An Optimizing Protocol Transformation for Constructor Finite Variant Theories in Maude-NPA[⋆]

Damián Aparicio-Sánchez[1], Santiago Escobar[1], Raúl Gutiérrez[2], and Julia Sapiña[1]

[1] VRAIN, Universitat Politècnica de València, Valencia, Spain
{daapsnc,sescobar,jsapina}@upv.es
[2] Universidad Politécnica de Madrid, Madrid, Spain
r.gutierrez@upm.es

Abstract. Maude-NPA is an analysis tool for cryptographic security protocols that takes into account the algebraic properties of the cryptosystem. Maude-NPA can reason about a wide range of cryptographic properties. However, some algebraic properties, and protocols using them, have been beyond Maude-NPA capabilities, either because the cryptographic properties cannot be expressed using its equational unification features or because the state space is unmanageable. In this paper, we provide a protocol transformation that can *safely* get rid of cryptographic properties under some conditions. The time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable. We also provide, for the first time, an encoding of the theory of bilinear pairing into Maude-NPA that goes beyond the encoding of bilinear pairing available in the Tamarin tool.

Keywords: crypto protocol analysis, Diffie-Hellman, exponentiation, bilinear pairing, protocol transformation

1 Introduction

Maude-NPA [13] is an analysis tool for cryptographic security protocols that takes into account the algebraic properties of the cryptosystem. Sometimes algebraic properties can uncover weaknesses of cryptosystems and, in other cases, they are part of the protocol security assumptions. Maude-NPA uses an approach similar to its predecessor, the NRL Protocol Analyzer (NPA) [24], i.e., it is based on unification and performs backwards search from an attack state pattern to

[⋆] Partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by the Spanish Generalitat Valenciana under grant PROMETEO/2019/098, and by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286. Julia Sapiña has been supported by the Generalitat Valenciana APOSTD/2019/127 grant.

determine whether or not it is reachable. However, unlike the original NPA, it has a theoretical basis on *rewriting logic* [12] and *narrowing* [7], and while NPA could only be used to reason about equational theories involving a fixed set of rewrite rules, Maude-NPA can be used to reason about a wide range of cryptographic properties [1, 14], including cancellation of encryption and decryption, Diffie-Hellman exponentiation [11], exclusive-or [30], and some approximations of homomorphic encryption [15, 34].

However, some algebraic properties and protocols using them have been beyond Maude-NPA capabilities, either because the cryptographic properties cannot be expressed using its equational unification features or because the state space is unmanageable. We provide a protocol transformation that can substantially reduce the search space, i.e., given some cryptographic properties, expressed using the equational unification features of Maude-NPA, and a protocol, we are able to transform the protocol in such a way that some cryptographic properties are no longer necessary, and thus can be safely removed. The time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable. We also provide, for the first time, an encoding of the theory of bilinear pairing into Maude-NPA that goes beyond the encoding of bilinear pairing available in Tamarin [2], the only crypto tool with such an equational theory.

Our protocol transformation relies on a program transformation from [28] for rewrite theories in Maude that we have improved by relaxing some of its applicability conditions. Such program transformation relies on *constructor term variants* [27], which is an extension of *term variants* [8, 17]. Nowadays, several crypto analysis tools rely on the variant-based equational unification capabilities of Maude, such as Maude-NPA but also Tamarin [10] and AKISS [5]. These tools may be benefited from our protocol transformation and, furthermore, from our encoding of the theory of bilinear pairing. Our contributions may even be useful for other tools with more limited crypto properties such as ProVerif [6], Scyther [9] or Scyther-proof [25].

The main contributions of this work are: (i) we provide a non-trivial protocol transformation based on [28]; (ii) since the protocols of Section 5 do not satisfy the conditions of [28], we provide a more powerful protocol transformation that we implemented, made available online, and pays off in practice; (iii) we provide an encoding of bilinear pairing that can handle all the protocols of Section 5 that Tamarin cannot handle; (iv) we implemented the algorithm of [32] for the computation of constructor variants [27] from scratch; and (v) there was no implementation of the program transformation of [28] and we implemented it.

After some preliminaries on Section 2, we present how Maude-NPA works in Section 3. We introduce our protocol transformation in Section 4. Section 5 presents several increasingly complex case studies: Diffie-Hellman protocol in Section 5.1, STR protocol in Section 5.2, Joux protocol in Section 5.3, and TAK protocols in Section 5.4. Our experiments are presented in Section 6 and we conclude in Section 7.

2 Preliminaries

We follow the classical notation and terminology for term rewriting [33], and for rewriting logic and order-sorted notions [26]. We assume an order-sorted signature Σ with a poset of sorts $(\mathsf{S}, \leq)$. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. For a term t, $\mathit{Var}(t)$ denotes the set of variables in t. Throughout this paper, Σ is assumed to be *preregular*, so each term t has a least sort, denoted $ls(t)$.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$, where the domain of σ is $\mathit{Dom}(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $\mathit{Ran}(\sigma)$. The identity substitution is denoted *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$ or $\sigma(t)$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions σ and σ' is written $\sigma\sigma'$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$. The E-equivalence class of a term t is denoted by $[t]_E$ and $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}$ denote the corresponding order-sorted term algebras modulo E. Throughout this paper we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s, because this affords a simpler deduction system. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. A set of substitutions $\mathit{CSU}_E(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E iff: (i) each $\sigma \in \mathit{CSU}_E(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is $\sigma \in \mathit{CSU}_E(t = t')$ and τ s.t. $\sigma\tau =_E \rho$; (iii) for all $\sigma \in \mathit{CSU}_E(t = t')$, $\mathit{Dom}(\sigma) \subseteq (\mathit{Var}(t) \cup \mathit{Var}(t'))$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there exist $p \in \mathit{Pos}_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The transitive (resp. transitive and reflexive) closure of $\to_{R,E}$ is denoted by $\to^+_{R,E}$ (resp. $\to^*_{R,E}$). A term t is (R, E)-irreducible if there is no t' s.t. $t \to_{R,E} t'$. The R, E-*narrowing* relation on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as $t \leadsto_{p,\sigma,R,E} t'$ ($\leadsto_\sigma$ if R, E are understood, and $\leadsto$ if σ is also understood) if there is a non-variable position $p \in \mathit{Pos}_\Sigma(t)$, a rule $l \to r \in R$ standardized apart (i.e., contains no variable previously met during any previous computation) and a unifier $\sigma \in \mathit{CSU}_E(t|_p = l)$, such that $t' = (t[r]_p)\sigma$. The transitive (resp. transitive and reflexive) closure of $\leadsto$ is denoted by $\leadsto^+$ (resp. $\leadsto^*$).

3 The Maude-NPA

Given a protocol $\mathcal{P}$ to be specified, protocol states are modeled as elements of an initial algebra $\mathcal{T}_{\Sigma_\mathcal{P}/\mathcal{E}_\mathcal{P}}$, i.e., each state is an equivalence class $[t]_{\mathcal{E}_\mathcal{P}} \in T_{\Sigma_\mathcal{P}/\mathcal{E}_\mathcal{P}}$ where $\Sigma_\mathcal{P}$ is the set of symbols defining the protocol $\mathcal{P}$, and $\mathcal{E}_\mathcal{P}$ specifies the *algebraic properties* of the cryptographic functions $\Sigma_\mathcal{P}$. The cryptographic properties $\mathcal{E}_\mathcal{P}$ may vary depending on different protocols.

The signature $\Sigma_\mathcal{P}$ incorporates some predefined symbols for protocol infrastructure. A state is a term of the form $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ where & is an associative-commutative union operator with identity symbol $\emptyset$.

The *intruder knowledge IK* of a state $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ is defined as a set of facts using the comma as an associative-commutative union operator with identity element $\emptyset$. There are two kinds of intruder facts: *positive* knowledge facts (the intruder knows m, i.e., $m \in \mathcal{I}$), and *negative* knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m \notin \mathcal{I}$), where m is a message expression.

Each S_i of a state $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ is called a strand and specifies the sequence of messages sent and received by a principal executing the protocol. *Strands* [18] are represented as a sequence of messages $[msg_1^\pm, msg_2^\pm, msg_3^\pm, \ldots, msg_{k-1}^\pm, msg_k^\pm]$ with $msg_i^\pm$ either msg_i^- (also written $-msg_i$) representing an input message, or msg_i^+ (also written $+msg_i$) representing an output message. Note that each msg_i is a term of a special sort Msg; this sort is extended by the user to allow any user-definable protocol syntax. Variables of a special sort Fresh are used to represent pseudo-random values (nonces) and Maude-NPA ensures that two distinct fresh variables will never be merged. Strands are extended with all the fresh variables created by that strand, i.e., $:: f_1, \ldots, f_k :: [msg_1^\pm, msg_2^\pm, \ldots, msg_k^\pm]$. Section 5 includes several examples of honest and Dolev-Yao strands.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and the actions of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA strands evolve over time; the symbol | is used to divide past and future. That is, given a strand $[\ msg_1^\pm,\ \ldots,\ msg_i^\pm \mid msg_{i+1}^\pm,\ \ldots,\ msg_k^\pm\ ]$, messages $msg_1^\pm, \ldots, msg_i^\pm$ are the *past messages*, and messages $msg_{i+1}^\pm, \ldots, msg_k^\pm$ are the *future messages* ($msg_{i+1}^\pm$ is the immediate future message). A strand $[msg_1^\pm, \ldots, msg_k^\pm]$ is shorthand for $[nil \mid msg_1^\pm, \ldots, msg_k^\pm, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the intruder knowledge has no fact of the form $m \in \mathcal{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m \notin \mathcal{I}$.

Since the number of states $T_{\Sigma_\mathcal{P}/\mathcal{E}_\mathcal{P}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{\mathcal{E}_\mathcal{P}} \in T_{\Sigma_\mathcal{P}/\mathcal{E}_\mathcal{P}}$ Maude-NPA explores *state patterns* $[t(x_1, \ldots, x_n)]_{\mathcal{E}_\mathcal{P}} \in T_{\Sigma_\mathcal{P}/\mathcal{E}_\mathcal{P}}(\mathcal{X})$ on the free $(\Sigma_\mathcal{P}, \mathcal{E}_\mathcal{P})$-algebra over a set of variables $\mathcal{X}$. In this way, a state pattern $[t(x_1, \ldots, x_n)]_{\mathcal{E}_\mathcal{P}}$ represents not a single concrete state but a possibly infinite set of such states, namely all the *instances* of the pattern $[t(x_1, \ldots, x_n)]_{\mathcal{E}_\mathcal{P}}$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

The semantics of Maude-NPA is expressed in terms of a Maude rewrite theory, including *rewrite rules* that describe how a protocol moves from one state to another via the intruder's interaction with it [14]. One uses Maude-NPA to find an attack by specifying an insecure state pattern called an *attack pattern*. Maude-NPA attempts to find a path from an initial state to the attack pattern via *backwards narrowing* (using the narrowing capabilities of Maude [7] but with the reversed orientation of the rewrite rules). That is, a sequence from an initial state to an attack state is searched *in reverse* as a *backwards path* from an attack state pattern to an initial state. Maude-NPA attempts to find paths until it can no longer form any backwards narrowing steps, at which point it terminates. If at that point it has not found an initial state, the attack pattern is judged *unreachable*; providing a proof of security rather than finding attacks. However, note that Maude-NPA places *no bound on the number of sessions*, so reachability is undecidable in general. Maude-NPA does not achieve termination by any data abstraction, e.g. a bounded number of nonces. Instead, the tool makes use of a number of sound and complete state space reduction techniques that help to identify unreachable and redundant states [16], and thus make termination more likely.

4 Protocol Transformation

Maude-NPA relies on equational unification to perform each backwards narrowing step. Some cryptographic properties often involve the development of dedicated algorithms (see [4]). Maude-NPA provides built-in support for theories involving symbols with any combination of associativity (A), commutativity (C), and identity (U) axioms. Furthermore, by relying on the variant-based equational unification [7,17], Maude-NPA allows users to augment the basic set of equational axioms supported with rewrite rules such as cancellation of encryption and decryption, Diffie-Hellman exponentiation [11], exclusive-or [30], and some approximations of homomorphic encryption [15,34].

4.1 Finite Variant Theories

An equational theory $(\Sigma, \mathcal{E})$ is often decomposed into a disjoint union $\mathcal{E} = E \uplus B$, where B is a set of algebraic axioms (which are implicitly expressed in Maude as operator attributes `assoc`, `comm`, and `id:` keywords) and E consists of variant equations that are implicitly oriented from left to right as a set $\vec{E}$ of rewrite rules (and operationally used as simplification rules modulo B).

Definition 1 (Decomposition [17]**).** *Let $(\Sigma, \mathcal{E})$ be an order-sorted equational theory. We call $(\Sigma, B, \vec{E})$ a* decomposition *of $(\Sigma, \mathcal{E})$ if $\mathcal{E} = E \uplus B$ and $(\Sigma, B, \vec{E})$ is an order-sorted rewrite theory satisfying the following properties:*

1. *B is* regular*, i.e., for each $t = t'$ in B, we have $Var(t) = Var(t')$, and* linear*, i.e., for each $t = t'$ in B, each variable occurs only once in t and in t'.*

2. *B is* sort-preserving*, i.e., for each $t = t'$ in B and substitution σ, we have $t\sigma \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ iff $t'\sigma \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$. Furthermore, for each equation $t = t'$ in B, all variables in $Var(t)$ and $Var(t')$ have a common top sort.*
3. *B has a finitary and complete unification algorithm.*
4. *The rewrite rules in $\vec{E}$ are* convergent*, i.e., confluent, terminating, and coherent modulo B, and* sort-decreasing*.*

In a decomposition, for each term $t \in \mathcal{T}_{\Sigma}(\mathcal{X})$, there is a unique (up to B-equivalence) $(\vec{E}, B)$-irreducible term that can be obtained by rewriting t to its *normal* form, which is denoted by $t\downarrow_{\vec{E},B}$. We often abuse notation and say that $(\Sigma, B, \vec{E})$ is a decomposition of an order-sorted equational theory $(\Sigma, \mathcal{E})$ even if $\mathcal{E} \neq E \uplus B$ but E is instead the explicitly extended B-coherent completion of a set E' such that $\mathcal{E} = E' \uplus B$ (see [17]).

Example 1. The property associated to Diffie-Hellman exponentiation is described using the following equational theory in Maude, including an auxiliary associative-commutative symbol $*$ for exponents so that $(z^x)^y = (z^y)^x = z^{x*y}$.

```
fmod DH-FVP is
  sorts Exp Nonce NeNonceSet Gen .
  subsort Nonce < NeNonceSet . subsort Gen < Exp .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm] .
  var X : Exp . vars Y Z : NeNonceSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

Note that X admits any exponentiation and Y and Z are restricted to non-empty multisets of nonces. For an arbitrary term g of sort Gen and three arbitrary terms n_A, n_B, n_C of sort Nonce, $t = exp(exp(exp(g, n_A), n_B), n_C)$ is simplified into $t\downarrow_{\vec{E},B} = exp(g, n_A * n_B * n_C)$.

In order to provide a finitary and complete unification algorithm for a decomposition $(\Sigma, B, \vec{E})$, the *folding variant narrowing* strategy is defined in [17]. Intuitively, an $(\vec{E}, B)$-*variant* of a term t is the $(\vec{E}, B)$-irreducible form of an *instance $t\sigma$* of t. That is, the variants of t are all of the possible $(\vec{E}, B)$-irreducible terms to which instances of t evaluate.

Definition 2 (Term Variant [8, 17]**).** *Given a term t and a decomposition $(\Sigma, B, \vec{E})$, we say that (t', θ) is a* variant *of t if $t' =_B (t\theta)\downarrow_{\vec{E},B}$, where $Dom(\theta) \subseteq Var(t)$ and $Ran(\theta) \cap Var(t) = \emptyset$.*

Example 2. Following Example 1, the set of variants for the term $exp(X, Y)$ is infinite, since we have $(exp(X', Y * Y'), \{X \mapsto exp(X', Y')\})$, $(exp(X'', Y * Y' * Y''), \{X \mapsto exp(exp(X'', Y''), Y')\})$, ...

It is possible to compute a complete and finite set of variants for some equational theories.

Definition 3 (Complete set of Variants [17]**).** *Given a decomposition* $(\Sigma, B, \vec{E})$ *and a term* t*, we write* $[\![t]\!]_{\vec{E},B}$ *for a* complete *set of variants of* t*, i.e., for any variant* (t_2, θ_2) *of* t*, there is a variant* $(t_1, \theta_1) \in [\![t]\!]_{\vec{E},B}$ *such that* $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$*, where* $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$ *iff there is a substitution* ρ *such that* $(\theta_1\rho)|_{Var(t)} =_B (\theta_2{\downarrow_{\vec{E},B}})|_{Var(t)}$ *and* $t_1\rho =_B t_2$*. An equational theory has the* finite variant property *(FVP) (also called* finite variant theory*) iff for all* $t \in \mathcal{T}_\Sigma(\mathcal{X})$*,* $[\![t]\!]_{\vec{E},B}$ *is a finite set.*

Example 3. Following Example 2, there exists a complete and finite set of variants for the term $exp(X, Y)$: the variant $(exp(X, Y), id)$ and the variant $(exp(X', Y * Y'), \{X \mapsto exp(X', Y')\})$. Any other variant includes a substitution not in irreducible form.

4.2 Constructor Finite Variant Theories

Quite often, the signature Σ of a decomposition $(\Sigma, B, \vec{E})$, on which $\mathcal{T}_{\Sigma/B}$ is defined, has a natural subsignature of *constructor* symbols Ω. The elements of the *canonical algebra* $C_{\Sigma/\vec{E},B} = \{[t{\downarrow_{\vec{E},B}}]_B \mid t \in \mathcal{T}_\Sigma\}$, i.e., the B-equivalence classes computed by $\vec{E}, B$-simplification, are Ω-terms, whereas the other symbols are viewed as functions which are simplified into constructor symbols.

Proverif [6] already incorporated this distinction between what they called *destructor and constructor symbols* time ago in contrast to other crypto tools such as AKISS [5], Maude-NPA [1], OFMC [29], Scyther [9], Scyther-proof [25], and Tamarin [2]. In the rest of the paper, we exploit this distinction in Maude-NPA without altering the tool.

A decomposition $(\Sigma, B, \vec{E})$ *protects* a *constructor decomposition* $(\Omega, B_\Omega, \vec{E}_\Omega)$ iff $\Omega \subseteq \Sigma$, $B_\Omega \subseteq B$, and $\vec{E}_\Omega \subseteq \vec{E}$, and for all $t, t' \in \mathcal{T}_\Omega(\mathcal{X})$ we have: (i) $t =_{B_\Omega} t' \iff t =_B t'$, (ii) $t = t{\downarrow_{\vec{E}_\Omega,B_\Omega}} \iff t = t{\downarrow_{\vec{E},B}}$, and (iii) $C_{\Omega/\vec{E}_\Omega,B_\Omega} = C_{\Sigma/\vec{E},B}|_\Omega$. A *constructor decomposition* $(\Omega, B_\Omega, \emptyset)$ is called *free.* A decomposition $(\Sigma, B, \vec{E})$ is called *sufficiently complete* with respect to a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$ iff for each $t \in \mathcal{T}_\Sigma$ we have: (i) $t{\downarrow_{\vec{E},B}} \in \mathcal{T}_\Omega$, and (ii) if $u \in \mathcal{T}_\Omega$ and $u =_B v$, then $v \in \mathcal{T}_\Omega$. This ensures that if any element in an equivalent class is a constructor term, all the other elements are also constructor.

Example 4. We can extend the equational theory of Example 1 to protect a constructor subsignature[3] by overloading symbol `exp` to use the former[4] sorts Exp and Gen.

```
fmod DH-CFVP is
  sorts Exp Nonce NeNonceSet Gen .
```

[3] Operator declarations labeled `ctor`, their associated sorts, and no equation.

[4] This equational theory, as well as all the ones in Section 5, should be parametric on sorts `Gen`, `GenP` and `Nonce` but we omit such more general-purpose definitions for simplicity (see [7] for details on parametric equational theories).

```
  subsort Nonce < NeNonceSet .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm ctor] .
  var X : Gen . vars Y Z : NeNonceSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

For an arbitrary term g of sort Gen and three arbitrary terms n_A, n_B, n_C of sort Nonce, $t = exp(exp(exp(g, n_A), n_B), n_C)$ is simplified into the constructor term $t{\downarrow_{\vec{E},B}} = exp(g, n_A * n_B * n_C)$.

The notion of a constructor variant, rather than a variant, is defined in [27].

Definition 4 (Constructor Variant [27]**).** *Given a decomposition $(\Sigma, B, \vec{E})$ protecting a constructor decomposition $(\Omega, B_\Omega, \vec{E}_\Omega)$ and a Σ-term t, we say that a variant (t', θ) of t is a* constructor variant *if $t' \in \mathcal{T}_\Omega(\mathcal{X})$.*

Example 5. Following Example 4, the set of constructor variants for the term $exp(X, Y)$ is infinite, as in Example 2, since we have $(exp(X', Y * Y'), \{X \mapsto exp(X', Y')\})$, $(exp(X'', Y * Y' * Y''), \{X \mapsto exp(exp(X'', Y''), Y')\})$, ...

Definition 5 (Complete set of Constructor Variants [27]**).** *Given a decomposition $(\Sigma, B, \vec{E})$ protecting a constructor decomposition $(\Omega, B_\Omega, \vec{E}_\Omega)$ and a Σ-term t, we write $[\![t]\!]^\Omega_{\vec{E},B}$ for a* complete *set of constructor variants of t, i.e., for any constructor variant (t_2, θ_2) of t, there is a constructor variant $(t_1, \theta_1) \in [\![t]\!]^\Omega_{\vec{E},B}$ such that $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$. A decomposition $(\Sigma, B, \vec{E})$ has the* constructor finite variant property *(CFVP) (or it is called a* constructor finite variant theory*) iff for all $t \in \mathcal{T}_\Sigma(\mathcal{X})$, $[\![t]\!]^\Omega_{\vec{E},B}$ is a finite set.*

Example 6. Following Example 5, there exists a finite and complete set of constructor variants for the term $exp(X, Y)$ where X is of sort Exp, since we have $(exp(XG, Y * Y'), \{X \mapsto exp(XG, Y')\})$ where XG is a new variable of sort Gen instead of sort Exp.

An algorithm for computing $[\![t]\!]^\Omega_{\vec{E},B}$ is provided in [32] for equational theories that are FVP. This algorithm assumes an extra condition called *preregular below*, i.e., a term cannot have a constructor typing above a non-constructor typing.

Definition 6 (Preregular below [27]**).** *Given a decomposition $(\Sigma, B, \vec{E})$ protecting a constructor decomposition $(\Omega, B_\Omega, \vec{E}_\Omega)$, the (preregular) order-sorted signature $(\Sigma, <)$ is called* preregular below *iff $\forall t \in \mathcal{T}_\Sigma(\mathcal{X})$, $ls_\Omega(t) = ls_\Sigma(t)$.*

Example 7. Consider the following equational theory

```
fmod DH-NoPreregularBelow is
  sorts Nonce NeNonceSet GenSub Gen ExpSub Exp .
  subsort GenSub < Gen . subsort ExpSub < Exp .
  subsort Nonce < NeNonceSet .
  op gSub : -> GenSub [ctor] .
  op g : -> Gen [ctor] .
  op exp : GenSub NeNonceSet -> ExpSub .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
endfm
```

The signature is not preregular below since, given an arbitrary term n_A of sort Nonce, the least sort of the term $exp(gSub, n_A)$ is ExpSub in the original signature but Exp in the constructor subsignature.

The set of constructor variants of the form $[\![\langle l, r\rangle]\!]^{\Omega}_{\vec{E},B}$, where l and r are, respectively, the lefthand and righthand sides of a rewrite rule in a rewrite theory, play a crucial role in the following theory transformation $\mathcal{R} \mapsto \mathcal{R}^{\Omega}_{l,r}$ from [28].

Definition 7 **($\mathcal{R} \mapsto \mathcal{R}^{\Omega}_{l,r}$ [28]).** *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is CFVP and preregular below, the rewrite theory $(\Sigma, B \uplus E, R^{\Omega}_{l,r})$ is defined as $R^{\Omega}_{l,r} = \{l' \to r' \mid l \to r \in R \wedge (\langle l', r'\rangle, \sigma) \in [\![\langle l, r\rangle]\!]^{\Omega}_{\vec{E},B}\}$.*

Section 5 shows how several protocols are transformed using a protocol transformation that relies on this program transformation.

Example 8. Any expression of the form $exp(X, Y)$, where X is of sort Exp and Y is of sort NeNonceSet, occurring in any lefthand or righthand side of a rule in a rewrite theory will be replaced by the constructor variant shown in Example 6.

Theorem 1 **(**[28, Theo. 7]**).** *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$, it is CFVP, it is sufficiently complete with respect to $(\Omega, B_\Omega, \emptyset)$, and Σ is preregular below, then the rewrite theory $(\Sigma, B_\Omega, R^{\Omega}_{l,r})$ is ground semantically equivalent to $(\Sigma, B \uplus E, R)$.*

The equational theory for Diffie-Hellman of Example 4 is sufficiently complete w.r.t. its constructor subsignature, since any ground term rooted by symbol `exp` is either already using the constructor typing or can be simplified into the constructor typing of `exp`. However, some other theories of interest are not.

Example 9. Consider the cancellation of encryption and decryption.

```
fmod DE is
  sorts Msg Key .
  op enc : Key Msg -> Msg [ctor] .
  op dec : Key Msg -> Msg .
  var K : Key . vars X : Msg .
  eq dec(K,enc(K,X)) = X [variant] .
endfm
```

Given arbitrary keys k_1, k_2 and an arbitrary term a, the term $dec(k_1, enc(k_2, a))$ cannot be reduced.

Terms that cannot be simplified into a constructor term are understood as an erroneous expression and discarded. This is the behaviour of *destructor symbols* in Proverif [6], i.e., functions that may fail. In the rest of the paper, we relax the condition on sufficiently completeness of Theorem 1 and follow the spirit of Proverif's approach[5]: a NF rewrite theory below ensures that erroneous expressions cannot occur in the righthand sides of rewrite rules or in equations, preventing any function to capture that any of its arguments fails. Typical security protocols do however not satisfy the conditions of [28], and in particular all protocols studied in Section 5 did not.

Definition 8 (NF Rewrite Theory). *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$,* erroneous terms *are defined as* $\mathcal{E}rr_\bot = \{t \in \mathcal{T}_\Sigma(\mathcal{X}) \mid \nexists\sigma : (t\sigma)\downarrow_{\vec{E},B} \in \mathcal{T}_\Omega(\mathcal{X})\}$ *whereas* possibly erroneous terms *are defined as* $\mathcal{E}rr_\top = \{t \in \mathcal{T}_\Sigma(\mathcal{X}) \mid \exists\sigma : (t\sigma)\downarrow_{\vec{E},B} \notin \mathcal{T}_\Omega(\mathcal{X})\}$. *We say the rewrite theory is* NF *if, for each $l = r \in E$, $l, r \notin \mathcal{E}rr_\bot$ and, for each $l \to r \in R$, $r|_p \in \mathcal{E}rr_\top \implies \exists q : l|_q =_B r|_p$.*

Theorem 2. *Given a NF rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$ and it is CFVP, then any term reachable from a constructor term is also constructor.*

Proof. By induction on the length of the narrowing sequence $t_0 \rightsquigarrow^n t_n$. If $n = 0$, then $t_n = t_0$ and t_0 is constructor. If $n > 0$, then $t_0 \rightsquigarrow_\sigma t_1 \rightsquigarrow^{n-1} t_n$ s.t. $\sigma \in CSU_{E \uplus B}(t_0|_p = l)$ and $t_1 = (t_0[r]_p)\sigma$. Since t_0 is a constructor term, there is no equation applicable to t_0, i.e., $(t_0|_p)\sigma =_B (l\sigma)\downarrow_{\vec{E},B}$. Since t_0 is a constructor term, the bindings in $\sigma|_{Var(t_0)}$ contain only constructor terms. Since erroneous expressions do not appear in the equations E, $\sigma|_{Var(l)}$ contains also constructor terms. Since r does not contain any extra possible erroneous expression, t_1 is constructor. The conclusion follows by the induction hypothesis. □

Corollary 1. *Given a NF rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$, it is CFVP, and Σ is preregular below, then the rewrite theory $(\Sigma, B_\Omega, R^\Omega_{l,r})$ is ground semantically equivalent to $(\Sigma, B \uplus E, R)$.*

We have implemented both the algorithm for computing $[\![t]\!]^\Omega_{\vec{E},B}$ provided in [32] for equational theories that are FVP and the rewrite theory transformation $\mathcal{R} \mapsto \mathcal{R}^\Omega_{l,r}$ from [28]. As far as we know, there was no implementation available of the rewrite theory transformation $\mathcal{R} \mapsto \mathcal{R}^\Omega_{l,r}$ of [28]. We have used $[\![t]\!]^\Omega_{\vec{E},B}$ and $\mathcal{R} \mapsto \mathcal{R}^\Omega_{l,r}$ to create a *protocol* transformation available online at `http://safe-tools.dsic.upv.es/cvtool`. This web page accepts a protocol specification, using the Maude-NPA syntax, and returns the transformed version, including strands and attack patterns. The proof of soundness and completeness of the protocol transformation is omitted but relies on Theorem 1 and

[5] A detailed comparison is outside the scope of this paper.

Corollary 1. Informally speaking, Maude-NPA internally transforms a protocol specification into a rewrite theory (see Section 3). This transformed rewrite theory is then transformed using the program transformation $\mathcal{R} \mapsto \mathcal{R}^{\Omega}_{l,r}$. And, finally, this resulting rewrite theory is mapped back into a protocol specification. Note that the web page assumes that the conditions of Corollary 1 are satisfied without enforcing them. All the protocols presented in the next section need the relaxed conditions of application of Corollary 1 to safely apply the protocol transformation. These relaxed conditions allow us to deal with more complex protocol specifications efficiently.

$$
\begin{aligned}
A &\longrightarrow B : g^{N_a}\\
B &\longrightarrow A : g^{N_b}
\end{aligned}
$$

$$
\begin{aligned}
K_A &= (g^{N_b})^{N_a}\\
K_B &= (g^{N_a})^{N_b}\\
K_{AB} &= g^{N_a * N_b}
\end{aligned}
$$

Fig. 1. DH

$$
\begin{aligned}
A &\longrightarrow B : g^{N_a}\\
B &\longrightarrow A : g^{N_b}\\
B &\longrightarrow C : g^{((g^{N_b})^{N_a})}\\
C &\longrightarrow A, B : g^{N_c}
\end{aligned}
$$

$$
\begin{aligned}
K_A &= (g^{N_c})^{(g^{((g^{N_b})^{N_a})})}\\
K_B &= (g^{N_c})^{(g^{((g^{N_a})^{N_b})})}\\
K_C &= (g^{((g^{N_b})^{N_a})})^{N_c}\\
K_{ABC} &= g^{\underline{g^{N_a * N_b}} * N_c}
\end{aligned}
$$

Fig. 2. STR

$$
\begin{aligned}
A &\longrightarrow B, C : aP\\
B &\longrightarrow A, C : bP\\
C &\longrightarrow A, B : cP
\end{aligned}
$$

$$
\begin{aligned}
K_A &= \hat{e}(bP, cP)^a\\
K_B &= \hat{e}(aP, cP)^b\\
K_C &= \hat{e}(aP, bP)^c\\
K_{ABC} &= \hat{e}(P, P)^{a*b*c}
\end{aligned}
$$

Fig. 3. Joux

$$
\begin{aligned}
A &\longrightarrow B, C : \ aP; \{xP\}_A\\
B &\longrightarrow A, C : \ bP; \{yP\}_B\\
C &\longrightarrow A, B : \ cP; \{zP\}_C
\end{aligned}
$$

$$
\begin{aligned}
K_A &= h(\hat{e}(bP, cP)^a; \hat{e}(yP, zP)^x)\\
K_B &= h(\hat{e}(aP, cP)^b; \hat{e}(xP, zP)^y)\\
K_C &= h(\hat{e}(aP, bP)^c; \hat{e}(xP, yP)^z)\\
K_{ABC} &= h(\hat{e}(P, P)^{a*b*c}; \hat{e}(P, P)^{x*y*z})
\end{aligned}
$$

Fig. 4. TAK1

$$
\begin{aligned}
K_A &= \hat{e}(bP, zP)^a \cdot \hat{e}(yP, cP)^a \cdot \hat{e}(bP, cP)^x\\
K_B &= \hat{e}(aP, zP)^b \cdot \hat{e}(xP, cP)^b \cdot \hat{e}(aP, cP)^y\\
K_C &= \hat{e}(aP, yP)^c \cdot \hat{e}(xP, bP)^c \cdot \hat{e}(aP, bP)^z\\
K_{ABC} &= \hat{e}(P, P)^{a*y*c} \cdot \hat{e}(P, P)^{x*b*c} \cdot \hat{e}(P, P)^{a*b*z}
\end{aligned}
$$

Fig. 5. TAK2

$$
\begin{aligned}
K_A &= \hat{e}(yP, cP)^x \cdot \hat{e}(bP, zP)^x \cdot \hat{e}(yP, zP)^a\\
K_B &= \hat{e}(aP, zP)^y \cdot \hat{e}(xP, cP)^y \cdot \hat{e}(xP, zP)^b\\
K_C &= \hat{e}(aP, yP)^z \cdot \hat{e}(xP, bP)^z \cdot \hat{e}(xP, yP)^c\\
K_{ABC} &= \hat{e}(P, P)^{x*y*c} \cdot \hat{e}(P, P)^{x*b*z} \cdot \hat{e}(P, P)^{a*y*z}
\end{aligned}
$$

Fig. 6. TAK3

$$
\begin{aligned}
K_A &= \hat{e}(bP + h(bP; yP)yP, cP + h(cP; zP)zP)^{a+(h(aP;xP)*x)}\\
K_B &= \hat{e}(aP + h(aP; xP)xP, cP + h(cP; zP)zP)^{b+(h(bP;yP)*y)}\\
K_C &= \hat{e}(aP + h(bP; yP)yP, bP + h(bP; yP)yP)^{a+(h(cP;cP)*c)}\\
K_{ABC} &= \hat{e}(P, P)^{(a+(h(aP;xP)*x))*(b+(h(bP;yP)*y))*(c+(h(cP;zP)*z))}
\end{aligned}
$$

Fig. 7. TAK4

5 Case studies

This section presents several increasingly complex case studies: Diffie-Hellman protocol in Section 5.1, STR protocol in Section 5.2, Joux protocol in Section 5.3, and TAK protocols in Section 5.4. The Joux and TAK protocols use bilinear pairing but TAK4 requires properties beyond the encoding of bilinear pairings available in Tamarin, the only crypto tool with such an equational theory.

5.1 The Diffie-Hellman Protocol

In this section, we describe the analysis performed on the Diffie-Hellman (DH) protocol. This protocol was already analysed using Maude-NPA in [11]. DH uses exponentiation to share a secret between two parties. The description of the protocol using an Alice & Bob notation is given in Figure 1.

Alice and Bob agree on a common generator g. Alice sends the generator g raised to the power of a new nonce generated by her. Bob sends the generator g raised to the power of a new nonce generated by him. Both Alice and Bob take the received nonce and raised it to the power of their own respective nonce. The cryptographic property here allows $(g^{N_A})^{N_B} = (g^{N_B})^{N_A} = g^{N_A * N_B}$. This cryptographic property is represented using the equational theory of Example 4.

The informal description of Figure 1 is specified using strands as follows. We represent an exponentiation x^y as $\mathtt{exp}(x, y)$. We represent a nonce N_A as $n(A, f)$ where f is a Fresh variable. We have added the identifiers of the participants in the message exchange for clarity. And we have appended a final encryption of some random secret using the generated key to make explicit the different keys used by the honest participants before and after the transformation.

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::&[+(A; B; exp(g, n(A, f_a))), -(B; A; X),\\ &+ (enc(exp(X, n(A, f_a)), sec(A, f)))]\\ (\textbf{Bob}) :: f_b ::&[-(A; B; Y), +(B; A; exp(g, n(B, f_b))),\\ &- (enc(exp(Y, n(B, f_b)), Sr))]\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::&[+(A; B; exp(g, n(A, f_a))), -(B; A; exp(G, N)),\\ &+ (enc(exp(G, n(A, f_a) * N), sec(A, f)))]\\ (\textbf{Bob}) :: f_b ::&[-(A; B; exp(G, N)), +(B; A; exp(g, n(B, f_b))),\\ &- (enc(exp(G, N * n(B, f_b)), Sr))]\end{aligned}$$

As explained in Example 6, the expression $exp(X{:}Exp, n(A, f_a))$ has only one constructor variant using substitution $X{:}Exp \mapsto exp(G{:}Gen, N{:}NeNonceSet)$. Similarly for $exp(Y{:}Exp, n(B, f_b))$. The duplication of symbols in one defined and one constructor, the coincidence that each defined symbol has only one equation, and the use of associativity and commutativity, makes each strand of the protocols of this paper is replaced by just one strand. This may not always be the case and a strand may be replaced by several new strands (see [28, Example 7]). The Dolev-Yao capabilities for exponentiation are as follows.

$$(\mathbf{DY\_exp\_ctor})[-(G{:}Gen), -(N{:}NeNonceSet), +(exp(G{:}Gen, N{:}NeNonceSet)]$$
$$(\mathbf{DY\_exp\_func})[-(E{:}Exp), -(N{:}NeNonceSet), +(exp(E{:}Exp, N{:}NeNonceSet)]$$

The second one is transformed as follows

$$\begin{aligned}(\mathbf{DY\_exp\_cvar})[&-(exp(G{:}Gen, X{:}NeNonceSet)), -(N{:}NeNonceSet),\\ &+(exp(G{:}Gen, X{:}NeNonceSet * N{:}NeNonceSet))]\end{aligned}$$

5.2 The STR protocol

One extension of the Diffie-Hellman protocol is to consider that every time a new member is joined the exchange key is repeated, allowing for an unbounded number of participants a priori. We consider the tree-party group key agreement protocol *STR* from [21], where *STR* is a short name for **S**kinny **TR**ee. The description of the protocol using an informal Alice & Bob notation is given in Figure 2. The only difference between the cryptographic properties of STR and DH is that we can have an exponentiation as an exponent, where DH could not. Therefore, the only difference to the equational theory of Example 4 is "`subsort Nonce Exp < NeNonceSet`". The equational theory still satisfies all the conditions of Corollary 1. The informal description of Figure 2 is specified using strands as follows, we remove the identifiers of the participants for simplicity.

$$\begin{aligned}(\mathbf{Alice}) :: f_a, f ::[&+(exp(g, n(A, f_a))), -(XB), -(XC),\\ &+(enc(exp(XC, exp(XB, n(A, f_a))), sec(A, f)))]\\ (\mathbf{Bob}) :: f_b ::[&-(XA), +(exp(g, n(B, f_b))), +(exp(g, exp(XA, n(B, f_b)))),\\ &-(XC), -(enc(exp(XC, exp(XA, n(B, f_b))), Sr))]\\ (\mathbf{Carol}) :: f_c ::[&-(XAB), +(exp(g, n(C, f_c))), -(enc(exp(XAB, n(C, f_c)), Sr))]\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}(\mathbf{Alice}) :: f_a, f ::[&+(exp(g, n(A, f_a))), -(exp(G1, NB)), -(exp(G2, NC)),\\ &+(enc(exp(G2, exp(G1, n(A, f_a) * NB) * NC), sec(A, f)))]\\ (\mathbf{Bob}) :: f_b ::[&-(exp(G, NA)), +(exp(g, n(B, f_b))),\\ &+(exp(g, exp(G, n(B, f_b) * NA))), -(exp(G, NC)),\\ &-(enc(exp(G, exp(G, n(B, f_b) * NA) * NC), Sr))]\\ (\mathbf{Carol}) :: f_c ::[&-(exp(G, NAB)), +(exp(g, n(C, f_c))),\\ &-(enc(exp(G, NAB * n(C, f_c)), Sr))]\end{aligned}$$

5.3 The Joux Protocol

When you want to keep the spirit of the Diffie-Hellman protocol, where no extra sharing is necessary apart of the initial broadcast information, an interesting alternative for three participants is the Joux protocol [20], which relies on bilinear pairing. The description of the protocol using an informal Alice & Bob notation is given in Figure 3.

Pairing-based cryptography makes use of a pairing function $\hat{e} : G_1 \times G_2 \rightarrow G_T$ of two cryptographic groups G_1 and G_2 into a third group G_T. Typically, $G_1 = G_2$ and it will be a subgroup of the group of points on an elliptic curve over a finite field, and G_T will be a subgroup of the multiplicative group of a related finite field and the map $\hat{e}$ will be derived from either the Weil or Tate pairing on the elliptic curve. When $G = G_1 = G_2$, the pairing is called *symmetric* and the pairing function $\hat{e}$ is commutative, i.e., if the participants agree on a generator $g \in G$, for any P, Q in G there exist integers i, j s.t. $P = g^i$, $Q = g^j$, $\hat{e}(P,Q) = \hat{e}(g^i, g^j) = \hat{e}(g,g)^{i*j} = \hat{e}(g^j, g^i) = \hat{e}(Q,P)$. In Figure 3, we follow the syntax of [20] and use letter P as the agreed generator. We write aP instead of P^a for P added to itself a times, also called scalar multiplication of P by a. Note that we write `[a]P` in the equational theory below for clarification. The bilinear pairing is specified as follows.

```
fmod BP-CFVP is
  sorts Nonce NeNonceSet Gen GenP Exp ExpP .
  subsort Nonce < NeNonceSet .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm ctor] .
  op p : -> GenP [ctor] .
  op em : GenP GenP -> Gen [ctor comm] .
  op em : ExpP ExpP -> Exp [comm] .
  op [_]_ : NeNonceSet GenP -> ExpP [ctor] .
  op [_]_ : NeNonceSet ExpP -> ExpP .
  var X : Gen . vars Y Z : NeNonceSet . vars P Q : GenP .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
  eq [Z]([Y]P) = [Z * Y]P [variant] .
  eq em([Y]P, [Z]Q) = exp(em(P,Q),Y * Z) [variant] .
endfm
```

We adapted the built-in theory of bilinear pairing of Tamarin [2, 31] to satisfy[6] the conditions of Corollary 1. The informal description of Figure 3 is specified using strands as follows.

[6] Confluence is proved by the absence of critical pairs between the lefthand sides of the three equations. Termination and FVP are proved by strongly right-irreducibility [17], i.e., righthand sides do not unify with any lefthand side. CFVP is proved because it is preregular below.

$$\begin{aligned}
(\textbf{Alice}) :: f_a, f ::& [+([n(A, f_a)]p), -(XB), -(XC),\\
& + (enc(exp(em(XB, XC), n(A, f_a)), sec(A, f))]\\
(\textbf{Bob}) :: f_b ::& [-(XA), +([n(B, f_b)]p), -(XC),\\
& - (enc(exp(em(XA, XC), n(B, f_b)), Sr)]\\
(\textbf{Carol}) :: f_c ::& [-(XA), -(XB), +([n(C, f_c)]p),\\
& - (enc(exp(em(XA, XB), n(C, f_c)), Sr)]
\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}
(\textbf{Alice}) :: f_a, f ::& [+([n(A, f_a)]p), -([NB]PB), -([NC]PC),\\
& + (enc(exp(em(PB, PC), n(A, f_a) * NB * NC), sec(A, f))]\\
(\textbf{Bob}) :: f_b ::& [-([NA]PA), +([n(B, f_b)]p), -([NC]PC),\\
& + (enc(exp(em(PA, PC), n(B, f_b) * NA * NC), Sr)]\\
(\textbf{Carol}) :: f_c ::& [-([NA]PA), -([NB]PB), +([n(C, f_c)]p),\\
& + (enc(exp(em(PA, PB), n(C, f_c) * NA * NB), Sr)]
\end{aligned}$$

5.4 The TAK Group Protocols

The Tripartite Authenticated Key group protocols [3] is a set of authenticated key agreement protocols that still require only one round of communication. It is an improvement of the Joux protocol. The four versions of TAK share the same exchanged message but the computation key is different for each version. The description of the TAK protocol using an informal Alice & Bob notation is given in Figure 4. However, the four different ways of computing the keys are given in Figures 4, 5, 6, and 7. These four protocols use the bilinear pairing cryptographic properties explained in Section 5.3 plus a hash function h and the following additive property (and its symmetric version, since $\hat{e}$ is commutative)

$$\hat{e}(Q, W + Z) = \hat{e}(Q, W) \cdot \hat{e}(Q, Z) \qquad (1)$$

where $+$ is the additive symbol for the group G and $\cdot$ is the additive symbol for the group G_T given $\hat{e} : G \times G \to G_T$. These properties are specified[7] as follows.

```
fmod BPAdd-CFVP is
  sorts Nonce NeNonceSet Gen GenP Exp ExpP ExpT .
  subsort Nonce < NeNonceSet . subsort Exp < ExpT .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [ctor assoc comm] .
  op p : -> GenP [ctor] .
  op em : GenP GenP -> Gen [ctor comm] .
  op em : ExpP ExpP -> Exp [comm] .
  op [_]_ : NeNonceSet GenP -> ExpP [ctor] .
```

[7] The additive property (1) is not supported by the bilinear pairing of Tamarin [2, 31].

```
  op [_]_ : NeNonceSet ExpP -> ExpP .
  op _+_ : NeNonceSet NeNonceSet -> NeNonceSet [ctor assoc comm] .
  op _+_ : ExpP ExpP -> ExpP .
  op _·_ : ExpT ExpT -> ExpT [ctor assoc comm] .
  var X : Gen . vars Y Z : NeNonceSet . vars P Q : GenP .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
  eq [Z]([Y]P) = [Z * Y]P [variant] .
  eq em([Y]P, [Z]Q) = exp(em(P,Q),Y * Z) [variant] .
  eq ([Y]P) + ([Z]P) = [Y + Z]P [variant] .
endfm
```

Note that Property 1 does not appear explicitly in the equational theory above and it is transformed as follows. The addition symbol + is split into two versions, one of them being an associative-commutative constructor and the other one being a defined symbol. A new equation relating these two versions of + is added. And symbol · is simply represented as an associative-commutative constructor. The last, new equation denotes a homomorphic addition and it is easily handled by variant-based unification because it is defined on disconnected sorts ExpP and NeNonceSet (see [34] for approximations of homomorphism following the same idea). For example, the key generated by Alice in TAK4

$$\begin{aligned} K_A = exp(&em([b]p + [h([b]p;[y]p) * y]p, [c]p + [h([c]p;[z]p) * z]p),\\ &a + (h([a]p;[x]p) * x)) \end{aligned}$$

is transformed into the common key

$$\begin{aligned} K_{ABC} = exp(em(p,p),&(a + (h([a]p;[x]p) * x))*\\ &(b + (h([b]p;[y]p) * y)) * (c + (h([c]p;[z]p) * z))) \end{aligned}$$

by applying the last equation two times, followed by the third and the first equations (we underline the replaced subterm)

$$\begin{aligned} &exp(em(\underline{[b]p + [h([b]p;[y]p) * y]p}, \underline{[c]p + [h([c]p;[z]p) * z]p}),\\ &\quad a + (h([a]p;[x]p) * x)) =\\ &exp(\underline{em([b + (h([b]p;[y]p) * y)]p, [c + (h([c]p;[z]p) * z)]p)},\\ &\quad a + (h([a]p;[x]p) * x))) =\\ &\underline{exp(exp(em(p,p), a + (h([a]p;[x]p) * x),}\\ &\quad \underline{(b + h([b]p;[y]p) * y) * (c + h([c]p;[z]p) * z))} =\\ exp(em(p,p),&(a + (h([a]p;[x]p) * x))*\\ &\quad (b + (h([b]p;[y]p) * y)) * (c + (h([c]p;[z]p) * z))) \end{aligned}$$

If the non-constructor version of + becomes associative-commutative, then the theory is not FVP. This equational theory works for all the TAK protocols even if it is not the most general possible; it is left for future work whether Property 1 can be encoded directly. This equational theory satisfies the conditions of Corollary 1. The original and transformed versions of TAK1, TAK2, and TAK3 are omitted but are available online. The informal description of the TAK4 protocol given in Figure 7 is specified using strands as follows.

$$\begin{aligned}(\textbf{Alice}) :: f_a, f_x, f :: [&+([n(A,f_a)]p), +([n(A,f_x)]p), -(BP), -(YP), -(CP), -(ZP),\\ &+(enc(exp(\hat{e}(BP + [h(BP;YP)]YP, CP + [h(CP;ZP)]ZP),\\ &\quad f_a + h([n(A,f_a)]p;[n(A,f_x)]p * f_x)), sec(A,f)))]\end{aligned}$$

$$\begin{aligned}(\textbf{Bob}) :: f_b, f_y :: [&-(AP), -(XP), +([n(B,f_b)]p), +([n(B,f_y)]p), -(CP), -(ZP),\\ &-(enc(exp(\hat{e}(AP + [h(AP;XP)]XP, CP + [h(CP;ZP)]ZP),\\ &\quad f_b + h([n(B,f_b)]p;[n(B,f_y)]p * f_y)), Sr))]\end{aligned}$$

$$\begin{aligned}(\textbf{Carol}) :: f_c, f_z :: [&-(AP), -(XP), -(BP), -(YP), +([n(C,f_c)]p), +([n(C,f_z)]p),\\ &-(enc(exp(\hat{e}(AP + [h(AP;XP)]XP, BP + [h(BP;YP)]YP),\\ &\quad f_c + h([n(C,f_c)]p;[n(C,f_c)]p * f_c)), Sr))]\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}(\textbf{Alice}) :: f_a, f_x, f :: [&+([n(A,f_a)]p), +([n(A,f_x)]p),\\ &-([NB]PB), -([NY]PB), -([NC]PC), -([NZ]PC),\\ &+(enc(exp(\hat{e}(PB,PC), (NB + (h([NB]PB;[NY]PB) * NY))\\ &\qquad * (NC + (h([NC]PC;[NZ]PC) * NZ))\\ &\qquad * (f_a + h([n(A,f_a)]p;[n(A,f_x)]p * f_x))), sec(A,f)))]\end{aligned}$$

$$\begin{aligned}(\textbf{Bob}) :: f_b, f_y :: [&-([NA]PA), -([NX]PA), +([n(B,f_b)]p), +([n(B,f_b)]p),\\ &-([NC]PC), -([NZ]PC),\\ &-(enc(exp(\hat{e}(PA,PC), (NA + (h([NA]PA;[NX]PA) * NX))\\ &\qquad * (NC + (h([NC]PC;[NZ]PC) * NZ))\\ &\qquad * (f_b + h([n(B,f_b)]p;[n(B,f_y)]p * f_y))), Sr))]\end{aligned}$$

$$\begin{aligned}(\textbf{Carol}) :: f_c, f_z :: [&-([NA]PA), -([NX]PA), -([NB]PB), -([NB]PB),\\ &+([n(C,f_c)]p), +([n(C,f_c)]p),\\ &-(enc(exp(\hat{e}(PA,PB), (NA + (h([NA]PA;[NX]PA) * NX))\\ &\qquad * (NB + (h([NB]PB;[NY]PB) * NY))\\ &\qquad * (f_c + h([n(C,f_c)]p;[n(C,f_z)]p * f_z))), Sr))]\end{aligned}$$

6 Experiments

We have evaluated all the protocols of Section 5, both before and after the transformation. For DH, STR and Joux, we consider two general attack patterns, one for authentication and another for secrecy of the session key. For TAKs we consider only a secrecy attack pattern. Both properties of DH are insecure,

		Before Transformation		*After Transformation*			
Protocol	*Property*	*States*	*Time (ms)*	*States*	*Time (ms)*	*States (%)*	*Speedup*
DH	auth	137	308,066	111	132,756	81.02	2.32
	secrecy	138	322,731	104	142,015	75.36	2.27
STR	auth	34	43,144	31	16,010	91.18	2.69
	secrecy	250	1,016,469	117	408,960	46.80	2.49
Joux	auth	38	85,579	37	30,012	97.37	2.85
	secrecy	55	247,712	58	78,384	105.45	3.16
TAK1	secrecy	25	259,619	20	126,998	80.00	2.04
TAK2	secrecy	67	365,797	46	152,842	68.66	2.39
TAK3	secrecy	117	670,775	67	216,350	57.26	3.10
TAK4	secrecy	57	371,770	48	181,850	84.21	2.04

Table 1. Experimental results for the transformed protocols.

authentication of STR is insecure but secrecy is secure [21], both properties of Joux are insecure [20], and TAK1, TAK2, TAK3, and TAK4 are secure [3].

In Table 1, we report both the number of states and the generation time of the search space associated to each attack pattern. The transformation itself is almost immediate, since the equational theories in these examples are not so complex. The time and space difference is shown in columns *States (%)* and *Speedup*. These columns demonstrate that the difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable in three different aspects. First, for the STR protocol, the transformed protocol produces only 46.80% of the total number of states of the untransformed version. Second, for the TAK3 protocol, the execution time of the transformed protocol is three times faster than the untransformed version. Third, for the Joux protocol, even if the analysis of the transformed protocol produces more states than the analysis of the untransformed protocol, the execution time is three times faster.

All the experiments were conducted on a PC with a 3.3GHz Intel Xeon E5-1660 and 64GB RAM. We used Maude v3.0 [7] and Maude-NPA v3.1.4 [1]. The protocol specifications of both before and after the transformation and the output of each analysis are available at `http://safe-tools.dsic.upv.es/cvtool`.

7 Conclusions

Our first contribution is a protocol transformation that can safely get rid of cryptographic properties under some mild conditions. We have demonstrated with experiments that the time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable (an average speedup of 2.54). A similar idea is presented in [23] for XOR and in [22] for DH. These works are however not comparable to ours, since they are not protocol transformations but classes of protocols were the analysis using Proverif is sound. In [19], protocol transformations are studied. However the goal it not to optimize the verification, but to ensure that a transformed protocol satisfies some security goals, when the source protocol did, focusing on incremental protocol construction. Our second contribution is an encoding of the theory of bilinear pairing into Maude-NPA. This encoding goes beyond the encoding of bilinear pairing available in the Tamarin tool, the only crypto tool with such an equational theory. Since Tamarin [10] and AKISS [5] use term variants, they could be adapted to use both our protocol transformation and our encoding of the theory of bilinear pairing. They may even be useful for other crypto tools with more limited crypto properties such as ProVerif [6], OFMC [29], Scyther [9] or Scyther-proof [25]. Specially, since Proverif [6] already incorporated the notion of destructors and constructors time ago. As future work, we plan to study how the protocol transformation applies to other families of protocols and crypto properties such as homomorphisms [34].

References

1. Maude-NPA manual v3.1, `http://maude.cs.illinois.edu/w/index.php/Maude_Tools:_Maude-NPA`
2. The Tamarin-Prover Manual June 4, 2019). Available on: `https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf`
3. Al-Riyami, S.S., Paterson, K.G.: Tripartite authenticated key agreement protocols from pairings. In Cryptography and Coding. pp. 332–359. Springer (2003)
4. Baader, F., Snyder, W.: Unification Theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 447–533. Elsevier Science (2001)
5. Baelde, D., Delaune, S., Gazeau, I., Kremer, S.: Symbolic verification of privacy-type properties for security protocols with XOR. In: 30th IEEE Computer Security Foundations Symposium, CSF 2017, pp. 234–248. IEEE Computer Society (2017).
6. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Found. and Trends in Privacy and Security **1**(1–2), 1–135 (2016)
7. Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martií-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.: Maude Manual (Version 3.0). Tech. rep., SRI Int. Computer Science Laboratory (2020), available at: `http://maude.cs.uiuc.edu`
8. Comon-Lundh, H., Delaune, S.: The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In: Proc. of the 16th Int. Conference on Rewriting Techniques and Applications (RTA 2005). LNCS, vol. 3467, pp. 294–307. Springer.
9. Cremers, C.J.F.: The scyther tool: Verification, falsification, and analysis of security protocols. In Computer Aided Verification, 20th Int. Conference, CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer (2008).
10. Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond subterm-convergent equational theories in automated verification of stateful protocols. In Principles of Security and Trust, 2017. LNCS, vol. 10204, pp. 117–140. Springer (2017).
11. Escobar, S., Hendrix, J., Meadows, C., Meseguer, J.: Diffie-Hellman Cryptographic reasoning in the Maude-NRL protocol analyzer. In: Proc. 2nd Int. Workshop on Security and Rewriting Techniques (SecReT 2007)
12. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. Theoretical Compute Science **367**(1–2), 162–202 (2006)
13. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In: FOSAD 2007/2008/2009 Tutorial Lectures. LNCS, vol. 5705, pp. 1–50. Springer (2009).

14. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2008/2009 Tutorial Lectures. LNCS, vol. 5705, pp. 1–50. Springer (2009)
15. Escobar, S., Kapur, D., Lynch, C., Meadows, C.A., Meseguer, J., Narendran, P., Sasse, R.: Protocol analysis in Maude-NPA using unification modulo homomorphic encryption. In Proc. of PPDP 2011, pp. 65–76. ACM (2011).
16. Escobar, S., Meadows, C.A., Meseguer, J., Santiago, S.: State space reduction in the Maude-NRL Protocol Analyzer. Inf. Comput. **238**, 157–186 (2014).
17. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. J. Log. Algebr. Program. **81**(7-8), 898–928 (2012).
18. Fabrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: why is a security protocol correct? In: Proc. IEEE Symp. on Security and Privacy, 160–171, 1998.
19. Guttman, J.D.: Security goals and protocol transformations. In Theory of Security and Applications. pp. 130–147. Springer (2012)
20. Joux, A.: A one round protocol for tripartite diffie–hellman. In: Bosma, W. (ed.) Algorithmic Number Theory. pp. 385–393. Springer (2000).
21. Kim, Y., Perrig, A., Tsudik, G.: Communication-efficient group key agreement. In Trusted Information. pp. 229–244. Springer (2001)
22. Küsters, R., Truderung, T.: Using proverif to analyze protocols with diffie-hellman exponentiation. In: IEEE Computer Security Foundations, pp. 157–171 (2009)
23. Küsters, R., Truderung, T.: Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. J. Autom. Reason. **46**(3-4), 325–352, 2011
24. Meadows, C.: The NRL protocol analyzer: An overview. Journal of Logic Programming **26**(2), 113–131 (1996)
25. Meier, S., Cremers, C., Basin, D.: Strong invariants for the efficient construction of machine-checked protocol security proofs. In: 2010 23rd IEEE Computer Security Foundations Symposium. pp. 231–245 (2010)
26. Meseguer, J.: Conditional Rewriting Logic as a United Model of Concurrency. Theoretical Computer Science **96**(1), 73–155 (1992).
27. Meseguer, J.: Variant-based satisfiability in initial algebras. Sci. Comput. Program. **154**, 3–41 (2018).
28. Meseguer, J.: Generalized rewrite theories, coherence completion, and symbolic methods. J. Log. Algebr. Meth. Program. **110** (2020).
29. Mödersheim, S., Viganò, L.: The open-source fixed-point model checker for symbolic analysis of security protocols. In: Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 166–194. Springer (2009).
30. Sasse, R., Escobar, S., Meadows, C., Meseguer, J.: Protocol analysis modulo combination of theories: A case study in maude-npa. In: Security and Trust Management. pp. 163–178. Springer (2011)
31. Schmidt, B., Sasse, R., Cremers, C., Basin, D.A.: Automated verification of group key agreement protocols. In: 2014 IEEE Symp. on Security and Privacy, SP 2014, pp. 179–194. IEEE Computer Society (2014).
32. Skeirik, S., Meseguer, J.: Metalevel algorithms for variant satisfiability. J. Log. Algebraic Methods Program. 96, 81–110 (2018)
33. TeReSe: Term Rewriting Systems. Cambridge University Press, Cambridge (2003)
34. Yang, F., Escobar, S., Meadows, C.A., Meseguer, J., Narendran, P.: Theories of homomorphic encryption, unification, and the finite variant property. In: Proc. of PPDP 2014. pp. 123–133. ACM (2014).