Special Topics in Security and Applied Logics I

Deducibility constraint systems. Verification of cryptographic properties

Mocanu Alexandru
University of Bucharest, Security and Applied Logics, Group 510

January 2022

Abstract

The verification of security protocols can be challenging because it involves skills from cryptography as well as from formal verification methods. Multiple techniques have been developed from both sides and ways to combine them have been investigated. There has been reached a point where formal verification models could have been used with no regard to cryptographic functions supposing that they respect a certain amount of security defined by a model from cryptography. Such verification models are deducibility constraint systems that are presented in this paper with no prior knowledge about them required. In addition, we study a specific issue in protocol verification, related to its interaction with cryptography. Even if strong cryptographic primitives are used, formal verification models should be extensible in order to allow verification of certain cryptographic properties. Such examples are presented in this paper, as well as their approach using the previously defined constraint system.

1 Introduction

Program verification is an important component in the process of software development and its importance has increased even more with the growing number of users. That is why it becomes a key component when talking about communication protocols because of their usage at a large scale and also due to the numerous threats that are present in the environment. Comparing to a server based application in which the developer can control the context and should only check the entry and exit points from the system, inside a network the attack surface is considerably larger. This aspect is modeled in logic using a Dolev-Yao adversary [5], that has the capability to control all the messages from the network regardless of their source or destination. Beside that, the intruder can also have control over the corrupted agents, which means he can access their knowledge and influence their actions (send, receive).

Different approaches have been taken for modeling the context of protocol execution. Beside the deducibility constraint system that will be presented in this paper, we should also mention the transition system model in which the states hold the adversary knowledge as well as the run status of the protocol. Both of these models as well as others have been implemented in tools such as AVISPA [2] and Scyther [4]. In this paper the focus will be on the constraint systems, starting with their definition and a method to reduce a constraint system to a simple form called solved.

Another important aspect of the paper is related to the cryptographic properties that can be expressed in the proposed model. Even though, in general, the formal methods for verifying protocols assume that the cryptographic primitives used are strong, particular flaws can appear when this primitives are used in a more complex system. That is why it is important to have an extensible model. Examples of such cryptographic particularities, key cycles and key ordering, will be analyzed.

As an outline of the paper, we will present the defining elements for a deducing constraint systems in section 2. The section 3 is reserved for the simplification techniques that can be used to reduce an arbitrary constraint system to its equivalent in solved form. Cryptographic applications, key cycles and key orderings, together with the deducible constraint systems are explored in section 4. An important remark is that the majority of the results presented in these sections are synthesized from the work of Hubert Comon-Lundh, Véronique Cortier and Eugen Zălinescu [3].

2 Deducibility constraint systems

We will begin by defining the syntax of the elements used in a constraint system. Let us denote $\mathcal{S} = \{s, s_1, ...\}$ the set of sorts (similar to types from a programming language), $\mathcal{X} = \{x, y, ...\}$ a set of variables and $\mathcal{N} = \{a, b, ...\}$ a set of names (used to identify agents, messages and so on). Each variable and each name will have a sort associated. As example of sorts, Msg, Key and Timestamp will be used, but other can be defined for various purposes. As function symbols, the following will be used: enc(m,k) for symmetric encryption, enca(m,k) for asymmetric encryption, sign(m,k) is the signature for message m using the key k, priv(a) represents the private key of the agent a, $\langle m_1, m_2 \rangle$ represents the pairing between the two messages. An arity function, ar, is used to express the signature for each function. For example, $ar(enc) = s_1 \times s_2 \rightarrow s$. For simplicity, the name of the agent and its public key will be used interchangeably.

Definition 2.1. *The terms for a sort s are defined by induction using the rules:*

- *x is a term, where x is a variable of sort s*
- *a is a term, where a is a name of sort s*
- *$f(t_1, t_2, ..., t_n)$ is a term, where f is a function with $ar(f) = s_1 \times ... \times s_n \rightarrow s$ and each term t_i has the sort s_i*

To denote the set of variables from a term t, we will use $\mathcal{V}(t)$. The definition can be extended for a set of terms, for which the set of variables will be equal with the union of the sets corresponding to each term. The subterms of a term t, or for a set of terms T, will be noted with $St(t)$, respectively $St(T)$. Also, we need to define substitutions σ as functions from variables to terms, written as $\sigma = \{{}^{t_1}/_{x_1}, ..., {}^{t_n}/_{x_n}\}$. The application of such a substitution to a term t is noted with $\sigma(t)$ or $t\sigma$.

Next step is to model the intruder capabilities. In the deducibility constraint systems, we use the rules from the Dolev-Yao adversaries. Let us define the actions allowed for the attacker by the following deduction rules:

$$\frac{S \vdash x \quad S \vdash y}{S \vdash \langle x, y \rangle} \qquad \text{can pair two known messages}$$

$$\frac{S \vdash x \quad S \vdash y}{S \vdash enc(x, y)} \qquad \text{can encrypt a known message with a known symmetric key}$$

$$\frac{S \vdash x \quad S \vdash y}{S \vdash enca(x, y)} \qquad \text{can encrypt a known message with a known asymmetric key}$$

$$\frac{S \vdash x \quad S \vdash y}{S \vdash sign(x, y)} \qquad \text{can sign a known message with a known signing key}$$

$\dfrac{S \vdash \langle x, y\rangle}{S \vdash x}$	can read the first message from a known pair
$\dfrac{S \vdash \langle x, y\rangle}{S \vdash y}$	can read the second message from a known pair
$\dfrac{S \vdash enc(x, y) \quad S \vdash y}{S \vdash x}$	can decrypt a known message with a known symmetric key
$\dfrac{S \vdash enca(x, y) \quad S \vdash priv(y)}{S \vdash x}$	can decrypt a known message with a known asymmetric key
$\dfrac{S \vdash sign(x, y)}{S \vdash x}$	can find the message from a known signature of it

The last deduction rule is optional, but it will be used since, in general, digital signatures do not assure cryptographic confidentiality. Using these deduction rules, proofs can be written for a term u starting from a set of finite terms T as axioms and the conclusion of the proof is denoted by $T \vdash u$.

Definition 2.2. *A proof can be defined inductively as follows:*

- *if $u \in T$, then $T \vdash u$ is a proof (because u is an axiom)*
- *if $\pi_1, ..., \pi_n$ are proofs with the respective conclusions $T \vdash u_1, ..., T \vdash u_n$ and there is a deduction rule with n premises ($\dfrac{S \vdash t_1 \ \ ... \ \ S \vdash t_n}{S \vdash t}$) for which there exists a substitution σ such that $\sigma(t_1) = u_1, ..., \sigma(t_n) = u_n$, then we have a proof $\dfrac{\pi_1 \ \ ... \ \ \pi_n}{T \vdash \sigma(t)}$ for $T \vdash \sigma(t)$*

Using the Dolev-Yao model, the adversary has control over the network, meaning that he can drop, alter, eavesdrop or create messages. These actions are possible in a network protocol because even though the format of a message is known by both the sender and the receiver, parts from the message content are unknown to the receiver. Usually, we mark that part of the message with a variable. For example, we can refer to the Needham-Schroeder protocol [6], in which agent A send to agent B the message $enca((A, n_a), B)$, but before agent B receives the message, he only knows that the message should have the format $enca((x, y), B)$. Below it can be seen the difference between the perspective of agent A and agent B during the protocol:

Agent A	**Agent B**
sends $enca((A, n_a), B)$	receives $enca((x, y), B)$
receives $enca((n_a, z), A)$	sends $enca((y, n_b), A)$
sends $enca(z, B)$	receives $enca(n_b, B)$

An approach to model the actions that can be executed by an intruder is the deducible constraint systems. In this model, we define at each step the knowledge of the attacker as well as the message format that he should produce to advance the attack. Going back to the Needham-Schroeder protocol, let us refer to a run in which we have three agents a, b and i from which the first two are honest and the last one is corrupted. In our run we will have 2 instantiation of the protocol: one between a and i, and one between a and b where the attack will take place. At the initial step, the attacker only know the public keys of all agents and the private key of the corrupted one ($T_0 = \{a, b, i, priv(i)\}$). After this, agent a sends the message $enca((a, n_a), i)$ to initiate a session with agent i. This new message is added to the initial knowledge of the attacker ($T_1 = \{a, b, i, priv(i), enca((a, n_a), i)\}$). An important remark is that even if the message wouldn't have been directed towards a corrupted agent, we should've added the new message regardless.

That is because the intruder has complete control over the network, so he can eavesdrop all the messages. At this point, it is time for the attacker to generate the first message from the second session, and that should be write as a deducibility constraint:

$$T_1 \Vdash enca((a, x), b)$$

After that we continue the execution alternating the messages from the honest agents with the ones generated by the intruder. Each message sent by an honest agent is added to the knowledge of the attacker and for each message generated by a corrupted agent we write a deducibility constraint obtaining the following rules:

$$T_2 = \{a, b, i, priv(i), enca((a, n_a), i), enca((x, n_b), a)\} \Vdash enca((n_a, y), a)$$

$$T_3 = \{a, b, i, priv(i), enca((a, n_a), i), enca((x, n_b), a), enca(y, i)\} \Vdash enca(n_b, b)$$

Definition 2.3. *A deducibility constraint system C is a finite set of expressions $T \Vdash u$, where T is a finite non empty set of terms and u is a term such that:*

1. *The sets of terms are totally ordered with regard to inclusion.*
2. *For every variable $x \in \mathcal{V}(T)$ for some $(T \Vdash u) \in C$, the set $T_x = min\{T^{'} | (T^{'} \Vdash u^{'}) \in C, x \in \mathcal{V}(u^{'})\}$ exists and $T_x \subsetneq T$.*

Intuitively, the first rule says that the attacker doesn't "forget" what he knows and that his knowledge can only increase. While the second rule states that a variable should appear first in the right-hand side of a constraint. That is because we want the attacker to have complete control over the network and the messages that are sent across it. A variable that appears for the first time in the left-hand side corresponds to a value given by an honest agent, therefore it cannot be controlled by the attacker.

Definition 2.4. *A solution for a deducibility constraint system C is a subtitution σ which interprets all variables from $\mathcal{V}(C)$ and for every $(T \Vdash u) \in C$, we have $\sigma(T) \vdash \sigma(u)$.*

With this in mind, we can see a solution for the Needham-Schroeder protocol as $\sigma = \{{}^{n_a}/_{x}, {}^{n_b}/_{y}\}$.

Deducibility constraint systems are not enough to detect vulnerabilities in protocol. That is because even though an attacker can forge packets, this doesn't necessary mean that result in a breach of authenticity, confidentiality or other security properties. That's why we need to express these properties through a formula.

Definition 2.5. *Let us take a deducibility constraint system C and a security property ϕ. We call an attack for ϕ and C, a closed substitution σ (that interprets all variables) from $\mathcal{V}(\phi) \cup \mathcal{V}(C)$ and is a solution for both ϕ and C.*

3 Simplifying deducibility constraint systems

The goal of this chapter is to present a way in which the deducibility constraint systems can be reduced to a simpler form that will have an immediate solution.

Definition 3.1. *A deducibility constraint system is solved if it is $\perp$ or all of its constraints have a variable in the right-hand side.*

If a deducibility constraint system has no constraints, then it is in solved form.

Sentence 3.2. *Except for $\perp$, all solved deducibility constraint systems have a solution.*

Proof. Let us take T_1 the smallest set of terms (from the left-hand side) from the constraint system. Because of the second rule from definition 2.3, T_1 can't contain variables and, also, must be non empty. Therefore, it contain a term t which can be used to define the substitution σ such that $\sigma(x) = t$, for each variable x. Since for every $i \in \mathbb{N}$, we have $T_1 \subseteq T_i$, then $t \in T_i$. Because of this, for every constraint $T \Vdash x$, $T \vdash \sigma(x) = t$ can be proved using the axiom rule. $\square$

To transition from an arbitrary constraint system to one that is in solved form, we will apply sequentially simplification rules. These were designed to conserve the solution of a deducibility constraint system.

R_1	$C \wedge T \Vdash u \rightsquigarrow C$	if $T \cup \{x \vert (T^{'} \Vdash x) \in C, T^{'} \subsetneq T\} \vdash u$
R_2	$C \wedge T \Vdash u \rightsquigarrow_\sigma \sigma(C) \wedge \sigma(T) \Vdash \sigma(u)$	if $\sigma = mgu(t, u)$, $t \in St(T)$, $t \neq u$, t,u not variables
R_3	$C \wedge T \Vdash u \rightsquigarrow_\sigma \sigma(C) \wedge \sigma(T) \Vdash \sigma(u)$	if $\sigma = mgu(t_1, t_2)$, $t_1, t_2 \in St(T)$, $t_1 \neq t_2$, t_1,t_2 not variables
$R_3^{'}$	$C \wedge T \Vdash u \rightsquigarrow_\sigma \sigma(C) \wedge \sigma(T) \Vdash \sigma(u)$	if $\sigma = mgu(t_2, t_3)$, $enca(t_1, t_2), priv(t_3) \in St(T)$, $t_2 \neq t_3$, t_2 or t_3 (or both) is a variable
R_4	$C \wedge T \Vdash u \rightsquigarrow \perp$	if $\mathcal{V}(T, u) = \emptyset$ and $T \nvdash u$
R_f	$C \wedge T \Vdash f(u, v) \rightsquigarrow C \wedge T \Vdash u \wedge T \Vdash v$	for $f \in \{\langle,\rangle; enc; enca; sign\}$

In the above rules, we noted with $C^{'} = C \wedge T \Vdash u$ if the system $C^{'}$ is obtained by adding the constraint $T \Vdash u$ to C. With the notation $mgu(t, u)$ we denote the most general unifier of the terms t and u. Looking informally at the simplification rules, we can observe that R_1 eliminates a constraint, which is a consequence of other constraints. The rule R_2 tries to match a message that will be sent by a corrupted agent with one of the subterms from the knowledge of the attacker. Similarly, rules R_3 and $R_3^{'}$, tries to match subterms from the knowledge. Rule R_4 offers the condition for a constraint system to be unsatisfiable. The last set of rules R_f gives are unpacking rules, based on the idea that the only posibility to know the results of the functions is to know both of its arguments.

We define $\rightsquigarrow^*$ the transitive and reflexive closure of the relation $\rightsquigarrow$. More than that, we write $C \rightsquigarrow_\sigma^* C^{'}$, where σ is the substitution obtained by composing all the intermediary substitutions used to made the transition from C to $C^{'}$. In addition, if we want to specify that the number of steps are n, we may write $C \rightsquigarrow_\sigma^n C^{'}$.

Theorem 3.3. *Let C be a deducibility constraint system, θ a substitution and ϕ a security property.*

1. *(Correctness) If $C \rightsquigarrow_\sigma^* C^{'}$ for some deducibility constraint system $C^{'}$ and some substitution σ. If θ is an attack for $\sigma(\phi)$ and $C^{'}$, then $\sigma\theta$ is an attack for ϕ and C.*
2. *(Completeness) If θ is an attack for ϕ and C, then there exists a deducibility constraint system $C^{'}$ in solved form and substitutions σ, $\theta^{'}$ such that $\theta = \sigma\theta^{'}$, $C \rightsquigarrow_\sigma^* C^{'}$ and $\theta^{'}$ is an attack for $\sigma(\phi)$ and $C^{'}$.*
3. *(Termination) There is no infinite derivation sequence $C \rightsquigarrow_{\sigma_1} C_1 \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_n} C_n \ldots$*

The previous result was proven by the authors of [3] and is very important because it validates the method of simplification. More precisely, it sais that the method is correct (no substitution is wrongfully reported as a solutions), complete (all solutions are found) and it terminates (for any system, the algorithm offers a solution). The only thing that remains to be analyzed is the time efficiency of the method, which it can be seen that in some cases can grow exponentially. An improvement to this approach has been designed by adding memorization. Basically, this technique keeps track of all constraints that were simplified or removed and prevents us from reprocessing them. Instead, because we know that the result of a simplification is deterministic, we can just remove that constraint from the system. By updating the transformation with the use of memorization, we obtain: if $C \rightsquigarrow_\sigma^* C^{'}$ then $C; D \rightsquigarrow_\sigma^* C^{'} \setminus D; D \cup (C \setminus C^{'})$.

Theorem 3.4. *Let C be a deducibility constraint system, θ a substitution and ϕ a security property.*

1. *(Correctness) If $C; \emptyset \rightsquigarrow_\sigma^* C^{'}; D^{'}$ for some deducibility constraint system $C^{'}$ and some substitution σ. If θ is an attack for $\sigma(\phi)$ and $C^{'}$, then $\sigma\theta$ is an attack for ϕ and C.*
2. *(Completeness) If θ is an attack for ϕ and C, then there exists a deducibility constraint system $C^{'}$ in solved form, a set of deducibility constraints $D^{'}$ and substitutions σ, $\theta^{'}$ such that $\theta = \sigma\theta^{'}$, $C; \emptyset \rightsquigarrow_\sigma^* C^{'}; D^{'}$ and $\theta^{'}$ is an attack for $\sigma(\phi)$ and $C^{'}$.*
3. *(Termination) If $C; \emptyset \rightsquigarrow_\sigma^n C^{'}; D^{'}$ for some deducibility constraint system $C^{'}$ and some substitution σ, then n is polinomially bounded in the size of C.*

This theorem is very similar to 3.3, the theorem that didn't contain the memorization component. It can be observed that by adding a "memory", the cases in which the complexity grows exponentially have been eliminated.

4 Applications to key cycles

After defining a model for verification of protocols, the main focus is to see how expressive the model is regarding the properties that can be checked. An area of particular interest is that of properties related to cryptography. That is because many models initially considered the cryptographic functions used in them as being safe enough. With time, multiple weaknesses have been found regarding encryption schemes or other cryptographic systems. Thus, a requirement for verification models to be able to include such properties has arisen.

An example of such cryptographic properties is related to key cycles. There are two definitions that were used in literature and to which we are going to refer. But before looking at the definitions, informally, key cycles can be viewed as a relation between cipher keys regarding which key is protecting (encrypting) which key.

For this part we will asume that beside the implicit sort called Msg used until now, we will also have another sort called Key and Key $\subset$ Msg. Let us define the following relation $\sqsubseteq$ as the least transitive and reflexive relation for which $s_1 \sqsubseteq \langle s_1, s_2 \rangle$, $s_2 \sqsubseteq \langle s_1, s_2 \rangle$ and $s_1 \sqsubseteq enc(s_1, s_2)$. This describes only appearances of a term that are not in key positions. For every list of terms L, we associate a set of terms called L_S. Also, for a set of terms S, we can define the $hidden(S) = \{k \in St(S) | k \text{ of sort Key}, S \nvdash k\}$. The hidden set holds the keys that cannot be deduced from the set. We also define an encryption relation between keys, for a given set of terms S and a given set of keys K, noted with $k\rho k^{'}$, if there exists a term $m^{'}$ such that $enc(m^{'}, k) \sqsubseteq m$ and $k^{'} \sqsubseteq m^{'}$.

Definition 4.1. *Let us take S a set of terms and K be a set of keys. We say that S contain a strict key cycle on K, if there exists a cycle in the "encryption" relation ρ restricted to K.*

To this definition we associate a security predicate P_{skc} define as: $L \in P_{skc}$ if and only if the set $\{m | L_S \vdash m\}$ has a strict key cycle on hidden(L_S). This definition corresponds to the one presented in [1].

Definition 4.2. *Let us take S a set of terms and K be a set of keys. We say that S does not contain a key cycle on K, if there exists a strict partial ordering relation $<$ on K such that for any appearance of a key $k \in K$ there exists a key $k^{'} \in K$ such that $k^{'} < k$ and $k^{'}$ encrypts the key k. In any other case, S contains a key cycle on K.*

Similar to the strict key cycle definition, we attach a predicate P_{kc} define as: $L \in P_{kc}$ if and only if the set $\{m|L_S \vdash m\}$ has a key cycle on hidden(L_S).

Beside the fact that we want to avoid key cycles because they are not computationally secure, there is another property related to keys that is worth being analyzed and that is key ordering. A case in which this property may be useful is when we want to have forward secrecy, meaning that we don't want a key to encrypt a newer key because if the old key will be broken, then the new key will be disclosed, too.

Definition 4.3. *Let S be a set of terms, K a set of keys and $<$ a strict partial ordering relation on it. We say that S is compatible with $<$ on K if $k \rho k^{'} \Rightarrow k^{'} \not< k$, for all k, $k^{'} \in K$.*

As for the key cycle, we can also model the key ordering using a predicate $P_<$ defined as: $L \in P_<$ if and only if the set $\{m|L_S \vdash m\}$ is compatible with $<$ on hidden(L_S). Referring to our previous example with the forward secrecy, we can define the relation $k < k^{'}$, if k is newer than $k^{'}$.

Sentence 4.4. *Let C be a solved deducibility constraint system, L a list of messages such that $\mathcal{V}(L_S) \subseteq \mathcal{V}(C)$ and $lhs(C) \subseteq L_S$ ($lhs(C)$ is the union of all sets of terms appearing in the left-hand side of C), and $<$ a strict partial ordering relation on a set of keys. Deciding whether there exists an attack for C and $P(L)$ can be done in $\mathcal{O}(|L^2|)$, for any $P \in \{P_{kc}, P_{skc}, \neg P_<\}$.*

The proof for this sentence can be found in [3]. This sentence offers an upper bound for the complexity of an algorithm that checks key cycles, strict key cycles and key orders incompatibilities. More than that, the proof offers a constructive way to find the attack, if it exists.

5 Conclusion

We have shown in this paper what defines a deducible constraint system and how it can be used to verify security protocols and find possible attacks for invalid ones. A method running in deterministic polynomial time for simplifying a constraint system to a trivial one with immediate solution has been given. In addition, we presented ways in which the constraint system can be used with formulas describing cryptographic related properties and how this properties can be checked in quadratic time.

Apart from these, the authors of [3] also show an example on how the authentication can be expressed using a basic logic which contains the connectives from the propositional logic, the notion of terms defined for deducibility constraint systems and the equality of terms. Another example showed how a new sort, called Time, can be integrated into the constraint system which can help to represent notions such as a fresh value or an alive connection.

References

[1] Martın Abadi and Phillip Rogaway. “Reconciling two views of cryptography”. In: *Proceedings of the IFIP International Conference on Theoretical Computer Science*. Springer. 2000, pp. 3–22.

[2] Alessandro Armando et al. “The AVISPA tool for the automated validation of internet security protocols and applications”. In: *International conference on computer aided verification*. Springer. 2005, pp. 281–285.

[3] Hubert Comon-Lundh, Véronique Cortier, and Eugen Zălinescu. “Deciding security properties for cryptographic protocols. application to key cycles”. In: *ACM Transactions on Computational Logic (TOCL)* 11.2 (2010), pp. 1–42.

[4] Cas JF Cremers. “The Scyther Tool: Verification, falsification, and analysis of security protocols”. In: *International conference on computer aided verification*. Springer. 2008, pp. 414–418.

[5] Danny Dolev and Andrew Yao. “On the security of public key protocols”. In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[6] Roger M Needham and Michael D Schroeder. “Using encryption for authentication in large networks of computers”. In: *Communications of the ACM* 21.12 (1978), pp. 993–999.