Model Checking for Authentication Protocols

Will Marrero Edmmd Clarke Somesh Jha

Present for Course CSG399

Jingsong Feng
Northeastern University

Introduction

- BAN logic need "protocol idealization" step when use its formalism.
- The model checking approach examines all possible execution traces of a security protocol in the presence of a malicious intruder with well defined capabilities, we can determine if a protocol does indeed enforce its security guarantees. If not, we can provide a sample trace of an attack on the protocol.

Perfect cryptograph Assumed

- The decryption key must be known in order to extract the plaintext from the cyphertext.
- There is enough redundancy in the cryptosystem that a cyphertext can only be generated using encryption with the appropriate key. This also implies that there are no encryption collisions

Specification

There are two kinds of properties that we currently are interested in.

- The first is a kind of **secrecy property.**

We provide the model checker with a set of terms which the intruder is not allowed to obtain. During the verification, we simply check that the intruder does not have possession of any of the terms in this set.

- The second property is a temporal property named **correspondence**

In order to check for this kind of property, we will augment the global state with counters. For each correspondence property X → Y we will maintain a separate counter which will keep track of the difference between the number of *Y* events and *X* events. If this counter ever turns negative (i.e. there are more *X* events than *Y* events) then the correspondence property will be violated at that point (there will be no one-to-one mapping from *X* events to *Y* events). Conversely, as long as the counter never goes negative there is always a one-to-one mapping from *X* events to *Y* events.

Messages

Typically, the messages exchanged during the run of a protocol are built up using pairing and encryption from smaller submessages. The smallest such submessages *(i.e. they contain no submessages themselves)* are called ***atomic messages***. There are four types of *atomic messages*.

- ***Keys*** are used to encrypt messages.
- ***Principal names*** are used to refer to the participants in a protocol.
- ***Nonces*** are randomly generated numbers. The intuition is that since they are randomly generated, any message containing a nonce can be assumed to have been generated after the nonce was generated. (It is not an ``old'' message.)
- ***Data*** which plays no role in how the protocol works but which is intended to be communicated between principals.

Rules for Messages

Let B be a subset of messages. The closure of B (denoted $\overline{B}$), representing the set of everything that can be derived from B, is defined by the following rules:

1. If $m \in B$ then $m \in \overline{B}$.
2. If $m_1 \in \overline{B}$ and $m_2 \in \overline{B}$ then $m_1 \cdot m_2 \in \overline{B}$ (pairing)
3. If $m_1 \cdot m_2 \in \overline{B}$ then $m_1 \in \overline{B}$ and $m_2 \in \overline{B}$ (projection).
4. If $m \in \overline{B}$ and $k \in \overline{B}$ then $\{m\}_k \in \overline{B}$ (encryption).
5. If $\{m\}_k \in \overline{B}$ and $k^{-1} \in \overline{B}$ then $m \in \overline{B}$ (decryption).

The Model

Formally, each principal is modeled as a 4-tuple <***N,p,I,B***> , where:

- $N \in$ ***names*** is the name of the principal.
- ***p*** is a process (similar in style to CSP) given as a sequence of actions to be performed.
- $I \subseteq M$ is a set of all messages known (which can be produced) by the principal. ***M*** is the set of all possible messages. Typically ***I*** will be infinite and in particular, it is closed under encryption, decryption, pairing (concatenation), and projection.
- B , where *vars*(***p***)→ ***I*** is the set of variables appearing in the process p , is a set of bindings.

Global state

- The global state is maintained as the composition of the participating principals, along with the intruder process, a list of permanent secrets, a list of temporary secrets, and a set of counters indexed by the pairs of principals participating in protocol runs. More formally, the global state is a 5-tuple $< \Pi, C_i, C_r, S_s, S_t >$

• $\prod$ is the product of the individual principals and the intruder process. This product is asynchronous, yielding an interleaving semantics, with the restriction that processes synchronize on messages.

• $C_i : names \times names \rightarrow \mathbb{N}$ gives the difference between the number of times some principal with name *A* has begun initiating a protocol with some other principal with name *B* and the number of times *B* has finished responding to principal *A* .

• $C_r : names \times names \rightarrow \mathbb{N}$ gives the difference between the number of times some principal named *A* has begun responding to some other principal named *B* and the number of times *B* has finished initiating a protocol with *A* .

Global state (cont,)

- $S_s \subseteq M$ is a set of messages that are are considered safe secrets. These are the set of words that the intruder is never allowed to know. This set remains constant and usually includes things like the private keys that principals use to communicate with a server.

- $S_t \subseteq M$ is a set of messages that are are considered temporary secrets. This is the set of new secrets generated during the run of the protocol. These are secrets which we assume the intruder may be able to discover by some outside means, but which the protocol should not reveal, such as session keys.

Principal's actions

The specific actions that a principal may perform can be divided into internal actions and communication actions.

- **The internal actions** are performed asynchronously. Any principal is allowed to perform an internal action and interleaving is used to model all possible behaviors when multiple principals can make a transition.

  For the most part internal actions are used to create or discover new information. For example, NEWNONCE is used to create a nonce.

- **Communication actions** consist of send and receive actions. Each receive action can potentially change the principal's local store, reflecting any new information it has ``learned." Communication actions can only occur in pairs and both principals make a transition simultaneously. These communication actions are also interleaved with the possible actions of other automata.

How counters updated

we have four special actions **BEGINIT, ENDINIT, BEGRESPOND**, and **ENDRESPOND.**

$$< A, BEGININIT(B).p', I_A, B_A > \rightarrow < A, p', I_A, B_A >$$

then we update the global state by setting the new value of $C_i(A,B)$

$$C_i\,'(A,B) = \begin{cases} C_i(A,B)+1 & \text{if } C_i \text{ is defined} \\ 1 & \text{otherwise} \end{cases}$$

Similarily

$$< B, ENDRESPOND(A).p', I_B, B_B > \rightarrow < B, p', I_B, B_B >$$

then we update the global state by setting the new value of $C_i(A,B)$

$$C_i\,'(A,B) = \begin{cases} C_i(A,B)-1 & \text{if } C_i > 0 \\ \text{error} & \text{otherwise} \end{cases}$$

A DFS algorithm to generate all traces

A trace is an alternating sequence of global states and actions and that we are interested in checking all possible traces. Clearly, there are a finite number of next states for each of the participants.

In addition, while the intruder can generate an infinite number of messages, it is only allowed to send a finite number because each **SEND** must match with a **RECEIVE.** Since the there are a finite number of possible next states, we only consider a finite number of runs, we can perform a depth first search of the state space to generate all possible traces.

The Model checking algorithm

proc DFS($global\text{-}state$)

$push\ (global\text{-}state, S)$

while(not empty(S)) do

$<\Pi, C_i, C_r, S_s, S_t> = pop(S)$

If $C_i(x, y) < 0$ for some x and y or

$C_i(x, y) < 0$ for some x and y or

$s \in I_z$ for some $s \in S_s \cup S_t$

/* where Iz is the intruder's information in Π */

then $report\text{-}error$

$L = next\text{-}states(<\Pi, C_i, C_r, S_s, S_t>)$

for each $l \in L$ $push(S, l)$

Model checking algorithm

How to maintain local stores?

The local store is accessed in three places.

- First, if principal $<A,p,I_A,B_A>$ sends a message *m* , then we must insure that $m \in I_A$.
- Second, if the principal receives message *m* , then we must update to I_A to I_A'.
- Finally, we check every global state to see if $s \in I_Z$ for some secrets(safe secrets and temp secrets) , where I_Z is the intruder's local store.

It turns that these local stores are infinite because of the closure operation. However, we never really need to compute the entire closure; we need only determine if a particular message is in the closure. So it suffices to represent the infinite set with a finite set of ``generators.''.

Normalized Derivations

If ***B*** represents some set of information that is known by a principal, then the principal also knows (can generate) all the information in ***B***'s clousre, which in general is an infinite set; however, we usually are not interested in the set of everything that a principal knows, but instead whether or not a specific message $x \in \boldsymbol{M}$ can be generated by a principal.

Let $x \in \overline{B}$ be a message, A derivation of x from B is an alternating sequence of sets of messages and rule instances written as follows:

$$B_0 \xrightarrow{R_0} B_1 \xrightarrow{R_1} \cdots B_{k-1} \xrightarrow{R_{k-1}} B_k$$

where

$$B = B_0,\ x \in B_k$$

and each rule instance Ri is written as $< I_i, N_i, O_i >$ where

- $I_i \subseteq B_i$, $B_{i+1} = B_i \cup O_i$
- N_i is one of the clore rules for $\overline{B}$ such that I_i satisfies the premise of the rule and O_i is the corresponding conclusion

Normalized Derivations (cont')

For example, let $B = \{\{a\}_k \cdot b, k^{-1}\}$, we derive $x = a \cdot b$ as follows:

$B_0 = B = \{\{a\}_k \cdot b, k^{-1}\}$

$R_0 = \{\{\{a\}_k \cdot b\}, 3, \{\{a\}_k \cdot b\}\}$ projection rule

$B_1 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b\}$

$R_1 = \{\{\{a\}_k, k^{-1}\}, 5, \{a\}\}$ decryption rule

$B_2 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b, a\}$

$R_2 = \{\{a, b\}, 2, \{a \cdot b\}\}$ paring rule

$B_2 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b, a, a \cdot b\}$ which contains $x = a \cdot b$

Normalized Derivations (cont')

We define a normalized derivation as follows:

$$B_0 \xrightarrow{R_0} B_1 \xrightarrow{R_1} \cdots B_{k-1} \xrightarrow{R_{k-1}} B_k$$

is a normalized derivation if and only if all $0 \le i < k$ is an expanding rule implies N_j is an expanding rule for all $i < j < k$; In other words all shrinking rules appear to the left of all expanding rules.

Theorems

Theorem 1.

Let $B \subseteq M$ be a set of messages. Then $x \in \overline{B}$ if and only if x has a normalized derivation from B

Theorem 2.

$m_1 \cdot m_2 \in \overline{B_s}$ if and only if $m_1 \cdot m_2 \in B_s$ or m_1 and $m_2 \in \overline{B_s}$

Theorem 3.

$\{m\}_k \in \overline{B_s}$ if and only if $\{m\}_k \in B_s$ or $m \in \overline{B_s}$ or $k \in \overline{B_s}$

How Intruders augment their knowledge

function add (I, m)
  for each $i \in I$
    if $i = \{x\}_y$ and $y^{-1} = m$
      then I=add(I, x)
    if $y \in I$ then $I = I - i$
  if $m = x \cdot y$
    then return add (add(I, x), y)
  if $m = \{x\}_y$ and $y^{-1} \in$ I
    then if $y \in I$
      then return add(I, x)
      else return add$(I \cup m, x)$
  return $I \cup m$

Complexity

The total time to augment the intruder's local store by getting a new message is $O(|B_s|^2)$

Intruders will augment their knowledge when protocol runs. This algorithm is used in DFS algorithm when protocol runs from one global state to another global states.

To Query Intruder's local store

function in (I, m)
  for $m \in I$
    then return *true*
  if $m = x \cdot y$
    then return in(I, x)
  if $m = \{x\}_y$
    then return in(I, x) && in (I, y)
  else return *false*.

Complexity

When searching for a derivation of *w* from B_s we first check to see if $w \in B_s$. This costs at most |Bs| time. If not, we break down *w* into two smaller pieces and recursively check those peices. The total number of recursive calls is bounded by the number of operations making up *w* , which is in turn bounded by $|w|$

The total time to check if $w \in$ closure of B_s is $O(|B_s||w|)$

This algorithm is used in DFS algorithm to check if safe message is known by Intruder

A verification example

We now consider an example to illustrate how the model checker works. We consider the simplified Needham-Schroeder protocol analyzed by Lowe given below:

1. $A \rightarrow B$: $A.B.\{N_a.A\}K_B$
2. $B \rightarrow A$: $B.A.\{N_a. N_b\}K_A$
3. $A \rightarrow B$: $A.B.\{N_b\}K_B$

Initiator process

```
((beginit (*p-var* b))
    (newnonce (*var* na))
    (send (*var* b)
            (concat a
                  (*var* b)
                   (encrypt (pubkey (*var* b)) (concat (*var* na) a))))
     (receive (*var* b)
                 (concat (*var* b)
                  a
                  (encrypt (pubkey a) (concat (*var* na) (*var* nb)))))
      (send (*var* b)
                 (concat a
                 (*var* b)
                 (encrypt (pubkey (*var* b)) (*var* nb))))
 (endinit (*var* b)))
```

Process description for the initiator

Intruder's initial knowledge

(a b *intruder* (pubkey a) (pubkey b)
(pubkey * intruder*) (privkey *intruder*))

Verification Result

"Lack of correspondence"

```
(B (BEGRESPOND A))
(A (BEGINIT *INTRUDER*))
 (A ((NEWNONCE (*VAR* NA)) (*NONCE* 245)))
 (A (CONCAT A *INTRUDER* (ENCRYPT (PUBKEY *INTRUDER*)
      (CONCAT   (*NONCE* 245) A))) INTRUDER)
 (INTRUDER (CONCAT A B (ENCRYPT (PUBKEY B) (CONCAT (*NONCE* 245) A))) B)
(B ((NEWNONCE (*VAR* NB)) (*NONCE* 260)))
(B (CONCAT B A (ENCRYPT (PUBKEY A)
      (CONCAT (*NONCE* 245) (*NONCE* 260)))) INTRUDER)
(INTRUDER (CONCAT *INTRUDER* A (ENCRYPT (PUBKEY A)
        (CONCAT (*NONCE* 245) (*NONCE* 260)))) A)
(A (CONCAT A *INTRUDER* (ENCRYPT (PUBKEY *INTRUDER*) (*NONCE* 260))) INTRUDER)
(A (ENDINIT *INTRUDER*))
(INTRUDER (CONCAT A B (ENCRYPT (PUBKEY B) (*NONCE* 260))) B)
```

Attack to this protocol

The model checker finds a violation of the security specification and generates a counter-example. The sequence of messages for two runs of the protocol are provided. The notation ***I*(*A*)** is meant to convey either *I* impersonating *A* if on the left of the arrow, or ***I*** intercepting a message meant for *A* if on the right of the arrow.

$$\alpha 1.\ A \rightarrow I:\ A.I.\{N_a.A\}K_I$$
$$\beta 1.\ I(A) \rightarrow B:\ A.B.\{N_a.A\}K_B$$
$$\beta 2.\ B \rightarrow I(A):\ B.A.\{N_a.N_b\}K_A$$
$$\alpha 2.\ I \rightarrow A:\ I.A.\{N_a.N_b\}K_A$$
$$\alpha 3.\ A \rightarrow I:\ A.I.\{N_b\}K_I$$
$$\beta 3.\ I(A) \rightarrow B:\ A.B.\{N_b\}K_B$$

A Fix for this protocol

1. $A \rightarrow B$: $A.B.\{N_a.A\}K_B$

2. $B \rightarrow A$: $B.A.\{N_a.\ N_b\ B\}K_A$

3. $A \rightarrow B$: $A.B.\{N_b\}K_B$

Lowe suggests fixing the protocol by changing the second message (adding the name of Principal)

Conclusion

- The way we model a protocol is very intuitive. We simply list the sequence of actions that each participant takes in the protocol. Unlike systems based on logics, we need not interpret the beliefs that each message is meant to convey, and we can generate counterexamples when an error is found.
- Unlike term rewriting approaches, we need not construct a set of rewrite rules to model how an intruder can manipulate participants to generate new messages.
- We simply model the protocol as a set of programs, one for each participant in the protocol. Because we separate the algorithms that maintain the intruder's knowledge from the state exploration algorithms, we also never need to encode the intruder for our models.

Reference

- W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols, Rutgers, USA, 1997.
- Gavin Lowe, Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR ,Tools and Algorithms for the Construction and Analysis of Systems ({TACAS}), Springer-Verlag, Berlin Germany,147-166,1996.