solution: cryptographic (client) puzzles, proof-of-work protocols

Lack of authenticity: adversary can steal computational work
    basic principle: include sender identity in message
Classifying DoS attacks

**Excessive use**

no abnormal protocol use
adversary consumes less resources than honest principals
(flooding, spam, ...)

**Malicious use**

adversary brings protocol to abnormal state
protocol goals not completed correctly
Modeling framework

AVANTSSAR

Automated Validation of Trust and Security of Service-Oriented Architectures

(EU FP7 research project)

- AVANTSSAR Specification Language (ASLan)
- three model checkers:
  - CL-Atse (INRIA Nancy): constraint-based
  - OFMC (ETHZ / IBM): on-the-fly
  - SATMC (U Genova): SAT-based
Sample model in ASLan

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : N_A, H(k_{AB}, N_A, N_B, A)$
4. $B \rightarrow A : H(k_{AB}, N_A)$

(MS-CHAP)

$\text{state}_A(A, ID, 1, B, Kab, H, \text{Dummy}_Na, \text{Dummy}_Nb)$

$.i\text{knows}(Nb) = [\text{exists } Na] =>$

$[\text{exists } Na] =>$

$\text{state}_A(A, ID, 2, B, Kab, H, Na, Nb)$

$.i\text{knows}(pair(Na, apply(H, pair(Kab, pair(Na, pair(Nb, A)))))$)

\text{i\text{knows}}: communication mediated by intruder
\text{exists}: generates fresh values
\text{state}: contains participant knowledge
ASLan in a nutshell

\[ \text{state}_A(A, ID, 1, B, Kab, H, \text{Dummy}_Na, \text{Dummy}_Nb) \]
\[ .i\text{knows}(Nb) \]
\[ = [\exists Na] \Rightarrow \]
\[ \text{state}_A(A, ID, 2, B, Kab, H, Na, Nb) \]
\[ .i\text{knows}(\text{pair}(Na, \text{apply}(H, \text{pair}(Kab, \text{pair}(Na, \text{pair}(Nb, A)))))]) \]

**state**: set of ground terms

**transition**:
- removes terms on LHS
- adds terms on RHS

intruder knowledge \textit{iknows} is persistent
Augmenting models with computation cost

1. in *protocol transitions*  

\[ \text{LHS}.\text{cost}(P, C_1) \Rightarrow \text{RHS}.\text{cost}(P, C_2) \]
Augmenting models with computation cost

1. in *protocol transitions*  
   \[ \text{LHS}.\text{cost}(P, C_1) \Rightarrow \text{RHS}.\text{cost}(P, C_2) \]

2. in *intruder deductions*  
   \[ \text{iknows}(X)\text{.iknows}(Y).\text{cost}(i, C_1).\text{sum}(C_1, c_{op}, C_2) \Rightarrow \text{iknows}(\text{op}(X, Y)).\text{cost}(i, C_2) \]
   
   for \( \text{op} \in \{\text{exp, enc, sig}\} \)
Augmenting models with computation cost

1. in *protocol transitions*  

\[ \text{LHS}.\text{cost}(P, C_1) \Rightarrow \text{RHS}.\text{cost}(P, C_2) \]

2. in *intruder deductions*

\[
\text{iknows}(X).\text{iknows}(Y).\text{cost}(i, C_1).\text{sum}(C_1, c_{op}, C_2) \Rightarrow \\
\quad \text{iknows}(\text{op}(X, Y)).\text{cost}(i, C_2)
\]

for \( \text{op} \in \{\text{exp, enc, sig}\} \)

\[
\text{iknows(crypt}(K, X)).\text{iknows}(K).\text{cost}(i, C_1).\text{sum}(C_1, c_{dec}, C_2) \Rightarrow \\
\quad \text{iknows}(X).\text{cost}(i, C_2)
\]

(for decryption)
Meadows: reference cost-based formalization of DoS attacks manual analysis, suggests possibility of automation

**Cost structure: monoid** \{0, cheap, medium, expensive\}

*expensive:* exponentiation (incl. signatures & checking)

*medium:* encryption, decryption

*cheap:* everything else

ASLan implementation: facts declared in initial state

\texttt{sum(cheap, cheap, cheap).}\n
\texttt{sum(cheap, medium, medium).}\n
...\n
\texttt{sum(medium, expensive, expensive).}\n
\texttt{sum(expensive, expensive, expensive).}
Formalizing excessive use

1. session is \textit{initiated by adversary} and
2. \textit{adversary cost less} than honest principal cost

\textbf{attack\_state} \ \texttt{dos\_excessive}(P) :=

\begin{align*}
\text{initiate}(i).\text{cost}(i, C_i).\text{cost}(P, C_P).\text{less}(C_i, C_P)
\end{align*}

Track session cost only if \textit{adversary-initiated} (ID):

\begin{align*}
\mathcal{LHS} . \text{initiate}(i, ID) . \text{cost}(P, C_1) . \text{sum}(C_1, c_{step}, C_2) & \Rightarrow \mathcal{RHS} . \text{cost}(P, C_2) \\
\mathcal{LHS} . \text{initiate}(A, ID) . \text{not}(\text{equal}(i, A)) & \Rightarrow \mathcal{RHS} \ [\text{unchanged}]
\end{align*}

Can also model \textit{distributed DoS}
Formalizing malicious use

In normal use *protocol events match* (injective agreement)

\[ L : S \rightarrow R : M \]

state\(_S\)(S, ID, L, R, ...) ... state\(_R\)(R, ID, L, S, ...) ...

send(S, R, M, L, ID) ⇔ recv(S, R, M, I, ID)

Mismatch is an attack on protocol functionality (authentication)

\[
\text{tampered}(R) := \exists S, M, L, ID. \text{recv}(S, R, M, L, ID).\text{not}(\text{send}(S, R, M, L, ID))
\]

\[
\text{attack\_state } \text{dos\_malicious}(P) := \text{initiate}(i).\text{tampered}(P).\text{cost}(i, C_i).\text{cost}(P, C_P).\text{less}(C_i, C_P)
\]

Adversary may insert value from a previous run

⇒ must track honest agent cost *only in compromised sessions*
Malicious use in multiple sessions

1. track \textit{per-session} cost for normal sessions

\[ \mathcal{LHS}.\text{not}(\text{bad}(ID)).\text{send}(S, P, M, L, ID) \]
\[ \quad \cdot \text{scost}(P, C_{ID}, ID).\text{sum}(C_{ID}, c_{\text{step}}, C'_{ID}). \]
\[ \Rightarrow \mathcal{RHS}.\text{recv}(S, P, M, L, ID).\text{scost}(P, C'_{ID}, ID) \]
Malicious use in multiple sessions

1. track *per-session* cost for normal sessions

\[
\mathcal{LHS}.\text{not(bad(ID))}.\text{send}(S, P, M, L, ID) \\
\quad .\text{scost}(P, C_{ID}, ID) . \text{sum}(C_{ID}, c_{\text{step}}, C'_{ID}). \\
\quad \Rightarrow \mathcal{RHS}.\text{recv}(S, P, M, L, ID) . \text{scost}(P, C'_{ID}, ID)
\]

2. *switch* from per-session to per-principal cost on tampering

\[
\mathcal{LHS}.\text{not(bad(ID))}.\text{not(send}(S, P, M, L, ID)) \\
\quad .\text{cost}(P, C_P) . \text{scost}(P, C_{ID}, ID) . \text{sum}(C_P, C_{ID}, C_1) . \text{sum}(C_1, c_{\text{step}}, C'_P) \\
\quad \Rightarrow \mathcal{RHS}.\text{recv}(S, P, M, L, ID) . \text{bad(ID)} . \text{cost}(P, C'_P)
\]
Malicious use in multiple sessions

1. track *per-session* cost for normal sessions

\[
\mathcal{LHS}.\text{not}(\text{bad}(\text{ID})).\text{send}(S, P, M, L, \text{ID})
\quad .\text{scost}(P, C_{ID}, \text{ID}).\text{sum}(C_{ID}, c_{\text{step}}, C'_{ID}).
\quad \Rightarrow \mathcal{RHS}.\text{recv}(S, P, M, L, \text{ID}).\text{scost}(P, C'_{ID}, \text{ID})
\]

2. *switch* from per-session to per-principal cost on tampering

\[
\mathcal{LHS}.\text{not}(\text{bad}(\text{ID})).\text{not}(\text{send}(S, P, M, L, \text{ID}))
\quad .\text{cost}(P, C_P).\text{scost}(P, C_{ID}, \text{ID}).\text{sum}(C_P, c_{ID}, C_1).\text{sum}(C_1, c_{\text{step}}, C'_P)
\quad \Rightarrow \mathcal{RHS}.\text{recv}(S, P, M, L, \text{ID}).\text{bad}(\text{ID}).\text{cost}(P, C'_P)
\]

3. track *per-principal* cost for tampered sessions

\[
\mathcal{LHS}.\text{bad}(\text{ID}).\text{cost}(P, C_P).\text{sum}(C_P, c_{\text{step}}, C'_P)
\quad \Rightarrow \mathcal{RHS}.\text{bad}(\text{ID}).\text{cost}(P, C'_P)
\]
Undetectable resource exhaustion

Excessive/malicious executions especially *dangerous if undetected* (cannot be distinguished from normal executions)
Modeled by checking that all instances of $P$ complete successfully

$$
\text{dos\_exc\_nd}(P) := \text{initiate}(i).\text{active\_cnt}(P, 0).
\quad \text{cost}(i, C_i).\text{cost}(P, C_P).\text{less}(C_i, C_P)
$$

$$
\text{dos\_mal\_nd}(P) := \text{tampered}(P).\text{active\_cnt}(P, 0).
\quad \text{cost}(i, C_i).\text{cost}(P, C_P).\text{less}(C_i, C_P)
$$

Can also characterize attacks undetectable by *any* participant
Case studies: Station-to-station protocol

1. $A \rightarrow B : \alpha^x$
2. $B \rightarrow A : \alpha^y, \text{Cert}_B, E_k(\text{sig}_B(\alpha^y, \alpha^x))$
3. $A \rightarrow B : \text{Cert}_A, E_k(\text{sig}_A(\alpha^x, \alpha^y))$

Reproduced Lowe’s attack: Adv impersonates $B$ to $A$:
1. $A \rightarrow \text{Adv}(B) : \alpha^x$
1'. $\text{Adv} \rightarrow B : \alpha^x$
2'. $B \rightarrow \text{Adv} : \alpha^y, \text{Cert}_B, E_k(\text{sig}_B(\alpha^y, \alpha^x))$
2. $\text{Adv}(B) \rightarrow A: \alpha^y, \text{Cert}_B, E_k(\text{sig}_B(\alpha^y, \alpha^x))$
3. $A \rightarrow \text{Adv}(B): \text{Cert}_A, E_k(\text{sig}_A(\alpha^x, \alpha^y))$

excessive use: $\text{Adv}$ initiates attack on $B$
malicious use: $A$ receives value from $B$’s session with $\text{Adv}$
[Smith et al. '06] strengthened from [Aiello et al. '04]

1. $I \rightarrow R : N'_I, g^i, ID'_R$
2. $R \rightarrow I : N'_I, N_R, g^r, grpinfo_R, ID_R, S_R[g^r, grpinfo_R], token, k$
3. $I \rightarrow R : N_I, N_R, g^i, g^r, token,$
   \[\{ID_I, sa, S_I[N'_I, N_R, g^i, g^r, ID_R, sa]\}^{K_e}_{K_a}, sol\]
4. $R \rightarrow I : \{S_R[N'_I, N_R, g^i, g^r, ID_I, sa], sa'^{'}\}^{K_e}_{K_a}, sol$
[Smith et al. '06] strengthened from [Aiello et al. '04]

1. $I \rightarrow R : N_I', g^i, ID_R'$
2. $R \rightarrow I : N_I', N_R, g^r, grpinfo_R, ID_R, S_R[g^r, grpinfo_R], \text{token}, k$
3. $I \rightarrow R : N_I, N_R, g^i, g^r, \text{token},$
   $$\{ID_I, sa, S_I[N_I', N_R, g^i, g^r, ID_R, sa]\}^{K_e}_{K_a}, \text{sol}$$
4. $R \rightarrow I : \{S_R[N_I', N_R, g^i, g^r, ID_I, sa], sa'\}^{K_e}_{K_a}, \text{sol}$

Analysis: malicious use exploiting the \textit{initiator}

\textbf{A} initiates session 1 with \textbf{Adv} (responder)

\textbf{Adv} \textit{initiates} session 2 with \textbf{B}

forwards \textbf{B}'s puzzle \textit{token} (step 2) to \textbf{A} in session 1

reuses \textbf{A}'s solution \textit{sol} (step 3) in session 2

\textbf{Flaw}: puzzle \textit{token} is \textbf{not bound} to identity of requester \textit{I}

(same for difficulty level \textit{k})
Part 2: Guessing attacks

*Important*

weak passwords are common
vulnerable protocols still in use

*Realistic*, if secrets have low entropy

*Few tools* can detect guessing attacks:
Lowe ’02, Corin et al. ’04, Blanchet-Abadi-Fournet ’08
(only offline attacks)
How to guess?

Two steps:
- guess a value for the secret $s$
- compute a \textit{verifier} value that confirms the guess

Low entropy $\Rightarrow$ can repeat over all values
How to guess?

Two steps:
- guess a value for the secret $s$
- compute a verifier value that confirms the guess

Low entropy $\Rightarrow$ can repeat over all values

Example guessing conditions [Lowe, 2002]

$Adv$ knows $v, E_s(v)$: guess $s$, and verify known value $v$
How to guess?

Two steps:
- guess a value for the secret $s$
- compute a verifier value that confirms the guess

Low entropy $\Rightarrow$ can repeat over all values

Example guessing conditions [Lowe, 2002]

$Adv$ knows $v, E_s(v)$: guess $s$, and verify known value $v$
$Adv$ knows $E_s(v.v)$: guess $s$, decrypt, verify equal parts
How to guess?

Two steps:
- guess a value for the secret \( s \)
- compute a \textit{verifier} value that confirms the guess

Low entropy \( \Rightarrow \) can repeat over all values

Example guessing conditions [Lowe, 2002]

\( \text{Adv} \) knows \( v, E_s(v) \): guess \( s \), and verify known value \( v \)

\( \text{Adv} \) knows \( E_s(v \cdot v) \): guess \( s \), decrypt, verify equal parts

\( \text{Adv} \) knows \( E_s(s) \): guess \( s \), and encrypt, verify result or decrypt, verify result is \( s \)
Goals for guessing theory and implementation

Detect both *on-line* and *off-line* attacks

Distinguish *blockable / non-blockable* on-line attacks

Deal with verifiers matching *more than one* secret

Allow chaining guesses of *multiple secrets*
From algebraic to symbolic properties

We can guess $s$ from $f(s)$ if $f$ is injective.

**Generalize:** consider pseudo-random one-way functions

$f(s, x)$ is *distinguishing* in $s$ (probabilistically)

if polynomially many $f(s, x_i)$ can distinguish any $s' \neq s$. 
From algebraic to symbolic properties

We can guess $s$ from $f(s)$ if $f$ is injective.

**Generalize:** consider pseudo-random one-way functions

$f(s, x)$ is *distinguishing* in $s$ (probabilistically)
if polynomially many $f(s, x_i)$ can distinguish any $s' \neq s$.

**Quantify:** $f(s, x)$ is *strongly distinguishing* in $s$ after $q$ queries
if $q$ values $f(s, x_i)$ can on average distinguish any $s' \neq s$.\[ \]
We can guess $s$ from $f(s)$ if $f$ is injective.

*Generalize:* consider pseudo-random one-way functions $f(s, x)$ is *distinguishing* in $s$ (probabilistically) if polynomially many $f(s, x_i)$ can distinguish any $s' \neq s$.

*Quantify:* $f(s, x)$ is *strongly distinguishing* in $s$ after $q$ queries if $q$ values $f(s, x_i)$ can on average distinguish any $s' \neq s$.

Two main guessing cases:
- know image of a *one-way function* on the secret
- know image of *trap-door one-way function* on the secret
Oracles and the adversary

**Oracle**: abstract view of a computation (function)

- **off-line**, constructing terms directly
- **on-line**, employing an honest principal
**Oracle**: abstract view of a computation (function)

- **off-line**, constructing terms directly
- **on-line**, employing an honest principal

An adversary:

- **observes** the oracle for a secret $s$
  
  if he knows a term that contains the secret $s$

  $$\text{ihears} (\text{Term}) \land \text{part}(s, \text{Term}) \Rightarrow \text{observes} (O_s^{\text{Term}}(\cdot))$$
Oracles and the adversary

**Oracle:** abstract view of a computation (function)
- **off-line,** constructing terms directly
- **on-line,** employing an honest principal

An adversary:

- **observes** the oracle for a secret $s$
  
  if he knows a term that contains the secret $s$
  
  $i\text{hears}(\text{Term}) \land \text{part}(s, \text{Term}) \Rightarrow \text{observes}(O_s^{\text{Term}}(\cdot))$

- **controls** the oracle for a secret $s$
  
  if he can generate terms with fresh replacements of secret $s$
  
  $i\text{hears}(\text{Term}(s)) \land i\text{knows}(s') \land i\text{knows}(\text{Term}(s')) \Rightarrow \text{controls}(O_s^{\text{Term}}(\cdot))$
What guesses can be verified? (1)

- an *already known* term:

  \[
  \text{vrfy}(\text{Term}) :\neg \text{iknows}(\text{Term})
  \]

- a *signature*, if the public key and the message are known:

  \[
  \text{vrfy}(\text{sign}(\text{inv}(\text{PK}), \text{Term})) :\neg \text{iknows}(\text{PK}), \text{iknows}(\text{Term})
  \]

- a term under a *one-way function* application:

  \[
  \begin{align*}
  \text{vrfy}(\text{STerm}) & :\neg \text{iknows}(h), \text{iknows}(\text{apply}(h, \text{Term})), \\
  & \quad \text{part}(\text{STerm}, \text{Term}), \text{controls}(\text{STerm}, \text{Term})
  \end{align*}
  \]
What guesses can be verified? (2)

- a ciphertext, if key is known (or decryption oracle controlled) and part of plaintext verifiable:

  \[
  \text{vrfy}(\text{scrypt}(K, \text{Term})) :\!- \!\text{iknows}(K), \\
  \text{splitknow}(\text{Term}, T_1, T_2), \text{vrfy}(T_2)
  \]

- a key, if ciphertext known and part of plaintext verifiable:

  \[
  \text{vrfy}(K) :\!- \!\text{ihears}(\text{scrypt}(K, \text{Term})), \\
  \text{splitknow}(\text{Term}, T_1, T_2), \text{vrfy}(T_2)
  \]

where \text{splitknow}(\text{Term}, T_1, T_2) splits \text{Term} and asserts \text{iknows}(T_1)
e.g., from \text{m.h}(m) with \text{iknows}(m) can verify \text{h}(m)
Modeling guessing rules

Protocol execution:

Intruder deductions as transitions: inefficient (state explosion)
Changing model checker built-in deductions: impractical

⇒ ASLan provides \{ transition rules, Horn clauses \}
are **re-evaluated after each protocol step** (transitive closure)

facts deduced from Horn clauses are non-persistent

```
hc part_left(T0, T1, T2, T3) :=
  split(pair(T0,T1), T2, pair(T3,T1)) :- split(T0, T2, T3)
```

```
hc part_right(T0, T1, T2, T3) :=
  split(pair(T0,T1), pair(T0,T2), T3) :- split(T1, T2, T3)
```

- natural modeling of recursive facts (e.g., term processing)
- multiple (intruder) deductions applied after each protocol step
- orders of magnitude more efficient than using transitions
Resulting guessing rules

- from one-way function images
  (allows guessing from $h(s)$, $m.h(s.m)$ etc.)

\[
\text{guess}(s) : \neg \text{observes}(O_s^f(\cdot)) , \text{controls}(O_s^f(\cdot))
\]

- by inverting one-way trapdoor functions
  (allows guessing from $\{m.m\}_s$, $m.\{h(m)\}_s$ etc.)

\[
\text{guess}(s) : \neg \text{observes}(O_s^{\{T\}_K}) , \text{controls}(O_s^{\{T\}_{K-1}}), \\
\text{splitknow}(T, T_1, T_2) , \text{vrfy}(T_2)
\]
Flavors of guessing

**off-line**: terms constructed directly by intruder

**on-line**: uses computations of honest protocol principals

(intruder *controls* computation oracles with arbitrary inputs)

**undetectable**

all participants terminate (no abnormal protocol activity)

modeled by checking that all instances reach final state

**multiple** secrets

a guessed secret becomes known to the intruder

allows chaining of guessing rules
Example 1: Norwegian ATM

Real case, described by Hole et al. (IEEE S&P 2007)

2001: money withdrawn \textit{within 1 hour} of stealing card
Did the thief have to know the PIN?

Card setup:

PIN and card-specific data DES-encrypted with \textit{unique bank key}

\[ \left\lfloor \text{DES}_{BK}(PIN.CV) \right\rfloor_{16} \]
Example 1: Norwegian ATM

Real case, described by Hole et al. (IEEE S&P 2007)

2001: money withdrawn \textit{within 1 hour} of stealing card

Did the thief have to know the PIN?

Card setup:

PIN and card-specific data DES-encrypted with \textit{unique bank key}

\text{card stores 56-bit result cut to 16 bits: } \left\lceil \text{DES}_{BK}(\text{PIN.CV}) \right\rceil_{16}

Suggested attack [Hole et al., 2007]: break bank key

DES search, verifier is a legitimate card owned by adversary

But: verifier only has 16 bits \( \Rightarrow 2^{56-16} = 2^{40} \) bank keys match

Insight: each honest card reduces key search space by 16 bits

\( \Rightarrow \left\lceil \frac{56}{16} \right\rceil = 4 \) cards suffice
Model and new attacks

New attack, if Adv can do unlimited PIN changes on own card

PIN Change Procedure:
1. $User \rightarrow ATM : [DES_{BK}(PIN_{old})]_{16}, PIN_{old}, PIN_{new}$
2. $ATM \rightarrow User : [DES_{BK}(PIN_{new})]_{16}$

simplified case: card encrypts just $PIN \Rightarrow$ card-independent
   $\Rightarrow$ observes and controls $f(PIN) \Rightarrow$ can guess $PIN$ directly

real case: card encrypts $PIN$ and card-specific value
   $\Rightarrow$ controls $f(BK, PIN)$ in argument $PIN$
1. use PIN-change procedure to guess $BK$ (average 4 PINs)
2. when $BK$ found, can trivially guess $PIN$
Example 2: MS-CHAP

Known insecure protocol from Microsoft, still in use

1. $A \rightarrow B : A$
   
   $i \rightarrow (a,1): N_b$

2. $B \rightarrow A : N_B$
   
   $i \rightarrow (a,1): N_b$

3. $A \rightarrow B : N_A, H(kab, N_A, N_B, A)$
   
   $i \rightarrow (b,1): N_a.h(kab.Na(3).Nb(2).a)$

4. $B \rightarrow A : H(kab, N_A)$
   
   $i \rightarrow (a,1): h(kab.Na(3))$

Man-in-the-middle attack: intruder observes $N_A$ and $H(k_{AB}, N_A)$

$\Rightarrow$ can guess $k_{AB}$

Similar guessing attack on NTLM protocol (v2-Session).
Example 3: Lomas et al.’89

Lowe’s replay attack: replace timestamp with constant 0

New typing attack, replacing the timestamp with a nonce

1. $A \rightarrow S : \{A, B, Na1, Na2, Ca, \{Ta\}_{pwdA}\}_{pks}$
2. $S \rightarrow B : A, B$
3. $B \rightarrow S : \{B, A, Nb1, Nb2, Cb, \{Tb\}_{pwdB}\}_{pks}$
4. $S \rightarrow A : \{Na1, k \oplus Na2\}_{pwdA}$
5–8. [... not relevant here ...]

$1'$. $Adv(A) \rightarrow S : \{A, B, Na1', Na2', Ca', \{Na1, k \oplus Na2\}_{pwdA}\}_{pks}$
$2'$. $S \rightarrow B : A, B$
$3'$. $B \rightarrow S : \{B, A, Nb1', Nb2', Cb', \{Tb\}'_{pwdB}\}_{pks}$
$4'$. $S \rightarrow Adv(A) : \{Na1', k' \oplus Na2'\}_{pwdA}$

... 

From last term, knowing $Na1'$, $pwdA$ can be guessed (and then $k'$)
Conclusions

Automated detection for two types of attacks (guessing, DoS) less represented in protocol verification toolsets.

Implemented by augmenting protocol models with transition costs / guessing rules (efficient as Horn clauses).

Flexible, no changes to model checker backends.

Insights for attack classification:
- off-line vs. on-line guessing attacks
- excessive vs. malicious use in DoS attacks
- attacks undetectable by protocol participants

Automated Validation of Trust and Security of Service-Oriented Architectures, FP7-ICT-2007-1 project 216471