Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols

Jannik Dreier, Charles Duménil, Steve Kremer, Ralf Sasse

Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols

Jannik Dreier[1], Charles Duménil[1], Steve Kremer[1], and Ralf Sasse[2]

(1) LORIA, CNRS & Inria & Université de Lorraine, France
(2) Department of Computer Science, ETH Zurich, Switzerland

Abstract. The TAMARIN prover is a state-of-the-art protocol verification tool. It supports verification of both trace and equivalence properties, a rich protocol specification language that includes support for global, mutable state and allows the user to specify cryptographic primitives as an arbitrary subterm convergent equational theory, in addition to several built-in theories, which include, among others, Diffie-Hellman exponentiation.
In this paper, we improve the underlying theory and the tool to allow for more general user-specified equational theories: our extension supports arbitrary convergent equational theories that have the finite variant property, making TAMARIN the first tool to support at the same time this large set of user-defined equational theories, protocols with global mutable state, an unbounded number of sessions, and complex security properties. We demonstrate the effectiveness of this generalization by analyzing several protocols that rely on blind signatures, trapdoor commitment schemes, and ciphertext prefixes that were previously out of scope.

1 Introduction

The goal of security protocols is to protect communications against malicious behavior of third parties which may monitor or completely control the network, and sometimes even legitimately participate in the protocol. Typical properties that such protocols aim to achieve are confidentiality, authentication, as well as anonymity or unlinkability. To this end, security protocols employ cryptographic primitives. The most usual primitives are encryption and signatures, either symmetric or asymmetric, and cryptographic hash functions. Some security goals may however require more advanced primitives: digital cash may rely on blind signatures to ensure anonymity [21], e-voting protocols may use trapdoor commitments [26] or plaintext equivalence tests [23] to achieve receipt-freeness, and verifiability may rely on zero-knowledge proofs [23,1].

Effective tools, e.g., [4,9,19,15,22,25,10], for automated analysis of security protocols exist, in particular in the case of simple authentication and confidentiality goals, standard cryptographic primitives, and protocols that do not rely on a global mutable state. There has been active research on extending the class of properties that can be verified, e.g., by considering complex forms of compromise [5], or the more expressive class of equivalence properties [7,10,12,28,6]. Many tools also support user-specified equational theories for modeling less usual cryptographic primitives [9,19,25,10]. Finally, tool support has been devised for protocols that allow for different sessions to update a global, mutable state [3,24].

The TAMARIN prover [25] is a state-of-the-art cryptographic protocol verifier which allows the user at the same time to specify complex security properties (both trace and equivalence properties), to model cryptographic primitives by means of an equational theory, and allows protocols to maintain state information. The class of equational theories supported by the tool is the class of *subterm-convergent* equational theories, in addition to built-in theories for Diffie-Hellman exponentiations, bilinear pairings, and multisets. While the class of subterm-convergent theories includes many usual cryptographic primitives, it does not include primitives such as blind signatures or trapdoor commitment schemes.

Our contributions. In this paper we significantly extend the supported class of equational theories in the TAMARIN prover. We remove the restriction of *subterm*-convergent theories, and now permit an arbitrary convergent theory which has the finite variant property. As the underlying problem is undecidable, we cannot guarantee termination of course. More technically, our extension generalizes *(i)* the underlying techniques used in the TAMARIN prover to reason about adversary knowledge, *(ii)* the normal form conditions that the TAMARIN prover imposes on traces to favor termination, and *(iii)* the correctness proof that the set of considered traces remains complete.

We have implemented these extensions in the TAMARIN prover and demonstrate that, with our generalization, the tool succeeds to effectively analyze diverse protocols that were previously out of scope of automated verification in TAMARIN.

- We studied Chaum's digital cash protocol [11] which uses blind signatures and whose modelling also requires the use of global state. We have verified anonymity, untraceability, as well as unforgeability, which states that no coins can be maliciously created. In previous work using PROVERIF [18], the proof of unforgeability could not be completed due to PROVERIF's difficulties in handling state.
- We also analyzed the FOO e-voting protocol [21] which relies on blind signatures. Vote privacy in this protocol could previously only be analyzed by the AKISS tool [10] and a recent extension of PROVERIF [8]. Using our new version of the TAMARIN prover we have been able to also check vote privacy (modeled as an equivalence property) and furthermore eligibility (modeled as a trace property).
- We also verified the Okamoto e-voting protocol [26] which relies on trapdoor commitments to achieve receipt-freeness. Voter anonymity of this protocol was previously analyzed using the AKISS tool, but we provide the first automated proof of receipt-freeness for this protocol, which was previously only shown manually [16][1].
- Finally, we analyzed the Denning-Sacco and Needham-Schroeder symmetric key protocols with an encryption scheme that has a prefix property, e.g., in CBC mode, as described in [14]. As expected we have found known attacks on these protocols when the prefix property is considered.

Related work. In terms of supported user-specified equational theories, our extension of the TAMARIN prover is comparable to the AKISS tool. While AKISS additionally guarantees termination for subterm-convergent theories, it is limited to a *bounded* number

[1] In a previous version we falsely stated here that PROVERIF does not support the equational theory for trapdoor commitments. PROVERIF does support the equational theory for trapdoor commitments, but was never used to analyze receipt-freeness for this protocol.

of sessions and does not support protocols with else branches. There are only few tools for automated verification for an *unbounded* number of sessions: Maude-NPA [19], Scyther [15], CPSA [22] and PROVERIF [9]. We will now discuss and compare our extension of TAMARIN with each of them.

Scyther [15] is restricted to a fixed set of cryptographic primitives and does not allow for user-specified equational theories. Moreover, it neither supports global mutable state nor verification of equivalence properties.

CPSA [22] was designed for analyzing, essentially, authentication and secrecy properties. The tool was used, in combination with the theorem prover PVS, to analyze stateful protocols [27]. However, like Scyther, it does neither support user-defined equational theories nor the verification of equivalence properties.

Maude-NPA [19] offers support for many equational theories. Regarding convergent theories, the support offered by Maude-NPA is comparable to our extension of the TAMARIN prover, as it also relies on the finite variant property. Maude-NPA treats algebraic properties, such as associative-commutative operators, in a more generic way than TAMARIN, which only offers support for built-in Diffie-Hellman and bilinear pairing theories. However, Maude-NPA does not support global mutable state.

PROVERIF is the reference tool in protocol verification. It offers support for user defined equational theories, and allows for the verification of a rich variety of security properties. Moreover, the abstractions (based on a translation of applied pi calculus processes into Horn clauses) underlying the theory of PROVERIF make it extremely efficient. However, these abstractions may also cause false attacks, which make the tool unsuitable to analyze protocols with global state. An extension of PROVERIF, called STATVERIF [3], tries to overcome this shortcoming. However, the support for stateful protocols that can be effectively analyzed by STATVERIF remains partial. For instance, only a fixed number of state cells may be declared and non-termination arises frequently. Moreover, only secrecy properties can be verified with STATVERIF.

We also want to mention SAPIC [24], a front-end to TAMARIN which permits to specify protocols in a stateful extension of the applied pi calculus and has been used successfully for stateful protocols. It will benefit from our extension of TAMARIN.

Outline. We present necessary preliminaries in Section 2. Our extensions of the theory and tool are described in Section 3, and we evaluate them with the case studies shown in Section 4. We give concluding remarks in Section 5.

2 Preliminaries

We explain our model of protocols and their security properties and the adversary deduction after covering the representation of messages as terms.

2.1 Representing messages as terms

As usual in symbolic analysis of cryptographic protocols we model messages and operations on them by terms in an order-sorted term algebra, equipped with an equational theory. We assume given a signature Σ_{Op} defining operators and their arity. Additionally, we use three sorts, a top sort *msg* with two incomparable subsorts: terms of sort

fr model nonces, keys, and random values in general; terms of sort *pub* model publicly known values. For each sort s there is a countable set of variables, $\mathcal{V}_s$, and we call their union $\mathcal{V}$. Similarly we suppose a countable set of names $\mathcal{N}_s$ per sort, and denote their union by $\mathcal{N}$. The set of terms $T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$ contains variables in $\mathcal{V}$, names in $\mathcal{N}$, and is closed under application of operators in Σ_{Op}. A term t is ground when it contains no variables and we denote the set of ground terms by $T_{\Sigma_{Op}}(\mathcal{N})$, or simply $T_{\Sigma_{Op}}$. We also use standard notations for *positions*: a position p in t is a finite sequence of integers, the empty sequence being denoted by $[]$, and we write $t|_p$ for the subterm of t at position p, where (1) if $p = []$, then $t|_p = t$, (2) if $p = [i] \cdot p'$, and $t = f(t_1, \ldots, t_n)$ for $f \in \Sigma_{Op}$ and $1 \leq i \leq n$ then $t|_p = t_i|_{p'}$, and (3) otherwise $t|_p$ is not defined and p is not a valid position. A *substitution* σ is a function from variables to terms. As usual, we homomorphically lift σ to terms and use postfix notations, i.e., we write $t\sigma$ for $\sigma(t)$.

For a signature Σ_{Op}, an *equation* is an unordered pair of terms $s, t \in T_{\Sigma_{Op}}(\mathcal{V})$ written $s = t$. For a set of equations E over Σ_{Op} the resulting *equational presentation* is $\mathcal{E} = (\Sigma_{Op}, E)$. We call the smallest Σ_{Op}-congruence closure containing all instances of E the corresponding *equational theory*, written $=_{\mathcal{E}}$. When it is clear from the context we often drop the Σ_{Op} and likewise write $=_E$ for the equational theory $=_{\mathcal{E}}$. Two terms s and t are equal modulo E iff $s =_E t$. For all operations on sets, sequences and multisets we use the subscript E to denote that this is to be considered modulo E. We write $\in_E$ for set membership modulo E for example.

We only consider equational theories that are convergent, i.e., confluent and terminating, when oriented left to right. This implies that every term t has a normal form denoted $t{\downarrow_E}$. Such equational theories are additionally called *subterm-convergent* when the right-hand side is either a ground term or a strict subterm of the left-hand side.

Example 1. To model asymmetric signatures, let Σ_{Op} be the signature consisting of the functions $sign(\cdot,\cdot)$, $checksign(\cdot,\cdot)$ and $pk(\cdot)$ together with the equation $checksign(sign(x,k), pk(k)) = x$. This theory, denoted T_{AS}, is subterm-convergent.

We are also interested in equational theories with the *finite variant property* (FVP) [13] of which subterm-convergent theories are a special case. When a theory has the FVP, then for any term t we can compute a finite set $t_1, \ldots, t_n$ of terms with the following property: for any substitution σ there exist i, θ such that $t\sigma{\downarrow_E} =\ t_i\theta$. This precomputation offers a way to get rid of the equational theory and enables efficient symbolic protocol analysis. TAMARIN uses this approach, which is also why our extension still requires the finite variant property. More precisely, the complete set of variants modulo E (which can be computed via folding variant narrowing [20]) for a term t is denoted $\lceil t \rceil^E$. By abuse of notation we extend this to the variants of all protocol rules (which will be defined in Section 2.2) for a protocol P and denote it $\lceil P \rceil^E$. Next we give an example that has the FVP, but is not subterm-convergent.

Example 2. To model blind signatures we extend T_{AS} from Example 1 with two operators $unblind(\cdot,\cdot)$ and $blind(\cdot,\cdot)$. To represent extracting an actual signature from a blinded signature, we add the equation $unblind(sign(blind(m,r),k),r) = sign(m,k)$, with random r as blinding factor. Then, $\{t,\ sign(y,k)\}$ is a complete set of variants for the term $t = unblind(sign(x,k),r)$. The second variant corresponds to all instances of the term $t[x \mapsto blind(y,r)]$. In this additional equation $sign(m,k)$ is not a subterm of $unblind(sign(blind(m,r),k),r)$, yielding a theory which is not subterm convergent.

2.2 Modeling protocols and adversaries using multiset rewriting rules

We model security protocols using *multiset rewriting rules*. These rules manipulate multisets of *facts*. Facts represent the current state of the system and are built by applying elements of the fact signature Σ_{Fact} to terms. Formally, the set of facts is defined as $\mathcal{F} = \{F(t_1, \ldots, t_n) \mid t_i \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}), F \in \Sigma_{Fact} \text{ of arity n}\}$. We partition $\mathcal{F}$ into linear and persistent facts: during rewriting linear facts can only be consumed once; persistent facts can be consumed arbitrarily often. The set of multisets of facts is denoted by $\mathcal{F}^\sharp$. The set of multisets of ground facts is written $\mathcal{G}^\sharp$. The function $set(\cdot)$ converts a multiset into a set.

The system's state transitions are then given by a set of labeled multiset rewriting rules. Such rules are given as a tuple (id, l, a, r) where id is a unique identifier and l, a, and r are multisets of facts. The resulting rule ri is written: $ri = id : l -[\, a \,]\!\!\rightarrow r$. We say its *name* is $\mathit{name}(ri) = id$, its *premises* are $\mathit{prems}(ri) = l$, its *conclusions* $\mathit{concs}(ri) = r$, and its *actions* $\mathit{acts}(ri) = a$. Given a set of multiset rewriting rules R its ground instances are represented as $\mathit{ginsts}(R)$. We denote by $\mathit{lfacts}(l)$ the multiset of linear facts and by $\mathit{pfacts}(l)$ the set of persistent facts in l.

The semantics of a set of multiset rewriting rules R are given by a *labeled transition relation* $\rightarrow_R \subseteq \mathcal{G}^\sharp \times \mathcal{G}^\sharp \times \mathcal{G}^\sharp$, defined by the following step rule, where S is the current state (a multiset of facts):

$$\frac{ri = id : l -[\, a \,]\!\!\rightarrow r \in_E \mathit{ginsts}(R) \qquad \mathit{lfacts}(l) \subseteq^\sharp S \qquad \mathit{pfacts}(l) \subseteq S}{S \xrightarrow{set(a)}_R ((S \setminus^\sharp \mathit{lfacts}(l)) \cup^\sharp r)}$$

Note that the initial state of a labeled transition system derived from multiset rewriting rules is the empty multiset of facts $\emptyset$. Each transition transforms a multiset of facts (S) into a new multiset of facts, as described by the rewriting rule. Additionally, the actions a of the rule are the label of each transition. These labels are used in our definition of security properties below. We perform multiset rewriting modulo equations E, so we use $\in_E$ for the rule instance modulo. Linear facts are consumed upon rewriting according to the multiplicity of their appearance, so we use multiset inclusion, written $\subseteq^\sharp$, to check that all facts in $\mathit{lfacts}(l)$ occur sufficiently often in S. For persistent facts, we only need to check that each fact in $\mathit{pfacts}(l)$ occurs in S. The successor state is derived by removing all consumed linear facts and adding the generated facts.

There is one distinguished (built-in) rule that generates fresh values, called the *fresh* rule: $\mathit{Fresh} : -[]\!\!\rightarrow \mathsf{Fr}(n)$. Note that the rule has no premise. This fresh rule is the only rule that can have a Fr fact in the conclusion. The argument n represents a fresh value and is unique. We enforce that the values generated by two separate instances of the fresh rule differ. For details see [30].

An *execution* e of a protocol, specified by a set of multiset rewriting rules P, is the alternating sequence of states (i.e., multisets of facts) and rule instances:

$$S_0, (l_1 -[\, a_1 \,]\!\!\rightarrow r_1), S_1, \ldots, S_{n-1}, (l_n -[\, a_n \,]\!\!\rightarrow r_n), S_n$$

such that $S_0 = \emptyset$, and that for all $i \in \{1, \ldots, n\}$ we have $(S_{i-1}, (l_i -[\, a_i \,]\!\!\rightarrow r_i), S_i)$ is a valid step according to the above step rule. The associated trace is the sequence of the

set of the labels: $trace(e) = [set(a_1), \ldots, set(a_n)]$. We denote the set of executions of P as $exec(P)$.

We consider a Dolev-Yao style adversary who has full control over the network and the ability to apply all cryptographic operators. It does so using the message deduction rules *MD* below. All messages sent by participants are put into Out facts and stored in the adversary knowledge K facts, before being sent to participants as In facts. The adversary can create its own random values and knows all public values. It can also apply functions from the signature using the rules in the third line of *MD*.

$$\begin{aligned} \mathit{MD} = \{\ & \mathsf{Out}(x) \multimap[]\to \mathsf{K}(x),\ \mathsf{K}(x) \multimap[\ \mathsf{K}(x)\ ]\to \mathsf{In}(x), \\ & \mathsf{Fr}(x:\mathit{fr}) \multimap[]\to \mathsf{K}(x:\mathit{fr}),\ [] \multimap[]\to \mathsf{K}(x:\mathit{pub})\ \} \\ \cup\ \{\ & \mathsf{K}(x_1), \ldots, \mathsf{K}(x_n) \multimap[]\to \mathsf{K}(f(x_1, \ldots, x_n)) \mid f \in \Sigma_{Op} \text{ with arity } n\ \} \end{aligned}$$

Note that in this message deduction we do not explicitly deal with the equations modeling the properties of cryptographic operators, as all terms are considered modulo the equational theory. Note that as an (efficient) representation of an execution, TAMARIN uses (normal) *dependency graphs* to present and reason about the protocol and adversary deduction rules that have been applied, and their relation to each other. We will explain normal dependency graphs later in more detail.

Example 3. Consider a protocol P_{basic} where agent A sends a nonce m on the network and then receives it, specified using the following rules:

$$P_{basic} = \left\{ \frac{\mathsf{Fr}(m)}{\mathsf{St}(A,m) \quad \mathsf{Out}(m)}[\mathsf{Start}(m)], \frac{\mathsf{St}(A,m) \quad \mathsf{In}(m)}{}[\mathsf{End}(m)] \right\}$$

Figure 1 gives a sample execution of this protocol as a dependency graph. It also illustrates how the dependency graph represents the trace and intermediate states.

2.3 Specifying security properties

We consider both trace and indistinguishability properties. Trace properties like secrecy and agreement are expressed as first-order logic formulas. Formulas introduce variables of an additional sort *temp* for reasoning about the ordering of actions and are evaluated on a trace. The atomic formulas and their informal semantics we consider are

- $\bot$: false;
- $t_1 \approx t_2$: t_1 and t_2 are equal in the equational theory;
- $F@i$: fact $F \in_E tr[i]$ where i is of sort *temp* and $tr[i]$ is the ith element of the trace tr on which we evaluate the formula;
- $i \doteq j$: timepoints i and j are equal;
- $i \lessdot j$: timepoints i occurs before timepoint j.

For a detailed definition of the semantics and the fragment of first order logic that the TAMARIN prover accepts, we refer the reader to [30]. We write $tr \models \varphi$ when φ holds on trace tr and lift the semantics to sets of traces: given a set of traces Tr we write $Tr \models^{\forall} \varphi$ if $tr \models \varphi$ for any $tr \in Tr$ and $Tr \models^{\exists} \varphi$ if $tr \models \varphi$ for some $tr \in Tr$.

We specify unlinkability, anonymity, and more generally equivalence properties by use of *diff*-terms (defining bi-systems, i.e., two systems differing only in some terms) and check their observational equivalence, see [6].

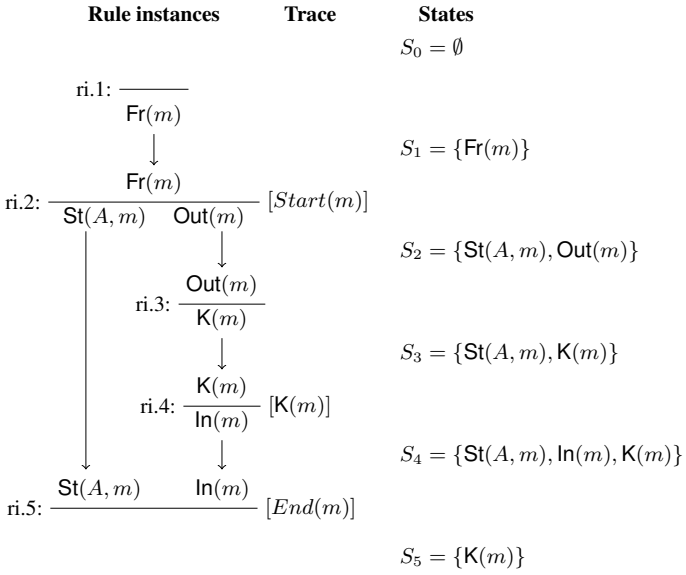

Fig. 1. Example execution of ($P_{basic} \cup MD$).

Example 4 ([6], Ex. 10). An equational theory representing probabilistic encryption is $pdec(penc(m, pk(k), r), k) = m$. This equation gives rise to the *decryption rule* for probabilistic encryption for the adversary which TAMARIN automatically generates:

$$Dpenc : \mathsf{K}(penc(m, pk(k), r)), \mathsf{K}(k) \text{—}[]\!\!\rightarrow \mathsf{K}(m) \,.$$

Consider now the following bi-system:

$$\begin{aligned} S = \{\ & GEN : \mathsf{Fr}(k) \text{—}[]\!\!\rightarrow \mathsf{Key}(k), \mathsf{Out}(pk(k)) \\ & ENC : \mathsf{Key}(k), \mathsf{Fr}(r_1), \mathsf{Fr}(r_2), \mathsf{In}(x) \text{—}[]\!\!\rightarrow \\ & \qquad \mathsf{Out}(\mathit{diff}[r_1, penc(x, pk(k), r_2)]) \quad \}\,. \end{aligned}$$

Here TAMARIN will compare the system where $\mathit{diff}[r_1, penc(x, pk(k), r_2)]$ is replaced by r_1 to the system where it is replaced by $penc(x, pk(k), r_2)$. If the adversary cannot distinguish both systems, they are said to be observationally equivalent. In this example, this means that he cannot distinguish a probabilistic encryption from a random value.

3 Beyond Subterm-Convergent Equational Theories

Example 2 illustrated that subterm-convergent theories are often insufficient to deal with the classical specifications of complex cryptographic operators. In this section we will explain how to extend the TAMARIN prover to work with more than subterm-convergent equational theories. To do that, we need to explain the way that *normal message deduction rules* are computed for the extension. We start by recalling how the TAMARIN prover handled the case of subterm-convergent equational theories before our extension.

3.1 Subterm-convergent Equational Theories

Even for simple subterm-convergent theories containing only the pairing function $\langle\cdot,\cdot\rangle$ and the fst and snd operators, we can see directly that non-normalized dependency graphs are not sufficient to automate the analysis of traces. For example, consider the case where the adversary deduces the first element a of a pair $\langle a, b\rangle$ by applying the function fst$(\cdot)$, then pairs it with an element c, and then deduces a from the new pair to next build the pair $\langle a, d\rangle$ (visualized in the left-most graph of Figure 2 – note that the topmost rule is actually an instance of the function application rule for fst$(\cdot)$ where the conclusion fst$(\langle a, d\rangle)$ reduced to a according to the equational theory). This is a legal dependency graph, but very much redundant, as the steps containing c could have been skipped. As this can be resolved in just one step we are in general interested in *normal* dependency graphs that exclude useless steps. Moreover, this kind of unnecessary derivation could continue indefinitely with arbitrary extra steps in between.

Construction and Deconstruction Rules. To improve efficiency and avoid the aforementioned redundancy, we make the equational theory explicit by dividing the adversary rules into two categories: construction rules and deconstruction rules. Deconstruction rules correspond to equations and are used by the adversary just after protocol rules to deduce messages from what has been sent on the network. Construction rules are, conversely, used to build messages from the knowledge of the adversary that are then sent on the network. To achieve this, we equip adversary knowledge K facts with an orientation, *up* and *down*, denoted $\mathsf{K}^{\uparrow}$ and $\mathsf{K}^{\downarrow}$. Deconstruction rules have premises with both $\mathsf{K}^{\downarrow}$ and $\mathsf{K}^{\uparrow}$ facts (as, e.g., decrypting a ciphertext that was received requires knowing the key) and a conclusion with a $\mathsf{K}^{\downarrow}$ fact. Construction rules, conversely, have premises with only $\mathsf{K}^{\uparrow}$ facts and their conclusion is a $\mathsf{K}^{\uparrow}$ fact as well. To match the purpose of construction and deconstruction rules, the new Out rule has a $\mathsf{K}^{\downarrow}$ fact as conclusion, while the In rule has $\mathsf{K}^{\uparrow}$ facts as premise. The transition from $\mathsf{K}^{\downarrow}$ to $\mathsf{K}^{\uparrow}$ is achieved by a special rule with label "Coerce", see below, but no direct conversion from $\mathsf{K}^{\uparrow}$ to $\mathsf{K}^{\downarrow}$ is possible to prevent loops. This enforces deconstruction rules to be used before construction rules.

In the context of a subterm-convergent theory $\mathcal{ST}$, the idea is to consider a construction rule for every operator in $\Sigma_{\mathcal{ST}}$, and deconstruction rules for each rewriting rule (induced by an ordered equality). The process for deriving deconstruction rules will be explained later. Additionally, we add construction rules for fresh and public name generation.

We give the minimal set of normal deduction rules (included in all subsequent normal deduction rule sets in this work) parametric on the set of operators Σ, including the usual pairing and unpairing operators:

$$ND_{\Sigma} = \left\{ \begin{array}{l} \dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}(x)} \quad \dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)] \quad \mathsf{Coerce}: \dfrac{\mathsf{K}^{\downarrow}(x)}{\mathsf{K}^{\uparrow}(x)} \quad \dfrac{\mathsf{Fr}(x:fr)}{\mathsf{K}^{\uparrow}(x:fr)} \quad \dfrac{}{\mathsf{K}^{\uparrow}(x:pub)} \\ \\ \dfrac{\mathsf{K}^{\downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\downarrow}(x)} \quad \dfrac{\mathsf{K}^{\downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\downarrow}(y)} \quad \dfrac{\mathsf{K}^{\uparrow}(x_1) \;\; \ldots \;\; \mathsf{K}^{\uparrow}(x_k)}{\mathsf{K}^{\uparrow}(f(x_1,\ldots,x_k))} \text{ for all } f \in \Sigma \end{array} \right\}$$

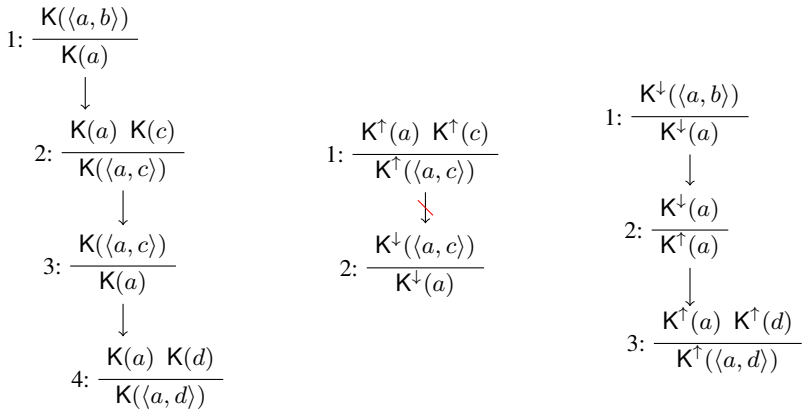

Fig. 2. Message deduction graphs for pairing: the left represents a redundant dependency graph, the middle an impossible deduction with ordered K-facts, and the right shows a shorter deduction with final conclusion equivalent to the left.

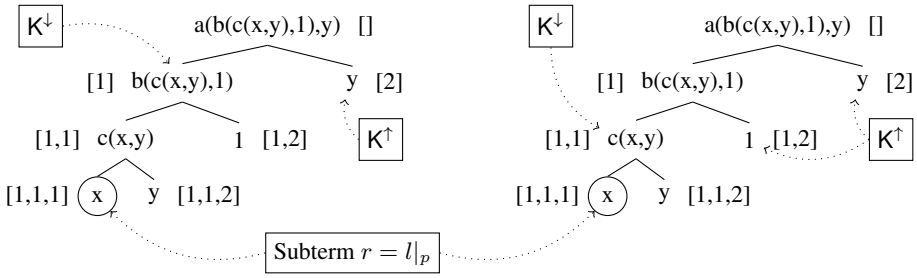

Fig. 3. Different possible positions of K-facts for deconstruction rules associated with $a(b(c(x, y), 1), y) \rightarrow x$.

Example 5. Let us consider the theory for asymmetric encryption called $\mathcal{ASE}$ which we define with the following subterm-convergent theory that includes an operator pk to derive the public key from a private key and equation: $adec(aenc(m, pk(k)), k) = m$.

The resulting set of normal message deduction rules is

$$ND_{\mathcal{ASE}} = \left\{ \frac{\mathsf{K}^{\downarrow}(aenc(m, pk(k))) \quad \mathsf{K}^{\uparrow}(k)}{\mathsf{K}^{\downarrow}(m)} \right\} \cup ND_{\Sigma_{\mathcal{ASE}}}.$$

We see that the deconstruction rule for decryption has $\mathsf{K}^{\uparrow}$ and $\mathsf{K}^{\downarrow}$ facts in its premises.

With such rules, the adversary avoids cases of redundancy as shown in Figure 2. For the full detail of computing the normal deduction rules we refer the reader to [29] but present its high-level motivation here. For a subterm-convergent rewriting system, a method to compute deconstruction rules is the following. Consider a subterm rewriting

rule $l \to r$ where r is not a ground term. Since it is a subterm rewriting rule, there is a position p in l such that $l|_p = r$. Then, for each position $p' \neq []$ strictly above p, we compute a deconstruction rule for which the term $l|_{p'}$ is in a $\mathsf{K}^{\downarrow}$ fact and the terms $l|_{\tilde{p}}$, where $\tilde{p}$ has a sibling equal or above p', are required in a $\mathsf{K}^{\uparrow}$ fact.

Example 6. Consider the rewriting rule $a(b(c(x,y),1),y) \to x$. The only position p of l such that $l|_p = r$ is $[1,1,1]$, so there are two positions strictly above p and different from $[]$, namely $p'_1 = [1,1]$ and $p'_2 = [1]$. For p'_1, we have $\tilde{p}_1 = [2]$ and $\tilde{p}_2 = [1,2]$ as positions which have a sibling above or equal to p'_1. For p'_2, we have only $\tilde{p}_1 = [2]$ as position which has a sibling above or equal to p'_2. We visualize this in Figure 3.

Thus, the two associated deconstruction rules are:

$$[\mathsf{K}^{\downarrow}(c(x,y)),\ \mathsf{K}^{\uparrow}(1),\ \mathsf{K}^{\uparrow}(y)] \dashrightarrow [\mathsf{K}^{\downarrow}(x)] \text{ and } [\mathsf{K}^{\downarrow}(b(c(x,y),1)),\ \mathsf{K}^{\uparrow}(y)] \dashrightarrow [\mathsf{K}^{\downarrow}(x)].$$

Generally, for each position p such that $l|_p = r$, we use the function $ctxtdrules$ extended from the one in [29] to compute the corresponding deconstruction rules, where $cprems(l,p')$ determines the sequence of $\mathsf{K}^{\uparrow}$ premises:

$$\begin{aligned} &ctxtdrules(l,p,r) = \\ &\quad \{[\mathsf{K}^{\downarrow}(l|_{p'})] \cdot cprems(l,p') \dashrightarrow [\mathsf{K}^{\downarrow}(r)] \mid p' \text{ strictly above } p \text{ and } p' \neq []\}, \\ &cprems(l,p') = seq(\{\mathsf{K}^{\uparrow}(l|_{\tilde{p}}) \mid \tilde{p} \neq [] \wedge \tilde{p} \text{ has a sibling above or equal to } p'\}) \end{aligned}$$

where seq converts sets to sequences. Clearly the deconstruction rules from $ND_{\mathcal{ASE}}$ match this construction. We will relax the requirement that $r = l|_p$ for this rule later.

Normal message deduction for non-orientable theories. We combine this with the built-in non-orientable ($\mathcal{NO}$) theory of bilinear pairing ($\mathcal{BP}$), which includes Diffie-Hellman ($\mathcal{DH}$) exponentiation (see [29] for details). We refer by $\mathcal{ACC}$ to the underlying equational axioms of associativity and commutativity for multiplication, bilinear pairing, and multisets as used in $\mathcal{DH}$ and $\mathcal{BP}$. Note that we suppose that the user-defined theory is disjoint from $\mathcal{DH}$, $\mathcal{BP}$, and $\mathcal{ACC}$. We denote by $dgraphs(P)$ the set of all dependency graphs of P. For each dependency graph d we define its trace, called $trace(d)$, as the list of the sets of the actions of the linearization of rule instances in d (see Figure 1). We say that a fact is in a conclusion in a dependency graph if it appears in the conclusion of any rule instance in the dependency graph, similarly for the premises. As proven in [30] as Lemma 4 we have $trace(exec(P)) =_E \{trace(dg) | dg \in dgraphs_E(P \cup \mathit{MD})\}$.

Normal Dependency Graphs. We integrate the concept of normal message deduction with construction and deconstruction rules and dependency graphs. This yields eleven normal form conditions to be enforced on dependency graphs, called **N1**-**N11**, and detailed in the technical report [17]. We use $\mathcal{R}_{\mathcal{BP}}$ to refer to the rules resulting from the built-in bilinear pairing theory.

Definition 1. *A normal dependency graph for a set of protocol rules P is a dependency graph dg such that $dg \in dgraphs(\lceil P \rceil^{\mathcal{R}_{\mathcal{BP}}}_{insts} \cup \mathit{ND})$ and the conditions* **N1**-**N11** *are satisfied. We denote the set of all normal dependency graphs for P with $ndgraphs(P)$.*

Let $\overline{tr}$ denote the subsequence, called observable trace, of all actions in a trace tr that are not equal to $\emptyset$. We have the following proposition which states that executions modulo the equational theory and normal dependency graphs have the same observable traces:

Proposition 1. *[29, Corollary 3.20] For all sets P of protocol rules,*

$$\overline{trace(execs(P \cup \mathit{MD}))}\downarrow_{\mathcal{R}_{\mathcal{BP}}} =_{\mathcal{ACC}} \overline{trace(ndgraphs(P))}.$$

Note that by relying on the observable trace we hide the adversary's deduction steps on both sides, but ensure that security properties (defined on actions) are carried over correctly. This proposition shows that by ordering the K-facts the adversary does not lose any power, and that we can simplify the deduction using the finite variant property.

3.2 Convergent Equational Theories

Now that we have shown that we can use normal dependency graphs for protocols involving a subterm-convergent theory, we will extend this for convergent theories with the FVP. Let $\mathcal{CT}$ be such a theory, and $\mathcal{R}_{\mathcal{CT}}$ the rules $l \to r$ induced by its equations.

Remark 1. For all convergent rules $l \to r$ there are k and $p_1, \ldots, p_k$ such that $r \in \mathcal{T}_{\Sigma_{\mathcal{CT}}}(l|_{p_1}, \ldots, l|_{p_k})$. This is due to the right-hand side not introducing new variables.

As running example, we take the blind signature theory $\mathcal{BS}$ introduced in Example 2, which is used in Chaum's online protocol for e-cash, and in the FOO and Okamoto protocols for e-voting that we will study in our case studies in Section 4.

Example 7. Continuing Example 2 we know that the blind signature permits to sign a blinded message with a secret key and then to unblind the signed blinded message to get the signed message without the blinding. This primitive can be modeled as follows:

$$\Sigma_{\mathcal{BS}} = \left\{ \begin{array}{ccccc} blind(\_,\_), & unblind(\_,\_), & sign(\_,\_), & checksign(\_,\_), \\ fst(\_), & snd(\_), & \langle \_,\_ \rangle, & pk(\_) \end{array} \right\}, and$$

$$\mathcal{R}_{\mathcal{BS}} = \left\{ \begin{array}{c} unblind(blind(m,r),r) \to m,\ checksign(sign(m,k),pk(k)) \to m, \\ unblind(sign(blind(m,r),k),r) \to sign(m,k), \\ fst(\langle x,y \rangle) \to x, \qquad snd(\langle x,y \rangle) \to y \end{array} \right\}.$$

The first rule models that blinding and then unblinding a message with the same key gives back the initial message, similar to symmetric encryption. The second rule extracts and verifies the message under a signature, as the signature is not supposed to hide the message. The third one is not a subterm rule and has been explained previously. The last two rules are the usual ones for projection on pairs.

To be as general as possible, we consider the combination of the existing built-in Diffie-Hellman ($\mathcal{DH}$) and bilinear pairing ($\mathcal{BP}$) theories (note that $\mathcal{DH}$ is included in $\mathcal{BP}$) and allow for disjoint user-defined extensions based on convergent rules. Previously, only subterm-convergent theories could be added to $\mathcal{DH}$ and $\mathcal{BP}$. So we consider $\mathcal{R}_{\mathcal{CT}'} = \mathcal{R}_{\mathcal{CT}} \cup \mathcal{R}_{\mathcal{BP}}$ and the equational theory (where $(\cdot)^{\simeq}$ turns the rule into an equality)

$$\mathcal{CT}' = (\Sigma_{\mathcal{CT}} \cup \Sigma_{\mathcal{BP}}, \mathcal{R}^{\simeq}_{\mathcal{CT}} \cup \mathcal{R}^{\simeq}_{\mathcal{BP}}).$$

We observe that key lemmas for $\mathcal{BP}$, namely [29, Lemma 3.10, Lemma 3.11], still hold for $\mathcal{CT}'$ since the subterm convergence property is not needed in their respective proofs.

The set of message deduction rules *MD* is defined as given in Section 2. To motivate why we derive normal deconstruction rules for convergent equational theories the way we do later, we use the following lemma adapted from [29]. It will also be helpful in the proof of our main theorem later. The lemma describes that the adversary can always convert a $\mathsf{K}^{\downarrow}$ fact into a $\mathsf{K}^{\uparrow}$ fact using the coerce rule. We call a *deduction extension* a dependency graph that has same the trace, state facts, and fresh values as the initial dependency graph, but can include additional intruder deduction rule instances (see the technical report [17] for details).

Lemma 1. *[29, Lemma A.15] For all $ndg \in ndgraphs(P)$ and conclusion facts $\mathsf{K}^{\downarrow}(m)$, there is a deduction extension ndg' with a conclusion fact $\mathsf{K}^{\uparrow}(m')$ with $m =_{ACC} m'$.*

We now define common subterms for use in adversary deduction rule derivation.

Definition 2. *A* common subterm *t of a rewriting rule $l \to r$ is a term such that there are p and q such that $t = l|_p = r|_q$.*

A common maximal subterm *t of a rewriting rule $l \to r$ is a common subterm of $l \to r$ such that there is no common subterm $t' \neq t$ such that t is a subterm of t'.*

For a given rewriting rule $l \to r$ where $vars(r) \neq \emptyset$, and for which there is a common maximal subterm $l|_p$, we use the function $ctxtdrules$ to compute the corresponding deconstruction rules. The set of deconstruction rules is given by:

$$Ctxtdrules(l, r) = \bigcup_{p \in P(l,r)} ctxtdrules(l, p, r)$$

where $P(l, r) = \{p \mid \exists q, l|_p = r|_q$, *and* $l|_p = r|_q$ *is a maximal common subterm*$\}$. The set $DR_{\mathcal{CT}}$ of deconstruction rules for $\mathcal{CT}$ is:

$$DR_{\mathcal{CT}} = \bigcup_{(l,r) \in \mathcal{R}_{\mathcal{CT}}} Ctxtdrules(l, r)$$

Thus, we get the set of normal deduction rules $ND_{\mathcal{CT}} = ND_{\Sigma_{\mathcal{CT}}} \cup DR_{\mathcal{CT}}$.

Example 8. We apply this to the blind signature rewriting rule

$$unblind(sign(blind(m, r), k), r) \to sign(m, k).$$

We have m and k as common maximal subterms on respective positions $[1, 1, 1]$ and $[1, 2]$. Then we consider the following deconstruction rules:

$$ctxtdrules(l, [1, 1, 1], r) = \left\{ \frac{\mathsf{K}^{\downarrow}(blind(m, r)) \;\; \mathsf{K}^{\uparrow}(k) \;\; \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(sign(m, k))}, \frac{\mathsf{K}^{\downarrow}(sign(blind(m, r), k)) \;\; \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(sign(m, k))} \right\},$$

$$ctxtdrules(l, [1, 2], r) = \left\{ \frac{\mathsf{K}^{\downarrow}(sign(blind(m, r), k)) \;\; \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(sign(m, k))} \right\}.$$

As two of the three deconstruction rules are identical, we thus get two rules, and $Ctxtdrules(l, \{[1,1,1],[1,2]\}, r) = ctxtdrules(l, [1,1,1], r)$. We show the set $ND_{\mathcal{BS}}$ of normal deduction message rules for $\mathcal{BS}$, which contains $ND_{\Sigma_{\mathcal{BS}}}$ and these rules:

$$\left\{ \begin{array}{cc} \dfrac{\mathsf{K}^{\downarrow}(blind(m,r)) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(m)} & , \quad \dfrac{\mathsf{K}^{\downarrow}(sign(m,k)) \quad \mathsf{K}^{\uparrow}(pk(k))}{\mathsf{K}^{\downarrow}(m)} \\ \dfrac{\mathsf{K}^{\downarrow}(blind(m,r)) \;\; \mathsf{K}^{\uparrow}(k) \;\; \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(sign(m,k))} & , \quad \dfrac{\mathsf{K}^{\downarrow}(sign(blind(m,r),k)) \;\; \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(sign(m,k))} \end{array} \right\}$$

For an extended example, see the technical report [17].

3.3 Further restrictions – normal form conditions

The need for additional normal-form conditions will become apparent with the following example using the equational theory for *trapdoor commitments*, needed for instance in Okamoto's voting protocol [26]. Trapdoor commitments are commitments that can be opened to return a different value than the one initially committed, using a special trapdoor. This is used to create fake receipts (see Section 4.3). To model the algebraic properties of trapdoor commitments, we use the equational presentation $\mathcal{BSTDC}_0 = (\Sigma_{\mathcal{BSTDC}}, \mathcal{R}^{\cong}_{\mathcal{BSTDC}_0})$ where

$$\Sigma_{\mathcal{BSTDC}} = \Sigma_{\mathcal{BS}} \cup \{tdcommit(\_,\_,\_), open(\_,\_), f(\_,\_,\_,\_)\}$$

and the rules are

$$\mathcal{R}_{\mathcal{BSTDC}_0} = \mathcal{R}_{\mathcal{BS}} \cup \left\{ \begin{array}{c} open(tdcommit(m,r,td),r) \to m, \\ tdcommit(m_2, f(m_1,r,td,m_2), td) \to tdcommit(m_1,r,td) \end{array} \right\}.$$

Note that the second equation is not subterm convergent as $tdcommit(m_1, r, td)$ is not a subterm of $tdcommit(m_2, f(m_1, r, td, m_2), td)$. Equations in $\mathcal{R}^{\cong}_{\mathcal{BSTDC}_0}$ model that the voter is able to replace m_2 by m_1 in his commitment, which is crucial to achieve the receipt-freeness property. Simply orienting the equations in $\mathcal{R}^{\cong}_{\mathcal{BSTDC}_0}$ yields a non confluent rewrite system though. Instead, we extend it to obtain a convergent system:

$$\mathcal{R}_{\mathcal{BSTDC}} = \mathcal{R}_{\mathcal{BSTDC}_0} \cup \left\{ \begin{array}{c} open(tdcommit(m_1,r,td), f(m_1,r,td,m_2)) \to m_2, \\ f(m_1, f(m,r,td,m_1), td, m_2) \to f(m,r,td,m_2) \end{array} \right\}.$$

Again, the last equation is not subterm convergent. We then compute the normal deconstruction rules as specified before. One of the resulting normal deconstruction rules is as follows and essentially shows that when one knows the previous content m_1 and the trapdoor td, one can replace the content by m_2:

$$\frac{\mathsf{K}^{\downarrow}(f(m,r,td,m_1)) \;\; \mathsf{K}^{\uparrow}(m_1) \;\; \mathsf{K}^{\uparrow}(td) \;\; \mathsf{K}^{\uparrow}(m_2)}{\mathsf{K}^{\downarrow}(f(m,r,td,m_2))}$$

We see that applying this rule naively again and again can lead to an infinite loop, the start of which is shown in Figure 4. Even though nothing changes except for the adversary-injected last argument, this leads to a looping behavior which we address next. The problem is that the conclusion $\mathsf{K}^{\downarrow}$ term unifies with the premise $\mathsf{K}^{\downarrow}$ term.

$$1: \frac{\mathsf{K}^{\downarrow}(f(m,r,td,m_1))\ \ \mathsf{K}^{\uparrow}(m_1)\ \ \mathsf{K}^{\uparrow}(td)\ \ \mathsf{K}^{\uparrow}(m_2)}{\mathsf{K}^{\downarrow}(f(m,r,td,m_2))}$$

$$\downarrow$$

$$2: \frac{\mathsf{K}^{\downarrow}(f(m,r,td,m_2))\ \ \mathsf{K}^{\uparrow}(m_2)\ \ \mathsf{K}^{\uparrow}(td)\ \ \mathsf{K}^{\uparrow}(m_3)}{\mathsf{K}^{\downarrow}(f(m,r,td,m_3))}$$

$$\downarrow$$

$$3: \frac{\mathsf{K}^{\downarrow}(f(m,r,td,m_3))\ \ \mathsf{K}^{\uparrow}(m_3)\ \ \mathsf{K}^{\uparrow}(td)\ \ \mathsf{K}^{\uparrow}(m_4)}{\mathsf{K}^{\downarrow}(f(m,r,td,m_4))}$$

Fig. 4. Loop using f.

Normal Form Conditions to Prevent Loops As we have seen, convergent equational theories give rise to a special case where we need to add a new normal form condition to help termination. For an equation $l = r$, the right-hand side r of the equation may be unifiable with a strict subterm $l|_p$, $p \neq []$ of the left-hand side. This can also occur in the subterm-convergent case, but there we have equality of $l|_p = r$, and an existing normal-form condition forbidding to derive the same adversary knowledge more than once (**N3**, see the technical report [17]) effectively prevents this problem.

In terms of adversary deduction (i.e., deconstruction rules) the above example of the trapdoor commitment shows that the right-hand $\mathsf{K}^{\downarrow}$ term is unifiable with the left-hand $\mathsf{K}^{\downarrow}$ term. This then leads to the infinite chain illustrated in Figure 4. The normal form condition to not derive the same term repeatedly does not apply, as the adversary adds in a different value each time. For the convergent theory case where such unification is possible the resulting derivation rule can thus be repeatedly applied as the derived knowledge does indeed change each time because $l|_p \neq r$. As one can see in the example, one does not actually need to apply the rule repeatedly to its intermediate results, but can rather apply it to the original term with different premises to get the same final result in one step. Thus we will now explain and prove that no chain (beyond a certain length) of applications of this rule are needed in general.

As the given convergent equational theory is by definition required to be terminating, there is a limit n for how often one needs to apply this rule in general. A conservative bound for n is the number of subterms of $l|_p$. Intuitively, with each application, some part of the original content of the term must be removed (due to termination), and if this has been done n times, no original subterm (of the initial term before applying this rule the first time) remains, and all the subterms are known to the adversary as $\mathsf{K}^{\uparrow}$ terms. Thus, instead of using this deconstruction rule, the adversary can simply use the construction rule for the root symbol and apply it to all the known subterms in the result of the deconstruction rule chain.

Example 9. Let us show with a simple example that this bound is really needed. For the equational theory with two function symbols h/2 and f/3 and the single equation:

$$h(f(x_1, x_2, x_3), z) = f(x_2, x_3, z)$$

we get one deconstruction rule:

$$\frac{\mathsf{K}^{\downarrow}(f(x_1, x_2, x_3)) \quad \mathsf{K}^{\uparrow}(z)}{\mathsf{K}^{\downarrow}(f(x_2, x_3, z))}$$

For this rule the conclusion $\mathsf{K}^{\downarrow}$-term obviously unifies with that in the premises. Now if the adversary receives $f(a, b, c)$ intuitively it should be possible to derive $f(c, x, y)$, for some x, y of the adversary's choosing, but using just one application of the deconstruction rule this is not possible. If we permit two applications on the other hand, it can be derived as expected.

Note that in the previous example, we can give f an arbitrary number of arguments and the form of the deconstruction rule will stay the same, so we need to permit the use of the deconstruction rule up to $n-1$ times, for n the number of strict subterms of the $\mathsf{K}^{\downarrow}$-term of the premises. Note that this number is of course fixed by the input equational theory and can thus be easily computed.[2]

This leads us to define a new normal form condition:

Definition 3. N12. *There is no chain of nodes repeatedly instantiating a rule of the form* $\mathsf{K}^{\downarrow}(l|_p), \mathsf{K}^{\uparrow}(t_1), \ldots, \mathsf{K}^{\uparrow}(t_i) \multimap \mathsf{K}^{\downarrow}(r)$ *of length at least equal to the number of subterms of* $l|_p$*, if* $l|_p$ *and* r *are unifiable.*

This limits the length of chains of derivation with such rules as motivated above. Do note that for the case of equality, i.e., $r = l|_p$, this does not add a restriction as there the condition "to not derive the same term more than once" is already in effect.

Note that in general we cannot guarantee termination for the intruder deduction as even for the class of optimally reducing convergent rewrite systems (which have the finite variant property) the deducibility problem is undecidable [2].

We next present the key theorem that states that the traces of dependency graphs modulo the equational theory and normal dependency graphs do actually coincide. This is an extension of the version for subterm-convergent theories [29, Lemma 3.19] to the convergent case:

Theorem 1. *For all sets* P *of protocol rules,*

$$\{\overline{trace(dg)} \mid dg \in dgraphs(\lceil P \cup \mathit{MD} \rceil^{\mathcal{C}\mathcal{T}'}_{insts}) \wedge dg \downarrow_{\mathcal{C}\mathcal{T}'} \textit{-normal}\} = \overline{trace(ndgraphs(P))}.$$

We give the full proof in the technical report [17], and present a short sketch highlighting the key points here.

Proof (Sketch). We need to show that the traces of the normal and non-normal dependency graphs coincide. As protocol rules can be used for dependency graphs and normal dependency graphs, the interesting part is the message deduction. Moreover, send and receive rules are available in both, so we have to analyze the construction and deconstruction rules.

[2] For *private* function symbols the deconstruction rule must be usable up to n times, as there is no corresponding construction rule.

For construction rules, there is always a normal version available due to Lemma 1 which allows us to obtain all knowledge in $\mathsf{K}^{\uparrow}$ format. The remaining case is the one where the output of the rule requires use of the equational theory, and here we focus on the deconstruction rules for convergent equations as all other rules are covered by the old proof from [29]. Here, generalizing the old proof, we can rely on a lemma stating that for any unknown subterm there is a position above, such that the subterm at that position appears as a $\mathsf{K}^{\downarrow}$-fact, allowing us to apply our new deconstruction rules.

For the new restriction **N12** the interesting case is when a derivation is possible in the regular dependency graph by using the deconstruction rule n times (n being the number of strict subterms), which is forbidden in the normal dependency graph. Our key observation is that the result of n derivations with such a deconstruction rule can be created by applying a construction rule for the operator as all subterms are known in $\mathsf{K}^{\uparrow}$ by the deconstruction rule structure.

4 Case Studies

The new version of TAMARIN together with the code used for the case studies is available on github [31, case studies in `examples/post17/`].

4.1 Chaum's Online e-Cash Protocol

Chaum's Online e-cash protocol allows a client to withdraw a coin blindly from the bank, and then spend it later in a payment without being traced even by the bank. The protocol is "on-line" in the sense that the seller does not accept the payment before contacting the bank to verify that the coin has not been deposited before, to prevent double spending [11].

We have three roles, the client C, the bank B and the seller S. In a first phase, the withdrawal phase, the client C blinds a coin x and sends it to the bank B. The bank deducts the money from the client's account, signs blindly the coin and sends the signature to the client. Then, in a second phase, the client unblinds the signature, and sends the coin x and the signature of x to the seller S who checks if the signature is correct. Then it sends the coin to the bank, which responds on a private channel with payment approval if the coin had not been deposited. Then the seller accepts the coin.

$$
\begin{aligned}
C &\longrightarrow B : blind(x, r) \\
B &\longrightarrow C : sign(blind(x, r), skB) \\
C &\longrightarrow S : \langle x, sign(x, skB) \rangle \\
S &\longrightarrow B : \langle x, sign(x, skB) \rangle \\
B &\xrightarrow[priv]{} S : x
\end{aligned}
$$

We use the equational theory for blind signatures from Example 2.

Unforgeability Unforgeability ensures that, in an e-cash protocol, a client is unable to create a coin without involving the bank, resulting in a fake coin, or to spend a valid

coin he withdrew from the bank twice [18]. We express unforgeability as follows:

$$\forall j,x.Spend(x)@j \Rightarrow (\exists i.\, Withdraw(x)@i \wedge i \lessdot j \wedge \neg(\exists l.Spend(x)@l \wedge l \neq j))$$

When verifying the protocol TAMARIN returns an attack that allows the client to withdraw multiple coins if the bank does not verify the correct format of the coin. This works as follows: the client submits $blind(blind(x, r_1), r_2)$ to the bank, which signs it. The client obtains a first valid coin $sign(blind(x, r_1), skB)$ by unblinding once, and a second coin $sign(x, skB)$ by unblinding again. He can spend both of them, although he should only have one valid coin. This attack can be prevented by the bank verifying the correct format of the coin before signing it. A similar problem arises when the seller receives a coin. After correcting both issues, TAMARIN manages to prove unforgeability, which was previously not possible in PROVERIF [18] due to problems in modeling the state of the bank, which needs to keep track of all previously spent coins.

Anonymity & Untraceability Anonymity and untraceability (called Weak and Strong Anonymity in [18]) are defined as observational equivalence properties. To define anonymity, we consider two clients C_1 and C_2 and the case where both of them withdraw a coin from the same bank, but only one of them makes a purchase. Anonymity is the property guaranteeing that neither the bank nor the seller are able to distinguish the case where C_1 makes the purchase from the case where it is C_2 who makes it.

For untraceability, we also consider two clients C_1 and C_2 and the case where both of them withdraw two coins and both spend the first coin, but only one of them makes a second purchase. Untraceability guarantees that neither the bank nor the seller are able to know whether C_1 or C_2 makes the second purchase.

To ensure anonymity, we have to add a synchronization point to synchronize both clients after the coin withdrawal, as the adversary can otherwise trace one of them. In that case, TAMARIN can prove both anonymity and untraceability.

4.2 The FOO Voting Protocol

The FOO (for Fujioka, Okamoto and Otha) voting protocol [21] allows a voter to publish a vote signed by the administration without being identified, even by the administrator. The protocol is designed to ensure that each published vote has been signed by the administrator guaranteeing eligibility, and at the same time ensuring anonymity of the voter even with respect to the administrator.

We consider three roles, the voter V, the administrator A, and the collector C. The protocol is split into three phases.

- In the first phase the administrator signs the voter's commitment to his vote: voter V chooses his vote v and computes a commitment $x = commit(v, r)$ for a random key r. He blinds the commitment using a random value b and obtains $e = blind(x, b)$. Then he signs e and sends the signature $sb_V = sign(e, ltkV)$ together with e and his identity to the administrator. The administrator checks if V has the right to vote and has not yet voted, and if the signature sb_V is correct. If all tests succeed, he signs $sb_A = sign(e, ltkA)$ and sends it back to V. V checks the signature, and unblinds it to obtain $s_A = unblind(sb_A, b) = sign(x, ltkA)$.

- In the second phase, the voter submits his ballot: voter V sends (x, s_A) to the collector C through an anonymous channel. The collector checks the administrator's signature and enters (x, s_A) as the l-th entry into a list.
- When all ballots are cast the counting phase begins: the collector publishes the list of correct ballots. V verifies that his commitment appears on the list and sends (l, r) to C using an anonymous channel. The collector C opens the l-th ballot using r and publishes the vote.

To model commitments, we use the equational theory $\mathcal{BSC} = (\Sigma_{\mathcal{BSC}}, \mathcal{R}^{\approx}_{\mathcal{BSC}})$ where $\Sigma_{\mathcal{BSC}} = \Sigma_{\mathcal{BS}} \cup \{commit(\_,\_), open(\_,\_)\}$ and

$$\mathcal{R}_{\mathcal{BSC}} = \mathcal{R}_{\mathcal{BS}} \cup \{open(commit(m, r), r) \rightarrow m\}.$$

Eligibility Eligibility ensures that, if a vote is published by the collector, then its commitment has been signed by the administration, denoted by the $Registered$ action. This is expressed as follows, and automatically verified by TAMARIN:

$$\begin{aligned}&\forall v, j.\, VotePublished(v)@j \Rightarrow \\ &\quad (\exists b, r, i.\, Registered(blind(commit(v, r), b))@i \wedge i \lessdot j)\end{aligned}$$

Vote privacy Following [16], to define vote privacy, we consider two voters V_1 and V_2 and the case where both of them commit a different vote, for example yes and no. Vote privacy is the property guaranteeing that neither the administrator nor the collector can distinguish the case where V_1 votes for yes from the case where he votes for no (and V_2 votes no or yes, so that there is one vote for yes and one for no in both cases) [16]. Again, we need to add synchronization to prevent trivial attacks, but then TAMARIN verifies observational equivalence for FOO.

4.3 The Okamoto Protocol

The Okamoto protocol [26] is similar to the FOO protocol, but it uses trapdoor commitments and it involves a *timeliness member* (i.e., a trusted third party) to achieve Receipt-Freeness. Receipt-Freeness means that a voter cannot construct a receipt proving to somebody else that he voted for a certain candidate, in order to prevent vote-buying.

The protocol works a follows. The first phase, during which the voter obtains a signature on his commitment x, is the same as for the FOO protocol, except that x is a trapdoor commitment.

- In the second phase the vote is submitted; the voter V sends the signed trapdoor commitment to the collector through an anonymous channel. The collector checks the administrator's signature and enters (x, s_A) into a list. The voter sends (v, r, x) to the timeliness member T through a secure anonymous channel.
- When all ballots are cast the counting phase begins: the collector publishes the list of correct ballots. V verifies that his commitment appears on the list. The timeliness member publishes the randomly shuffled list of votes.

To model the algebraic properties of trapdoor commitments, we use again the signature $\mathcal{R}_{\mathcal{BSTDC}}$ defined in Section 3.3. We can show eligibility using the same property as for FOO, and TAMARIN succeeds in proving the property. We can also show vote privacy using the same approach as for FOO.

Receipt-freeness Following [16], to model receipt-freeness, we compare a case where a voter V_1 votes yes and honestly sends all his secret values (the blinding factor, the trapdoor, his secret keys, and so on) as a receipt, to the case where he votes no and sends fake values instead. If an adversary cannot distinguish both cases, then the voter cannot produce a meaningful receipt.

In case of the Okamoto protocol, the trapdoor allows the voter to open his commit differently to fool the adversary. In the first case, he reveals his vote yes, his blinding factor r, the trapdoor td and his secret signing key $ltkV$ (used in his first message to the administrator). In the second case, he still reveals yes (although he voted no), a newly generated blinding factor $f(no, r, td, yes)$ (instead of r), the trapdoor td and his secret signing key $ltkV$. In both cases, we have that

$$\begin{aligned} open(tdcommit(yes, r, td), r, td) &= yes \\ &= open(tdcommit(no, r, td), f(no, r, td, yes), td) \end{aligned}$$

thus to the adversary it looks like the voter voted yes in both cases.

With our extension and the new normal form condition, TAMARIN proves that both cases are observationally equivalent, showing that the Okamoto protocol guarantees receipt-freeness.

4.4 Prefix Property: Denning-Sacco and Needham-Schroeder Protocols

The prefix property models the fact that in certain cryptographic schemes (like CBC) one can extract from encrypted messages their encrypted prefix: given the ciphertext $enc(\langle x, y\rangle, k)$, one can deduce its prefix $enc(x, k)$. For more details see [14].

Using this property, a confusion attack exists for the Denning-Sacco symmetric key protocol with CBC and the key secrecy is violated for the Needham-Schroeder symmetric key protocol with CBC. These are known attacks, but they can now be automatically exhibited with TAMARIN. As the equational theory for prefix extraction (see Equation (1)) is not subterm-convergent, these protocols could not have been analyzed without our new extension.

The equational theory under consideration is that of symmetric encryption (enc) and decryption (dec), permitting one to decrypt an encrypted message with the right key: $dec(enc(m, k), k) = m$. We add an additional operator $prefix$ to the signature which allows one to extract the first part of an encrypted message as encrypted ciphertext under the same key:

$$prefix(enc(\langle x, y\rangle, k)) = enc(x, k) \tag{1}$$

We use this theory to model and analyze the Denning-Sacco and Needham-Schroeder protocols. The results are reported in the table below and the details for both are available in the technical report [17].

Protocol	Property	Result	Time	Proof Steps
Chaum	Unforgeability	Verified	0.2s	10
Chaum	Anonymity	Verified	7.6s	673
Chaum	Untraceability	Verified	1m13.7s	2769
FOO	Eligibility	Verified	10.3s	9
FOO	Vote Privacy	Verified	4m11.1s	6946
Okamoto	Eligibility	Verified	8.4s	5
Okamoto	Vote Privacy	Verified	1m20.3s	3332
Okamoto	Receipt-Freeness	Verified	13m35.8s	19691
Denning-Sacco	Session matching	Attack	0.3s	4
Needham-Schroeder	Key secrecy	Attack	24.0s	8

Table 1. Summary of case study results. Timings are done on a standard dual-core laptop (requiring less than 8GB RAM) and include precomputations.

4.5 Summary of Case Studies

Altogether, the set of case studies presented shows that the expansion of admissible equational theories for TAMARIN prover is quite general and useful for many, very different protocols. Table 1 presents our verification results.

5 Conclusion

In this paper, we significantly extend the scope of the protocols that can be handled by the TAMARIN prover: we allow users to specify arbitrary convergent equational theories that have the finite variant property. This extension strictly generalizes the original theory underlying the TAMARIN prover which is restricted to subterm convergent theories. From a more technical side, we generalize the theory for dealing with message deduction, introduce a new normal form condition on dependency graphs to avoid non-termination issues and prove the completeness of the generalized normal message deduction rules and additional normal form condition. All our results have been implemented in the TAMARIN prover and their effective applicability is demonstrated on several, quite different case studies: Chaum's digital cash protocol, the FOO and Okamoto e-voting protocols, and consideration of a prefix property for encryption in two classical authentication protocols.

An interesting line for future work is to add more support for equational theories that have associative-commutative operators, such as the built-in theory for Diffie-Hellman and bilinear pairings. Including support for exclusive or (xor) seems particularly challenging. Backward reasoning on the message deduction for xor leads easily to non-termination. We however believe that our new normal form condition may serve as a promising starting point for this extension.

Acknowledgments. This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC), and by the CNRS project PEPS JCJC VESPA.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: Proceedings of the 17th USENIX Security Symposium. pp. 335–348. USENIX Association (2008)
2. Anantharaman, S., Narendran, P., Rusinowitch, M.: Intruders with caps. In: Proceedings of the 18th International Conference on Term Rewriting and Applications. pp. 20–35. RTA'07, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1779782.1779786
3. Arapinis, M., Ritter, E., Ryan, M.: StatVerif: Verification of stateful processes. In: Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11). pp. 33–47. IEEE Press (2011)
4. Armando, A., Basin, D.A., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA tool for the automated validation of internet security protocols and applications. In: CAV. pp. 281–285 (2005)
5. Basin, D., Cremers, C.: Know your enemy: Compromising adversaries in protocol analysis. ACM Trans. Inf. Syst. Secur. 17(2), 7:1–7:31 (Nov 2014), http://doi.acm.org/10.1145/2658996
6. Basin, D., Dreier, J., Sasse, R.: Automated symbolic proofs of observational equivalence. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM (2015)
7. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming 75(1), 3–51 (Feb–Mar 2008)
8. Blanchet, B., Smyth, B.: Automated reasoning for equivalences in the applied pi calculus with barriers. In: Proc. 29th Computer Security Foundations Symposium (CSF'16). pp. 310–324. IEEE Computer Society (2016)
9. Blanchet, B., Smyth, B., Cheval, V.: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial (2016)
10. Chadha, R., Cheval, V., Ciobâcă, Ş., Kremer, S.: Automated verification of equivalence properties of cryptographic protocol. ACM Transactions on Computational Logic (2016), https://hal.inria.fr/hal-01306561/document, to appear
11. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of CRYPTO '82. pp. 199–203. Plenum Press (1982)
12. Cheval, V., Comon-Lundh, H., Delaune, S.: Trace equivalence decision: Negative tests and non-determinism. In: 18th Conference on Computer and Communications Security (CCS'11). ACM, Chicago, Illinois, USA (Oct 2011)
13. Comon-Lundh, H., Delaune, S.: The finite variant property: How to get rid of some algebraic properties. In: Giesl, J. (ed.) Term Rewriting and Applications, 16th International Conference, RTA. LNCS, vol. 3467, pp. 294–307. Springer (2005)
14. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
15. Cremers, C.J.F.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: 20th International Conference on Computer Aided Verification (CAV'08). LNCS, vol. 5123, pp. 414–418. Springer (2008)
16. Delaune, S., Kremer, S., Ryan, M.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17, 435–487 (December 2009)
17. Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols. Tech. rep., HAL (2017), https://hal.inria.fr/hal-01430490/

18. Dreier, J., Kassem, A., Lafourcade, P.: Formal analysis of e-cash protocols. In: SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography. pp. 65–75. SciTePress (2015)
19. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: Foundations of Security Analysis and Design V. LNCS, vol. 5705, pp. 1–50. Springer (2009)
20. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. Journal of Logic and Algebraic Programming 81(7-8), 898–928 (2012)
21. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: International Workshop on the Theory and Application of Cryptographic Techniques. Springer (1992)
22. Guttman, J.D., Ramsdell, J.D.: CPSA: a cryptographic protocol shapes analyzer (2009), http://hackage.haskell.org/package/cpsa
23. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: ACM workshop on Privacy in the electronic society (WPES'05). pp. 61–70. ACM (2005)
24. Kremer, S., Künnemann, R.: Automated analysis of security protocols with global state. Journal of Computer Security (2016), https://hal.inria.fr/hal-01351388, to appear
25. Meier, S., Schmidt, B., Cremers, C.J.F., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: CAV. LNCS, vol. 8044, pp. 696–701. Springer (2013)
26. Okamoto, T.: An electronic voting scheme. In: IFIP World Conference on IT Tools. pp. 21–30 (1996)
27. Ramsdell, J.D., Dougherty, D.J., Guttman, J.D., Rowe, P.D.: A hybrid analysis for security protocols with state. In: Proc. 11th International Conference on Integrated Formal Methods (IFM'14). LNCS, vol. 8739, pp. 272–287. Springer (2014)
28. Santiago, S., Escobar, S., Meadows, C., Meseguer, J.: A formal definition of protocol indistinguishability and its verification using Maude-NPA. In: Security and Trust Management (STM) 2014. pp. 162–177. Springer (2014)
29. Schmidt, B.: Formal Analysis of Key Exchange Protocols and Physical Protocols. PhD dissertation, ETH Zurich (2012)
30. Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Computer Security Foundations Symposium (CSF). pp. 78–94. IEEE (2012)
31. Tamarin website, https://tamarin-prover.github.io/