# 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

(EU FP7 research project)

Automated Validation of Trust and Security of Service-Oriented Architectures

- 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)

```
state_A(A,ID,1,B,Kab,H,
                 Dummy_Na,Dummy_Nb)
 .iknows(Nb)
 =[exists Na]=>
state_A(A,ID,2,B,Kab,H,Na,Nb)
 .iknows(pair(Na,
           apply(H,pair(Kab,
             pair(Na,pair(Nb,A))))))
```

`iknows`: communication mediated by intruder

`exists`: generates fresh values

`state`: contains participant knowledge

# ASLan in a nutshell

```
state_A(A,ID,1,B,Kab,H,Dummy_Na,Dummy_Nb)
 .iknows(Nb)
 =[exists Na]=>
state_A(A,ID,2,B,Kab,H,Na,Nb)
 .iknows(pair(Na,apply(H,pair(Kab,pair(Na,pair(Nb,A))))))
```

- state: set of ground terms
- transition:
  - removes terms on LHS
  - adds terms on RHS
- intruder knowledge `iknows` is persistent

# Augmenting models with computation cost

1. in *protocol transitions* [more to follow]

$$\mathcal{LHS}.\mathtt{cost}(\mathtt{P}, \mathtt{C}_1) \Rightarrow \mathcal{RHS}.\mathtt{cost}(\mathtt{P}, \mathtt{C}_2)$$

# Augmenting models with computation cost

1. in *protocol transitions* [more to follow]

$$\mathcal{LHS}.\mathtt{cost}(\mathtt{P},\mathtt{C_1}) \Rightarrow \mathcal{RHS}.\mathtt{cost}(\mathtt{P},\mathtt{C_2})$$

2. in *intruder deductions*

$$\mathtt{iknows}(\mathtt{X}).\mathtt{iknows}(\mathtt{Y}).\mathtt{cost}(\mathtt{i},\mathtt{C_1}).\mathtt{sum}(\mathtt{C_1},\mathtt{c_{op}},\mathtt{C_2}) \Rightarrow \mathtt{iknows}(\mathtt{op}(\mathtt{X},\mathtt{Y})).\mathtt{cost}(\mathtt{i},\mathtt{C_2})$$

for $\mathtt{op} \in \{\mathtt{exp}, \mathtt{enc}, \mathtt{sig}\}$

# Augmenting models with computation cost

1. in *protocol transitions* [more to follow]

$$\mathcal{LHS}.\mathtt{cost}(\mathtt{P}, \mathtt{C}_1) \Rightarrow \mathcal{RHS}.\mathtt{cost}(\mathtt{P}, \mathtt{C}_2)$$

2. in *intruder deductions*

$$\mathtt{iknows}(\mathtt{X}).\mathtt{iknows}(\mathtt{Y}).\mathtt{cost}(\mathtt{i}, \mathtt{C}_1).\mathtt{sum}(\mathtt{C}_1, \mathtt{c}_{\mathtt{op}}, \mathtt{C}_2) \Rightarrow \mathtt{iknows}(\mathtt{op}(\mathtt{X}, \mathtt{Y})).\mathtt{cost}(\mathtt{i}, \mathtt{C}_2)$$

for op $\in \{\mathtt{exp}, \mathtt{enc}, \mathtt{sig}\}$

$$\mathtt{iknows}(\mathtt{crypt}(\mathtt{K}, \mathtt{X})).\mathtt{iknows}(\mathtt{K}).\mathtt{cost}(\mathtt{i}, \mathtt{C}_1).\mathtt{sum}(\mathtt{C}_1, \mathtt{c}_{\mathtt{dec}}, \mathtt{C}_2) \Rightarrow \mathtt{iknows}(\mathtt{X}).\mathtt{cost}(\mathtt{i}, \mathtt{C}_2)$$

(for decryption)

# Cost model [Meadows '01]

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

```
sum(cheap, cheap, cheap).
sum(cheap, medium, medium).
...
sum(medium, expensive, expensive).
sum(expensive, expensive, expensive)
```

# Formalizing excessive use

1. session is *initiated by adversary* and
2. *adversary cost less* than honest principal cost

$$\texttt{attack\_state}\ \ \texttt{dos\_excessive(P)} := \texttt{initiate(i).cost(i}, \texttt{C}_\texttt{i}\texttt{).cost(P}, \texttt{C}_\texttt{P}\texttt{).less(C}_\texttt{i}, \texttt{C}_\texttt{P}\texttt{)}$$

Track session cost only if *adversary-initiated* (ID):

$$\mathcal{LHS}.\texttt{initiate(i, ID).cost(P}, \texttt{C}_1\texttt{).sum(C}_1, \texttt{c}_{\texttt{step}}, \texttt{C}_2\texttt{)} \Rightarrow \mathcal{RHS}.\texttt{cost(P}, \texttt{C}_2\texttt{)}$$

$$\mathcal{LHS}.\texttt{initiate(A, ID).not(equal(i, A))} \Rightarrow \mathcal{RHS} \qquad [\textit{unchanged}]$$

Can also model *distributed DoS*

# Formalizing malicious use

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

$$L: \quad S \rightarrow R : M$$

$$\texttt{state\_S}(\texttt{S}, \texttt{ID}, \texttt{L}, \texttt{R}, \ldots) \;\ldots \qquad \texttt{state\_R}(\texttt{R}, \texttt{ID}, \texttt{L}, \texttt{S}, \ldots) \;\ldots$$

$$\texttt{send}(\texttt{S}, \texttt{R}, \texttt{M}, \texttt{L}, \texttt{ID}) \quad \Longleftrightarrow \quad \texttt{recv}(\texttt{S}, \texttt{R}, \texttt{M}, \texttt{I}, \texttt{ID})$$

Mismatch is an attack on protocol functionality (authentication)

$$\texttt{tampered}(\texttt{R}) :=$$

$$\exists\, \texttt{S}, \texttt{M}, \texttt{L}, \texttt{ID} \,.\, \texttt{recv}(\texttt{S}, \texttt{R}, \texttt{M}, \texttt{L}, \texttt{ID}).\texttt{not}(\texttt{send}(\texttt{S}, \texttt{R}, \texttt{M}, \texttt{L}, \texttt{ID}))$$

$$\texttt{attack\_state} \quad \texttt{dos\_malicious}(\texttt{P}) :=$$

$$\texttt{initiate}(\texttt{i}).\texttt{tampered}(\texttt{P}).\texttt{cost}(\texttt{i}, \texttt{C}_\texttt{i}).\texttt{cost}(\texttt{P}, \texttt{C}_\texttt{P}).\texttt{less}(\texttt{C}_\texttt{i}, \texttt{C}_\texttt{P})$$

Adversary may insert value from a previous run

$\Rightarrow$ must track honest agent cost *only in compromised sessions*

# Malicious use in multiple sessions

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

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

# Malicious use in multiple sessions

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

$$\mathcal{LHS}.\mathtt{not}(\mathtt{bad}(\mathtt{ID})).\mathtt{send}(\mathtt{S},\mathtt{P},\mathtt{M},\mathtt{L},\mathtt{ID})$$
$$.\mathtt{scost}(\mathtt{P},\mathtt{C}_{\mathtt{ID}},\mathtt{ID}).\mathtt{sum}(\mathtt{C}_{\mathtt{ID}},\mathtt{c}_{\mathtt{step}},\mathtt{C}'_{\mathtt{ID}}).$$
$$\Rightarrow \mathcal{RHS}.\mathtt{recv}(\mathtt{S},\mathtt{P},\mathtt{M},\mathtt{L},\mathtt{ID}).\mathtt{scost}(\mathtt{P},\mathtt{C}'_{\mathtt{ID}},\mathtt{ID})$$

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

$$\mathcal{LHS}.\mathtt{not}(\mathtt{bad}(\mathtt{ID})).\mathtt{not}(\mathtt{send}(\mathtt{S},\mathtt{P},\mathtt{M},\mathtt{L},\mathtt{ID}))$$
$$.\mathtt{cost}(\mathtt{P},\mathtt{C}_{\mathtt{P}}).\mathtt{scost}(\mathtt{P},\mathtt{C}_{\mathtt{ID}},\mathtt{ID}).\mathtt{sum}(\mathtt{C}_{\mathtt{P}},\mathtt{c}_{\mathtt{ID}},\mathtt{C}_1).\mathtt{sum}(\mathtt{C}_1,\mathtt{c}_{\mathtt{step}},\mathtt{C}'_{\mathtt{P}})$$
$$\Rightarrow \mathcal{RHS}.\mathtt{recv}(\mathtt{S},\mathtt{P},\mathtt{M},\mathtt{L},\mathtt{ID}).\mathtt{bad}(\mathtt{ID}).\mathtt{cost}(\mathtt{P},\mathtt{C}'_{\mathtt{P}})$$

# Malicious use in multiple sessions

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

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

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

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

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

$$\mathcal{LHS}.\mathtt{bad}(\mathtt{ID}).\mathtt{cost}(\mathtt{P},\mathtt{C_P}).\mathtt{sum}(\mathtt{C_P},\mathtt{c_{step}},\mathtt{C'_P})$$
$$\Rightarrow \mathcal{RHS}.\mathtt{bad}(\mathtt{ID}).\mathtt{cost}(\mathtt{P},\mathtt{C'_P})$$

# 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

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

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

Can also characterize attacks undetectable by *any* participant

# Case studies: Station-to-station protocol

1. $A \rightarrow B : \alpha^x$
2. $B \rightarrow A : \alpha^y, Cert_B, E_k(sig_B(\alpha^y, \alpha^x))$
3. $A \rightarrow B : Cert_A, E_k(sig_A(\alpha^x, \alpha^y))$

Reproduced Lowe's attack: *Adv* impersonates *B* to *A*:

1. $A \rightarrow Adv(B):$ $\alpha^x$
1′. $Adv \rightarrow B:$ $\alpha^x$
2′. $B \rightarrow Adv:$ $\alpha^y, Cert_B, E_k(sig_B(\alpha^y, \alpha^x))$
2. $Adv(B) \rightarrow A:$ $\alpha^y, Cert_B, E_k(sig_B(\alpha^y, \alpha^x))$
3. $A \rightarrow Adv(B):$ $Cert_A, E_k(sig_A(\alpha^x, \alpha^y))$

excessive use: *Adv* initiates attack on *B*

malicious use: *A* receives value from $B'$s session with *Adv*

# Just Fast Keying with client puzzles

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

# Just Fast Keying with client puzzles

[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. $R \rightarrow I : \{S_R[N'_I, N_R, g^i, g^r, ID_I, sa], sa'\}^{K_e}_{K_a}, sol$

Analysis: malicious use exploiting the *initiator*

*A* initiates session 1 with *Adv* (responder)

*Adv* *initiates* session 2 with *B*

forwards *B*'s puzzle token (step 2) to *A* in session 1

reuses *A*'s solution sol (step 3) in session 2

Flaw: puzzle *token* is not bound to identity of requester *I*
(same for difficulty level *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 *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 *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

$Adv$ 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 $s' \neq s$.

# 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 $s' \neq s$.

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

# 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 $s$

  $$ihears(Term) \wedge part(s, Term) \Rightarrow observes(O_s^{Term}(\cdot))$$

# 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 $s$
  $$ihears(Term) \wedge part(s, Term) \Rightarrow observes(O_s^{Term}(\cdot))$$

- *controls* the oracle for a secret $s$
  if he can generate terms with fresh replacements of secret $s$

$$ihears(Term(s)) \wedge iknows(s') \wedge iknows(Term(s')) \Rightarrow controls(O_s^{Term}(\cdot))$$

# What guesses can be verified ? (1)

- an *already known* term:

```
vrfy(Term) :- iknows(Term)
```

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

```
vrfy(sign(inv(PK),Term)) :- iknows(PK) , iknows(Term)
```

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

```
vrfy(STerm) :- iknows(h) , iknows(apply(h,Term)) ,
               part(STerm,Term) , controls(STerm,Term)
```

# What guesses can be verified ? (2)

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

```
vrfy(scrypt(K, Term)) :- iknows(K) ,
                         splitknow(Term, T1, T2) , vrfy(T2)
```

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

```
vrfy(K) :- ihears(scrypt(K, Term)) ,
           splitknow(Term, T1, T2) , vrfy(T2)
```

where `splitknow(Term, T1, T2)` splits `Term` and asserts `iknows(T1)`
e.g., from $m.h(m)$ with $iknows(m)$ can verify $h(m)$

# Modeling guessing rules

Protocol execution: protocol step / intruder deductions

Intruder deductions as transitions: inefficient (state explosion)

Changing model checker built-in deductions: impractical

$\Rightarrow$ ASLan provides
- transition rules
- *Horn clauses*

# Modeling with 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.)

$$guess(s) \;\text{:-}\; observes(O_s^f(\cdot))\;,\; controls(O_s^f(\cdot))$$

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

$$guess(s) \;\text{:-}\; observes(O_s^{\{T\}_K})\;,\; controls(O_s^{\{T\}_{K^{-1}}})\;,\; splitknow(T, T_1, T_2)\;,\; vrfy(T_2)$$

# 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 *within 1 hour* of stealing card
Did the thief have to know the PIN ?

Card setup:

PIN and card-specific data DES-encrypted with *unique bank key*
card stores 56-bit result cut to 16 bits: $\lfloor DES_{BK}(PIN.CV) \rfloor_{16}$

# Example 1: Norwegian ATM

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

2001: money withdrawn *within 1 hour* of stealing card
Did the thief have to know the PIN ?

Card setup:

PIN and card-specific data DES-encrypted with *unique bank key*
card stores 56-bit result cut to 16 bits: $\lfloor DES_{BK}(PIN.CV) \rfloor_{16}$

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
$\Rightarrow \lceil 56/16 \rceil =$ *4 cards* suffice

# Model and new attacks

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

*PIN Change Procedure:*

1.$User \rightarrow ATM : \lfloor DES_{BK}(PIN_{old}) \rfloor_{16}, PIN_{old}, PIN_{new}$
2.$ATM \rightarrow User : \lfloor DES_{BK}(PIN_{new}) \rfloor_{16}$

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. $A \to B : A$
2. $B \to A : N_B$
3. $A \to B : N_A, H(kab, N_A, N_B, A)$
4. $B \to A : H(kab, N_A)$

(a,1) → i: a
i → (b,1): a
(b,1) → i: Nb(2)
i → (a,1): Nb(2)
(a,1) → i: Na(3).h(kab.Na(3).Nb(2).a)
i → (b,1): Na(3).h(kab.Na(3).Nb(2).a)
(b,1) → i: h(kab.Na(3))
i → (a,1): h(kab.Na(3))
i → (i,1): h(kab_repl.Na(3))
i → (i,1): kab.dummy

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. $A \to S : \{A, B, Na1, Na2, Ca, \{Ta\}_{pwdA}\}_{pks}$
2. $S \to B : A, B$
3. $B \to S : \{B, A, Nb1, Nb2, Cb, \{Tb\}_{pwdB}\}_{pks}$
4. $S \to A : \{Na1, k \oplus Na2\}_{pwdA}$

5–8. [... not relevant here ...]

---

$1'$. $Adv(A) \to S : \{A, B, Na1', Na2', Ca', \{Na1, k \oplus Na2\}_{pwdA}\}_{pks}$

$2'$. $S \to B : A, B$

$3'$. $B \to S : \{B, A, Nb1', Nb2', Cb', \{Tb'\}_{pwdB}\}_{pks}$

$4'$. $S \to Adv(A) : \{Na1', k' \oplus Na2'\}_{pwdA}$

...

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)

*Flexibile*, 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

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