Efficient Decision Procedures for Message Deducibility and Static Equivalence

Bruno Conchinha[1], David Basin[2], and Carlos Caleiro[3]

[1] Information Security Group, ETH Zürich, Zürich, Switzerland[1,2]
brunoco@inf.ethz.ch[1], basin@inf.ethz.ch[2]
[2] SQIG - Instituto de Telecomunicações, Department of Mathematics, IST, TU Lisbon, Portugal
ccal@math.ist.utl.pt[3]

Abstract. We consider two standard notions in formal security protocol analysis: message deducibility and static equivalence under equational theories. We present polynomial-time algorithms for deciding both problems under subterm convergent equational theories and under a theory representing symmetric encryption with the prefix property. For subterm convergent theories, polynomial-time algorithms for both problems are well-known. However, we achieve a significantly better asymptotic complexity than existing approaches. For the prefix theory, we are not aware of any polynomial-time algorithms for static equivalence.
As an application, we use our algorithm for static equivalence to discover off-line guessing attacks on the Kerberos protocol when implemented using a symmetric encryption scheme for which the prefix property holds.

Key words: security protocols, equational theories, deducibility, static equivalence

1 Introduction

Formal methods and associated tools are now well established for analyzing security protocols. In symbolic approaches, the messages exchanged by agents are represented by terms in an algebra and properties of cryptographic operators are formalized equationally. This sets the scene for different analysis approaches.

Message deducibility and static equivalence are two relations, formulated in the equational setting, with direct applications to security protocol analysis. Procedures for analyzing the security of cryptographic protocols use algorithms for deduction [5, 6, 14, 17], and static equivalence has been used to study cryptographic indistinguishability [9] and to model and reason about off-line guessing attacks [1, 7, 20].

Contributions In this paper, we give new algorithms for deciding both deduction and static equivalence in two practically relevant cases: subterm convergent theories and theories for symmetric encryption with the prefix property. Our algorithms have better asymptotic complexity than existing approaches.

Our algorithms for the prefix theory are a simple generalization of those for subterm convergent theories. To the best of our knowledge, no polynomial-time algorithm for static equivalence under such theories was previously known.

As an application, we analyze the security of the Kerberos protocol against off-line guessing attacks. Although it was previously known that Kerberos admits an off-line guessing attack [24], we are able to find numerous new attacks by considering an implementation with a symmetric encryption scheme for which the prefix property holds, *e.g.*, if ECB or CBC modes of operation are used. Identifying such attacks highlights design and implementation issues (such as message ordering, modes, etc.) that may not appear important. Automating such analysis requires static equivalence procedures for different theories and highlights the importance of efficient, automated methods for this task.

Background and Related Work The notions of deduction and static equivalence that we consider were originally introduced in the context of the pi-calculus [4].

Considerable work has been devoted to proving the decidability of both problems for a wide range of equational theories. [2] gives polynomial-time algorithms for solving both problems under equational theories generated by subterm convergent rewriting systems. For other equational theories there are few polynomial-time results; however, decidability results exist under fairly general conditions, e.g., [3, 14–18, 22, 23, 25–27].

Despite the considerable prior work in this area, there have been few implementations of these algorithms, particularly for static equivalence. For some time, ProVerif [10, 11] was the only tool capable of deciding static equivalence. General algorithms for deduction and static equivalence have more recently been implemented by the YAPA [8] and KISS [16] tools. Although the precise set of equational theories and conditions under which these algorithms terminate is unclear, they seem able to handle most theories previously studied. We provide a detailed comparison of our algorithms with those implemented by these tools.

Even less attention has been devoted to improving the efficiency of algorithms for these problems. Given the interest in tools for solving these problems, complexity issues are not only theoretically interesting but also practically relevant.

Organization In Section 2, we introduce basic definitions and notation. In Section 3, we present our algorithms and their properties. In Section 4, we extend the methods of Section 3 to handle symmetric encryption with the prefix property. As a case study, we analyze the security of the Kerberos protocol against off-line guessing attacks. We draw conclusions in Section 5. For reasons of space and readability, full proofs are given in the technical report [19].

2 Background and Basic Definitions

Given a function g, we denote by $dom(g)$ and by $ran(g)$ its domain and range, respectively. When $X \subseteq dom(g)$, we write $g[X]$ for the image of X under g.

We consider signatures $\Sigma = \biguplus_{n\in\mathbb{N}} \Sigma_n$ consisting of a finite number of function symbols, where Σ_i contains the functions symbols of arity i. For each $f \in \Sigma$, the

function $\mathsf{ar}\colon \Sigma \to \mathbb{N}$ returns the arity $\mathsf{ar}(f)$ of f. We also fix infinite, disjoint sets Var and Name of variables and names. Intuitively, names represent fresh data (such as nonces) and constant symbols (*i.e.*, symbols in Σ_0) represent publicly known constants. We assume that $x, y, z \in \mathsf{Var}$ and that $\{x_i \mid i \in \mathbb{N}\} \subseteq \mathsf{Var}$.

Example 1. The signature $\Sigma^{\mathcal{DY}}$, representing a Dolev-Yao model with an hash function h, a pairing function pair, the projections π_1 and π_2, and symmetric and asymmetric encryption and decryption, is given by $\Sigma^{\mathcal{DY}} = \Sigma_1^{\mathcal{DY}} \cup \Sigma_2^{\mathcal{DY}}$, where $\Sigma_1^{\mathcal{DY}} = \{\mathsf{h}, \pi_1, \pi_2, \mathsf{pub}, \mathsf{priv}\}$ and $\Sigma_2^{\mathcal{DY}} = \{\mathsf{enc_{sym}}, \mathsf{dec_{sym}}, \mathsf{pair}, \mathsf{enc_{asym}}, \mathsf{dec_{asym}}\}$.

The functions pub and priv represent the generation of public and private keys. We will use the following abbreviations: x_{pub} for $\mathsf{pub}(x)$; x_{priv} for $\mathsf{priv}(x)$; $\langle x, y\rangle$ for $\mathsf{pair}(x, y)$; $\{P\}^{s}_{K}$ for $\mathsf{enc_{sym}}(P, K)$; $\{C\}^{s,-1}_{K}$ for $\mathsf{dec_{sym}}(C, K)$; $\{P\}_{K}$ for $\mathsf{enc_{asym}}(P, K)$; $\{C\}^{-1}_{K}$ for $\mathsf{dec_{asym}}(C, K)$; and $\langle x_1, \ldots, x_n\rangle$ for $\langle \ldots \langle x_1, x_2\rangle \ldots, x_n\rangle$.

As usual, given a set X, $T(\Sigma, X)$ is the set of Σ-terms over X, *i.e.*, the smallest set such that $X \subseteq T(\Sigma, X)$ and $f(t_1, \ldots, t_n) \in T(\Sigma, X)$ for all $t_1, \ldots, t_n \in T(\Sigma, X)$ and all $f \in \Sigma_n$. We use the symbol $=$ to denote syntactic equality. Given $t \in T(\Sigma, X)$, we define the set $sub(t)$ of subterms of t as usual: if $t \in X$, then $sub(t) = \{t\}$; if $t = f(t_1, \ldots, t_n)$ for some $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T(\Sigma, X)$, then $sub(t) = \{t\} \cup \bigcup_{i=1}^{n} sub(t_i)$. We denote by $vars(t) = sub(t) \cap \mathsf{Var}$ the set of variables occurring in t.

We use the standard notion of substitution as a partial function $\sigma\colon \mathsf{Var} \nrightarrow T(\Sigma, X)$. We abuse notation by using the same symbol σ for a substitution and its homomorphic extension to $T(\Sigma, X)$, where $dom(\sigma) \subseteq X$. As usual, we write $t\sigma$ instead of $\sigma(t)$.

A frame is a pair $(\tilde{n}, \sigma)$, written $\upsilon\tilde{n}.\sigma$, where $\tilde{n} \subseteq \mathsf{Name}$ is a finite set of names and $\sigma\colon \mathsf{Var} \nrightarrow T(\Sigma, \mathsf{Name})$ is a substitution with finite domain. Intuitively, names in $\tilde{n}$ represent fresh data generated by agents and thus unavailable to the attacker, while σ represents the messages learned by the attacker by eavesdropping on the network. Given a frame $\phi = \upsilon\tilde{n}.\sigma$, we define $T_\phi = T(\Sigma, (\mathsf{Name} \setminus \tilde{n}) \cup dom(\sigma))$. We say that terms in T_ϕ are *ϕ-recipes*. The terms in $\sigma[T_\phi]$ are the concrete terms that the attacker can obtain and we refer to them as terms *constructible* from ϕ.

A rewriting system R over Σ is a set of rewrite rules of the form $l \to r$, where $l, r \in T(\Sigma, \mathsf{Var})$. We assume that rewriting systems have only finitely many rules. Given a rewriting system R, we define the relation $\to_R \subseteq T(\Sigma, \mathsf{Name}) \times T(\Sigma, \mathsf{Name})$ as the smallest relation such that:

- if $(l \to r) \in R$ and $\sigma\colon vars(l) \to T(\Sigma, \mathsf{Name})$ is a substitution, then $l\sigma \to_R r\sigma$, and
- if $t_1, \ldots, t_n, t_i' \in T(\Sigma, \mathsf{Name})$, $t_i \to_R t_i'$, and $f \in \Sigma_n$, then $f(t_1, \ldots, t_i, \ldots, t_n) \to_R f(t_1, \ldots, t_i', \ldots, t_n)$.

We drop the R from $\to_R$ when it is clear from context.

A rewriting system R is convergent if it is terminating and confluent. In this case, each term t has a unique normal form $t{\downarrow_R} \in T(\Sigma, \mathsf{Name})$. Given a convergent rewriting system R, we define $\approx_R \subseteq T(\Sigma, \mathsf{Name}) \times T(\Sigma, \mathsf{Name})$ as

the relation such that $t \approx_R t'$ if and only if $t{\downarrow_R} = t'{\downarrow_R}$. Note that we adopt the usual convention of writing $t \approx_R t'$ instead of $(t, t') \in \approx_R$. It is simple to check that $\approx_R$ is an equational theory (*i.e.*, an equivalence relation closed under the application of contexts). We call $\approx_R$ the *equational theory generated by* R. A rewriting system R is subterm convergent if it is convergent and, for each $(l \to r) \in R$, either $r \in sub(l)$ or $r \in T(\Sigma, \emptyset)$ is a term in normal form. Permitting terms in $T(\Sigma, \emptyset)$ on the right-hand side follows [8].

Example 2. The rewriting system $R_{\mathcal{DY}}$ over $\Sigma^{\mathcal{DY}}$, formalizing the standard capabilities of the Dolev-Yao intruder, is given by

$$R_{\mathcal{DY}} = \left\{ \pi_1(\langle x, y\rangle) \to x, \pi_2(\langle x, y\rangle) \to y, \left\{\{x\}_y^s\right\}_y^{s,-1} \to x, \left\{\{x\}_{y_{\mathsf{pub}}}\right\}_{y_{\mathsf{priv}}}^{-1} \to x \right\}.$$

The rewriting system R_{p}, given by $R_{\mathsf{p}} = R_{\mathcal{DY}} \cup \{\pi_1(\{\langle x, y\rangle\}_z^s) \to \{x\}_z^s\}$, represents a Dolev-Yao attacker in the presence of symmetric encryption satisfying the prefix property. $R_{\mathcal{DY}}$ and R_{p} are convergent rewriting systems. However, only $R_{\mathcal{DY}}$ is subterm convergent. For readability, we write $\approx_{\mathcal{DY}}$ and $\approx_{\mathsf{p}}$ instead of $\approx_{R_{\mathcal{DY}}}$ and $\approx_{R_{\mathsf{p}}}$, respectively.

Our definitions of deduction and static equivalence differ slightly from those introduced in [4] and used, *e.g.*, in [2, 3, 20]. However, they are equivalent to the original ones (in particular, our definition of deduction is analogous to the characterization provided by Proposition 1 of [2]) and fit our methods better.

Definition 1. *Given a frame ϕ, a term $t \in T(\Sigma, \mathsf{Name})$, and an equational theory $\approx$, we say that t is* deducible from ϕ under $\approx$*, and write $\phi \vdash_{\approx} t$, if there is a $t' \in \sigma[T_\phi]$ such that $t' \approx t$.*

The equational theories $\approx$ that we use are those generated by rewriting systems R; thus, we write $\phi \vdash_R t$ instead of $\phi \vdash_{\approx_R} t$.

Definition 2. *Given two frames $\phi = \nu\tilde{n}.\sigma$ and $\phi' = \nu\tilde{n}'.\sigma'$ and an equational theory $\approx$, we say that ϕ and ϕ' are* statically equivalent under $\approx$*, and write $\phi \approx^s \phi'$, if $T_\phi = T_{\phi'}$ (*i.e., *$\tilde{n} = \tilde{n}'$ and $dom(\sigma) = dom(\sigma')$) and, for all $t, t' \in T_\phi$, $t\sigma \approx t'\sigma$ if and only if $t\sigma' \approx t'\sigma'$.*

The corresponding decision problems are defined as expected.

Definition 3 (Deduction Problem). *Given a frame ϕ, an equational theory $\approx$, and a term t, does $\phi \vdash_{\approx} t$ hold?*

Definition 4 (Static Equivalence Problem). *Given frames ϕ and ϕ' and an equational theory $\approx$, does $\phi \approx^s \phi'$ hold?*

Static equivalence is well-suited for modeling off-line guessing attacks [1,7,20]. Suppose that a nonce g has low entropy: for example, g is a human-picked password. Then, an attacker may choose a small set of bitstrings with a high probability of containing the bitstring represented by g. The attacker can then use each of these bitstrings as his guess for the password. The attack is successful if he can verify which of these guesses is correct. The following definition, in the spirit of [7, 20], captures this intuition.

Definition 5. *Let $\approx$ be an equational theory, $\phi = \nu\tilde{n}.\sigma$ be a frame, and $g \in \mathsf{Name}$ be a name. Fix some fresh name $w \in \mathsf{Name} \setminus (sub(ran(\sigma)) \cup \{g\})$ and define ϕ_g and ϕ_w by*

$$\phi_g = \nu(\tilde{n} \cup \{w\}).(\sigma \cup \{x_{n+1} \mapsto g\}),$$
$$\phi_w = \nu(\tilde{n} \cup \{w\}).(\sigma \cup \{x_{n+1} \mapsto w\}).$$

We say that ϕ allows an *off-line guessing attack* of g under $\approx$ *if $\phi_g \not\approx^s \phi_w$.*

Note that this definition is independent of the particular choice of the name w.

Intuitively, the attacker's guess can be seen as a message in the network. The attacker does not know beforehand if his guess is correct, but he can check this if he can distinguish a frame in which x_{n+1} stands for a random name w from a frame in which x_{n+1} stands for the guessed name g. Section 4.1 presents an application of static equivalence to the study of off-line guessing attacks.

In order to obtain polynomial complexity bounds for our algorithms, we will work with DAG (directed acyclic graph) representations of terms, as in [2].

Definition 6. *Let $t \in T(\Sigma, X)$ be a term. Let V be a set of vertices, $\lambda : V \to \Sigma \cup X$ a labeling function, $E \subseteq V \times V \times \mathbb{N}$ a set of edges, and $v \in V$ a vertex.*

If $t \in X$, then (V, λ, E, v) is a DAG-representation of t if $\lambda(v) = t$ and $(v, v', n) \notin E$ for all $v' \in V$ and all $n \in \mathbb{N}$.

If $t = f(t_1, \ldots, t_n)$, then (V, λ, E, v) is a DAG-representation of t if $\lambda(v) = f$, there are edges $(v, v_1, 1), \ldots, (v, v_n, n) \in E$ such that, for each $i \in \{1, \ldots, n\}$, (V, λ, E, v_i) is a DAG-representation of t_i, and there are no other edges $(v, v', m) \in E$ for any $v' \in V$ and any $m \in \mathbb{N}$.

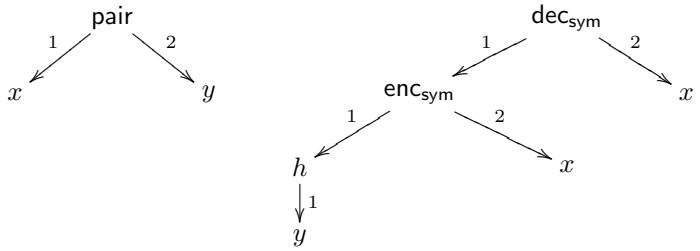
A tuple $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest if, for all $v \in V$, (V, λ, E, v) represents some term $t \in T(\Sigma, X)$. If $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest and $v \in V$, we use the following notions:

- $\mathsf{term}_{\mathcal{T}}(v)$ is the (unique) term represented by (V, λ, E, v);
- $\mathsf{e}_{i,\mathcal{T}}(v)$ is the (only) $v' \in V$ such that $(v, v', i) \in E$;
- $\mathsf{in}_{\mathcal{T}}(v) = \{w \in V \mid (w, v, i) \in E, \text{ for some } i\}$;
- $\mathsf{out}_{\mathcal{T}}(v) = \{w \in V \mid (v, w, i) \in E, \text{ for some } i\}$;
- $\mathsf{leaves}(\mathcal{T}) = \{v \in V \mid \mathsf{out}_{\mathcal{T}}(v) = \emptyset\}$;
- $\mathsf{roots}(\mathcal{T}) = \{v \in V \mid \mathsf{in}_{\mathcal{T}}(v) = \emptyset\}$.

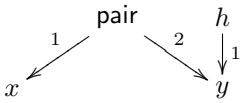
If $\mathcal{T}$ has only one root, we may refer to it as a DAG-tree.

Let $\mathcal{T} = (V, \lambda, E)$ be a DAG-forest. If $\mathsf{roots}(\mathcal{T}) = \{v\}$, we say that $\mathcal{T}$ is a DAG-representation of the term $\mathsf{term}_{\mathcal{T}}(v)$. When no confusion can arise, we may abuse notation and use the same symbol for such a DAG-forest and the term represented by it. $\mathcal{T}$ is *minimal* if there are no distinct vertices $v, v' \in V$ such that $\mathsf{term}_{\mathcal{T}}(v) = \mathsf{term}_{\mathcal{T}}(v')$. $\mathcal{T}$ is in normal form if, for each vertex v in $\mathcal{T}$, $\mathsf{term}_{\mathcal{T}}(v)$ is in normal form. A *normal form* of $\mathcal{T}$ is a DAG-forest $\mathcal{T}_{\mathsf{nf}}$ such that, for all $v \in \mathsf{roots}(\mathcal{T})$, there is a vertex v_{nf} in $\mathcal{T}_{\mathsf{nf}}$ such that $\mathsf{term}_{\mathcal{T}}(v)\!\downarrow = \mathsf{term}_{\mathcal{T}_{\mathsf{nf}}}(v_{\mathsf{nf}})$, and for each $v_{\mathsf{nf}} \in \mathsf{roots}(\mathcal{T}_{\mathsf{nf}})$, there is a $v \in \mathsf{roots}(\mathcal{T})$ such that $\mathsf{term}_{\mathcal{T}}(v)\!\downarrow = \mathsf{term}_{\mathcal{T}_{\mathsf{nf}}}(v_{\mathsf{nf}})$. The minimal normal form of a DAG-forest is unique up to renaming of vertices.

Example 3. The diagram



depicts a DAG-forest containing DAG-representations of the terms $\langle x, y\rangle$ and $\{\{h(y)\}_x^s\}_x^{s,-1}$. Its minimal normal form is shown below.



Our complexity results depend on the rewriting system and are stated in terms of the size of terms and frames. If $t \in T(\Sigma, \mathsf{Name})$ is a term, then the *size* $|t|$ of t is 1 if $t \in X$ and $1 + \sum_{i=1}^{n} |t_i|$ if $t = f(t_1, \ldots, t_n)$. If $\phi = \upsilon\tilde{n}.\sigma$ is a frame, then the *size* $|\phi|$ of ϕ is given by $|\phi| = \sum_{x \in dom(\sigma)} |x\sigma|$. If $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest, we define $|\mathcal{T}| = |V|$. If R is a rewriting system, we define $nvars(R) = \max_{(l \to r) \in R} |vars(l)|$. In order to take advantage of DAGs, we always assume a random-access machine model in our complexity analysis.

3 Decision Procedures for Subterm Convergent Rewriting Systems

Throughout this section we assume fixed a subterm convergent rewriting system R, such that $nvars(R) \geq 1$, and a frame $\phi = \upsilon\tilde{n}.\sigma$, such that $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ and the terms t_i are all in normal form. We also fix a set $\Upsilon = \{\tau_1, \ldots, \tau_{nvars(R)}\}$ of fresh names, *i.e.*, $\Upsilon \cap (\tilde{n} \cup sub(ran(\sigma))) = \emptyset$.

3.1 Frame Saturation

In this section, we present our frame saturation algorithm. Frame saturation is an established technique for deciding both deduction and static equivalence [2, 8, 16]. Our procedure is less general than those implemented in [8, 16], but it is more efficient for subterm convergent equational theories. The existence of a saturation, as defined below, is closely related to the property of *local stability* in [3].

Definition 7. *We say that t is a ϕ-*recipe (under R) for t' *if t is a ϕ-recipe and $t\sigma \approx_R t'$. We say that a frame $\phi_s = \upsilon\tilde{n}.\sigma_s$ is a* saturation of ϕ (under R) *if there is a ϕ-recipe for each $t \in ran(\sigma_s)$ and, for all $t \in T_\phi$, $(t\sigma)\!\downarrow \in \sigma_s[T_{\phi_s}]$.*

The following simple lemma justifies the usefulness of saturation.

Lemma 1. *Let ϕ_s be a saturation of ϕ and $t \in T(\Sigma, \mathsf{Name})$ be a term. Then, $\phi \vdash_R t$ if and only if $t{\downarrow} \in \sigma_s[T_{\phi_s}]$.*

The first step in our saturation algorithm is to build a forest $\mathcal{T}_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R})$ (line 1 in Algorithm 1). $\mathcal{T}_{\phi,R}$ is a minimal DAG-forest containing DAG representations of all terms $l\sigma_l$, where l is the left-hand side of some rewrite rule $(l \to r) \in R$ and $\sigma_l \colon vars(l) \to sub(ran(\sigma)) \cup \Upsilon$ is a substitution. We also use the functions ζ and rw. ζ is such that, for each vertex $v \in V_{\phi,R}$ representing a term $t \in ran(\sigma) \cup \Upsilon$, $\zeta(v)$ is a DAG-representation of a ϕ-recipe for t. rw is such that, for each vertex v representing a term $l\sigma_l$ as described above, $\mathsf{rw}(v)$ is the (unique) vertex representing $r\sigma_l$. Algorithms for computing $\mathcal{T}_{\phi,R}$, rw and ζ are given in the technical report [19]. Lemma 2 summarizes their relevant properties.

Lemma 2. *The forest $\mathcal{T}_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R})$ and the functions rw and ζ are such that:*

(1) *$\mathcal{T}_{\phi,R}$ is minimal, can be obtained in time $\mathcal{O}(|\phi|^{nvars(R)} \log^2 |\phi|)$, and $|\mathcal{T}_{\phi,R}| \in \mathcal{O}(|\phi|^{nvars(R)})$;*
(2) *rw can be computed in time $\mathcal{O}(\log |\phi|)$;*
(3) *ζ can be computed in time $\mathcal{O}(\log |\phi|)$; for each $v \in dom(\zeta)$, $|\zeta(v)| = 1$;*
(4) *for each $s \in sub(ran(\sigma)) \cup \Upsilon$, there is an unique v such that $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) = s$;*
(5) *for each $v \in dom(\mathsf{rw})$, $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) \to_R \mathsf{term}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v))$;*
(6) *for each $t \in ran(\sigma) \cup \Upsilon$, there is a v such that $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) = t$ and $v \in dom(\zeta)$;*
(7) *for each $v \in dom(\zeta)$, $\mathsf{term}_{\zeta(v)}(v)$ is a ϕ-recipe for $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v)$;*
(8) *for each rule $(l \to r) \in R$ and each substitution $\sigma_l \colon vars(l) \to sub(ran(\sigma)) \cup \Upsilon$, there is a unique $v \in V_{\phi,R} \cap dom(\mathsf{rw})$ such that $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) = l\sigma_l$ and $\mathsf{term}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v)) = r\sigma_l$.*

Our saturation algorithm traverses the forest $\mathcal{T}_{\phi,R}$ bottom-up. At each vertex v, whenever a recipe for $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v)$ is found, v is added to the range of ζ, and $\zeta(v)$ is a DAG-representation of a ϕ-recipe for that term. A recipe is found if one has recipes $\zeta(v_1), \ldots, \zeta(v_n)$ for all vertices v_i that have an incoming edge (v, v_i, i) from v. If the term represented by v is an instance of the left-hand side of a rule, then this recipe is also stored under $\zeta(\mathsf{rw}(v))$ (note that $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) \to_R \mathsf{term}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v))$). Thus, throughout the saturation process, the function ζ associates each vertex v in its domain to a DAG-representation of a ϕ-recipe for $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v)$. Whenever we add a vertex v to the domain of ζ, we add all vertices v' with an outgoing edge (v', v, i) to v to the list of vertices to visit in the next iteration of the visiting loop. At the end of the process, a term $t \in sub(ran(\sigma))$ is deducible from ϕ if and only if the (unique) vertex representing that term is in the domain of ζ.

The algorithm also stores the functions σ_s and ζ_s. σ_s is such that $\phi_s = v\tilde{n}.\sigma_s$ is a saturation of ϕ, and ζ_s is such that $dom(\zeta_s) = dom(\sigma_s)$ and, for each $x \in dom(\sigma_s)$, $\zeta_s(x)$ is a DAG-representation of a ϕ-recipe for $x\sigma_s$.

Furthering our abuse of notation, we use the symbol ζ_s as the substitution that assigns, to each $x \in dom(\zeta_s)$, the term represented by (the DAG-forest) $\zeta_s(x)$. In this case we use postfix notation and write $x\zeta_s$.

The tree $\mathcal{T}_{\phi,R}$ has at most $\mathcal{O}(|\phi|^{nvars(R)})$ vertices, and each vertex $v \in V_{\phi,R}$ is visited at most $|\mathsf{in}_{\mathcal{T}_{\phi,R}}(v)|$ times. Thus, the total number of visits to vertices is at most $\mathcal{O}(|\phi|^{nvars(R)})$. By using suitable data structures, we can ensure that each visit takes at most time $\mathcal{O}(\log^2 |\phi|)$. We thus obtain an asymptotic complexity of $\mathcal{O}(|\phi|^{nvars(R)} \log^2 |\phi|)$.

We state the algorithm's properties and complexity in Lemma 3.

Algorithm 1 (Saturating a Frame)
Input: *a frame* $\phi = v\tilde{n}.\sigma$*, with* $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
Output: *a saturation* $\phi_s = v\tilde{n}.\sigma_s$ *of* ϕ *and a function* ζ_s

compute $\mathcal{T}_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R}), \mathsf{rw}, \zeta$
$\zeta_s \leftarrow \{x \mapsto (\{v_x\}, \{v_x \mapsto x\}, \emptyset) \mid x \in dom(\sigma)\}$,
where the v_x *are such that* $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v_x) = x\sigma$
$\sigma_s \leftarrow \sigma$
$\mathsf{visitnow} \leftarrow \mathsf{leaves}(\mathcal{T}_{\phi,R}) \cup (\bigcup_{v \in dom(\zeta)} \mathsf{in}_{\mathcal{T}_{\phi,R}}(v))$, $\mathsf{visitnext} \leftarrow \emptyset$
while $\mathsf{visitnow} \neq \emptyset$
  for all $v \in \mathsf{visitnow}$
    if $\lambda(v) \in X \setminus \tilde{n}$ **and** $v \notin dom(\zeta)$ **then**
      $\zeta \leftarrow \zeta \cup \{v \mapsto (v, \{v \mapsto \lambda(v)\}, \emptyset)\}$
      $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,R}}(v)$
    if $\mathsf{out}_{\mathcal{T}_{\phi,R}}(v) \subseteq dom(\zeta)$ **and** $v \notin dom(\zeta)$ **then**
      let $(V_i, \lambda_i, E_i) = \zeta(\mathsf{e}_{i,\mathcal{T}_{\phi,R}}(v))$ *for* $i \in \{1, \ldots, \mathsf{ar}(\lambda_{\phi,R}(v))\}$
      $\zeta \leftarrow \zeta \cup \{v \mapsto (v \cup \bigcup_{i=1}^{\mathsf{ar}(\lambda_{\phi,R}(v))} V_i,$
        $\{v \mapsto \lambda(v)\} \cup \bigcup_{i=1}^{\mathsf{ar}(\lambda_{\phi,R}(v))} \lambda_i,$
        $\bigcup_{i=1}^{v} \{(v, \mathsf{e}_{i,\mathcal{T}_{\phi,R}}(v), i)\} \cup \bigcup_{i=1}^{\mathsf{ar}(\lambda_{\phi,R}(v))} E_i)\}$
      if $v \in dom(\mathsf{rw})$ **and** $\mathsf{rw}(v) \notin dom(\zeta)$ **then**
        $\zeta \leftarrow \zeta \cup \{\mathsf{rw}(v) \mapsto \zeta(v)\}$
        if $\mathsf{term}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v)) \in sub(ran(\sigma))$
          then *choose* $x \in \mathsf{Var} \setminus dom(\sigma_s)$
            $\sigma_s \leftarrow \sigma_s \cup \{x \mapsto \mathsf{term}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v))\}$
            $\zeta_s \leftarrow \zeta_s \cup \{x \mapsto \zeta(\mathsf{rw}(v))\}$
            $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,R}}(\mathsf{rw}(v))$
      else $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,R}}(v)$
return ζ_s, $\phi_s = v\tilde{n}.\sigma_s$

Lemma 3. *Algorithm 1 terminates in time* $\mathcal{O}(|\phi|^{nvars(R)} \log^2 |\phi|)$.

ϕ_s is a saturation of ϕ (under R), $dom(\zeta_s) = dom(\sigma_s)$, and, for each $x \in dom(\sigma_s)$, $\zeta_s(x) \in T_\phi$ and $\zeta_s(x)$ is a DAG-representation of a ϕ-recipe for $x\sigma_s$ with size $|\zeta_s(x)| \in \mathcal{O}(|\phi|)$.

For each $v \in dom(\zeta)$, there is a ϕ_s-recipe t for $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v)$ such that $\zeta(v) = t\zeta_s$ is a ϕ-recipe for $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v)$. If $\mathsf{term}_{\mathcal{T}_{\phi,R}}(v) \in \sigma_s[T_{\phi_s}]$, then $v \in dom(\zeta)$.

3.2 Decision Procedure for Deduction

In light of Lemma 1, to solve the deduction problem under R for a frame ϕ and a term t, it suffices to compute $t{\downarrow_R}$ and the saturated frame $\phi_s = \nu\tilde{n}.\sigma_s$ (using Algorithm 1) and then decide whether $t{\downarrow_R} \in \sigma_s[T_{\phi_s}]$. In the technical report [19] we show that the complexities of these three computations are, respectively, $\mathcal{O}(|t| \log^2 |t|)$, $\mathcal{O}(|\phi|^{nvars(R)} \log^2 |\phi|)$, and $\mathcal{O}(|t||\phi|^2)$. Theorem 1 sums up these observations.

Theorem 1. *Given a frame ϕ and a term t, the complexity of deciding whether $\phi \vdash_R t$ is at most $\mathcal{O}(|t| \log^2 |t| + |t||\phi|^2 + |\phi|^{nvars(R)} \log^2 |\phi|)$.*

3.3 Decision Procedure for Static Equivalence

Throughout this section, $\mathcal{T}_{\phi,R}$ is as described in the previous sections, ϕ_s and ζ_s are as output by Algorithm 1, and $\phi' = \nu\tilde{n}.\sigma'$ is a frame such that $dom(\sigma') = dom(\sigma) = \{x_1, \ldots, x_n\}$. We assume that $dom(\sigma_s) = \{x_1, \ldots, x_{m_s}\}$ and that σ_s is an extension of σ. Υ will be used as in the previous section.

Algorithm 2 summarizes our procedure for deciding static equivalence. Note that some of the operations performed by this algorithm must use the DAG-representation of terms to ensure polynomial-time complexity. For simplicity, we refer to the technical report [19] for the exposition of such details.

Algorithm 2 (Decision Procedure for $\approx_R^s$)
Input: *two frames $\phi = \upsilon\tilde{n}.\sigma$ and $\phi' = \upsilon\tilde{n}.\sigma'$ such that $dom(\sigma) = dom(\sigma') = \{x_1, \ldots, x_n\}$*
Output: **true** *if $\phi \approx_R^s \phi'$ and* **false** *otherwise*

1: *compute $\mathcal{T}_{\phi,R}$, ζ, rw, ζ_s and ϕ_s*
2: *choose a permutation $\pi\colon \{1, \ldots, m_s\} \to \{1, \ldots, m_s\}$ such that $1 \le i < j \le m_s \Rightarrow |x_{\pi(i)}\sigma_s| \le |x_{\pi(j)}\sigma_s|$*
3: *for each $k \in \{1, \ldots, m_s\}$, let $\phi_{s,k} = \upsilon\tilde{n}.\{x_{\pi(1)} \mapsto x_{\pi(1)}\sigma_s, \ldots, x_{\pi(k)} \mapsto x_{\pi(k)}\sigma_s\}$*
4: **for all** $k \in \{1, \ldots, m_s\}$
5: **if** $x_{\pi(k)}\sigma_s \in \sigma_s[T_{\phi_{s,k-1}}]$ **then**
6: *choose $t \in T_{\phi_{s,k-1}}$ such that $x_{\pi(k)}\sigma_s = t\sigma_s$*
7: **if** $x_{\pi(k)}\zeta_s\sigma' \not\approx_R t\zeta_s\sigma'$ **then return false**
8: **for all** $v \in dom(\mathsf{rw})$
9: **then if** $(\zeta(v))\sigma' \not\approx_R (\zeta(\mathsf{rw}(v)))\sigma'$
10: **then return false**
11: *Repeat once lines 1–10, exchanging ϕ and ϕ'*
12: **return true**

The first loop (lines 4–7) tests whether syntactical equality between terms yielded by two distinct ϕ-recipes under ϕ implies that these two recipes yield equationally equal terms under ϕ'. The condition in lines 8–10 tests whether there are two recipes representing instances of the left and right-hand sides of a rule under ϕ but that do not represent equal terms (modulo R) under ϕ'. If either

of the two loops outputs **false** then the two frames are not statically equivalent. Otherwise, we conclude that all equalities (between recipes, modulo R) satisfied by ϕ are also satisfied by ϕ'. Repeating the procedure, exchanging the roles of ϕ and ϕ', allows one to decide whether $\phi \approx_R^s \phi'$. The correctness of this procedure and its complexity analysis are provided by Theorem 2.

Theorem 2. *Algorithm 2 decides whether $\phi \approx_R^s \phi'$ in time*

$$\mathcal{O}((|\phi| + |\phi'|)^3 \log^2(|\phi| + |\phi'|) + (|\phi| + |\phi'|)^{nvars(R)+1} \log^2(|\phi| + |\phi'|)).$$

$\mathcal{O}((|\phi| + |\phi'|)^3)$ is an upper bound for the complexity of the first loop (lines 4–7) of Algorithm 2; $\mathcal{O}((|\phi| + |\phi'|)^{nvars(R)+1} \log^2(|\phi| + |\phi'|))$ is an upper bound for the complexity of the second (lines 8–10).

4 The Prefix Theory

We now investigate how our methods can be extended to deal with theories resulting from other convergent rewriting systems. In particular, we examine the system R_{p} presented in Example 2, which represents symmetric encryption with the prefix property. Encryption modes designed to encrypt large messages using block ciphers often have the prefix property, namely ECB and CBC. Although the decidability of this theory has been studied [14], we are not aware of polynomial-time results for static equivalence.

As before, we assume here that $\phi = \nu\tilde{n}.\sigma$ is a frame, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, and fix a set $\Upsilon = \{\tau_1, \tau_2, \tau_3\}$ of fresh names (note that $nvars(R_{\mathsf{p}}) = 3$, since $|vars(\pi_1(\{\langle x, y\rangle\}_z^s))| = 3$).

We begin by defining p-subterms, which extend the usual notion of subterm.

Definition 8. *Let $t \in T(\Sigma^{\mathcal{D}\mathcal{Y}}, \mathsf{Name})$. We define the set of* p-subterms *of t inductively as follows:*

- *if $t \in \mathsf{Name}$, then $sub_{\mathsf{p}}(t) = \{t\}$;*
- *if $t = \{\langle t_1, t_2\rangle\}_{t_3}^s$ for some $t_1, t_2, t_3 \in T(\Sigma, \mathsf{Name})$, then $sub_{\mathsf{p}}(t) = \{t, t_3\} \cup sub_{\mathsf{p}}(\{t_1\}_{t_3}^s) \cup sub_{\mathsf{p}}(\langle t_1, t_2\rangle)$;*
- *if $t = f(t_1, \ldots, t_n)$ for some f and some $t_1, \ldots, t_n$ and $t \neq \{\langle t_1', t_2'\rangle\}_{t_3'}^s$ for all $t_1', t_2', t_3' \in T(\Sigma, \mathsf{Name})$, then $sub_{\mathsf{p}}(t) = \{t\} \cup \bigcup_{i=1}^n sub_{\mathsf{p}}(t_i)$.*

We extend this definition to sets of terms as usual.

Our algorithms for the prefix theory use a forest $\mathcal{T}_{\phi,\mathsf{p}}$ analogous to the tree $\mathcal{T}_{\phi,R}$ used for subterm convergent theories. The construction process is also similar. We first compute the substitution $\sigma_{\mathsf{nf}} = \{x_1 \mapsto t_1\!\downarrow, \ldots, x_n \mapsto t_n\!\downarrow\}$. Then, we build a minimal DAG-forest $\mathcal{T}_{\phi,\mathsf{p}}$ containing DAG-representations of all terms $l\sigma_l$, where l is the left-hand side of a rewrite rule $(l \to r) \in R$ and σ_l: $vars(l) \to sub_{\mathsf{p}}(ran(\sigma)) \cup \Upsilon$. The functions rw_{p} and ζ_{p} are defined analogously to rw and ζ for the tree $\mathcal{T}_{\phi,R}$. $\mathcal{T}_{\phi,\mathsf{p}}$ satisfies properties analogous to those given in Lemma 2, after replacing some critical instances of sub by sub_{p}. A summary

of these properties and details of the construction can be found in the technical report [19].

Let Algorithm 1_{p} be Algorithm 1 after replacing R by R_{p}, $\mathcal{T}_{\phi,R}$ by $\mathcal{T}_{\phi,\mathsf{p}}$, and *sub* by sub_{p} in line 15. Algorithm 1_{p} computes a saturated frame under R_{p}. Our algorithms for deduction and static equivalence also work for the prefix theory. We summarize these results in the following theorems. Note, in our complexity analysis, that $nvars(R_{\mathsf{p}}) = 3$. Hence, the complexity of our algorithms is not affected by the fact that R_{p} is not subterm convergent.

Lemma 4. *Algorithm* 1_{p} *terminates in time* $\mathcal{O}(|\phi|^3 \log^2 |\phi|)$.

ϕ_s is a saturation of ϕ (under R_{p}), $dom(\zeta_s) = dom(\sigma_s)$, and, if $x \in dom(\zeta_s)$, then $x\zeta_s$ is a DAG-representation of a ϕ-recipe for $x\sigma_s$ with size $|x\zeta_s| \in \mathcal{O}(|\phi|)$.

For each $v \in dom(\zeta)$, there is a ϕ_s-recipe t for $\mathsf{term}_{\mathcal{T}_{\phi,\mathsf{p}}}(v)$ such that $\zeta(v) = t\zeta_s$ is a ϕ-recipe for $\mathsf{term}_{\mathcal{T}_{\phi,\mathsf{p}}}(v)$. If $\mathsf{term}_{\mathcal{T}_{\phi,\mathsf{p}}}(v) \in \sigma_s[T_{\phi_s}]$, then $v \in dom(\zeta)$.

Theorem 3. *Given a frame ϕ and a term t, the complexity of deciding whether $\phi \vdash_{R_{\mathsf{p}}} t$ is at most $\mathcal{O}(|t| \log^2 |t| + |t||\phi|^2 + |\phi|^3 \log^2 |\phi|)$.*

Theorem 4. *Algorithm 2 decides whether $\phi \approx^s_{R_{\mathsf{p}}} \phi'$ in time*

$$\mathcal{O}((|\phi| + |\phi'|)^4 \log^2(|\phi| + |\phi'|)).$$

4.1 Off-line Guessing Attacks on a Version of Kerberos

We now present multiple off-line guessing attacks on a version of Kerberos. Most of our attacks rely on the prefix property. Kerberos is known to be insecure against off-line guessing attacks — *e.g.*, [24] describes an attack relying only on standard properties of symmetric encryption that is also captured by our model. However, our formal analysis of its security when implemented with a symmetric encryption scheme satisfying the prefix property is novel.

Kerberos Variant The version of Kerberos we consider is adapted from [12]. We present a short description of the protocol, in standard notation.

1. $\mathsf{A} \to \mathsf{KAS}$: $\langle \mathsf{A}, \mathsf{KAS}, T_1 \rangle$
2. $\mathsf{KAS} \to \mathsf{A}$: $\left\{ \langle \{ \langle \mathsf{A}, \mathsf{TGS}, K_{\mathsf{A},\mathsf{TGS}}, T_2 \rangle \}^s_{K_{\mathsf{KAS},\mathsf{TGS}}}, K_{\mathsf{A},\mathsf{TGS}}, \mathsf{TGS}, T_2 \rangle \right\}^s_{K_{\mathsf{A},\mathsf{KAS}}}$
3. $\mathsf{A} \to \mathsf{TGS}$: $\langle \{ \langle \mathsf{A}, \mathsf{TGS}, K_{\mathsf{A},\mathsf{TGS}}, T_2 \rangle \}^s_{K_{\mathsf{KAS},\mathsf{TGS}}}, \{ \langle \mathsf{A}, T_3 \rangle \}^s_{K_{\mathsf{A},\mathsf{TGS}}}, \mathsf{B} \rangle$
4. $\mathsf{TGS} \to \mathsf{A}$: $\left\{ \langle K_{\mathsf{A},\mathsf{B}}, \mathsf{B}, T_4, \{ \langle \mathsf{A}, \mathsf{B}, K_{\mathsf{A},\mathsf{B}}, T_4 \rangle \}^s_{K_{\mathsf{B},\mathsf{TGS}}} \rangle \right\}^s_{K_{\mathsf{A},\mathsf{TGS}}}$
5. $\mathsf{A} \to \mathsf{B}$: $\langle \{ \langle \mathsf{A}, \mathsf{B}, K_{\mathsf{A},\mathsf{B}}, T_4 \rangle \}^s_{K_{\mathsf{B},\mathsf{TGS}}}, \{ \langle \mathsf{A}, T_5 \rangle \}^s_{K_{\mathsf{A},\mathsf{B}}} \rangle$
6. $\mathsf{B} \to \mathsf{A}$: $\{T_5\}^s_{K_{\mathsf{A},\mathsf{B}}}$

A is a name (*e.g.*, of a client of a single-sign-on service). KAS is the Kerberos authentication server. TGS is the ticket-granting server. B is some service provider. $K_{\mathsf{A},\mathsf{KAS}}$ (respectively $K_{\mathsf{KAS},\mathsf{TGS}}$, $K_{\mathsf{B},\mathsf{TGS}}$) is a long-term key shared between A and KAS (respectively between KAS and TGS and between B and TGS). $K_{\mathsf{A},\mathsf{TGS}}$

(respectively $K_{\mathsf{A,B}}$) is a short-term key shared between A and TGS (respectively between A and B), freshly generated for each session by KAS (respectively by TGS). Finally, $T_1, \ldots, T_5$ are timestamps. Note that for our purposes, it suffices to view them as freshly generated nonces, even if this representation is imprecise.

The only difference between the protocol presented and the model of Kerberos (version IV) presented in [12] is that, in the term sent in step 2, the encryption $\{\langle \mathsf{A}, \mathsf{TGS}, K_{\mathsf{A,TGS}}, T_2\rangle\}^s_{K_{\mathsf{KAS,TGS}}}$ is the first (instead of the last) term in the encrypted tuple. Although this may may appear to be a small and harmless change, it gives rise to guessing attacks if a symmetric encryption scheme with the prefix property is used. It is interesting to note that one of our attacks relies on the double encryption used in version IV of the protocol. Version V eliminates the double encryption and thereby prevents this attack.

The signature Σ^{ker} we use is obtained by simply adding the set $\Sigma_0 = \{\mathsf{A}, \mathsf{B}, \mathsf{KAS}, \mathsf{TGS}\}$ of agent names to the signature $\Sigma^{\mathcal{DY}}$. We assume that $\{T_i \mid i \in \mathbb{N}\} \subseteq \mathsf{Name}$. We also represent the long-term keys $K_{\mathsf{A,KAS}}, K_{\mathsf{KAS,TGS}} \in \mathsf{Name}$ and the short-term keys $K_{\mathsf{A,TGS}}, K'_{\mathsf{A,TGS}} \in \mathsf{Name}$ (corresponding to the keys generated by KAS for use between A and TGS in two distinct sessions) as names.

The Attacker's Knowledge We consider an attacker who eavesdrops on two different protocol sessions, both executed between an agent A and the server KAS. For simplicity, we assume that the attacker stores only the second message of each session. This is enough to present our off-line guessing attacks. We represent the attacker's knowledge by the frame $\phi = \upsilon\tilde{n}.\sigma$, where $\tilde{n} = \left\{K_{\mathsf{A,TGS}}, K'_{\mathsf{A,TGS}}, K_{\mathsf{A,KAS}}, K_{\mathsf{KAS,TGS}}, T_1, T_2\right\}$ and

$$\sigma = \{\ x_0 \mapsto \left\{\langle\{\langle \mathsf{A}, \mathsf{TGS}, K_{\mathsf{A,TGS}}, T_1\rangle\}^s_{K_{\mathsf{KAS,TGS}}}, K_{\mathsf{A,TGS}}, \mathsf{TGS}, T_1\rangle\right\}^s_{K_{\mathsf{A,KAS}}},$$
$$x_1 \mapsto \left\{\langle\left\{\langle \mathsf{A}, \mathsf{TGS}, K'_{\mathsf{A,TGS}}, T_2\rangle\right\}^s_{K_{\mathsf{KAS,TGS}}}, K'_{\mathsf{A,TGS}}, \mathsf{TGS}, T_2\rangle\right\}^s_{K_{\mathsf{A,KAS}}}\ \}.$$

We are interested in determining whether this frame allows an off-line guessing attack of $K_{\mathsf{A,KAS}}$. This is a (potentially) weak key, since it is often chosen by human users or derived from such a key. We take $g = K_{\mathsf{A,KAS}}$ and $w = a_0$ and, throughout our example, we work with the frames $\phi_g = \upsilon\tilde{n}'.\sigma_g$ and $\phi_w = \upsilon\tilde{n}'.\sigma_w$, where $\tilde{n}' = \tilde{n} \cup \{a_0\}$, $\sigma_g = \sigma \cup \{x_2 \mapsto K_{\mathsf{A,KAS}}\}$, and $\sigma_w = \sigma \cup \{x_2 \mapsto a_0\}$.

Saturation of ϕ_w and ϕ_g In [19] we present tables with the results of saturating the frames.

Off-line Guessing Attacks to Kerberos It is clear that $\phi_w \not\approx^s_{R_\mathsf{p}} \phi_g$. Thus, Kerberos allows an off-line-guessing attack of $K_{\mathsf{A,KAS}}$. In fact, an attacker has multiple pairs of recipes $t, t' \in T_{\phi_g}$ that he can use to validate his guess; we present a few illustrative examples in Table 1. Note that, of the four attacks presented, all but Attack 1 rely on the prefix property of the encryption scheme.

Table 1. (Sample) Off-line Guessing Attacks to Kerberos

Attack	t	t'
1	$\pi_2(\pi_1(\{x_0\}^{s,-1}_{x_2}))$	TGS
2	$\left\{\langle\{\pi_1(x_0)\}^{s,-1}_{x_2}, \pi_2(\{x_0\}^{s,-1}_{x_2})\rangle\right\}^{s}_{x_2}$	x_0
3	$\langle\{\pi_1(\pi_1(x_0))\}^{s,-1}_{x_2}, \mathsf{TGS}\rangle$	$\pi_1(\{x_0\}^{s,-1}_{x_2})$
4	$\pi_1(\pi_1(\pi_1(\pi_1(\{x_0\}^{s,-1}_{x_2}))))$	$\pi_1(\pi_1(\pi_1(\pi_1(\{x_1\}^{s,-1}_{x_2}))))$

Only Attack 4 relies on the fact that we use version IV instead of version V and exchange the order of the messages of the original Kerberos protocol.

How feasible are these attacks in practice? First of all, CBC encryption mode uses a random initialization vector. To prevent Attack 4 it is enough that the initialization vectors used in the two messages are not the same (even if they are public). This can be captured in our model by representing symmetric encryption as a function $\mathsf{enc_{sym}}$ with three arguments (an initialization vector, the message, and the key) and writing the rewriting rule representing the prefix property as

$$\pi_1(\mathsf{enc_{sym}}(IV, \langle M_1, M_2\rangle, K)) \to \mathsf{enc_{sym}}(IV, M_1, K).$$

Attack 4 does not arise in this model.

Furthermore, consider the recipes t and t' given in Attack 4. We have $t\sigma_g = t'\sigma_g = \{\langle \mathsf{A}, \mathsf{TGS}\rangle\}^{s}_{K_{\mathsf{KAS,TGS}}}$. However, in practice, all that the attacker can do is obtain an encryption of the first block of the plaintext. Thus, this attack is only feasible if the first encrypted block is equal in both messages. If the size of the first encrypted block is smaller than the encryption of $\langle \mathsf{A}, \mathsf{TGS}\rangle$, then the attack succeeds. Otherwise, its success depends on whether and how padding techniques are applied: if the rest of the first block is padded with a sequence of 0's, then the two blocks are equal and the attack succeeds. If the rest of the first block is padded with a random sequence of freshly generated bits or used for storing the encryption of $K_{\mathsf{A,TGS}}$, then the attack fails. Finally, note that Attack 2 (respectively 3) is only feasible if the encryption of the first three (respectively two) elements of the tuple occupies disjoint encryption blocks from the encryption of the last one (respectively two) elements. Note that modeling such details would fall outside of the scope of the theories we consider.

The relevance of details such as initialization vectors, block length, and padding techniques in the study of off-line guessing attacks has been previously pointed out [13]. We believe that it is an important challenge for symbolic methods to be able to reason about these kinds of possible weaknesses.

5 Related Work and Conclusion

Our algorithms compare favorably to previously existing algorithms. [2] presents the first proof that deduction and static equivalence under subterm convergent equational theories can be decided in polynomial-time. However, efficiency is not

their main concern and it is not surprising that our algorithms has a much better asymptotic complexity. For example, for the theory $\approx_{\mathcal{DY}}$, the complexities of our algorithms are $\mathcal{O}(|t|\log|t|+|\phi|^2|t|+|\phi|^2\log^2|\phi|)$ and $\mathcal{O}((|\phi|+|\phi'|)^3\log^2(|\phi|+|\phi'|))$ for deduction and static equivalence (respectively), whereas our best estimates for the complexity of the algorithms in [2] are $\mathcal{O}(|\phi|^{10}+|\phi|^2|t|+|t|^4)$ and $\mathcal{O}((|\phi|+|\phi'|)^{15})$ for the same problems.

The complexity of the YAPA tool [8] is not polynomial, as it uses a straightforward representation of terms, as opposed to DAGs. Furthermore, our saturation technique is also more efficient: in YAPA, for each (n,p,q)-decomposition of the left-hand side of a rewrite rule and each assignment of the first $n+p$ parameters to recipes in the frame, it may be necessary to compute the normal form of a term t. We are not aware of any general algorithms for this task that have a better complexity than $\mathcal{O}(|t|^4)$ (see discussion below). If we denote by $Y(R)$ the greatest value of $n+p$ for all (n,p,q)-decompositions of rewriting rules in R, we obtain a complexity of $\mathcal{O}(|\phi|^{Y(R)+4})$ for YAPA's saturation procedure; this is significantly worse than the complexity of $\mathcal{O}(|\phi|^{nvars(R)}\log^2|\phi|)$ achieved by our algorithm (note that $nvars(R) \leq Y(R)$ in general). For the rewriting system $R_{\mathcal{DY}}$ we obtain an estimated complexity of $\mathcal{O}(|\phi|^7)$ for the saturation procedure in YAPA and $\mathcal{O}(|\phi|^2\log|\phi|)$ for ours. Note that this estimate assumes that DAGs are implemented; the exact implementation of DAGs and the algorithms to compute normal forms may affect the complexity of the procedure. It may also be possible to provide better bounds on the number of recipes for which we need to perform this reduction to a normal form.

Our saturation procedure is also more efficient than that implemented by the KISS tool. In this tool, the rule Narrowing generates a number of deduction facts for each rewriting rule in R. If we denote by $K(R)$ the maximum number of side conditions in deduction facts generated by any rewriting rule in R, we again have $nvars(R) \leq K(R)$ in general: for example, $K(R_{\mathcal{DY}}) = 3$. The terms in these side-conditions must then be instantiated (via the rule $\mathsf{F}-\mathsf{Solving}$) with terms that are heads of some deduction fact. There are at least $\mathcal{O}(|\phi|)$ such terms, which implies that the saturated frame may have $\mathcal{O}(|\phi|^{K(R)})$ deduction facts. Testing the premise of the rules F-Solving, E-Solving, and Unifying requires selecting two deduction facts from the frame and performing an operation with linear-time complexity. Since there are $\mathcal{O}(|\phi|^{2K(R)})$ such pairs, we obtain a complexity of at least $\mathcal{O}(|\phi|^{2K(R)+1})$. For the rewriting system $R_{\mathcal{DY}}$, this amounts to a complexity of $\mathcal{O}(|\phi|^7)$ for KISS, in contrast to the complexity of $\mathcal{O}(|\phi|^2\log^2|\phi|)$ for our algorithms. Here it may also be possible to improve this complexity bound, for example by limiting the number of pairs of rules that must be tested.

Finally, we note that all the algorithms discussed here require deciding the equality of two terms t and t' under the equational theory. Our algorithms rely on the subterm convergence of the rewriting system to perform this task with complexity $\mathcal{O}((|t|+|t'|)\log(|t|+|t'|))$. This constitutes a marked advantage over algorithms for more general rewriting systems, for which we are not aware of any algorithm improving the complexity of $\mathcal{O}((|t|+|t'|)^4)$ achieved in [2].

As future work we plan to implement our algorithms. We also plan to study how our approach can be adapted to other equational theories. The simplicity of extending our methods to the prefix theory suggests that it may be possible to generalize our approach for a much larger class of equational theories, possibly improving upon existing complexity results for the two decision problems, when such results exist.

Acknowledgements

This work was partly supported by FCT and EU FEDER, namely via the project KLog PTDC/MAT/68723/2006 of SQIG-IT and the project UTAustin/MAT/0057/2008 AMDSC of IST.

References

1. Abadi, M., M. Baudet and B. Warinschi, *Guessing Attacks and the Computational Soundness of Static Equivalence*, Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS06), Lecture Notes in Computer Science **3921** (2006), 398–412
2. Abadi, M. and V. Cortier, *Deciding Knowledge in Security Protocols Under Equational Theories*, Proceedings 31st International Colloquium on Automata, Languages and Programming (ICALP'04), Lecture Notes in Computer Science **3142** (2004), 46–58
3. Abadi, M. and V. Cortier, *Deciding Knowledge in Security Protocols Under (Many More) Equational Theories*, Proceedings of the 18th Workshop on Computer Security Foundations (CSFW'2005) (2005), 62–76
4. Abadi, M. and C. Fournet, *Mobile Values, New Names and Secure Communications*, ACM SIGPLAN Notices **36** (2001), 104-115
5. A. Alessandro, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron, *The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications*, CAV (2005), 281-285
6. Basin, D., S. Mödersheim and L. Viganò, *OFMC: A Symbolic Model Checker for Security Protocols*, International Journal of Information Security **4(3)** (2005), 181-208
7. Baudet, M., *Deciding Security of Protocols against Off-line Guessing Attacks*, Proceedings of the 12th ACM Conference on Computer and Communications Security (2005), 16–25
8. Baudet, M., V. Cortier and S. Delaune, *YAPA: A Generic Tool for Computing Intruder Knowledge*, Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA'09), Lecture Notes in Computer Science **5595** (2009), 148–163
9. Baudet, M., V. Cortier and S. Kremer, *Computationally Sound Implementations of Equational Theories against Passive Adversaries*, Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP05), Lecture Notes in Computer Science **3580** (2005), 652–663

10. Blanchet, B., *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*, 14th Computer Security Foundations Workshop (CSFW'01) (2001), 82–96
11. Blanchet, B., M. Abadi and C. Fournet, *Automated Verification of Selected Equivalences for Security Protocols*, Symposium on Logic in Computer Science (2005), 331–340
12. Bella, G. and L.C. Paulson, *Using Isabelle to Prove Properties of the Kerberos Authentication System*, DIMACS Workshop on Design and Formal Verification of Security Protocols (1997)
13. Bellovin, S. M. and M. Merritt, *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*, IEEE Symposium on Research in Security and Privacy (1992), 72–84
14. Chevalier, Y., R. Küsters, M. Rusinowitch and M. Turuani, *An NP Decision Procedure for Protocol Insecurity with XOR*, Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03) (2003), 261-270
15. Chevalier, Y., R. Küsters, M. Rusinowitch and M. Turuani, *Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Product in Exponents*, Lecture Notes in Computer Science **2914** (2003), 124–135
16. Ciobâcă, Ş, S. Delaune and S. Kremer, *Computing Knowledge in Security Protocols Under Convergent Equational Theories*, CADE'09, LNAI (2009), 355–370
17. Comon-Lundh, H. and V. Shmatikov, *Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive Or*, 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03) (2003), 271–280
18. Comon-Lundh, H. and R. Treinen, *Easy Intruder Deductions*, Verification: Theory & Practice, Essays Dedicated to Zohar Manna on the Occasion of his 64th Birthday, Lecture Notes in Computer Science **2772** (2003), 225-242
19. Conchinha, B., D. Basin and C. Caleiro, *Efficient Algorithms for Deciding Deduction and Static Equivalence*, Technical Reports 680, ETH Zrich, Information Security Group D-INFK (2010)
20. Corin, R., J. Doumen and S. Etalle, *Analyzing Password Protocol Security Against Off-line Dictionary Attacks*, Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS05), Electronic Notes in Theoretical Computer Science **121** (2005), 47–63
21. Cortier, V., S. Delaune and P. Lafourcade, *A Survey of Algebraic Properties Used in Cryptographic Protocols*, Journal of Computer Security **14** (2006), 1–43
22. Cortier, V. and S. Delaune, *Deciding Knowledge in Security Protocols for Monoidal Equational Theories*, Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'07), Lecture Notes in Computer Science **4790** (2007), 196–210
23. Delaune, S., P. Lafourcade, D. Lugiez and R. Treinen, *Symbolic Protocol Analysis for Monoidal Equational Theories*, Information and Computation **206** (2009), 312–351
24. Gong, L., M. A. Lomas, R. M. Needham and J. H. Saltzer, *Protecting Poorly Chosen Secrets From Guessing Attacks*, IEEE Journal on Selected Areas in Communications **11** (1993), 648–656
25. Lafourcade, P., *Intruder Deduction for the Equational Theory of Exclusive-or with Commutative and Distributive Encryption*, Proceedings of the 1st International Workshop on Security and Rewriting Techniques (SecReT 2006), Electronic Notes in Theoretical Computer Science **171** (2007), 37–57
26. Lafourcade, P., D. Lugiez and R. Treinen, *Intruder Deduction for AC-like Equational Theories with Homomorphisms*, Proceedings of the 16th International Confer-

ence Rewriting Techniques and Applications (RTA05), Lecture Notes in Computer Science **3467** (2005), 308–322

27. Millen, J. and V. Shmatikov, *Symbolic Protocol Analysis With an Abelian Group Operator or Diffie–Hellman Exponentiation*, Journal of Computer Security **13** (2005), 515–564