AGVI
Automatic Generation, Verification, and Implementation of security protocols

By: Dawn Song, Adrian Perrig, and Doantam Phan.
In: 13th Conference on Computer Aided Verification (CAV), 2001

Presenter:
AliReza Namvar

Motivation

Rapid design & deployment of security protocol
- eCommerce systems

Security protocol are difficult to:
- Design
- Verify the correctness
- Implement correctly

Introduction

Optimal security protocol:
- System aspects:
  - Computation overhead
  - Communication overhead
  - Power consumptions
- Guarantee the correctness by Automatic Verifiers
- Automatic implementation to reduce the number of bugs
Overview

- Specify security requirement:
  - Authentication and secrecy
  - System specification
    - System spec:
      - Sym/asym enc/decryption
      - Low bandwidth

Protocol Specification

- Metric function
  - cost or overhead of the protocol
  - monotonically increasing

- A specification of the initial setup
  - defines which cryptographic primitives are available
  - to the principals
  - what keys each principal possesses

Representation

- $\text{Message} \rightarrow \text{Message: Encrypted}$
  | $\text{Concatenated}$
- $\text{Message} \rightarrow \text{Principal: Name \& Key}$
- $\text{Encrypted} \rightarrow (\text{Message, Key})$
- $\text{Key} \rightarrow \text{Public/Private}$
  | $\text{Symmetric-Key}$
- $\text{Concatenated} \rightarrow \text{Message List}$
- $\text{Message List} \rightarrow \text{Message}$
  | $\text{Message, Message List}$
Representation
- Each msg can be represented as a tree
  - atomic messages as leaves
  - operations as intermediate nodes.

Protocol Generator
- Generates candidate security protocol
  - Satisfying the system requirements
  - Using exhaustive search in a combinatorial protocol space \( 4 \times 10^{12} \)
  - Discard obviously flawed protocols
    - Powerful reduction tech. \( \frac{10000}{s} \)

Protocol Generator (cntd)
- Iterative deepening
  - Each iteration: cost threshold
  - Search < given threshold
  - Sorted protocols passed to the protocol screener
  - practical APG
    - Using efficient reduction techniques and heuristics
    - Each security property is therefore accompanied by:
      - pruning algorithm (which efficiently discards most severely flawed protocols)
      - a verification condition for the screener.

**Protocol Screener**

- Analyze protocol executions with any arbitrary protocol configuration
- Analysis Termination
  - Gives a proof for satisfaction of sec. Props.
  - or Generates a counterexample
- Exploits state space reduction techniques to achieve high efficiency
  - Backward search, symbolic representation
- Average: 5-10/s [500Mhz PentIII linux]

---

**Protocol Screener (Cntd)**

- Athena: represent execution Strand SM
- Logic: to reason about strand spaces & bundles
- Automate procedure to evaluate well-formed formulae in this logic
- Forced termination:
  - Bounding # of concurrent protocol runs
  - Length of msgs
- No state explosion caused by assyn.
  - Composition & symmetry redundancy
- Uncertainty theorems:
  - Prune state space at early stage

---

**Athena: Logic**

- Terms:
  - Node constant (n, n1, ...)
  - Strand constants (s, s1, ...)
  - Bundle constant (c, c1, ...)
  - Bundle variable (C, C1, ...)
- Prepositional Formula:
  - n < s < c, n < c, s < c are atomic
  - ¬f1 and f1∧f2 are p.f. if f1 and f2 are p.f.
- Well-formed formulae (wff)
  - f, ¬f1, F1∧F2

**Code Generator**

- translates the formal specification into Java code
- final implementation is correct (proof by construction)
- implementation is secure
  - Buffer overruns
  - False input attacks
  - Type flaws
  - Replay attacks and freshness attacks

**Conclusions**

- Variation of system speciation for experiments
- Iterative deepening is Greedy. Any proof for optimality?
- Is formal security protocol analysis is NP?
- Automatic code generation.