Verification of Copy-Protection Cryptographic Protocol using Approximations of Term Rewriting Systems

Thomas Genet* Yan-Mei Tang-Talpin† Valérie Viet Triem Tong*

*IRISA / Université de Rennes 1
Campus de Beaulieu, F-35042 Rennes Cedex
{genet,viettrie}@irisa.fr

†Thomson Multimedia R&D
F-35510 Cesson Sévigné
tangy@thmulti.com

Abstract. This paper presents the verification of a cryptographic protocol of *SmartRight*, a copy protection system designed by Thomson. The property of the protocol is that the digital content can be viewed only once. The interesting aspect of this property is that it depends on but is not limited to secrecy and authentication: freshness plays a central role. The main contribution of the paper is to achieve the proof in an automated way on a Term Rewriting model of the protocol using Timbuk – a verification tool using abstract interpretation over tree automata domains.

1 The *SmartRight* system

The *SmartRight* system [Sma01] offers a high protection against illegal copies within a digital home network. Recently, the *SmartRight* system was proposed to the Digital Video Broadcasting (DVB) ad-hoc group [DVB] on Copy Protection Technology in reply to a call for proposals on Copy Protection and Content Management technologies. The DVB Copy Protection Technology ad-hoc group is currently working on defining the specification of the system that will be the standard for copy protection and content management for digital television environment in countries that employ the DVB technology, i.e. Europe and many countries in Latin America and Asia.

The 'view-only' protocol is one of the cryptographic protocols used in *SmartRight*. Figure 1 shows a complete execution of the protocol (called a session). This protocol is deployed between an access device (e.g. a decoder) receiving a scrambled digital content (e.g. video) and a presentation device (e.g. television) which is supposed to descramble the content before rendering it. The keys used to scramble the content are called *control words* (CW) and change periodically (typically every 10 seconds). Both access device and presentation device are equiped with a smartcard, called Converter Card (CC) and Terminal Card (TC), respectively. CC and TC share a secret symmetric encryption key Kc which is embedded in the card. The access device encrypts CW and sends it together with the scrambled content to the presentation device (step 2). The presentation device extracts CW (step 6) and descrambles the content if it received a good response (step 5) to the challenge (step 4).

	Converter Card	Communication	Terminal Card
1.	get new CW generate random $VoKey$, VoR		
2.		$\{VoKey, Xor(CW, VoR)\}_{K_c} \longrightarrow$	
3.			decrypt using K_c extract $VoKey$, $Xor(CW, VoR)$ generate random $VoRi$
4.		$\longleftarrow VoRi$	
5.	delete $VoKey$, VoR	$VoR, \{Hash(VoRi)\}_{VoKey} \longrightarrow$	
6.			verify $\{Hash(VoRi)\}_{VoKey}$ using $VoKey$, then extract CW using VoR

Fig. 1. specification of the protocol

The property to prove is the following: the control word CW (input of CC at step 1) may be extracted by TC (at step 6.) only once at the time where the protocol is played. This property guarantees that the content can be viewed only once. Even if the content (encrypted CW + scrambled content) is recorded, the protocol ensures that the sequence of CW cannot be replayed. This property has to be verified under assumptions following the Dolev-Yao model [DY83]: a hacker can impersonate the Converter Card and the Terminal Card and it can attack all exchanged messages of the protocol: modify, delete a message during a session or replay a message of a session in following sessions. However, it is impossible for him to know K_c or to modify the behaviour of CC and TC since they are sealed in the smart card.

2 Using Timbuk for verifying protocols

Timbuk [GVTT01] is a tool we develop for proving properties on Term Rewriting Systems (TRS for short) using abstract interpretation over tree automata domain. Using abstract interpretation rather than induction or Knuth-Bendix completion permits, in particular, to prove properties on non terminating TRSs.

In our setting, a configuration of the protocol contains the actors' and intruder's knowledge as well as the exchanged messages, and is represented by a term. A step of the protocol is described using a rule rewriting a configuration into another. Thus, executing the protocol consists in rewriting repeatedly a term representing the current configuration of the system. Intruder's actions are also modeled using rewrite rules. The model is close to the model used in [DMT98] but our verification technique is different since it is not limited to attack discovery but can also prove their absence. Starting from an initial configuration t and using Timbuk, we compute a tree automata recognizing an over-approximation of all the configurations that can be reached by applying the TRS (finitely or not) on t. Then, to verify a property on the protocol, it is enough to represent faulty configurations by terms and check that they are not recognized by the automaton. This method has already been used in [GK00] for verifying cryptographic protocols but, since then, the tool has been totally re-implemented and contruction of the approximations is more easy [GVTT01] (see section 4, for a small example).

3 Modeling the *SmartRight* protocol

Both the Converter Card and the Terminal Card are represented by a term containing their respective knowledge. The term $CC(l, v_1, v_2)$ will represent a Converter Card whose list of incoming control words is l and whose current *VoKey* is v_1 and current *VoR* is v_2. Similarly the Terminal Card is represented by a term $TC(i, v, l)$ where v is the current *Vori*, l is a term of the form the $Read(CW_n, \ldots Read(CW_1, nil))$ where $CW_1, \ldots, CW_n$ are control words that have been accepted by the Terminal Card, in this order. In i is stored the last pair ($VoKey$, $Xor(CW, VoR)$) received by the Terminal Card. Keys *VoKey*, *VoR* (resp. *VoRi*) must be fresh for each new session of the Converter Card (resp. Terminal Card).

A safe modeling choice for a key generated by a card c is to make it depend on an information known by c and that is changing at the beginning of every new session of c. For instance, the Converter Card starts a new session for every new control word cw it receives. Thus we represent randomly generated keys of the Converter Card by the terms $VoKey(cw)$ and $VoR(cw)$. On the other hand, when the Terminal Card generates $VoRi$ it does not have access to cw so it is not possible to make depend $VoRi$ on this parameter. A safe choice is to build this key as follows: $VoRi(v, l)$ where v is the last $VoKey$ the Terminal Card received and l is its list of accepted control words. Note that $VoRi(v)$ would not be fresh enough since if the intruder replays the first message of the Converter Card with a key v and a control word cw, then the reply sent by the Terminal Card will be identical: $VoRi(v)$, so will be the third message and we will get an attack: cw will be read twice. The two smart cards are supposed to communicate in an insecure world. According to the Dolev-Yao model [DY83], the intruder can read all traffic, modify or delete the messages and perform cryptographic operations. We model an intruder by its knowledge: a term of the form $Store(e_1, \ldots Store(e_n, emptystore))$ where $e_1, \ldots, e_n$ are the pieces of messages the intruder knows. Finally a configuration in *SmartRight* is represented by a term of the form $State(m, cc, tc, st)$ where m is the current message, cc, tc and st are respectively the Converter Card, the terminal card and the intruder.

Steps of the protocols are described with 4 rewrite rules: one for steps (1) and (2), one for steps (3) and (4), one for step (5) and one for step (6). We here show the first rule that initializes the protocol session: the converter card receives some control word and sends the first message (all small letters denote variables and $x :: l$ denotes the list whose first element is x and whose remainder is l).

$$State(m, CC(cw :: l, v_1, v_2), tc, i) \rightarrow$$
$$State(\{VoKey(cw), Xor(cw, VoR(cw))\}_{K_c}), CC(cw :: l, VoKey(cw), VoR(cw)), tc, i)$$

This rule models steps (1) and (2) of the protocol where the converter card has at least one control-word cw to send. The other steps of the protocol are encoded in the same way. The intruder's actions are also encoded by rewrite rules. Here are some examples[1]:

– The intruder can read and store every sent message and send every data he knows:

$$State(x, CC, TC, st) \rightarrow State(x, CC, TC, Store(x, st))$$

$$State(y, CC, TC, Store(x, st)) \rightarrow State(x, CC, TC, Store(x, st))$$

– The other rules of the intruder, encoding intruder's abilities to construct, deconstruct, crypt, decrypt, etc., do not depend on the protocol to be studied. The intruder can flip elements in its store (he can reorder it):

$$Store(x, Store(y, z)) \rightarrow Store(y, Store(x, z))$$

– The intruder can build encrypted messages

$$Store(x, store(k, st)) \rightarrow Store(\{x\}_k, Store(x, Store(k, st)))$$

– The intruder can decrypt a message m crypted with k if he has k:

$$Store(\{m\}_k, Store(k, st)) \rightarrow Store(m, Store(\{m\}_k, Store(k, st)))$$

– The intruder can deconstruct a Xor if he has at least one of the two elements:

$$Store(Xor(x, y), Store(x, st)) \rightarrow Store(y, Store(Xor(x, y), Store(x, st)))$$

$$Store(Xor(x, y), Store(y, st)) \rightarrow Store(x, Store(Xor(x, y), Store(y, st)))$$

4 The verification of *SmartRight*

Our technique allows us to analyze the protocol starting from different hypothesis or different initial configurations. It is possible to study what happens if the intruder knows some keys, or some control word by adding them to the initial configuration of the intruder. Similarly, we can study the behaviour of the protocol for different capacity of the intruder by incrementally adding some rewrite rules to the intruder behaviour.

Starting from the initial configuration and from the TRS modeling both the protocol and the intruder actions, we use Timbuk to compute the set of reachable terms. It can be used in two different modes: exact or approximated. In exact mode, reachable terms are computed iteratively and exactly. Of course, since TRSs are not necessarily terminating, the exact mode may diverge. In

[1] Note that these assumptions of the Dolev-Yao model can easily be weakened, if necessary.

approximated mode, we compute an over-approximation of reachable terms even if the TRS is not terminating. In practice, the exact mode is used to detect attacks and the approximated mode is used, when attacks are no longer found, to prove that there are no more. For the *SmartRight* protocol verification, our methodology for building approximation is the following:

"Observe one protocol session among an over-approximation[2] of an unbounded number of past and future protocol sessions, all attacked by an unbounded number of intruder's actions".

As an example, in the approximation we built, the list of control words received by the Converter Card is approximated into [*old_cw*, ..., *old_cw*, *current_cw*, *next_cw*, ..., *next_cw*]. Similarly, during approximation construction, new sets of reachable terms are approximated by so-called approximation rules. Approximation rules define equivalence classes between some sets of terms [GVTT01]. For instance, here are some of the rules defining intruder's knowledge approximation.

```
[...]
[store(x, y) -> z] -> [x -> z  y -> z]
[cons(x, y) -> qstore1] -> [x -> qstore1 y -> qstore1]
[...]
```

To comment on these rules, first recall that the approximation of reachable terms is recognized by a tree automaton. In a tree automaton, states represent sets of terms: each state can be seen as a canonical representative of an equivalence class of terms. Thus, merging two states in a tree automaton A, merges two sets of terms in A and produces an automaton A' whose language is an over-approximation of the language of A. For instance, the first approximation rule above states that for every term of the form $store(x, y)$ recognized by a state z (i.e. in the equivalence class of z), terms matched by x and y should be recognized by the same state z (i.e. added to the class of z). Similarly, the second rule merges $cons(x, y)$, x and y in the state $qstore1$, where *cons* is a pairing operator and $qstore1$ represents the intruder's knowledge before the converter card sends the message containing the control word *current_cw*. This rule merges together message components and pairs of message components in an intruder knowledge set. This approximation is natural since when the intruder knows a pair of messages he can obtain each part. There are similar rules for $qstore2$ and $qstore3$ (for intruder's knowledge during and after *current_cw* has been sent).

Once the approximation automaton is completed for the protocol, to check the *anti-replay* property, it is enough to use Timbuk's pattern matching[3] on the approximation automaton and verify that a pattern of the form $Read(x, Read(y, z))$ does only accept solutions where *old_cw*, *current_cw*, *next_cw* are accepted in this order and *current_cw* at most once. For instance, one has to verify that

[2] which is safe, since approximations built with Timbuk are safe by construction [Gen98].

[3] Syntactic matching of a term t_1 over every term t_2 (and its subterms) recognized by a tree automaton.

no solution of the form $Read(old\_cw, Read(cur\ rent\_cw, \ldots))$ is obtained (this solution represents an old control word that has been accepted after the current control word). For *SmartRight*, we have computed several tree automata recognizing the over-approximations of reachable configurations of the protocol under different hypothesis. First, we have verified the model of the protocol without any intruder action. This permitted to validate the model by verifying that if the control words are *all* emitted by the Converter Card then some of them[4] are received, at most once and in the right order by the Terminal Card. Secondly, we computed an approximation with a stronger intruder able to listen, erase and replay messages. On the computed automaton, while searching for a pattern $Read(x, Read(y, z))$, among normal solutions, one may find:

```
Solutions:
[...]
solution 4: x <- qcurrent_cw, y <- qold_cw, z <- qnil,
solution 5: x <- qcurrent_cw, y <- qcurrent_cw, z <- qreadcurrent,
solution 6: x <- qcurrent_cw, y <- qcurrent_cw, z <- qreadold,
[...]
```

where $Read(current\_cw, Read(current\_cw, \ldots))$, shows that control words may be accepted repeatedly (but in the right order). A careful study of the automaton pointed out that this attack was due to an element missing in the protocol specification. After step (6) of the protocol, if the intruder replays the message $\{VoR, \{Hash(VoRi)\}_{VoKey}\}_{K_c}$ before the terminal card starts its new session and changes its key $VoRi$, then this message can be read several times by the Terminal Card and the control word may also be extracted several times. One should note that this reflects a possible behavior of an implementation based on this incomplete protocol specification. Indeed, as long as the terminal card has not received a new message corresponding to step (2) of the protocol, it is supposed to wait in step (6). This problem is no longer possible if the terminal card deletes its key $VoRi$ as soon as it has extracted the control word. This is exactly what had been added to the Thomson specification to fix the problem. On this fixed specification the *anti-replay* property has been proved.

Finally, we have computed a third approximation automaton for an intruder having maximal capabilities w.r.t. the verification assumption. We have given to the intruder some keys (but not K_c) and the ability to listen, erase, replay, construct, deconstruct, decrypt (if he has the key), encrypt messages and also the ability to construct or deconstruct Xors. On this approximation automaton, it was possible to prove several properties.

Secrecy: K_c, Vo_Key and every control word CW remain secret. This can simply be done by checking that no pattern of the form $Store(K_c, x)$, $Store(VoKey(x), y)$, etc. can be found in the automaton.

[4] They are not all received since a Converter Card may start a new session with a new control word before the previous session was completed. Note that, we also modeled the case where TC can complete its session before CC starts a new one and we proved a liveness property where all the sent control words are received, at most once and in the right order.

Authentication Control words sent by the Terminal Card are all correctly authenticated, i.e. no control word sent by the intruder is accepted. This can be done by checking that a pattern of the form $Read(x,y)$ accepts no solution mapping x to a data owned by the intruder.

Anti-replay As before, we can check that control words are received at most once and in the right order using a pattern of the form $Read(x, Read(y, z))$.

5 Related work

In this paper, we present a case study of the Timbuk verification tool on a cryptographic protocol used in *SmartRight*. *SmartRight* may be a future norm for copy protection of DVB. Timbuk is a tool for proving some properties over TRSs by abstract interpretation. Provided that the user gives some approximation rules, Timbuk is able to build automatically a tree automaton recognizing the superset of all reachable terms for every kind of system or program whose behavior can be encoded into a TRS [GVTT01]. In this paper, this is applied to the particular case of a cryptographic protocol. TRSs are a well known and convincing model for cryptographic protocols [DMT98,JRV00,CDM$^+$00,Cir99]. In these works, cryptographic protocols are encoded into TRSs by hand or in an automatic way using an intermediate specification language dedicated to cryptographic protocols. The formal model used is similar to ours but the verification purpose is not: the produced TRSs are used for attack discovery on a bounded rewriting derivation tree. Thus, like with methods based on model-checking [Low96], if no attack is found nothing can be said about the correctness of the protocol. On the opposite, in our setting since we are dealing with approximations, if an attack is found we cannot prove that it is a real attack but if no attack is found, this is a proof that there is no flaw. Since the formal models are similar, we believe that our technique may be used as a complement to attack discovery methods. After discovering and fixing all attacks, our method can be used to prove that there is no more attack, i.e. *prove* the correctness of the protocol.

Some other works, like [Mon99,GL00,Bla01], are also using abstract interpretation with tree automata for cryptographic protocol verification. These works focus on the proof of secrecy but prove it very efficiently and without any help of the user. Furthermore, we believe that the underlying theoretical concepts of their tools could be used to prove some other properties, as we do. However, the approximation needed for proving the 'anti-replay' property are more precise than what is necessary for proving secrecy. Thus, this would lead to a complete transformation of their respective systems since the rules guiding the approximation are built-in whereas they are given by the user in our case. This comes the fact that Timbuk is *not* a tool specialized into the verification of cryptographic protocols and thus there is no built-in approximation rule dedicated to a specific domain. As a result, it is more easy for us to adapt the approximation to the property we want to prove. We believe that this is important since properties to prove on the protocols are to become more sophisticated.

With regards to some other verification methods, as far as we know, the 'anti-replay' property (control words are received at most once, and in the right order) seems difficult to tackle, on an unbounded number of sessions, with a *fully automatic verification tool* [CM01]. This property is likely to be in the scope of some other semi-automatic tools like the NRL Protocol Analyzer (NPA) [Mea96]. The main difference between Timbuk and NPA is that NPA is abble to find and reconstruct attacks whereas it remains difficult from an approximation tree automaton. The NPA provide induction techniques whereas Timbuk is restricted to simple abstract interpretations, and thus may necessitate fewer user interaction to obtain a finite search space. The NPA starts from a forbidden state and tries to check if it can be reached from an initial state, whereas Timbuk builds once for all a tree automaton recognizing every reachable state and then provides tools to check if some particular forbidden states or patterns are recognized by the automaton. Last but not least our tool is clearly not at the same state of development as the NRL Protocol Analyser is. For the moment, it is a more or less only a prototype with no protocol description facilities and few optimisations. As a result, whereas every method and tools cited above needs at most seconds to prove secrecy on bunches of protocols, Timbuk needs hours and two hundred megabytes to build a completed approximation automaton on the 'view-only' protocol. However, it is worth achieving this effort since this automaton is computed once for all and is precise enough to prove the secrecy, the authentication and the 'anti-replay' property at the same time.

Those property could have also been proved using a proof assistant like in [Pau97,Bol96] since induction is well adapted to handle properties where freshness of keys and nonces is crucial. Nevertheless, we believe that using a proof assistant would have needed more time and energy to achieve the same goal. Note that a more recent approach consists in using both abstract interpretation and proof assistants. In [OS01], starting from a Isabelle/HOL specification of the protocol, some lemmas about secrecy are proved using Timbuk [5] and then integrated as axioms into Isabelle/HOL for proving some other security properties. In [BLP03], abstract interpretation is used to construct a proof of secrecy which is to be certified afterwards with the Coq proof assistant.

6 Conclusion

To conclude, this experimentation revealed several benefits of using the approximation technique and the Timbuk tool for verifying this protocol. First of all, the specification language (first order rewrite rules) are general enough to capture the protocol particularities and simple enough to convince the protocol designers. Secondly, the specification is executable with Timbuk and thus permits to detect very early some specification omissions (like it was shown in section 4) or even attacks. Finally, using rewrite approximations permits to prove automatically the property in a general setting: unbounded number of sessions, unbounded number of intruder actions (and even unbounded number of agents [GK00]).

On the opposite, the formal model used for the verification, namely tree automata, revealed to be to hard for the protocol designers to handle. Thus, we

[5] where approximation rules are automatically generated from the specification.

plan to develop a verification tool based on Timbuk, with algorithmic optimisation and a higher level description language for protocols, for sets of terms and for approximation rules.

References

[Bla01] B. Blanchet. Abstracting Cryptographic Protocols by Prolog Rules. In *In 8th International Static Analysis Symposium (SAS'01)*, volume 2126 of *LNCS*, pages 433–436. Springer-Verlag, 2001.

[BLP03] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based Abstraction for Verifying Secrecy in Protocols. In *Proceedings of TACAS'03*, volume To appear of *LNCS*, 2003.

[Bol96] Dominique Bolignano. An Approach to the Formal Verification of Cryptographic Protocols. In *ACM Conference on Computer and Communications Security*, pages 106–118, 1996.

[CDM+00] I. Cervesato, N.A. Durgin, J.C. Mitchell, P.D. Lincoln, and A. Scedrov. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In *13-th IEEE Computer Security Foundations Workshop*, pages 35–51, Cambridge, U.K., July 2000.

[Cir99] H. Cirstea. Specifying authentication protocols using elan. Technical Report 99-R-246, LORIA, 1999.

[CM01] Catherine Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. *LNCS*, 2052:237–250, 2001.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. 2nd WRLA Workshop, Pont à Mousson (France)*, 1998.

[DVB] Digital Video Broadcasting Project. `http://www.dvb.org`.

[DY83] D.Dolev and A. Yao. On the security of public key protocols. In *Proc. IEEE Transactions on Information Theory*, pages 198–208, 1983.

[Gen98] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[GK00] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000.

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *In Proceedings of 15th IPDPS Workshop*, volume 1800 of *LNCS*, pages 977–984, 2000.

[GVTT01] T. Genet and Valérie Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with timbuk. In *Proc. 8th LPAR Conf., Havana (Cuba)*, volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proceedings of 7th Conference on Logic for Programming and Automated Reasoning*, volume 1955 of *LNAI*. Springer-Verlag, 2000.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Proc. 2nd TACAS Conf., Passau (Germany)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.

[Mea96] Catherine Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2), February 1996.

[Mon99] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Proc. 6th SAS, Venezia (Italy)*, 1999.

[OS01] F. Oehl and D. Sinclair. Combining two approaches for the formal verification of cryptographic protocols. In *Proceedings of ICLP Workshop on Specification, Analysis and Validation for Emerging technologies in computational logic*, 2001.

[Pau97] L. Paulson. Proving Properties of Security Protocols by Induction. In *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[Sma01] Smartright Technical White Paper v1.0. Thomson, October 2001. http://www.smartright.org.