TAPS: A First-Order Verifier for Cryptographic Protocols

Ernie Cohen

Telcordia Technologies
ernie@research.telcordia.com

1 Introduction

In recent years, a number of cryptographic protocols have been mechanically verified using a variety of inductive methods (e.g., [4,3,5]). These proofs typically require defining a number of recursive sets of messages, and require deep insight into why the protocol is correct. As a result, these proofs often require days to weeks of expert effort.

We have developed an automatic verifier, TAPS, that seems to overcome these problems for many cryptographic protocols. TAPS uses the protocol text to construct a number of first-order invariants; the proof obligations justifying these invariants, along with any user-specified protocol properties (e.g. message authentication), are proved from the invariants with a resolution theorem prover. The only flexibility in constructing these invariants is to guess, for each type of nonce[1] and encryption generated by the protocol, a formula capturing conditions necessary for that nonce/encryption to be published (i.e., sent in the clear). TAPS chooses these formulas heuristically, attempting to match the designer's intent as expressed in traditional protocol notations (for example, the choice can be influenced by the formally irrelevant use of the same variable name in different transitions). When necessary, the user can override these choices, but TAPS usually needs these hints only for recursive protocols and certain types of nested encryptions. Justifying the invariants usually requires substantial protocol reasoning; proving the invariants in a single simultaneous induction is critical to making the proofs work.

TAPS has verified properties of about 60 protocols, including (variants of) all but 3 protocols from the Clark & Jacob survey [1]. 90% of these protocols require no hints from the user; the remainder an average about 40 bytes of user input (usually a single formula). The average verification time for these protocols is under 4 seconds (on a 266MHz PC), and TAPS verifications seem to require about an order of magnitude less user time than equivalent Isabelle verifications. Although TAPS cannot generate counterexamples, it can quickly verify many protocols without the artificial limitations on protocol or state space required by model checking approaches.

For a more complete description of TAPS, see [2].

[1] We use "nonce" to describe any freshly generated unguessable value (e.g. including session keys).

E.A. Emerson and A.P. Sistla (Eds.): CAV 2000, LNCS 1855, pp. 568–571, 2000.

```
Protocol NeedhamSchroederLowe     /* 0.7 sec */
  /* k(X) =  X's public key,   dk(k(X)) <=> X has been compromised */
Definitions {
  m0 = {A,Na}_k(B)   m1 = {B,Na,Nb}_k(A)    m2 = {Nb}_k(B)
}
Transitions {
  /* A->B  */  Na: pub(A) /\ pub(B)     -p0-> m0
  /* B->A  */  Nb: pub(B) /\ pub(m0)    -p1-> m1
  /* A->B  */      p0 /\ pub(m1)        -p2-> m2
  /* B     */      p1 /\ pub(m2)        -p3-> {}
  /* oopsNa*/      p0 /\ dk(k(A))   -oopsNa-> Na
  /* oopsNb*/      p1 /\ dk(k(B))   -oopsNb-> Nb
}
Axioms { k injective }
Goals { /* If either A or B has executed his last step and neither is
           compromised, then his partner has executed the preceding
           step, with agreement on A,B,Na,Nb                          */
    p2 => p1 \/ dk(k(A)) \/ dk(k(B))
    p3 => p2 \/ dk(k(A)) \/ dk(k(B))
}
```

Fig. 1. TAPS input for the Needham-Schroeder-Lowe protocol

2 The Protocol Model

Figure 1 shows TAPS input for the Needham-Schroeder-Lowe (NSL) protocol. Each protocol makes use of an underlying set of *messages* whose structure is given by a first-order theory. Identifiers starting with uppercase letters (A,B,Na,...) are first-order variables, ranging over messages; the remainder are first-order functions (k), first-order predicates, history predicates ($p0, p1, p2, p3$), and the unary predicate *pub*. The message theory includes the list constructors *nil* and *cons*, *enc* (encryption), and the predicates *atom* (unary) and d ($d(X, Y)$ means that messages encrypted using X can be decrypted using Y), as well as any functions mentioned in the protocol (like k in the example). The first-order theory says that *nil*, *cons*, and *enc* are injective, with disjoint ranges, and do not yield atoms. The user can provide arbitrary first-order axioms in the `Axioms` section. Lists in braces are right-associated *cons* lists, and `_` is infix encryption (e.g., `{Nb}_k(B)` abbreviates $enc(k(B), cons(Nb, nil))$).

Each protocol defines an (infinite state) transition system. The state of the system is given by interpretations assigned to the history predicates and *pub*. These interpretations grow monotonically, so any positive formula (one in which history predicates and *pub* occur only with positive polarity) is guaranteed to be stable (once true, it remains true). The abbreviation $dk(X)$ ("X is a decryptable key") is defined by $dk(X) \Leftrightarrow (\exists\, Y : d(X, Y) \wedge pub(Y))$.

The transitions of the protocol are each of the form

$$\mathsf{nv_p} : \mathsf{g_p} \xrightarrow{\mathsf{p}} \mathsf{M_p}$$

where p is a history predicate, $\mathsf{nv_p}$ is an optional list of variables (the *nonce variables* of p), $\mathsf{g_p}$ is a positive formula, and $\mathsf{M_p}$ is a message term. For each p, TAPS generates a minimal signature (list of distinct variables) Σ_p that includes all the free variables in $\mathsf{nv_p}$, $\mathsf{g_p}$, and $\mathsf{M_p}$; used as a predicate, p abbreviates $\mathsf{p}(\Sigma_\mathsf{p})$[2]. For example, in NSL, $\Sigma_{p0} = \Sigma_{oopsNa} = A, B, Na$, and $\Sigma_{p1} = \Sigma_{p2} = \Sigma_{p3} = \Sigma_{oopsNb} = A, B, Na, Nb$.

A transition describes two separate atomic actions. In the first action, the system (1) chooses arbitrary values for all variables in Σ_p, such that variables in $\mathsf{nv_p}$ are assigned fresh, distinct atoms; (2) checks that $\mathsf{g_p}$ holds in the current state; (3) adds the tuple $\langle\Sigma_\mathsf{p}\rangle$ to the interpretation of p, and (4) checks that all the axioms hold. In the second action, the system (1) chooses an arbitrary tuple from the interpretation of p, (2) publishes the corresponding message $\mathsf{M_p}$, and (3) checks that the axioms hold. Execution starts in the state where all history predicates have the empty interpretation, no messages are published, and all the axioms hold.

In addition, each protocol implicitly includes transitions modeling the ability of the spy to generate (and publish) new messages; these transitions generate fresh atoms, tuple, untuple, and encrypt previously published messages, and decrypt previously published messages encrypted under decryptable keys.

3 Generating the Invariants

To generate the invariants, TAPS has to choose a formula $\mathsf{L_v}$ for each nonce variable v (giving conditions under which a freshly generated v atom might be published) and a formula for each encryption subterm of each $\mathsf{M_p}$ (giving conditions under which the subterm might be published). The user can influence these choices by providing formulas for some of the $\mathsf{L_v}$'s or providing explicit labels for some of the subterms of the $\mathsf{M_p}$'s. TAPS calculates these formulas as follows. Let S initially be the set of all formulas $\mathsf{p}(\Sigma_\mathsf{p}) \Rightarrow ok(\mathsf{M_p})$, and let T initially be the empty set; TAPS repeatedly

- replaces a formula $\mathsf{f} \Rightarrow ok(cons(\mathsf{X}, \mathsf{Y}))$ in S with $\mathsf{f} \Rightarrow ok(\mathsf{X})$ and $\mathsf{f} \Rightarrow ok(\mathsf{Y})$;
- removes a formula $\mathsf{f} \Rightarrow ok(nil)$ from S;
- replaces a formula $\mathsf{f} \Rightarrow ok(\mathsf{X})$ in S, where X is explicitly labeled by the user with the formula g, with $\mathsf{f} \Rightarrow \mathsf{g}$ and $\mathsf{g} \Rightarrow ok(\mathsf{X})$;
- replaces a formula $\mathsf{f} \Rightarrow ok(enc(\mathsf{X}, \mathsf{Y}))$ in S with $\mathsf{f} \wedge dk(\mathsf{X}) \Rightarrow ok(\mathsf{Y})$ and adds $\mathsf{f} \Rightarrow primeEnc(enc(\mathsf{X}, \mathsf{Y}))$ to T

For example, applying this procedure to the $p2$ transition of NSL has the net effect of adding the formula $p2 \wedge dk(k(B)) \Rightarrow ok(Nb)$ to S and adding the formula $p2 \Rightarrow primeEnc(m2)$ to T.

When this process has terminated, TAPS defines $\mathsf{L_v}$ to be the disjunction of all formulas f for which $\mathsf{f} \Rightarrow ok(\mathsf{v})$ is a formula of S (unless the user has defined

[2] TAPS provides an explicit substitution operator to allow history predicates to take arbitrary arguments. Default arguments make large protocols much easier to read and modify.

L_v explicitly), and defines *primeEnc* to be the strongest predicate satisfying the formulas of T (universally quantified). For example, for NSL, TAPS defines

$$\mathsf{L}_{Na} \Leftrightarrow (p0 \wedge dk(k(B))) \vee (p1 \wedge dk(k(A))) \vee oopsNa$$
$$\mathsf{L}_{Nb} \Leftrightarrow (p1 \wedge dk(k(A))) \vee (p2 \wedge dk(k(B))) \vee oopsNb$$
$$primeEnc(X) \Leftrightarrow$$
$$(\exists\, A, B, Na, Nb : (X = m0 \wedge p0) \vee (X = m1 \wedge p1) \vee (X = m2 \wedge p2))$$

TAPS then proposes the following invariants (in addition to the axioms of the underlying first-order theory, and any axioms specified in the `Axioms` section):

(1) $\mathsf{p}(\Sigma_\mathsf{p}) \Rightarrow \mathsf{g}_\mathsf{p} \wedge (\forall \mathsf{v} : \mathsf{v} \in \mathsf{nv}_\mathsf{p} : atom(\mathsf{v}))$

(2) $\mathsf{p}(\Sigma_\mathsf{p}) \wedge \mathsf{p}(\Sigma_\mathsf{p}{}') \wedge \mathsf{v} = \mathsf{v}' \Rightarrow \Sigma_\mathsf{p} = \Sigma_\mathsf{p}{}'$

(3) $\mathsf{p}(\Sigma_\mathsf{p}) \wedge \mathsf{q}(\Sigma_\mathsf{q}) \Rightarrow \mathsf{v} \neq \mathsf{w}$ for distinct $\mathsf{v} \in \Sigma_\mathsf{p},\ \mathsf{w} \in \Sigma_\mathsf{q}$

(4) $pub(X) \Rightarrow ok(X)$

(5) $ok(nil)$

(6) $ok(cons(X, Y)) \Leftrightarrow ok(X) \wedge ok(Y)$

(7) $ok(enc(X, Y)) \Leftrightarrow primeEnc(enc(X, Y)) \vee (pub(X) \wedge pub(Y))$

(8) $\mathsf{p}(\Sigma_\mathsf{p}) \Rightarrow (ok(\mathsf{v}) \Leftrightarrow (\exists\, V : \mathsf{L}_\mathsf{v}))$ for $\mathsf{v} \in \mathsf{nv}_\mathsf{p}$
where V is the set of free variables of L_v not in Σ_p

These formulas say (1) each history predicate implies its corresponding guard, and nonce variables are instantiated to atoms; (2)-(3) no atom is used more that once as the instantiation of a nonce variable; (4) all published messages are *ok*; (5)-(6) a tuple is *ok* iff each of its components is *ok*; (7) an encryption is *ok* iff it is either a *primeEnc* or the encryption of published messages; and (8) an atom used to instantiate a nonce variable v is *ok* iff L_v holds.

The remaining formulas of S (universally quantified) are left as proof obligations; if these formulas follow from the invariants, then the invariants hold in all reachable states. The invariants are then used to prove the goals.

References

[1] J. Clark and J. Jacob. A survey of the authentication protocol literature, version 1.0. Unpublished Monograph, 1997.

[2] Ernie Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Computer Security Foundations Workshop XIII*. IEEE Computer Society Press, 2000.

[3] C. Meadows. The NRL Protocol Analyzer: An overview. In *2nd Intl. Conf. on Practical Applications of Prolog*, 1993.

[4] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 1998.

[5] S. Schneider. Verifying authentication protocols with CSP. In *Computer Security Foundations Workshop X*. IEEE Computer Society Press, 1997.