FAST: An Efficient Decision Procedure for Deduction and Static Equivalence

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

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

Abstract

Message deducibility and static equivalence are central problems in symbolic security protocol analysis. We present FAST, an efficient decision procedure for these problems under subterm-convergent equational theories. FAST is a C++ implementation of an improved version of the algorithm presented in our previous work [10]. This algorithm has a better asymptotic complexity than other algorithms implemented by existing tools for the same task, and FAST's optimizations further improve these complexity results.

We describe here the main ideas of our implementation and compare its performance with competing tools. The results show that our implementation is significantly faster: for many examples, FAST terminates in under a second, whereas other tools take several minutes.

Keywords and phrases Efficient algorithms, Equational theories, Deducibility, Static equivalence, Security protocols

Digital Object Identifier 10.4230/LIPIcs.RTA.2011.11

Category System Description

1 Introduction

Cryptographic protocols are widely used to provide secure network communication. It is therefore important that such protocols are proven correct. Automated tools for this task rely on symbolic protocol models, in which cryptographic primitives are modeled as function symbols and messages exchanged over the network are represented by terms in a term algebra. Properties of cryptographic primitives are represented as equational theories relating terms in the term algebra.

Message deducibility and static equivalence are two important problems in the analysis of security protocols. The deducibility problem consists of deciding whether an attacker can use a set of messages (for instance, the messages exchanged over a network) to compute a secret message. The knowledge of an attacker is often expressed in terms of deducibility, *i.e.*, the set of messages he can deduce, and most automated methods for protocol analysis rely on this notion [5, 4]. Static equivalence is used to model the notion that an attacker cannot distinguish between two sequences of messages [3]. It has been used to model several indistinguishability related security properties, including security against off-line guessing attacks [12, 6, 2] and anonymity in e-voting protocols [13].

22nd International Conference on Rewriting Techniques and Applications.
Editor: M. Schmidt-Schauß; pp. 11–20



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We present an implementation of the algorithm proposed in [10] for efficiently deciding deduction and static equivalence. Although existing tools [7, 9] solve these problems for more general equational theories, our algorithm has significantly better asymptotic complexity for the class of subterm-convergent equational theories. This class is general enough to model many equational theories relevant in practice. FAST implements several optimizations to the algorithm presented in [10], further improving our asymptotic complexity results. We have analyzed its performance using as benchmarks different practically relevant examples of subterm-convergent theories. As expected, the results show that FAST is significantly faster than other algorithms for the equational theories it handles.

In Section 2 we formally introduce the notions of message deducibility and static equivalence. In Section 3 we describe our implementation and the improvements we made to the algorithm given in [10]. In Section 4 we present our benchmark tests and compare the performance of FAST with existing tools. We draw conclusions in Section 5.

2 Background and decision problems

We consider *signatures* $\Sigma = \biguplus_{n\in\mathbb{N}} \Sigma_n$ consisting of finitely many function symbols, where Σ_i contains the functions symbols of arity i. We also fix countably infinite, disjoint sets Var and Name of variables and names.

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 $f \in \Sigma_n$. 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. The *size* $|t|$ of a term $t \in T(\Sigma, X)$ is given by $|t| = 1$ if $t \in X$ and $|t| = 1 + \sum_{i=1}^{n} |t_i|$ if $t = f(t_1, \ldots, t_n)$.

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

A *frame* is a pair $(\tilde{n}, \sigma)$, written $\nu\tilde{n}.\sigma$, 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 a finite domain. The *size* $|\phi|$ of a frame ϕ is given by $|\phi| = \sum_{x\in dom(\sigma)} |x\sigma|$. Given a frame $\phi = \nu\tilde{n}.\sigma$, we define $T_\phi = T(\Sigma, (\mathsf{Name} \setminus \tilde{n}) \cup dom(\sigma))$. We say that terms in T_ϕ are *ϕ-recipes*. A term t can be *constructed* from ϕ if there is a ϕ-recipe t' such that $t\sigma = t'$ (syntactically). Frames are used to represent an attacker's knowledge: the names in $\tilde{n}$ represent randomly generated nonces unknown to the attacker, and the terms in σ's range represent the messages learned by the attacker, for instance, the messages exchanged over a network.

A rewrite rule is a pair (l, r), written as $l \to r$, where $l, r \in T(\Sigma, \mathsf{Var})$. A rewriting system R over Σ is a set of rewrite rules. We always assume that rewrite systems are finite. 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 $f(t_1, \ldots, t_i, \ldots, t_n) \to_R f(t_1, \ldots, t_i', \ldots, t_n)$.

If the rewriting system $\to_R$ is convergent, then each term t has a unique normal form $t{\downarrow_R} \in T(\Sigma, \mathsf{Name})$. In this case, we define $\approx_R \subseteq T(\Sigma, \mathsf{Name}) \times T(\Sigma, \mathsf{Name})$ as the equational theory such that $t \approx_R t'$ if and only if $t{\downarrow_R} = t'{\downarrow_R}$. As usual, we write $t \approx_R t'$ instead of $(t, t') \in \approx_R$.

A rewriting system R is subterm convergent if $\rightarrow_R$ is convergent and, for each $(l \rightarrow r) \in R$, $r \in sub(l)$. It is weakly subterm convergent if, for each $(l \rightarrow r) \in R$, either $r \in sub(l)$ or $r \in T(\Sigma, \emptyset)$ is in normal form.

Deducibility and static equivalence

▶ **Definition 2.1.** Given a frame $\phi = \nu\tilde{n}.\sigma$, a term $t \in T(\Sigma, \mathsf{Name})$, and a rewriting system R, we say that t is *deducible from* ϕ *under* R, and write $\phi \vdash_R t$, if there is a ϕ-recipe t' such that $t'\sigma \approx_R t$.

▶ **Definition 2.2.** Given two frames $\phi = \nu\tilde{n}.\sigma$ and $\phi' = \nu\tilde{n}'.\sigma'$ and a rewriting system R, we say that ϕ and ϕ' are *statically equivalent under* R, and write $\phi \approx^s_R \phi'$, if $T_\phi = T_{\phi'}$ (*i.e.*, $\tilde{n} = \tilde{n}'$ and $dom(\sigma) = dom(\sigma')$) and, for all $t, t' \in T_\phi$, $t\sigma \approx_R t'\sigma$ if and only if $t\sigma' \approx_R t'\sigma'$.

The message deducibility problem is concerned with whether an attacker who has learned the messages represented by the terms in the range of σ can use those messages to compute (*deduce*) a secret message t without using the (secret) names in $\tilde{n}$. Static equivalence formalizes that two sequences of messages are indistinguishable from an attacker's point of view. This is modeled as the condition that there are no two recipes that yield (equationally) equal terms under the substitution σ but not under σ' (or vice-versa). This is useful, for instance, in modeling off-line guessing attacks [12, 10].

3 Fast algorithm

Fast is a C++ implementation of the algorithm described in [10]. It solves the message deducibility and static equivalence problems for weakly subterm-convergent rewriting systems. Such rewriting systems are sufficiently expressive to represent a standard Dolev-Yao equational theory with one-way functions, pairing, projections and symmetric and asymmetric encryption and decryption. They can also model idempotent functions and signature schemes, among others.

Other existing tools for solving these two decision problem are Yapa [7], Kiss [9], and ProVerif [8]. However, ProVerif is designed for solving the harder problem of protocol security under active adversaries. Therefore, it is not surprising that it performs significantly slower for these two problems. Moreover, ProVerif is not guaranteed to terminate even under subterm-convergent equational theories, and Yapa claims a performance between one and two orders of magnitude faster than ProVerif [7]. We thus compare our algorithm with Yapa and Kiss.

Compared to Yapa and Kiss, Fast has a better asymptotic complexity, but a narrower application scope. For instance, Yapa and Kiss can handle rewriting systems representing blind signatures and homomorphic encryption. Kiss can also handle trapdoor committments. Fast cannot handle these rewriting systems. We believe, however, that many of the techniques and results that allow Fast to achieve a better asymptotic complexity do not depend on the subterm-convergent hypothesis (*cf.* Section 5).

3.1 Procedures and complexity

Data structures

Fast represents terms as DAGs (Directed Acyclic Graphs). Each term is represented by a C++ object which we call a *term vertex*. Term vertices contain as a class member a C++ object representing a recipe, which we will call a *recipe vertex*. Furthermore, each recipe

vertex contains as a class member an array of term vertices. These term vertices represent the normal forms of the terms corresponding to the recipe in each of the input frames. These normal forms are computed whenever a recipe vertex is created. Whenever a recipe is added to a vertex v, all the parent vertices of v are checked; if all children of a parent vertex pv have recipes, then a new recipe is computed and added to pv.

Saturation procedure

Similar to YAPA and KISS, FAST relies on a frame saturation procedure. However, FAST's saturation procedure generates less terms and is therefore more efficient than these other procedures. Namely, FAST only adds to the saturated frame those terms that are instances of right-hand sides of rules (rather than all deducible subterms). Furthermore, FAST only instantiates a rule if there is some subterm of the left-hand side that is mapped to a term already in the frame and each variable is mapped to either a subterm of a term in the original frame or a fresh nonce. A more detailed description of the saturation procedure, as well as a comparison with the procedures used in other tools, is given in [11].

Given a frame $\phi = \upsilon\tilde{n}.\sigma$, FAST's saturation procedure computes two sets Γ_s and Γ_l of term vertices. Γ_s represents a set of subterms in the range of σ that are deducible from ϕ. Γ_l represents the set of instantiations $l\sigma_l$ of left-hand sides l of rewrite rules such that $\sigma_l \colon vars(l) \to sub(ran(\sigma)) \cup \Upsilon$, where Υ is a finite set of fresh nonces computed from the rewrite system, and there is some $s \in sub(l)$ such that $s\sigma_l \in \Gamma_s$.

Γ_s is initialized as $ran(\sigma)$. Recipes are added to each vertex representing a term in $ran(\sigma)$ and to each leaf vertex. The program then tries to create instances $l\sigma_l$ of left-hand sides of rules which verify the conditions above. To do this, for each $t \in \Gamma_s$, each rewrite rule $l \to r$, and each $s \in sub(l)$, FAST checks whether t and s can be unified, and then attempts to extend the substitution unifying these terms to all the variables in l.

Whenever such an instance is found, the term vertices needed to represent that instance are created, and the term vertex representing $l\sigma_l$ is added to Γ_l. When a recipe ζ is added to a term vertex in Γ_l representing one such instance $l\sigma_l$, if $r\sigma_l$ does not have a recipe yet, then the recipe ζ is added to $r\sigma_l$, and $r\sigma_l$ is added to Γ_s. $r\sigma_l$ can then be used to find more instances of left-hand sides of rules.

Deducibility and static equivalence

After obtaining the saturated frame ϕ_s, deciding whether a term is deducible from ϕ amounts to deciding whether it can be constructed from the terms represented by term vertices in Γ_s. From Γ_s and Γ_l one also obtains a finite set Eq_ϕ of equations between ϕ-recipes satisfied by ϕ and such that, if another frame ϕ' with $T_\phi = T_{\phi'}$ satisfies all equations in Eq_ϕ, then ϕ' satisfies all equations satisfied by ϕ. These sets of equations are the same as those tested by the algorithm described in [10]. Therefore, given frames ϕ and ϕ' such that $T_\phi = T_{\phi'}$, one can decide whether $\phi \approx^s_R \phi'$ by building the corresponding saturated frames, obtaining the sets of equations Eq_ϕ and $\mathsf{Eq}_{\phi'}$ described above, and then testing whether ϕ satisfies all equations in $\mathsf{Eq}_{\phi'}$ and vice-versa.

Improvements over [10]

Our implementation improves the algorithm described in [10] in several ways. The most relevant improvement is that FAST only considers instances $l\sigma_l$ of left-hand sides of rules if there is some $s \in sub(l)$ such that $s\sigma_l$ is in the range of σ_s. This makes the saturation procedure faster and also reduces the number of equations that the algorithm must check

to decide static equivalence. Another important improvement is that recipes are computed at most once for each vertex. Furthermore, whenever a new recipe $t \in T_\phi$ is added, the implementation creates a recipe object containing pointers to the vertices $(t\sigma)\downarrow$ and $(t\sigma')\downarrow$. The proof of correctness of the algorithm presented in [10] shows that one must only consider instances $l\sigma_l$ of left-hand sides of rules if $r\sigma_l$ is in normal form; therefore, this only requires matching the term resulting from the recipe with each left-hand side of a rule, discarding it if the term obtained after one step of rewriting is not in normal form. This can be done in constant time.

3.2 Asymptotic complexity

The above improvements reduce the asymptotic complexity of the algorithm for some equational theories. For instance, consider the rewriting system

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

used in [10] for comparing the asymptotic complexity of the algorithm introduced there with YAPA and KISS. The asymptotic complexity of KISS is estimated in [10] to be $\mathcal{O}(|\phi|^7)$ under $\mathcal{DY}_{asym}$. For this rewrite system, YAPA has exponential complexity in the worst case scenario, since it does not implement DAG representation of terms. Even if DAGs were implemented, the asymptotic complexity of YAPA's saturation procedure is estimated (again in [10]) to be $\mathcal{O}(|\phi|^7)$. The asymptotic complexity of the algorithm presented in [10] for this rewrite system is estimated to be $\mathcal{O}((|\phi| + |\phi'|)^3 \log^2(|\phi| + |\phi'|)$ for static equivalence. Given that each recipe is associated with the normal forms of the terms represented by the recipe in each frame (thus eliminating the overhead of computing normal forms), FAST's complexity for static equivalence under $\mathcal{DY}_{asym}$ is only $\mathcal{O}((|t| + |\phi|)^2 \log^2(|t| + |\phi|))$. A detailed analysis of FAST's asymptotic complexity is given in [11].

4 Performance analysis

We have considered several families of interesting and practically relevant examples to compare the performance of our algorithm with YAPA and KISS. The results show great disparities in the performance of the three algorithms. Neither KISS nor YAPA show a clear advantage over the other: depending on the example, either algorithm may perform significantly faster than the other. As expected from the complexity results, FAST generally performs much better than either of these algorithms, particularly for static equivalence. Even for artificial equational theories designed to produce worst case performance for our algorithm, FAST is still more efficient for static equivalence, sometimes significantly so. For message deducibility under such equational theories, FAST performs better in most examples; however, in a few, it appears to be slower by a small constant.

All our tests were performed on a computer with an Intel Core 2 Duo processor running at 2.53GHz and with 4Gb memory. In all our static equivalence tests, we consider two equal frames. Similarly, in all our deduction tests, the input term is a secret that does not occur in the range of the substitution of the input frame. Therefore, the result is positive in all static equivalence tests and negative in all deducibility tests. This does not affect the algorithm's performance significantly, as both frames still have to be saturated in all implementations — that is, deducible subterms must still be added to the saturation, and the sets of equations which must be tested to check for static equivalence must still be generated (*cf.* Section 3). Static equivalence takes a slightly longer time in this case because all equations must be

checked rather than stopping as soon as a counter-example is found. However, the difference is minor. We present here an illustrative sample of the tests performed. For a more complete report on our results, see [1].

4.1 Chained keys

This family of tests uses the signature $\Sigma^{\mathcal{DY}}$, with $\Sigma_1^{\mathcal{DY}} = \{\pi_1, \pi_2\}$ and $\Sigma^{\mathcal{DY}} = \{\{\cdot\}_\cdot, \{\cdot\}_\cdot^{-1}, \langle\cdot,\cdot\rangle\}$, and a rewriting system representing a standard Dolev-Yao intruder without asymmetric encryption, specified by the set of rewrite rules

$$\mathcal{DY} = \left\{\pi_1(\langle x, y\rangle) \to x, \pi_2(\langle x, y\rangle) \to y, \left\{\{x\}_y\right\}_y^{-1} \to x\right\}.$$

For $n \in \mathbb{N}$, we define the frame $\phi_n^{ck} = \upsilon\tilde{n}_n^{ck}.\sigma_n^{ck}$, where $\tilde{n}_n^{ck} = \{k, k_0, \ldots, k_n\}$ and $\sigma = \left\{x_1 \mapsto \{k_0\}_{k_1}, \ldots, x_n \mapsto \{k_{n-1}\}_{k_n}, x_{n+1} \mapsto k_n\right\}$. For each parameter n, the deduction problem is to decide whether $\phi_n^{ck} \vdash_{\mathcal{DY}} k$, and the static equivalence problem is to decide whether $\phi_n^{ck} \approx_{\mathcal{DY}}^s \phi_n^{ck}$.

Fast has a much better performance than both Yapa and Kiss for these examples. Yapa also performs much better than Kiss. Tables 1 and 2 illustrate these relationships.

Table 1 Performance on *chained keys* for deduction (time in ms)

Parameter	50	100	200	500	1000	2000	5000
Fast	11	20	40	143	224	474	1526
Kiss	259	1730	12655	288606	> 300000	> 300000	> 300000
Yapa	31	108	415	4624	11297	62457	> 300000

Table 2 Performance on *chained keys* for static equivalence (time in ms)

Parameter	50	100	200	500	750	1500	2500
Fast	20	41	88	247	424	1020	1546
Kiss	1341	12185	127828	> 300000	> 300000	> 300000	> 300000
Yapa	143	744	5516	18467	44451	197648	> 300000

4.2 Composed keys

This family of examples uses the same signature $\Sigma^{\mathcal{DY}}$ and the same rewriting system $\mathcal{DY}$ used in Section 4.1.

For $n, s, i \in \mathbb{N}$, define $t_{n,s}^i$ recursively by

$$t_{n,s}^0 = \{\langle k_{2s-1}, k_{2s-2}\rangle\}_{\langle k_{2s}, k_{s2+1}\rangle},$$
$$t_{n,s}^i = \left\{\langle t_{n,s}^{i-1}\langle k_{2s+1+2i(n-1)}, k_{2s+2i(n-1)}\rangle, \right\}_{\langle k_{2s+2+2i(n-1)}, k_{2s+3+2i(n-1)}\rangle}.$$

For $k > 0$, the frame $\phi_k^c = \upsilon\tilde{n}_n^c.\sigma_n^c$ is such that $\tilde{n}_n^c = \{k, k_0, \ldots, k_{2n^2+1}\}$ and $\sigma_n^c = \left\{x_1 \mapsto t_{n,1}^{n-1}, \ldots, x_n \mapsto t_{n,n}^{n-1}, x_{n+1} \mapsto k_{2n^2}, x_{n+2} \mapsto k_{2n^2+1}\right\}$. The deduction problem corresponding to parameter n considered in our tests is to decide whether $\phi_n^c \vdash_{\mathcal{DY}} k$. The static equivalence problem corresponding to parameter n is to decide whether $\phi_n^c \approx_{\mathcal{DY}}^s \phi_n^c$.

This family of examples is particularly challenging because the decryption keys are pairs of secrets. At each point of the algorithm's execution, decrypting the right message yields

a pair of previously unknown secrets. This pair may then be used to compose the next decryption key by exchanging the order of the terms in the pair. As illustrated in Tables 3 and 4, the difference in FAST's performance is particularly marked in this example. KISS also performs much better than YAPA.

Table 3 Performance on *composed* for deduction (time in ms)

Parameter	3	4	5	7	9	10	20
FAST	7	11	17	34	61	126	945
KISS	138	867	3760	46369	245207	> 300000	> 300000
YAPA	158	34118	> 300000	> 300000	> 300000	> 300000	> 300000

Table 4 Performance on *composed* for static equivalence (time in ms)

Parameter	3	4	5	6	8	10	20
FAST	12	21	28	48	92	148	1635
KISS	469	2625	10428	252000	> 300000	> 300000	> 300000
YAPA	936	157358	> 300000	> 300000	> 300000	> 300000	> 300000

4.3 Denning-Sacco shared key protocol

The Denning-Sacco symmetric key protocol [14] is used to establish session keys in a network with a single server and multiple agents. Each agent shares a (secret) symmetric key with the server, but there are no shared keys between agents. In Alice&Bob notation, the protocol is as follows.

1. $\mathsf{A} \to \mathsf{S}$: A, B
2. $\mathsf{S} \to \mathsf{A}$: $\{\mathsf{A}, K_{\mathsf{A},\mathsf{B}}, T, \{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}\}_{K_{\mathsf{S},\mathsf{A}}}$
3. $\mathsf{A} \to \mathsf{B}$: $\{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}$

Here, A and B are two participants, and S is the server. A requests from the server a session key to communicate with B. The server generates a new session key, $K_{\mathsf{A},\mathsf{B}}$, and sends it to A, encrypted with the (symmetric) key shared between A and S. This message also contains a timestamp T, used to determine the validity of the new session key, and the ticket $\{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}$. A then forwards this ticket to B, who can decrypt it using the key $K_{\mathsf{S},\mathsf{B}}$ shared between B and S, to obtain the new session key $K_{\mathsf{A},\mathsf{B}}$, the name A of the intended communication partner, and the time T of the request.

This example uses the result of executing multiple sessions of the Denning-Sacco protocol. For the parameter n we assume a network with $3n$ participants, each of which initiates one session with each other participant. We assume that one third of the shared keys between the server and the agents are compromised, *i.e.*, available to the attacker.

We will once again use the signature $\Sigma^{\mathcal{DY}}$ and the rewriting system $\mathcal{DY}$ from Section 4.1. For parameter n, we thus use the frame $\phi_n^{ds} = \tilde{n}_n^{ds}.\sigma_n^{ds}$, with $\sigma_n^{ds} = \sigma_n^1 \cup \sigma_n^2 \cup \sigma_n^3 \cup \sigma_n^4$, where

- $\sigma_n^1 = \left\{x_i^1 \mapsto K_{\mathsf{S},i} \mid i \in \{1,\ldots,n\}\right\}$;
- $\sigma_n^2 = \left\{x_{i,j}^2 \mapsto \langle \mathsf{A}_i, \mathsf{A}_j\rangle \mid i,j \in \{1,\ldots,3n\}, i \neq j\right\}$;
- $\sigma_n^3 = \left\{x_{i,j}^3 \mapsto \left\{\langle \mathsf{A}_j, \langle K_{i,j}, \langle T_{i,j}, \{\langle K_{i,j}, \langle \mathsf{A}_i, T_{i,j}\rangle\rangle\}_{K_{\mathsf{S},j}}\rangle\rangle\right\}_{K_{\mathsf{S},i}} \mid i,j \in \{1,\ldots,3n\}, i \neq j\right\}$;
- $\sigma_n^4 = \left\{x_{i,j}^4 \mapsto \{\langle K_{i,j}, \langle \mathsf{A}_i, T_{i,j}\rangle\rangle\}_{K_{\mathsf{S},j}} \mid i,j \in \{1,\ldots,3n\}, i \neq j\right\}$,

and $\tilde{n}_n^{ds} = \{K_{i,j}, T_{i,j}, K_{\mathsf{S},i} \mid i,j \in \{1,\ldots,3n\}\}$.
Here, σ_n^1 represents the keys compromised by the attacker and σ_n^2, σ_n^3, and σ_n^4 represent the messages exchanged as part of the execution of the first, second, and third steps of the protocol, respectively. The deduction problem is to decide whether $\phi_n^{ds} \vdash_{\mathcal{DY}} K_{\mathsf{S},3n}$ and the static equivalence problem is to decide whether $\phi_n^{ds} \approx^s_{\mathcal{DY}} \phi_n^{ds}$.

YAPA performs noticeably better than KISS in this example. FAST, as before, is significantly faster than both. The results are shown in Tables 5 and 6.

Table 5 Performance on *Denning-Sacco* for deduction (time in ms)

Parameter	5	7	9	11	12	14	24
FAST	337	768	1336	2588	2239	5083	20073
KISS	6637	30195	91743	232093	> 300000	> 300000	> 300000
YAPA	2409	10320	30732	68845	74298	172249	> 300000

Table 6 Performance on *Denning-Sacco* for static equivalence (time in ms)

Parameter	3	5	7	9	11	13	20
FAST	181	585	1281	2300	6598	7507	24614
KISS	1219	8543	34726	158717	> 300000	> 300000	> 300000
YAPA	446	2836	12300	52506	134391	269781	> 300000

4.4 FAST worst case

In this family of examples, for the parameter n, we use the signature Σ^{wc}, where $\Sigma_1^{wc} = \{f\}$ and $\Sigma_n^{wc} = \{h\}$. The rewriting system is given by the set $wc_n = \{h(f(x_1),\ldots,f(x_n)) \to x_1\}$. We define the frame $\phi_n^{wc} = \tilde{n}_n^{wc}.\sigma_n^{wc}$, where $\tilde{n}_n^{wc} = \{k, k_1, \ldots, k_n\}$ and $\sigma_n^{wc} = \{x_1 \mapsto f(k_1), \ldots, x_n \mapsto f(k_n)\}$. The deduction problem is to decide whether $\phi_n^{wc} \vdash_{wc_n} k$ and the static equivalence problem is to decide whether $\phi_n^{wc} \approx^s_{wc_n} \phi_n^{wc}$.

This example is challenging because, to saturate this frame, FAST must instantiate each element of the tuple with each of the secret names. Therefore, the asymptotic complexity of FAST for this family is $\mathcal{O}(n^n)$. Note that this does not contradict the fact that, for a given rewriting system, FAST has polynomial-time complexity; the exponential complexity results from the fact that the size of the rewriting system itself increases with the parameter n.

Table 7 Performance on *worst case* for deduction (time in ms)

Parameter	3	4	5	6
FAST	9	72	1192	32487
KISS	10	47	866	21446
YAPA	11	161	6607	> 300000

None of the existing algorithms perform well on this example: FAST's performance is comparable to that of KISS and YAPA performs significantly worse. This is illustrated in Tables 7 and 8.

Table 8 Performance on *worst case* for static equivalence (time in ms)

Parameter	3	4	5	6
Fast	15	142	2199	56312
Kiss	16	146	2125	69533
Yapa	16	297	8862	> 300000

Nonlinear terms

It is interesting to note that Fast's complexity depends chiefly on the number of different variables in the rewriting system. Therefore, it's performance is not significantly affected if the left-hand sides of rewrite rules are non-linear. This is not the case for the other algorithms, whose performance degrades when the complexity of the terms in the rewriting system increases, even when the number of variables remains the same. Tables 9 and 10 illustrate this point. Here, the rewriting system considered is $wc2_n = \{h(f(x_1), f(x_1), \ldots, f(x_n), f(x_n)) \to x_1\}$. The frames and problems considered here are the same as above.

Table 9 Performance on *worst case 2* for deduction (time in ms)

Parameter	2	3	4	5	6
Fast	7	9	148	1567	43282
Kiss	9	99	4381	183236	> 300000
Yapa	45	> 300000	> 300000	> 300000	> 300000

Table 10 Performance on *worst case 2* for static equivalence (time in ms)

Parameter	2	3	4	5	6
Fast	4	17	396	6197	47937
Kiss	10	292	16135	> 300000	> 300000
Yapa	56	> 300000	> 300000	> 300000	> 300000

5 Discussion and future work

Fast is an efficient algorithm for deciding deduction and static equivalence under weakly subterm-convergent rewriting systems. The implementation and our benchmarks are available for download at [1]. Fast's scope is narrower than that of other existing tools for these problems, but is broad enough to represent many practically relevant theories. As expected from the results in [10], Fast is significantly faster than both Yapa and Kiss in almost all our tests. Even for the artificial examples designed to degrade its performance, Fast still compares favorably to other algorithms: it is either faster, or slower by only a small constant. This constitutes a significant advantage since the problematic cases for Yapa and Kiss degrade these algorithms' performances dramatically.

We believe that many of the ideas and results that allow Fast to achieve better asymptotic results may still be valid with weaker hypotheses on the rewriting system. Therefore, extending the algorithm to handle more general equational theories without significantly degrading its performance is an important research goal.

Acknowledgements This work was partly supported by FCT and EU FEDER, namely via the project PTDC/EIA-CCO/113033/2009 ComFormCrypt of SQIG-IT and the project UTAustin/MAT/0057/2008 AMDSC of IST. The first author acknowledges the support of FCT via the PhD grant SFRH/BD/44204/2008 and the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation. We would also like to thank Mohammad Torabi Dashti and Benedikt Schmidt for their helpful comments on this paper.

References

1 `http://www.infsec.ethz.ch/people/brunoco`, 2011.
2 Martin Abadi, Mathieu Baudet, and Bogdan Warinschi. Guessing attacks and the computational soundness of static equivalence. *Journal of Computer Security*, pages 909–968, December 2010.
3 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *SIGPLAN Not.*, 36:104–115, January 2001.
4 Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Pierre-Cyrille Heám, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*. Springer, 2005.
5 Charu Arora and Mathieu Turuani. Validating Integrity for the Ephemerizer's Protocol with CL-Atse. In *Formal to Practical Security: Papers Issued from the 2005-2008 French-Japanese Collaboration*, volume 5458 of *LNCS*, pages 21–32. Springer, 2009.
6 Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 16–25, New York, NY, USA, 2005. ACM.
7 Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. In Ralf Treinen, editor, *RTA*, volume 5595 of *Lecture Notes in Computer Science*, pages 148–163. Springer, 2009.
8 Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, CSFW '01, pages 82–96, Washington, DC, USA, 2001. IEEE Computer Society.
9 Stefan Ciobâca, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. In *CADE*, pages 355–370, 2009.
10 Bruno Conchinha, David Basin, and Carlos Caleiro. Efficient algorithms for deciding deduction and static equivalence. In *Proc. 7th Int. Workshop on Formal Aspects of Security and Trust (FAST'10)*, 2010.
11 Bruno Conchinha, David Basin, and Carlos Caleiro. Efficient algorithms for deciding deduction and static equivalence. volume 680 of *ETH Technical Reports*. ETH Zürich, Information Security Group D-INFK, September 2010.
12 Ricardo Corin, Jeroen Doumen, and Sandro Etalle. Analysing password protocol security against off-line dictionary attacks. *Electron. Notes Theor. Comput. Sci.*, 121:47–63, February 2005.
13 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17:435–487, December 2009.
14 Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24:533–536, August 1981.