Finite Models in FOL-Based Crypto-Protocol Verification

Jan Jürjens[1] and Tjark Weber[2]

[1] Open University (UK), Microsoft Research, Cambridge, and Robinson College (Univ. Cambridge)
http://www.jurjens.de/jan
[2] Computer Laboratory, University of Cambridge
tw333@cam.ac.uk

Abstract. Cryptographic protocols can only be secure under certain inequality assumptions. Axiomatizing these inequalities explicitly is problematic: stating too many inequalities may impair soundness of the verification approach. To address this issue, we investigate an alternative approach (based on first-order logic) that does not require inequalities to be axiomatized. A derivation of the negated security property exhibits a protocol attack, and absence of a derivation amounts to absence of the investigated kind of attack.

We establish a fragment of FOL strictly greater than Horn formulas in which the approach is sound. We then show how to use finite model generation in this context to prove the absence of attacks. To demonstrate its practicality, the approach is applied to several well-known protocols, including ones relying on non-trivial algebraic properties. We show that it can be used to deal with infinitely many principals (and thus sessions).

1 Introduction

Cryptographic protocol analysis often models operations on messages in terms of a free algebra. Encryption of a message m with a key k, for instance, may be represented as the term $e(k, m)$. Decryption can be represented either implicitly, by including a rule that says that if a principal knows a (symmetric) key k and an encrypted message $e(k, m)$, then he can also learn m, or explicitly, using a decryption operator d that is assumed to satisfy a cancellation rule $d(k, e(k, m)) = m$. The knowledge of a Dolev-Yao attacker is defined inductively: he knows any messages that are transmitted over an insecure channel, and he can learn new messages by performing, e.g., encryption, decryption (provided he knows the key), and by sending messages to other principals to learn their response.

Typically automated or interactive theorem proving is used with the free algebra model to establish security properties of protocols [1]. Security proofs, however, must make crucial use of freeness: to show that transmitting a message m over an insecure channel does not reveal a secret s, we of course have to assume that s is distinct from m. Since the attacker can encrypt, we also have to assume that s is distinct from $e(m, m)$, $e(m, e(m, m))$, etc. Moreover, to show that

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 155–172, 2009.

the attacker cannot learn s, we have to assume that the attacker's *only* means of gaining knowledge are according to the rules of the inductive definition, i.e., that his knowledge is understood as a least fixed point—because without this understanding, the attacker *might* know everything. Theorem provers usually implement an "open world" semantics, where only those statements are provably false that hold in no model of the axioms. To show secrecy, distinctness and least fixed-point axioms therefore have to be asserted explicitly in these systems.

This is sometimes done automatically [2]. Otherwise, however, it can be tedious and error-prone. Stating too few inequalities may impair completeness of the verification approach, and stating too many may impair soundness. One has to be careful not to assert erroneous axioms, which might render the protocol formalization inconsistent and allow one to prove anything (e.g., security of a protocol that is in fact insecure).

In this paper we investigate the alternative approach that one can negate the conjecture about a protocol's security: instead of showing that the attacker does not know a secret s, we attempt to prove that he knows s. A proof of this conjecture corresponds to an attack, and the proven absence of a proof (equivalently, by soundness and completeness of first-order logic, a counterexample to the conjecture) corresponds to security of the protocol.[1]

Note that this approach allows us to establish security of the protocol without stating any freeness or fixed-point axioms. It is related to the famous "closed world assumption", which was popularized by Prolog and is now also used in, e.g., the ProVerif protocol verifier [4]. The closed world assumption gives rise to the treatment of negation as failure: every statement that cannot be proved is considered false.

Thus model generation can be used to prove protocols secure: a simple 2-element model (in which m and s are interpreted differently) suffices to show that transmission of m does not necessarily reveal the secret s to the attacker. In Sect. 2, we will show that this is in fact sufficient to conclude secrecy of s in the free algebra model.

In Sect. 3, we demonstrate practicality of the approach by applying it to abstract first-order formalizations of three well-known protocols: the RSA Probabilistic Signature Scheme (RSA-PSS), the Wired Equivalent Privacy (WEP) protocol, and the Needham-Schroeder-Lowe (NSL) authentication protocol. Section 4 concludes.

2 FOL-Based Crypto-Protocol Verification: Derivations vs. Models

We consider protocol formalizations in first-order logic, as e.g. in [4,5]. We restrict ourselves to secrecy properties in this paper. Blanchet [6] discusses how authentication can be treated in this framework.

[1] A similar approach was taken in [3], although without investigating the questions we consider in this paper.

Central to the formalization of a protocol is a unary predicate knows (defined inductively) that describes which messages are known to a Dolev-Yao attacker. Showing security of a protocol then amounts to showing that the attacker does *not* know a certain secret in the free algebra model.

However, when giving a proof that the axioms of a protocol formalization imply that the attacker does not know the secret, we implicitly have to consider *all* models of the axioms. In some of these models, equalities may hold that one would expect an implementation of the protocol to avoid: e.g., that the secret is equal to a publicly known value. To consider the free algebra model only, traditionally further axioms must be asserted: namely freeness of operations (to prevent unwanted *confusion* in the model), and a least fixed-point axiom for the knows predicate (to prevent the attacker from knowing messages without reason).

Stating these axioms explicitly seems inelegant. It introduces the risk of including too few distinctness axioms, impairing completeness of the verification approach by giving rise to spurious attacks (based on accidental equalities between different terms). Including too many axioms, on the other hand, might lead to an inconsistent protocol specification, thereby impairing soundness of the verification approach. We would like to avoid these dangers.

There is an alternative to asserting these axioms, and consequently proving security in the free algebra model. We will show that for many protocols, security can be established by exhibiting just *one* (arbitrary) model in which the attacker does not know the secret.

2.1 Model-Theoretic Foundations

Note that automated theorem provers implicitly consider every possible model satisfying the given axioms to see whether it satisfies the given conjecture, not only the quotients of the free algebra under the axioms. This means that in the models considered, additional properties not following from the given axioms may hold. In the case of cryptographic protocols, this may mean that a secret key coincides with a public value and therefore becomes known to the adversary. This is of course something which one would assume an implementation of the protocol to avoid, and therefore one would like to analyze the protocol under the assumption that this does not happen. There are two ways to deal with this situation:

1. If one wants to formulate the conjecture in a way that a proof of the conjecture means that the protocol is secure, one needs to explicitly include axioms which prevent an unwanted collapse of the model (for example, axioms requiring each constant in the model describing a secret value to be different from any other value).
2. Alternatively, one can formulate the conjecture in a negated way so that a proof of the conjecture corresponds to an attack, and the proven absence of a proof (equivalently, by soundness and completeness of FOL, a counterexample to the formula) corresponds to the security of the protocol. This

makes sure that, when considering a given protocol execution (i.e., a given instantiation of the message variables), all models of the formulas have to fulfill the attack conjecture in order for an attack to be detected, in particular also the quotient of the free algebra under the formula, which does not satisfy any equations that would be assumed not to hold in an implementation of the protocol (for example, between a secret key and a public value) and which we call a *confusion-free model* (to be defined precisely below). That way, false positives arising in this way can be avoided.

We would like to avoid having to introduce the distinctness axioms in the first option, since it would introduce the risk of including too few distinctness axioms (again giving rise to false counter-examples to the security of the specification), or too many (which may lead to an inconsistency of the formula on the whole).

The second option, however, raises an issue with respect to the completeness of the analysis: Note that in general it is *not* the case that any formula which holds in the quotient of a free algebra under given axioms also holds in every other model of the axioms. Therefore, in the second approach explained above, finding a counter-example which satisfies the axioms but not the conjecture does not in general have to mean that the confusion-free models of the axioms do not satisfy the conjecture either. This, however, is what one would like to establish here to show that the program is secure in a certain sense. Intuitively speaking, having a counter-example just means that there exists an implementation of the protocol which happens to be secure against the conjecture, maybe only because certain non-standard equalities happen to hold, whereas one would like to show that any reasonable implementation of the protocol, in particular the ones which correspond to the confusion-free models, is secure.

Therefore, we would like to establish under which conditions on the set of axioms and which conditions on the conjecture it is indeed the case that the following logical equivalence holds: There exists a model which satisfies the axioms but not the conjecture if and only if the confusion-free models satisfying the axioms do not satisfy the conjecture.

We consider the negated security conjecture, i.e., that the attacker knows the secret. A proof of this "insecurity conjecture" corresponds to an attack on the protocol. Since the proof implicitly considers *all* models of the protocol axioms, the attack cannot depend on accidental equalities; it will be possible in the free algebra model as well.

Recall that if a conjecture cannot be derived from a given set of axioms, this does not mean that one can derive the negation of the conjecture from the axioms. The conjecture may just be independent of the axioms, i.e., it may hold in some models satisfying the axioms, but not in others. In general it is not the case that any formula which holds in the free algebra model also holds in every other model of the axioms.

We now identify conditions on protocol formalizations that are sufficient to conclude security (i.e., that the attacker does not know the secret in the free algebra model) from the proven absence of a proof of the "insecurity conjecture" (or equivalently, by soundness and completeness of first-order logic, from

a counterexample). Note that this is closely related to the treatment of negation as failure, which is justified in, e.g., Prolog by the "closed world assumption".[2]

Let us recall some concepts from mathematical logic. We observe that a formalization of security protocols in first-order logic (or more precisely, of the corresponding knows predicate describing the attacker's knowledge) can often be given by *strict Horn clauses*. The following definition is standard [9].

Definition 1 (Strict Horn Clause). *A* strict Horn clause *is a formula that consists of universal (first-order) quantifiers followed by a quantifier-free formula of the form φ (a* fact*), or $\varphi_1 \wedge \ldots \wedge \varphi_n \Longrightarrow \varphi$ (a* rule*), where the formulas φ_1, $\ldots$, φ_n, φ are all atomic.*

Theories consisting of strict Horn clauses always have an *initial model*. This is known as the *initial model theorem* [9].

Theorem 1 (Initial Model Theorem). *Let $\mathcal{T}$ be a theory consisting of strict Horn clauses. Then $\mathcal{T}$ has a model A with the property that for every model B of $\mathcal{T}$ there is a unique homomorphism from A to B. (Such a model A is called an* initial model *of $\mathcal{T}$. It is unique up to isomorphism.)*

The initial model satisfies the *no junk* and *no confusion* properties; its universe is indeed the freely generated term algebra of messages. Moreover, the initial model (also known as the *least Herbrand model*) satisfies only those atomic sentences that are derivable from $\mathcal{T}$. Thus, the initial model is precisely the free algebra model that we are interested in. Moreover, if we can find *any* model B in which the attacker does not know the secret, then the existence of a homomorphism from the initial model A to B implies that the attacker does not know the secret in the initial model either. (Recall that a homomorphism h from A to B, by definition, preserves satisfaction: if $A \models \mathsf{knows}(s)$, then $B \models \mathsf{knows}(h(s))$. Therefore, by contraposition, if $B \not\models \mathsf{knows}(h(s))$, then $A \not\models \mathsf{knows}(s)$.)

Theorem 1 also applies to theories that contain equality. (For a proof sketch, note that equality is a predicate that can be axiomatized by strict Horn clauses. We then obtain the initial model by taking equivalence classes of elements. A detailed proof can be found in [9].) Therefore the theorem covers both implicit and explicit decryption (cf. Sect. 1). More generally it covers *varieties*, i.e., algebraic structures defined by identities.

We would like to investigate now to what extent Thm. 1 can be strengthened to cover a larger fragment of FOL exceeding strict Horn clauses, in particular wrt. confusion-freeness in crypto-protocol verification.

For the purposes of this paper, we are only concerned with the additional knowledge that the adversary may gain from exploiting certain *equalities* in specific models of the axioms (which correspond to implementations of the protocol specification). One could also investigate other kind of relations besides equalities which the adversary may exploit to gain knowledge, although we believe

[2] It is interesting to note that there are non-standard interpretations of classical logic formulas such as inductive logic [7] that also enforce certain "closed world" assumptions and that have recently been used for crypto-protocol verification [8].

that in many cases these could be modeled by equalities, if necessary introducing extra function operators.

Therefore, we are interested in the models M of a given set $\mathcal{A}$ of axioms in the (logical) signature Σ (containing function and relation symbols) that are *confusion-free* in the following sense: for all terms $t1$, $t2$ in Σ with free variables $\boldsymbol{x}$, if the formula $\forall \boldsymbol{x}.\, t1 = t2$ holds in M, then it holds in all models of $\mathcal{A}$.

Suppose we are given a signature Σ and a set $\mathcal{A}$ of axioms and a conjecture c. We would like to establish sufficient conditions on $\mathcal{A}$ and c that imply existence of a set $\mathcal{G}$ of confusion-free models such that if all models in $\mathcal{G}$ satisfy c, then each model of $\mathcal{A}$ satisfies c.

Our approach is to find a way to construct all models of $\mathcal{A}$ out of the models in $\mathcal{G}$ in a way that preserves satisfaction of c. We recall a few definitions from algebraic model theory [10]. A *limit sentence* is a sentence of the form

$$(\forall x_1, \ldots, x_n)(\phi(x_1, \ldots, x_n) \Longrightarrow (\exists! y_1, \ldots, y_m)\psi(x_1, \ldots, x_n, y_1, \ldots, y_m)).$$

Note that, in particular, universal Horn sentences are limit sentences. However, limit sentences are strictly more expressive than universal Horn sentences: for example, one can show that the class of algebras satisfying the limit formula $\forall x.\, (\alpha(x) \wedge \beta(x) \Longrightarrow \exists! y.\, \rho(x, y))$ is not definable by a universal Horn theory [10, p.210].

A subclass $\mathcal{S}$ of a class $\mathcal{M}$ of models is called *reflective* if for every structure M in $\mathcal{M}$ there exists a structure S in $\mathcal{S}$ and a homomorphism $r : M \to S$ (the *reflection homomorphism*) such that for every structure S' in $\mathcal{S}$ and for every homomorphism $f : M \to S'$ there exists a unique homomorphism $f' : S \to S'$ such that $f = f' \circ r$.

Theorem 2. *Suppose that $\mathcal{A}$ is a set of limit sentences, c a universally quantified conjunction of atomic formulas, and $\mathcal{G}$ the set of reflections of the free Σ-structures on finitely many generators under the sentences in $\mathcal{A}$. Then the structures in $\mathcal{G}$ are confusion-free models such that if all models in $\mathcal{G}$ satisfy c, then each model of $\mathcal{A}$ satisfies c.*

Proof. Let T be a set of limit sentences and $\mathcal{A}$ the class of models of T. One can show that the class of $\mathcal{A}$-structures is closed under limits and directed colimits in the class of Σ-structures. By [10, 2.48], this implies that the class of $\mathcal{A}$-structures is reflective in the class of Σ-structures.[3] Therefore, the set $\mathcal{G}$ of reflections of the free Σ-structures on finitely many generators under the sentences in $\mathcal{A}$ is confusion-free. Note that every Σ-structure is a directed colimit of the free Σ-structures on finitely many generators. This implies that every $\mathcal{A}$-structure is a directed colimit of the structures in $\mathcal{G}$. Also, it follows that every $\mathcal{A}$-structure satisfies any universally quantified conjunction c of atomic formulas satisfied by all structures in $\mathcal{G}$ (because these are preserved by quotients and, as limit sentences, by directed colimits). This proves the above theorem.

[3] Note that it follows from [11] that the converse does not hold, that is a reflective subclass of a class of structures which is closed under colimits need not be definable by a limit theory.

Theorem 2 covers approaches such as [12] (formalizing BAN logic),[4] as well as [3,13] (that use similar formalizations as ours here). [13] also remarks that a significant number of protocols and security requirements can in fact be expressed using Horn formulas. [14] uses the (arguably inelegant) approach that the protocol specifier has to explicitly introduce relevant inequalities (such as different public keys being unequal). The paper does not comment on how to deal with a possible incompleteness or inconsistency of the resulting axioms. All of the above approaches do not explicitly consider the issue of soundness with respect to cryptographic inequalities raised above.

It would be worthwhile to consider whether Thm. 2 can be generalized from limit theories to basic theories defined in [10, 5.31], but we do not need this here.

Note that the restrictions on $\mathcal{A}$ cannot be relaxed arbitrarily. For instance, with $\mathcal{A} = \{e(k,a) \neq e(k,b) \Longrightarrow \mathsf{knows}(s)\}$ and $c = \mathsf{knows}(s)$ one obtains a model which satisfies the axiom (by falsifying its premise) but not the conjecture, although the corresponding protocol should be regarded as insecure (because one would usually assume an implementation to satisfy $e(k,a) \neq e(k,b)$, unless $a = b$). The reason is that limit sentences do not admit negation.

In cryptographic protocols, the usage of negated equations in the protocol formalization applies for example to the error treatment when a cryptographic certificate verification fails. Certificate verification is typically performed by checking an equation $c = c'$, where c and c' are cryptographic expressions. For error handling one would need to specify a behavior that is executed when $c = c'$ fails to hold, i.e., when $c \neq c'$ holds.

To deal with the fact that this is not supported by limit sentences, we simply leave out the precondition $c \neq c'$ from the formalization. This abstraction is safe (because the attacker needs to do less work to achieve his goal), and it is not overly unrealistic, because one would usually assume that the attacker would anyhow be able to provoke a failed certificate check (simply by producing a wrong certificate) to try to exploit a possible security weakness in the error treatment. Nevertheless, we are currently considering whether one can make use of ideas on elimination of negation in term algebras in this context [15,16].

Also, the restrictions on c cannot be relaxed arbitrarily, for example because quotients of free algebras do not in general preserve inequalities and implications. Thus, for $\mathcal{A} = \emptyset$ and $c = (e(k,a) = e(k,b) \Longrightarrow \mathsf{knows}(s))$ a counterexample is found, although one would usually assume an implementation to satisfy c (vacuously, since a and b are distinct because nothing forces them to be equal here).

However, in our practical examples it has so far always been possible to formalize the conjecture in a way that the restrictions do not become a problem. If they would, one could still take the approach of considering the security conjecture directly (without negating) and asserting suitable distinctness axioms. If

[4] Note that there a conjecture formalizes a security (rather than insecurity) property, but our theorem easily transforms to that case, since BAN logic does not formalize the adversary knowledge but rather the protocol participants assurance and can thus be treated in a dual way.

the model generator then finds counterexamples to both the conjecture and its negation, one knows that the conjecture is independent of the axioms; thus, more distinctness axioms should be introduced. However, this has not happened yet in our usage of this approach in a variety of examples and application case-studies.

2.2 Using Finite Model Generation for Verifying Crypto-Protocols

In Sect. 2.1, we have given sufficient conditions under which security of a protocol (i.e., that the attacker does not know the secret in the free algebra model) can be concluded from existence of just one (arbitrary) model in which the attacker does not know the secret. Unlike the free algebra model, which is necessarily infinite, this model may even be finite. Thus finite model generation [17] can be used to search for it.

A finite model generator is an automatic software tool that attempts to build a finite model of a (typically first-order) formula. In a sense, model generation is dual to theorem proving: while the latter establishes validity, the former establishes satisfiability (and may provide counterexamples by considering the negation of a formula).

Note that some formulas have infinite models only. Thus, failure to find a finite model does not prove unsatisfiability. Validity and finite satisfiability of first-order formulas are semi-decidable, general satisfiability however is not. In the following Sect. 3 we will see that these theoretical restrictions are of limited importance in practice.

3 Case Studies

To further illustrate the approach for protocol verification discussed in this paper, and to demonstrate its practicality, we have applied the approach to three well-known (and frequently studied) protocols: the RSA Probabilistic Signature Scheme (RSA-PSS) [18], the Wired Equivalent Privacy (WEP) protocol [19], and the Needham-Schroeder-Lowe (NSL) authentication protocol [20].

We present abstract protocol formalizations in TPTP [21] syntax. The TPTP library is a collection of standard benchmark problems for first-order theorem provers. The concrete syntax uses `&` for conjunction, `=>` for implication, and `!` (followed by a list of variables in square brackets) for universal quantification.

Vampire 10.0, an automatic theorem prover for first-order logic, was used to find attacks, and Paradox 2.3 [22], a finite model generator, was employed to search for models that show security. Both Vampire (which was invoked via Sutcliffe's "System on TPTP" [23] web interface) and Paradox support TPTP syntax as their input format.

We have also formalized these protocols in Isabelle/HOL [2], an interactive theorem prover and model generator for higher-order logic. Since the Isabelle/HOL formalization differs from the TPTP formalization only in terms of concrete syntax, we do not show it in this paper.

As is often done in protocol analysis, we assume *perfect cryptography* and consider abstract versions of these protocols only, i.e., we do not aim to verify

an actual implementation, nor the mathematics underlying the cryptographic primitives (except where stated otherwise).

3.1 RSA-PSS

RSA-PSS [18] is a digital signature scheme that follows the usual "hash-then-sign" paradigm. RSA refers to the now classic algorithm for public-key cryptography devised by Rivest, Shamir, and Adleman [24]. PSS stands for "Probabilistic Signature Scheme", first described by Bellare and Rogaway [25]. Starting with a message m that is to be signed, the RSA-PSS protocol—at a very abstract level—proceeds in two steps:

1. Apply a one-way hash function to the message m to produce an encoded message $\mathsf{hash}(m)$.
2. Apply a signature function to the encoded message, using a private key k, to produce a signature $\mathsf{sign}(\mathsf{hash}(m), k)$.

The message m is then sent together with its signature, $\mathsf{sign}(\mathsf{hash}(m), k)$. The signature can be verified by the receiver using the sender's public key k^{-1}. Note that RSA-PSS is not an encryption algorithm; m becomes publicly known.

A detailed Isabelle/HOL formalization of the RSA-PSS protocol by Lindenberg and Wirt is available [26]. For our purposes, however, it will be sufficient to model hashing and signing as uninterpreted functions. Our analysis is therefore not specific to PSS hashing. A third function, conc, forms the concatenation of two messages.

We assume a naive implementation of the RSA signature function that suffers from an undesirable homomorphism property, which allows the attacker to compute the signature of concatenated messages from signatures for their components (and vice versa):

$$\mathsf{sign}(\mathsf{conc}(a, b), k) = \mathsf{conc}(\mathsf{sign}(a, k), \mathsf{sign}(b, k)). \tag{1}$$

If we consider a modified protocol without PSS hashing, then it is easy to show from (1) that the attacker can forge the signature for $\mathsf{conc}(b, a)$ if he knows the signature for $\mathsf{conc}(a, b)$. Vampire finds a proof of this result in less than a second.

Our goal is rather simple: we want to show that PSS hashing breaks this homomorphism property, thereby improving security of the signature scheme. We consider a protocol run where the message $\mathsf{conc}(a, b)$ is hashed, signed with some private key k, and then transmitted over an insecure connection. The first-order formulas that model the RSA-PSS protocol and the abilities of a potential Dolev-Yao attacker are shown in Fig. 1. Note that we do *not* assume sign to satisfy a homomorphism property similar to (1) wrt. hash.

Can we conclude from these axioms that the attacker knows the signature $\mathsf{sign}(\mathsf{hash}(\mathsf{conc}(b, a)), k)$? (In our formalization, it is certainly *possible* that the attacker knows this signature—since the knows predicate, as discussed in Sect. 1, may be true everywhere—but is it also *necessary*?) The answer is: No. Paradox finds a counterexample with just four elements (shown in Fig. 2) in about two

```
fof(knows_hash, axiom,(![X]:(knows(X)=>knows(hash(X))))).
fof(knows_sign,axiom,(![X,K]:((knows(X)&knows(K))=>knows(sign(X,K))))).
fof(remove_sign,axiom,(![X,K]:((knows(sign(X,K))&knows(invs(K)))=>knows(X)))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
fof(sign_hom,axiom,(![X,Y,K]:(sign(conc(X,Y),K)=conc(sign(X,K),sign(Y,K))))).
fof(protocol_msg,axiom,(knows(conc(conc(a,b),sign(hash(conc(a,b)),k))))).
fof(public_key,axiom,(knows(invs(k)))).
fof(attack,conjecture,(knows(sign(hash(conc(b,a)),k)))).
```

Fig. 1. TPTP encoding of the RSA-PSS protocol

$\mathsf{hash}(1)$	$= 2$	$\mathsf{hash}(2)$	$= 4$	$\mathsf{hash}(3)$	$= 3$	$\mathsf{hash}(4)$	$= 4$
$\mathsf{invs}(1)$	$= 3$	$\mathsf{invs}(2)$	$= 3$	$\mathsf{invs}(3)$	$= 1$	$\mathsf{invs}(4)$	$= 3$
$\mathsf{sign}(1,1)$	$= 2$	$\mathsf{sign}(1,2)$	$= 2$	$\mathsf{sign}(1,3)$	$= 1$	$\mathsf{sign}(1,4)$	$= 2$
$\mathsf{sign}(2,1)$	$= 2$	$\mathsf{sign}(2,2)$	$= 2$	$\mathsf{sign}(2,3)$	$= 3$	$\mathsf{sign}(2,4)$	$= 1$
$\mathsf{sign}(3,1)$	$= 1$	$\mathsf{sign}(3,2)$	$= 1$	$\mathsf{sign}(3,3)$	$= 3$	$\mathsf{sign}(3,4)$	$= 2$
$\mathsf{sign}(4,1)$	$= 2$	$\mathsf{sign}(4,2)$	$= 2$	$\mathsf{sign}(4,3)$	$= 4$	$\mathsf{sign}(4,4)$	$= 1$
$\mathsf{conc}(1,1)$	$= 1$	$\mathsf{conc}(1,2)$	$= 2$	$\mathsf{conc}(1,3)$	$= 3$	$\mathsf{conc}(1,4)$	$= 1$
$\mathsf{conc}(2,1)$	$= 2$	$\mathsf{conc}(2,2)$	$= 2$	$\mathsf{conc}(2,3)$	$= 3$	$\mathsf{conc}(2,4)$	$= 2$
$\mathsf{conc}(3,1)$	$= 3$	$\mathsf{conc}(3,2)$	$= 3$	$\mathsf{conc}(3,3)$	$= 3$	$\mathsf{conc}(3,4)$	$= 3$
$\mathsf{conc}(4,1)$	$= 4$	$\mathsf{conc}(4,2)$	$= 2$	$\mathsf{conc}(4,3)$	$= 3$	$\mathsf{conc}(4,4)$	$= 1$
a	$= 4$	b	$= 1$	k	$= 3$		
$\mathsf{knows}(1)$		$\mathsf{knows}(2)$		$\neg\mathsf{knows}(3)$		$\mathsf{knows}(4)$	

Fig. 2. Model showing security of RSA-PSS hashing

seconds on a current personal computer. This proves that the attack mentioned above (exploiting (1)) is no longer possible with hashing.[5] Of course the interpretation of, e.g., conc is not the usual one in this finite model. By Thm. 1, however, the result also holds in the free algebra model (modulo (1)).

3.2 Wired Equivalent Privacy

The Wired Equivalent Privacy (WEP) protocol [19] was introduced in 1997 to provide confidentiality for wireless networks. Later serious weaknesses were identified, and the protocol is now considered deprecated. In the context of this paper the protocol is interesting because it employs the "exclusive or" function, which can be characterized algebraically by associativity, commutativity, existence of a neutral element, and nilpotency:

$$\mathsf{xor}(\mathsf{xor}(x,y),z) = \mathsf{xor}(x,\mathsf{xor}(y,z)), \qquad \mathsf{xor}(x,0) = x,$$
$$\mathsf{xor}(x,y) = \mathsf{xor}(y,x), \qquad \mathsf{xor}(x,x) = 0.$$

Note that all four axioms are universally quantified identities.

[5] We have also shown that this remains true even if the hashing function is reversible, i.e., if we add an axiom `![X]:(knows(hash(X))=>knows(X))`.

```
fof(xor_assoc,axiom,(![X,Y,Z]:(xor(xor(X,Y),Z)=xor(X,xor(Y,Z))))).
fof(xor_comm,axiom,(![X,Y]:(xor(X,Y)=xor(Y,X)))).
fof(xor_neutral,axiom,(![X]:(xor(X,zero)=X))).
fof(xor_nilpotent,axiom,(![X]:(xor(X,X)=zero))).
fof(knows_xor,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(xor(X,Y))))).
fof(knows_zero,axiom,(knows(zero))).
fof(knows_rc4,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(rc4(X,Y))))).
fof(knows_c,axiom,(![X,Y]:(knows(X)=>knows(c(X))))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
fof(protocol_msg,axiom,(knows(conc(v,xor(conc(m,c(m)),rc4(v,k)))))).
% an arbitrary delta message
fof(knows_d,axiom,(knows(d))).
% homomorphism property for c
fof(c_hom,axiom,(![X,Y]:(c(xor(X,Y))=xor(c(X),c(Y))))).
% homomorphism property for conc
fof(conc_hom,axiom,(![X1,Y1,X2,Y2]:(xor(conc(X1,Y1),conc(X2,Y2))
  =conc(xor(X1,X2),xor(Y1,Y2))))).
% the protocol is malleable
fof(attack,conjecture,
  (knows(conc(v,xor(conc(xor(m,d),c(xor(m,d))),rc4(v,k)))))).
```

Fig. 3. TPTP encoding of the Wired Equivalent Privacy protocol

Our formalization is based on [27, Sect. 3.3]. The WEP protocol uses the RC4 algorithm, which generates a sequence of pseudo-random bits from an initial vector v and a shared secret key k. Moreover, it uses a checksum algorithm c, e.g., cyclic redundancy check. We model both RC4 and c by uninterpreted functions.

To encrypt a message m, principal A chooses an initial vector v, computes $\mathsf{RC4}(v,k)$ and $\mathsf{c}(m)$, encrypts $\mathsf{conc}(m,\mathsf{c}(m))$ with $\mathsf{RC4}(v,k)$ (using xor), and sends both v and the ciphertext to principal B. To decrypt, B (who must know the shared key k) then computes $\mathsf{RC4}(v,k)$, obtains $\mathsf{conc}(m,\mathsf{c}(m))$ from the ciphertext (again using xor), and verifies the checksum $\mathsf{c}(m)$.

If both conc and c satisfy a homomorphism property with respect to xor (see Fig. 3), the protocol becomes malleable: a Dolev-Yao attacker can modify a ciphertext arbitrarily without disrupting the checksum [28]. Vampire automatically proves the attack in about three seconds.

The attack is no longer possible if we drop the `conc_hom` axiom. Paradox then finds a counterexample (of size 8) to the `attack` conjecture in less than a second. We omit the model for space reasons.

If we instead drop the `c_hom` axiom, Paradox—perhaps surprisingly—fails to refute the attack. We used it to exhaustively check all models of size 15 or less (which took several hours on a current personal computer); none of them constitute a counterexample to the conjecture. Possibly, all counterexamples are infinite.

3.3 Needham-Schroeder-Lowe with ECB

The Needham-Schroeder-Lowe (NSL) protocol [20] is perhaps the most frequently studied authentication protocol of all. Our formalization is based on [27, Sect. 3.5]. The NSL protocol consists of three messages.

1. First, principal A sends $\mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(B))$ to principal B, where $\mathsf{pk}(B)$ is B's public key, and N_A is a fresh (e.g., random) value.
2. B decrypts this message, using his secret key $\mathsf{sk}(B)$, to learn N_A. He then sends $\mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, B)), \mathsf{pk}(A))$ to A, where N_B is again a fresh value.
3. Third, A decrypts B's message to learn N_B, and replies to B with $\mathsf{encrypt}(N_B, \mathsf{pk}(B))$. B verifies receipt of this message.

At the end of the protocol, A and B are convinced to talk with each other, and to share the secrets N_A and N_B. The protocol, however, is flawed if Electronic Code Book (ECB) is used for encryption. ECB is a block cipher mode that simply encrypts each message block separately. ECB (formalized by the uninterpreted function encrypt below) renders the NSL protocol insecure because it satisfies a homomorphism property:

$$\mathsf{encrypt}(\mathsf{conc}(a, b), k) = \mathsf{conc}(\mathsf{encrypt}(a, k), \mathsf{encrypt}(b, k)). \tag{2}$$

The attack works as follows. Assuming that principal A initiates a protocol session S_1 with the intruder by sending $\mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(I))$, the intruder can then

1. impersonate A to initiate a session S_2 with principal B:

$$(S_2.1) \quad \mathrm{I(A)} \rightarrow \mathrm{B}: \quad \mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(B))\,,$$

2. learn the secret N_B by abusing A to decrypt B's response:

$$\begin{array}{lll} (S_2.2) & \mathrm{B} \rightarrow \mathrm{I(A)}: & \mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, B)), \mathsf{pk}(A)) \\ (S_1.2) & \mathrm{I} \rightarrow \mathrm{A}: & \mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, I)), \mathsf{pk}(A)) \\ (S_1.3) & \mathrm{A} \rightarrow \mathrm{I}: & \mathsf{encrypt}(N_B, \mathsf{pk}(I)), \end{array}$$

3. and finally use N_B to trick B (who is actually communicating with the intruder) into believing that he is communicating with A:

$$(S_2.3) \quad \mathrm{I(A)} \rightarrow \mathrm{B}: \quad \mathsf{encrypt}(N_B, \mathsf{pk}(B))\,.$$

The ECB homomorphism property (2) is used in step 2 of the attack.

The protocol formalization in TPTP syntax is shown in Fig. 4. We assume that exactly one principal, A, initiates a session with the intruder. To model freshness of N_B, we use a Skolem function (rather than a constant) nb that takes three arguments, which correspond to the three parameters in the protocol's first message, i.e., N_A, A, and B. Vampire automatically proves the attack in about two seconds.

```
fof(knows_encrypt,axiom,(![X,K]:((knows(X)&knows(K))=>knows(encrypt(X,K))))).
fof(remove_encrypt,axiom,(![X,Y]:((knows(encrypt(X,pk(Y)))&knows(sk(Y)))
  =>knows(X)))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
% principal A initiates a session with the intruder I
fof(protocol_msg_1,axiom,(knows(encrypt(conc(na,a),pk(i))))).
fof(protocol_msg_2,axiom,(![Na,A,B]:(knows(encrypt(conc(Na,A),pk(B)))
  =>knows(encrypt(conc(Na,conc(nb(Na,A,B),B)),pk(A)))))).
% principal A will respond to the intruder's reply
fof(protocol_msg_3,axiom,(![Nb]:(knows(encrypt(conc(na,conc(Nb,i)),pk(a)))
  =>knows(encrypt(Nb,pk(i)))))).
fof(knows_i,axiom,(knows(i))).
fof(knows_pka,axiom,(knows(pk(a)))).
fof(knows_pkb,axiom,(knows(pk(b)))).
fof(knows_pki,axiom,(knows(pk(i)))).
fof(knows_ski,axiom,(knows(sk(i)))).
% ECB homomorphism property
fof(ecb_hom,axiom,(![X,Y,K]:(encrypt(conc(X,Y),K)
  =conc(encrypt(X,K),encrypt(Y,K))))).
fof(attack,conjecture,(knows(nb(na,a,b)))).
```

Fig. 4. TPTP encoding of the Needham-Schroeder-Lowe protocol

On the other hand, if we do not assume (2), the attacker can no longer gain knowledge of $\mathsf{nb}(N_A, A, B)$. Paradox finds a counterexample of size 4 in just about a second. We omit the model for space reasons.

Our formalization does not model the state of principal A. In reality, the intruder can abuse A to decrypt a value only once. In our formal model, the intruder is more powerful: there appears to be an unbounded number of protocol sessions initiated by A available to him. However, this is not used in Vampire's proof of the attack, and it does not affect security of the protocol when (2) is omitted.

Infinitely Many Principals

We can model a potentially infinite number of principals by introducing a unary function next and a predicate $\mathsf{principal}$, and requiring $\mathsf{principal}(\mathsf{i})$ as well as $\forall x.\ (\mathsf{principal}(x) \implies \mathsf{principal}(\mathsf{next}(x)))$. Note that these assertions are strict Horn clauses. If we modify the NSL formalization accordingly (by replacing `protocol_msg_1`, `protocol_msg_3`, and `knows_pkX` from Fig. 4 with suitably quantified versions), Paradox still finds a counterexample of size 4. Thus, the NSL protocol is (by Thm. 1) secure even in the presence of infinitely many principals.

4 Conclusion

In this paper, we have proposed an approach to crypto-protocol verification that proceeds by negating the security conjecture, and then employs finite model generation to show security by means of a single (counter-) model. This idea was

independently pioneered in an earlier paper by Selinger [29]. Here we have established conditions under which the approach is sound, thereby perhaps remedying the "uneasy feeling" that Selinger in that paper mentioned that he had because of the models' potential simplicity. Our result also shows that approaches such as [3] are sound in the sense that our Theorem 2 applies to it. We have also shown that the search for models can be automated not just "in principle", but that current model generators are extremely valuable for this task.

Theorem proving, in this approach, can be used to find attacks. Unlike in the traditional free algebra approach, no freeness axioms or least fixed-point axioms for the attacker's knowledge are required.

In contrast to ProVerif, we do not merely claim security of protocols, but produce "security certificates": models that can be verified independently. This has recently been explored further by Goubault-Larrecq [30], who translates models into Coq proofs of protocol correctness. Furthermore, our use of a standard, highly efficient theorem prover and model generator leads to effortless support for a larger fragment of first-order logic, including, e.g., equality.

The proposed technique has both theoretical and practical limitations. It is sound for protocols that can be formalized by strict Horn clauses (including identities), and more generally, limit theories (cf. Sect. 2). On the practical side, finite model generators may fail to find a counterexample (which would demonstrate security of the protocol) because of resource (i.e., runtime, memory) constraints, or simply because no finite model exists. Therefore the approach is not complete; it may fail to demonstrate the security of a secure protocol. Nevertheless, its successful application to three well-known protocols in this paper provides evidence that the approach is both sufficiently versatile and practical. It can simplify protocol formalizations and security proofs significantly.

In this paper, we only considered secrecy properties. However, the general results carry over to other properties that can be formalized within limit theories (such as authentication by correspondence properties).

In future work, it would be interesting to investigate how the ideas discussed here could be used in the context of verifying implementations, rather than specifications of crypto protocols, e.g., in the context of [31], or the verification of secure information flow, e.g., based on [32].

References

1. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
2. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
3. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS, vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
4. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: CSFW-14, pp. 82–96. IEEE Computer Society, Los Alamitos (2001)

5. Jürjens, J.: A domain-specific language for cryptographic protocols based on streams. Journal of Logic and Algebraic Programming (JLAP) (2009)
6. Blanchet, B.: From secrecy to authenticity in security protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 342–359. Springer, Heidelberg (2002)
7. Comon, H., Nieuwenhuis, R.: Induction = I-axiomatization + first-order consistency. Technical report, ENS Cachan (1998)
8. Steel, G., Bundy, A., Maidl, M.: Attacking a protocol for group key agreement by refuting incorrect inductive conjectures. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 137–151. Springer, Heidelberg (2004)
9. Hodges, W.: Model Theory. Cambridge University Press, Cambridge (1993)
10. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. London Math. Soc. Lect. Note Ser., vol. 189. Cambridge University Press, Cambridge (1994)
11. Jürjens, J.: On a problem of Gabriel and Ulmer. Journal of Pure and Applied Algebra 158, 183–196 (2001)
12. Schumann, J.: Automatic verification of cryptographic protocols with SETHEO. In: CADE (1999)
13. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. Sci. Comput. Program. 50(1-3), 51–71 (2004)
14. Cohen, E.: First-order verification of cryptographic protocols. Journal of Computer Security 11(2), 189–216 (2003)
15. Lassez, J.L., Maher, M.J., Marriott, K.: Elimination of negation in term algebras. In: Tarlecki, A. (ed.) MFCS 1991. LNCS, vol. 520, pp. 1–16. Springer, Heidelberg (1991)
16. Comon, H., Fernández, M.: Negation elimination in equational formulae. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629. Springer, Heidelberg (1992)
17. Weber, T.: SAT-based Finite Model Generation for Higher-Order Logic. PhD thesis, Technische Universität München (2008)
18. RSA Laboratories: PKCS #1: RSA Cryptography Standard Version 2.1. (June 2002)
19. Institute of Electrical and Electronics Engineers: IEEE Std 802.11-1997 (1997)
20. Lowe, G.: An attack on the Needham-Schroeder public key authentication protocol. Information Processing Letters 56(3), 131–136 (1995)
21. Sutcliffe, G., Suttner, C.B.: The TPTP problem library: CNF release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)
22. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: CADE, Workshop W4 (2003)
23. Sutcliffe, G.: System on TPTP, `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP` (accessed January 9, 2009)
24. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
25. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
26. Lindenberg, C., Wirt, K.: SHA1, RSA, PSS and more. In: Klein, G., Nipkow, T., Paulson, L. (eds.) The Archive of Formal Proofs (May 2005), `http://afp.sourceforge.net/entries/RSAPSS.shtml`, Formal proof development

27. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
28. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: The insecurity of 802.11. In: MOBICOM, pp. 180–188 (2001)
29. Selinger, P.: Models for an adversary-centric protocol logic. Electr. Notes Theor. Comput. Sci. 55(1) (2001)
30. Goubault-Larrecq, J.: Towards producing formally checkable security proofs, automatically. In: Computer Security Foundations (CSF), pp. 224–238 (2008)
31. Jürjens, J.: Security analysis of crypto-based java programs using automated theorem provers. In: ASE, pp. 167–176. IEEE Computer Society, Los Alamitos (2006)
32. Jürjens, J.: Secure information flow for concurrent processes. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 395–409. Springer, Heidelberg (2000)

A Appendix

This appendix contains finite models that demonstrate security of the Needham-Schroeder-Lowe protocol (in Fig. 5) and of the Wireless Equivalent Privacy protocol (in Fig. 6). These models were mentioned in Sect. 3.3 and Sect. 3.2, respectively, of the paper. They were omitted from the paper for space reasons.

$\mathsf{conc}(1,1) = 1$		$\mathsf{conc}(1,2) = 2$		$\mathsf{conc}(1,3) = 1$		$\mathsf{conc}(1,4) = 2$	
$\mathsf{conc}(2,1) = 2$		$\mathsf{conc}(2,2) = 2$		$\mathsf{conc}(2,3) = 2$		$\mathsf{conc}(2,4) = 2$	
$\mathsf{conc}(3,1) = 3$		$\mathsf{conc}(3,2) = 2$		$\mathsf{conc}(3,3) = 1$		$\mathsf{conc}(3,4) = 4$	
$\mathsf{conc}(4,1) = 4$		$\mathsf{conc}(4,2) = 2$		$\mathsf{conc}(4,3) = 2$		$\mathsf{conc}(4,4) = 4$	
$\mathsf{encrypt}(1,1) = 3$		$\mathsf{encrypt}(1,2) = 4$		$\mathsf{encrypt}(1,3) = 1$		$\mathsf{encrypt}(1,4) = 4$	
$\mathsf{encrypt}(2,1) = 4$		$\mathsf{encrypt}(2,2) = 4$		$\mathsf{encrypt}(2,3) = 2$		$\mathsf{encrypt}(2,4) = 4$	
$\mathsf{encrypt}(3,1) = 3$		$\mathsf{encrypt}(3,2) = 1$		$\mathsf{encrypt}(3,3) = 3$		$\mathsf{encrypt}(3,4) = 4$	
$\mathsf{encrypt}(4,1) = 4$		$\mathsf{encrypt}(4,2) = 3$		$\mathsf{encrypt}(4,3) = 3$		$\mathsf{encrypt}(4,4) = 4$	
$\mathsf{nb}(1,1,1) = 1$		$\mathsf{nb}(1,1,2) = 3$		$\mathsf{nb}(1,1,3) = 3$		$\mathsf{nb}(1,1,4) = 1$	
$\mathsf{nb}(1,2,1) = 1$		$\mathsf{nb}(1,2,2) = 1$		$\mathsf{nb}(1,2,3) = 2$		$\mathsf{nb}(1,2,4) = 1$	
$\mathsf{nb}(1,3,1) = 3$		$\mathsf{nb}(1,3,2) = 1$		$\mathsf{nb}(1,3,3) = 3$		$\mathsf{nb}(1,3,4) = 1$	
$\mathsf{nb}(1,4,1) = 1$		$\mathsf{nb}(1,4,2) = 3$		$\mathsf{nb}(1,4,3) = 4$		$\mathsf{nb}(1,4,4) = 2$	
$\mathsf{nb}(2,1,1) = 1$		$\mathsf{nb}(2,1,2) = 4$		$\mathsf{nb}(2,1,3) = 1$		$\mathsf{nb}(2,1,4) = 1$	
$\mathsf{nb}(2,2,1) = 4$		$\mathsf{nb}(2,2,2) = 2$		$\mathsf{nb}(2,2,3) = 4$		$\mathsf{nb}(2,2,4) = 2$	
$\mathsf{nb}(2,3,1) = 1$		$\mathsf{nb}(2,3,2) = 4$		$\mathsf{nb}(2,3,3) = 2$		$\mathsf{nb}(2,3,4) = 3$	
$\mathsf{nb}(2,4,1) = 1$		$\mathsf{nb}(2,4,2) = 2$		$\mathsf{nb}(2,4,3) = 1$		$\mathsf{nb}(2,4,4) = 2$	
$\mathsf{nb}(3,1,1) = 4$		$\mathsf{nb}(3,1,2) = 1$		$\mathsf{nb}(3,1,3) = 1$		$\mathsf{nb}(3,1,4) = 3$	
$\mathsf{nb}(3,2,1) = 3$		$\mathsf{nb}(3,2,2) = 1$		$\mathsf{nb}(3,2,3) = 4$		$\mathsf{nb}(3,2,4) = 3$	
$\mathsf{nb}(3,3,1) = 1$		$\mathsf{nb}(3,3,2) = 4$		$\mathsf{nb}(3,3,3) = 1$		$\mathsf{nb}(3,3,4) = 1$	
$\mathsf{nb}(3,4,1) = 1$		$\mathsf{nb}(3,4,2) = 1$		$\mathsf{nb}(3,4,3) = 4$		$\mathsf{nb}(3,4,4) = 4$	
$\mathsf{nb}(4,1,1) = 4$		$\mathsf{nb}(4,1,2) = 4$		$\mathsf{nb}(4,1,3) = 2$		$\mathsf{nb}(4,1,4) = 3$	
$\mathsf{nb}(4,2,1) = 4$		$\mathsf{nb}(4,2,2) = 1$		$\mathsf{nb}(4,2,3) = 2$		$\mathsf{nb}(4,2,4) = 1$	
$\mathsf{nb}(4,3,1) = 2$		$\mathsf{nb}(4,3,2) = 2$		$\mathsf{nb}(4,3,3) = 2$		$\mathsf{nb}(4,3,4) = 1$	
$\mathsf{nb}(4,4,1) = 4$		$\mathsf{nb}(4,4,2) = 1$		$\mathsf{nb}(4,4,3) = 3$		$\mathsf{nb}(4,4,4) = 4$	
$\mathsf{pk}(1) = 3$		$\mathsf{pk}(2) = 4$		$\mathsf{pk}(3) = 1$		$\mathsf{pk}(4) = 2$	
$\mathsf{sk}(1) = 2$		$\mathsf{sk}(2) = 4$		$\mathsf{sk}(3) = 1$		$\mathsf{sk}(4) = 4$	
$a = 1$		$b = 1$		$i = 3$		$N_A = 3$	
$\mathsf{knows}(1)$		$\neg\mathsf{knows}(2)$		$\mathsf{knows}(3)$		$\neg\mathsf{knows}(4)$	

Fig. 5. Model showing security of NSL

xor(1, 1)	= 5	xor(1, 2)	= 7	xor(1, 3)	= 4	xor(1, 4)	= 3
xor(1, 5)	= 1	xor(1, 6)	= 8	xor(1, 7)	= 2	xor(1, 8)	= 6
xor(2, 1)	= 7	xor(2, 2)	= 5	xor(2, 3)	= 8	xor(2, 4)	= 6
xor(2, 5)	= 2	xor(2, 6)	= 4	xor(2, 7)	= 1	xor(2, 8)	= 3
xor(3, 1)	= 4	xor(3, 2)	= 8	xor(3, 3)	= 5	xor(3, 4)	= 1
xor(3, 5)	= 3	xor(3, 6)	= 7	xor(3, 7)	= 6	xor(3, 8)	= 2
xor(4, 1)	= 3	xor(4, 2)	= 6	xor(4, 3)	= 1	xor(4, 4)	= 5
xor(4, 5)	= 4	xor(4, 6)	= 2	xor(4, 7)	= 8	xor(4, 8)	= 7
xor(5, 1)	= 1	xor(5, 2)	= 2	xor(5, 3)	= 3	xor(5, 4)	= 4
xor(5, 5)	= 5	xor(5, 6)	= 6	xor(5, 7)	= 7	xor(5, 8)	= 8
xor(6, 1)	= 8	xor(6, 2)	= 4	xor(6, 3)	= 7	xor(6, 4)	= 2
xor(6, 5)	= 6	xor(6, 6)	= 5	xor(6, 7)	= 3	xor(6, 8)	= 1
xor(7, 1)	= 2	xor(7, 2)	= 1	xor(7, 3)	= 6	xor(7, 4)	= 8
xor(7, 5)	= 7	xor(7, 6)	= 3	xor(7, 7)	= 5	xor(7, 8)	= 4
xor(8, 1)	= 6	xor(8, 2)	= 3	xor(8, 3)	= 2	xor(8, 4)	= 7
xor(8, 5)	= 8	xor(8, 6)	= 1	xor(8, 7)	= 4	xor(8, 8)	= 5
zero	= 5						
RC4(1, 1)	= 5	RC4(1, 2)	= 8	RC4(1, 3)	= 2	RC4(1, 4)	= 5
RC4(1, 5)	= 1	RC4(1, 6)	= 1	RC4(1, 7)	= 2	RC4(1, 8)	= 7
RC4(2, 1)	= 5	RC4(2, 2)	= 2	RC4(2, 3)	= 3	RC4(2, 4)	= 2
RC4(2, 5)	= 2	RC4(2, 6)	= 5	RC4(2, 7)	= 1	RC4(2, 8)	= 7
RC4(3, 1)	= 5	RC4(3, 2)	= 1	RC4(3, 3)	= 1	RC4(3, 4)	= 1
RC4(3, 5)	= 5	RC4(3, 6)	= 1	RC4(3, 7)	= 5	RC4(3, 8)	= 7
RC4(4, 1)	= 1	RC4(4, 2)	= 2	RC4(4, 3)	= 2	RC4(4, 4)	= 3
RC4(4, 5)	= 5	RC4(4, 6)	= 5	RC4(4, 7)	= 1	RC4(4, 8)	= 7
RC4(5, 1)	= 5	RC4(5, 2)	= 4	RC4(5, 3)	= 1	RC4(5, 4)	= 5
RC4(5, 5)	= 1	RC4(5, 6)	= 4	RC4(5, 7)	= 1	RC4(5, 8)	= 7
RC4(6, 1)	= 5	RC4(6, 2)	= 5	RC4(6, 3)	= 2	RC4(6, 4)	= 4
RC4(6, 5)	= 5	RC4(6, 6)	= 7	RC4(6, 7)	= 1	RC4(6, 8)	= 7
RC4(7, 1)	= 6	RC4(7, 2)	= 1	RC4(7, 3)	= 1	RC4(7, 4)	= 2
RC4(7, 5)	= 1	RC4(7, 6)	= 7	RC4(7, 7)	= 6	RC4(7, 8)	= 7
RC4(8, 1)	= 7	RC4(8, 2)	= 7	RC4(8, 3)	= 7	RC4(8, 4)	= 7
RC4(8, 5)	= 7	RC4(8, 6)	= 7	RC4(8, 7)	= 7	RC4(8, 8)	= 7
c(1)	= 5	c(2)	= 2	c(3)	= 1	c(4)	= 1
c(5)	= 5	c(6)	= 7	c(7)	= 2	c(8)	= 7
conc(1, 1)	= 1	conc(1, 2)	= 2	conc(1, 3)	= 7	conc(1, 4)	= 8
conc(1, 5)	= 1	conc(1, 6)	= 3	conc(1, 7)	= 2	conc(1, 8)	= 8
conc(2, 1)	= 3	conc(2, 2)	= 8	conc(2, 3)	= 7	conc(2, 4)	= 7
conc(2, 5)	= 4	conc(2, 6)	= 3	conc(2, 7)	= 4	conc(2, 8)	= 2
conc(3, 1)	= 2	conc(3, 2)	= 8	conc(3, 3)	= 7	conc(3, 4)	= 8
conc(3, 5)	= 2	conc(3, 6)	= 4	conc(3, 7)	= 2	conc(3, 8)	= 2
conc(4, 1)	= 8	conc(4, 2)	= 7	conc(4, 3)	= 7	conc(4, 4)	= 7
conc(4, 5)	= 2	conc(4, 6)	= 3	conc(4, 7)	= 4	conc(4, 8)	= 2
conc(5, 1)	= 5	conc(5, 2)	= 8	conc(5, 3)	= 6	conc(5, 4)	= 2
conc(5, 5)	= 5	conc(5, 6)	= 2	conc(5, 7)	= 2	conc(5, 8)	= 7
conc(6, 1)	= 8	conc(6, 2)	= 2	conc(6, 3)	= 4	conc(6, 4)	= 2
conc(6, 5)	= 3	conc(6, 6)	= 4	conc(6, 7)	= 6	conc(6, 8)	= 7
conc(7, 1)	= 6	conc(7, 2)	= 7	conc(7, 3)	= 3	conc(7, 4)	= 6
conc(7, 5)	= 2	conc(7, 6)	= 2	conc(7, 7)	= 3	conc(7, 8)	= 7
conc(8, 1)	= 7	conc(8, 2)	= 7	conc(8, 3)	= 7	conc(8, 4)	= 7
conc(8, 5)	= 7	conc(8, 6)	= 7	conc(8, 7)	= 7	conc(8, 8)	= 7
m	= 2	v	= 1	k	= 2	d	= 1
knows(1)		¬knows(2)		¬knows(3)		¬knows(4)	
knows(5)		¬knows(6)		¬knows(7)		¬knows(8)	

Fig. 6. Model showing security of WEP