Unification modulo a 2-sorted Equational theory for Cipher-Decipher Block Chaining

Siva Anantharaman, Christopher Bouchard, Paliath Narendran, Michaël Rusinowitch

Logical Methods in Computer Science
Vol. 10(1:5)2014, pp. 1–26
www.lmcs-online.org

Submitted Aug. 7, 2013
Published Feb. 6, 2014

UNIFICATION MODULO A 2-SORTED EQUATIONAL THEORY FOR CIPHER-DECIPHER BLOCK CHAINING

SIVA ANANTHARAMAN [a], CHRISTOPHER BOUCHARD [b], PALIATH NARENDRAN [c], AND MICHAËL RUSINOWITCH [d]

[a] LIFO, Université d'Orléans (France)
e-mail address: siva@univ-orleans.fr

[b,c] University at Albany–SUNY (USA)
e-mail address: {cb829983,dran}@albany.edu

[d] Loria-INRIA Grand Est, Nancy (France)
e-mail address: rusi@loria.fr

Abstract. We investigate unification problems related to the Cipher Block Chaining (CBC) mode of encryption. We first model chaining in terms of a simple, convergent, rewrite system over a signature with two disjoint sorts: *list* and *element*. By interpreting a particular symbol of this signature suitably, the rewrite system can model several practical situations of interest. An inference procedure is presented for deciding the unification problem modulo this rewrite system. The procedure is modular in the following sense: any given problem is handled by a system of 'list-inferences', and the set of equations thus derived between the element-terms of the problem is then handed over to any ('black-box') procedure which is complete for solving these element-equations. An example of application of this unification procedure is given, as attack detection on a Needham-Schroeder like protocol, employing the CBC encryption mode based on the associative-commutative (AC) operator XOR. The 2-sorted convergent rewrite system is then extended into one that fully captures a block chaining encryption-decryption mode at an abstract level, using no AC-symbols; and unification modulo this extended system is also shown to be decidable.

1. Introduction

The technique of *chaining* is applicable in many situations. A simple case is e.g., when we want to calculate the partial sums (resp. products) of a (not necessarily bounded) list of integers, with a given 'base' integer; such a list of partial sums (resp. products) can be calculated, incrementally, with the help of the following two equations:

$$bc(nil,\, z) = nil, \qquad bc(cons(x, Y),\, z) = cons(h(x, z),\, bc(Y,\, h(x, z)))$$

2012 ACM CCS: [**Theory of computation**]: Logic—Logic and verification / Equational logic and rewriting.

Key words and phrases: Equational unification, Block chaining, Protocol.

[b,c] Research supported in part by NSF grant CNS-0905286.

[d] Research supported in part by FP7 NESSOS Project.

DOI:10.2168/LMCS-10(1:5)2014

where *nil* is the empty list, z is the given base integer, x is an integer variable, and Y is the given list of integers. The partial sums (resp. products) are returned as a list, by evaluating the function bc, when $h(x, z)$ is interpreted as the sum (resp. product) of x with the given base integer z.

A more sophisticated example is the Cipher Block Chaining encryption mode (CBC, in short), employed in cryptography, a mode which uses the AC-operator exclusive-or (XOR) for 'chaining the ciphers across the message blocks'; here is how this is done: Let $\oplus$ stand for XOR (which we let distribute over block concatenation), and let $M = p_1 \dots p_n$ be a message given as a list of n 'plaintext' message subblocks. Then the encryption of M, with any given public key k and an initialization vector v, is defined as the list $c_1 \dots c_n$ of ciphertext message subblocks, where: $c_1 = e_k(p_1 \oplus v)$, and $c_i = e_k(p_i \oplus c_{i-1})$, for any $1 < i \leq n$. (Note: It is usual in Cryptography to see a message as a sequence of "records", each record being decomposed into a sequence of blocks of the same size; what we refer to as 'message' in this paper, would then correspond to a 'record' in the sense of cryptography.) The above set of equations also models this CBC encryption mode: for this, we interpret the function $h(x, y)$ as the encryption $e_k(x \oplus y)$ of any single block message x, XOR-ed with the initialization vector y, using the given public key k. Under such a vision, a message M is decomposed as the concatenation of its first message block m with the rest of the message list M', i.e., we write $M = m \cdot M'$; then, the encryption of M with any given public key k, with x taken as initialization vector (IV), is derived by $bc(M, x) = h(m, x) \cdot bc(M', h(m, x))$.

Actually, our interest in the equational theory defined by the above two equations was motivated by the possibility of such a modeling for Cipher Block Chaining, and the fact that rewrite as well as unification techniques are often employable, with success, for the formal analysis of cryptographic protocols (cf. e.g., [1, 3, 7, 8, 9], and also the concluding section).

This paper is organized as follows. In Section 2 we introduce our notation and the basic notions used in the sequel; we shall observe, in particular, that the two equations above can be turned into rewrite rules and form a convergent rewrite system over a 2-sorted signature: *lists* and *elements*. Our concern in Section 3 is the unification problem modulo this rewrite system, that we denote by $\mathcal{BC}$; we present a 2-level inference system (corresponding, in a way, to the two sorts of the signature) for solving this problem. Although our main aim is to investigate the unification problem for the case where h is an interpreted function symbol (as in the two situations illustrated above), we shall also be considering the case where h is a free uninterpreted symbol. The soundness and completeness of our inference procedure are established in Section 4. While the complexity of the unification problem is polynomial over the size of the problem when h is uninterpreted, it turns out to be NP-complete when h is interpreted so that the rewrite system models CBC encryption. We then present, in Section 5, a 2-sorted convergent system $\mathcal{DBC}$ that fully models at an abstract level, a block chaining cipher-decipher mode without using any AC-operators; this is done by adding a couple of equations to the above two: one for specifying a left-inverse g for h (g does the deciphering), and the other for specifying the block chaining mode for deciphering. A 2-level inference procedure extending the one given in Section 3 is presented, and is shown to be sound and complete for unification modulo this extended system $\mathcal{DBC}$; unification modulo $\mathcal{DBC}$ also turns out to be NP-complete. In the concluding section we briefly evoke possible lines of future work over these systems $\mathcal{BC}$ and $\mathcal{DBC}$.

Note: The first part of this paper, devoted to unification modulo $\mathcal{BC}$, is a more detailed version of the work we presented at LATA 2012 ([2]).

2. Notation and Preliminaries

We consider a ranked signature Σ, with two *disjoint* sorts: τ_e and τ_l, consisting of binary functions *bc, cons, h*, and a constant *nil*, and typed as follows:

$$bc: \tau_l \times \tau_e \to \tau_l \quad , \quad cons: \tau_e \times \tau_l \to \tau_l \quad , \quad h: \tau_e \times \tau_e \to \tau_e \quad , \quad nil: \tau_l.$$

We also assume given a set $\mathcal{X}$ of countably many variables; the objects of our study are the (well-typed) terms of the algebra $\mathcal{T}(\Sigma, \mathcal{X})$; terms of the type τ_e will be referred to as *elements*; and those of the type τ_l as *lists*. It is assumed that the only constant of type list is *nil*; the other constants, if any, will all be of the type element. For better readability, the set of variables $\mathcal{X}$ will be divided into two subsets: those to which 'lists' can get assigned will be denoted with upper-case letters as: $X, Y, Z, U, V, W, \ldots$, with possible suffixes or primes; these will be said to be variables of type τ_l; variables to which 'elements' can get assigned will be denoted with lower-case letters, as: $x, y, z, u, v, w, \ldots$, with possible suffixes or primes; these will be said to be variables of type τ_e. The theory we shall be studying first in this paper is defined by the two axioms (equations) already mentioned in the Introduction:

$$bc(nil,\, z) = nil, \qquad bc(cons(x, Y),\, z) = cons(\, h(x, z),\; bc(Y,\, h(x, z))\,)$$

It is easy to see that these axioms can both be oriented left-to-right under a suitable *lexicographic path ordering (lpo)* (cf. e.g., [10]), and that they form then a convergent — i.e., confluent and terminating — 2-sorted rewrite system.

As mentioned in the previous section, we consider two theories that contain the above two axioms. The first is where these are the only axioms; we call that theory $\mathcal{BC}_0$. The other theory is where h is interpreted as for CBC, i.e., where $h(x, y) = e_k(x \oplus y)$ where $\oplus$ is exclusive-or and e_k is encryption using some (fixed) given key k. This theory will be referred to as $\mathcal{BC}_1$. We use the phrases "$\mathcal{BC}$-unification" and "unification modulo $\mathcal{BC}$" to refer to unification problems modulo both the theories, collectively.

Note that in the case where h is a free uninterpreted symbol (i.e., $\mathcal{BC}_0$) h is fully cancellative in the sense that for any terms s_1, t_1, s_2, t_2, $h(s_1, t_1) \approx_{\mathcal{BC}} h(s_2, t_2)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $t_1 \approx_{\mathcal{BC}} t_2$. But when h is interpreted for CBC, this is no longer true; in such a case, h will be only *semi-cancellative*, in the sense that for all terms s_1, s_2, t, the following holds:

h is right-cancellative: $h(s_1, t) \approx_{\mathcal{BC}} h(s_2, t)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$, and

h is also left-cancellative: $h(t, s_1) \approx_{\mathcal{BC}} h(t, s_2)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$.

Thus, in the sequel, when we look for the unifiability of any set of element equations modulo $\mathcal{BC}_0$ (resp. modulo $\mathcal{BC}_1$) the cancellativity of h (resp. the semi-cancellativity of h) will be used as needed, in general without any explicit mention.

Our concern in this section, and the one following, is the equational unification problems modulo $\mathcal{BC}_0$ and $\mathcal{BC}_1$. We assume without loss of generality (wlog) that any given $\mathcal{BC}$-unification problem $\mathcal{P}$ is in *standard form,* i.e., $\mathcal{P}$ is given as a set of equations $\mathcal{EQ}$, each having one of the following forms:

$$U =^? V,\; U =^? bc(V, y),\; U =^? cons(v, W),\; U =^? nil,$$
$$u =^? v,\;\; v =^? h(w, x),\; u =^? const$$

where *const* stands for any ground constant of sort τ_e. The first four kinds of equations — the ones with a list-variable on the left-hand side — are called *list-equations,* and the rest (those which have an element-variable on the left-hand side) are called *element-equations.* For any problem $\mathcal{P}$ in standard form, $\mathcal{L}(\mathcal{P})$ will denote the subset formed of its list-equations, and $\mathcal{E}(\mathcal{P})$ the subset of element-equations. A set of element-equations is said to be in *dag-solved form* (or *d-solved form*) ([14]) if and only if they can be arranged as a list $x_1 =^? t_1, \ldots, x_n =^? t_n$, such that:

$\forall\, 1 \leq i < j \leq n$: x_i and x_j are distinct variables, and x_i does not occur in t_i nor in any t_j.

Such a notion is naturally extended to sets of list-equations as well. In the next section we give an inference system for solving any $\mathcal{BC}$-unification problem in standard form. For any given problem $\mathcal{P}$, its rules will transform $\mathcal{L}(\mathcal{P})$ into one in *d*-solved form. The element-equations at that point can be passed on to an algorithm for solving them — thus in the case of $\mathcal{BC}_1$ what we need is an algorithm for solving the *general* unification problem modulo the theory of exclusive-or.

Any development presented below — without further precision on h — is meant as one which will be valid for both $\mathcal{BC}_0$ and $\mathcal{BC}_1$.

3. Inference System for $\mathcal{BC}$-Unification

The inference rules have to consider two kinds of equations: the rules for the *list-equations* in $\mathcal{P}$, i.e., equations whose left-hand sides (lhs) are variables of type τ_l, and the rules for the *element-equations,* i.e., equations whose lhs are variables of type τ_e. Our method of solving any given unification problem will be 'modular' on these two sets of equations: The list-inference rules will be shown to terminate under suitable conditions, and then all we will need to do is to solve the resulting set of element-equations for h.

A few technical points need to be mentioned before we formulate our inference rules. Note first that it is not hard to see that *cons* is cancellative; by this we mean that $cons(s_1, T_1) \approx_{\mathcal{BC}} cons(s_2, T_2)$, for terms s_1, s_2, T_1, T_2, if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $T_1 \approx_{\mathcal{BC}} T_2$. On the other hand, it can be shown by structural induction (and the semi-cancellativity of h) that *bc* is *conditionally* semi-cancellative, depending on whether its first argument is *nil* or not; for details, see *Appendix-1.* This property of *bc* will be assumed in the sequel.

The inference rules given below will have to account for cases where an 'occur-check' succeeds on some list-variable, and the problem will be unsolvable. The simplest among such cases is when we have an equation of the form $U =^? cons(z, U)$ in the problem. But one could have more complex unsolvable cases, where the equations involve both *cons* and *bc*; e.g., when $\mathcal{P}$ contains equations of the form: $U =^? cons(x, V), U =^? bc(V, y)$; the problem will be unsolvable in such a case: indeed, from the axioms of $\mathcal{BC}$, one deduces that V must be of the form $V =^? cons(v, V')$, for some v and V', then x must be of the form $x =^? h(v, y)$, and subsequently $V =^? bc(V', x)$, and we are back to a set of equations of the same format. We need to infer failure in all such cases. With that purpose, we define the following relations on the list-variables of the equations in $\mathcal{P}$:

- $U >_{cons} V$ iff $U =^? cons(z, V)$, for some z.
- $U >_{bc} V$ iff there is an equation $U =^? bc(V, x)$
- $U \sim_{bc} V$ iff $U =^? bc(V, w)$, or $V =^? bc(U, w)$, for some w.

Note that $\sim_{bc}$ is the symmetric closure of the relation $>_{bc}$; its reflexive, symmetric and transitive closure is denoted as $\sim^*_{bc}$. The transitive closure of $>_{bc}$ is denoted as $>^+_{bc}$; and its reflexive transitive closure as $>^*_{bc}$.

Note, on the other hand, that $U =^? bc(U, x)$ is solvable by the substitution $\{U := nil\}$; in fact this equation forces U to be nil, as would also a set of equations of the form $U =^? bc(V, y),\ V =^? bc(U, x)$. Such cycles (as well as some others) have to be checked to determine whether a list-variable is forced to be nil. This can be effectively done with the help of the relations defined above on the type τ_l variables. We define, recursively, a set **nonnil** of the list-variables of $\mathcal{P}$ that cannot be nil for any unifying substitution, as follows:

- if $U =^? cons(x, V)$ is an equation in $\mathcal{P}$, then $U \in$ **nonnil**.
- if $U =^? bc(V, x)$ is an equation in $\mathcal{P}$, then $U \in$ **nonnil** if and only if $V \in$ **nonnil**.

We have then the following obvious result:

Lemma 3.1. *A variable $U \in$ **nonnil** if and only if there are variables V and W such that $U \sim^*_{bc} V$ and $V >_{cons} W$.*

Some of the inference rules below will refer to a graph whose nodes are the list-variables of the given problem $\mathcal{P}$, 'considered equivalent up to equality'; more formally: for any list-variable U of $\mathcal{P}$, we denote by $[U]$ the equivalence class of list-variables that get equated to U in $\mathcal{P}$, in the following sense:

$$[U] = \{V \mid U =^? V \in \mathcal{P} \text{ or } V =^? U \in \mathcal{P}\}.$$

Any relation $\mathcal{R}$ defined over the list-variables of $\mathcal{P}$ is then extended naturally to these equivalence classes, by setting: $\mathcal{R}([U_1], \ldots, [U_n])$ iff $\exists V_1 \in [U_1] \ldots \exists V_n \in [U_n]\colon \mathcal{R}(V_1, \ldots, V_n)$.

Definition 3.2. Let $G_l = G_l(\mathcal{P})$ be the graph whose nodes are the equivalence classes on the list-variables of $\mathcal{P}$, with arcs defined as follows: From a node $[U]$ on G_l there is a *directed* arc to a (not necessarily different) node $[V]$ on G_l if and only if:

- Either $U >_{cons} V$: in which case the arc is labeled with $>_{cons}$
- $U >_{bc} V$: in which case the arc is labeled with $>_{bc}$.

In the latter case, G_l will also have a *two-sided (undirected)* edge between $[U]$ and $[V]$, which is labeled with $\sim_{bc}$. The graph G_l is called the *propagation graph* for $\mathcal{P}$.

A node $[U]$ on G_l is said to be a *bc/bc*-peak if $\mathcal{P}$ contains two different equations of the form $U =^? bc(V, x), U =^? bc(W, y)$; the node $[U]$ is said to be a *cons/bc*-peak if $\mathcal{P}$ has two different equations of the form $U =^? cons(x, V_1)$, $U =^? bc(V, z)$.

On the set of nodes of G_l, we define a partial relation $\succ_l$ by setting: $[U] \succ_l [V]$ iff there is a path on G_l from $[U]$ to $[V]$, at least one arc of which has label $>_{cons}$. In other words,

$$\succ_l = \sim^*_{bc} \circ >_{cons} \circ (\sim_{bc} \cup >_{cons})^*$$

A list-variable U of $\mathcal{P}$ is said to *violate occur-check* iff $[U] \succ_l [U]$ on G_l. For instance, the variable U violates occur-check in the problem:

$$U =^? bc(W, z),\ W =^? cons(x, U),$$

as well as in the problem:

$$U =^? bc(V, z),\ V =^? bc(W, a), W =^? cons(a, L),\ L =^? bc(U, b)$$

It can be checked that both the problems are unsatisfiable.

3.1. **Inference System $\mathcal{INF}_l$ for List-Equations.**

(L1) *Variable Elimination*:

$$\frac{\{U =^? V\} \uplus \mathcal{EQ}}{\{U =^? V\} \cup [V/U](\mathcal{EQ})} \qquad \text{if } U \text{ occurs in } \mathcal{EQ}$$

(L2) *Cancellation on cons*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? cons(x, V)\}}{\mathcal{EQ} \cup \{U =^? cons(x, V),\ v =^? x,\ W =^? V\}}$$

(L3.a) *Nil solution-1*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.b) *Nil solution-2*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ V =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.c) *Nil solution-3*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}} \qquad \text{if } V >^*_{bc} U$$

(L4.a) *Semi-Cancellation on bc*, at a *bc*/*bc*-peak:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, x)\}}{\mathcal{EQ} \cup \{U =^? bc(V, x),\ W =^? V\}}$$

(L4.b) *Push bc below cons*, at a **nonnil** *bc*/*bc*-peak:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\mathcal{EQ} \cup \{V =^? cons(v, Z),\ W =^? cons(w, Z),\ U =^? cons(u, U'),\ U' =^? bc(Z, u),\ u =^? h(v, x),\ u =^? h(w, y)\}}$$

if $U \in$ **nonnil**

(L5) *Splitting*, at a *cons*/*bc*-peak:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(x, U_1),\ U =^? bc(V, z)\}}{\mathcal{EQ} \cup \{U =^? cons(x, U_1),\ V =^? cons(y, V_1),\ x =^? h(y, z),\ U_1 =^? bc(V_1, x)\}}$$

(L6) *Occur-Check Violation*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } U \text{ occurs in } \mathcal{P}, , \text{ and } [U] \succ_l [U] \text{ on the graph } G_l$$

(L7) *Size Conflict*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? nil\}}{FAIL}$$

The symbol '$\uplus$' in the premises of the above inference rules stands for disjoint set union (and '$\cup$' for usual set union). The role of the Variable Elimination inference rule (L1) is to keep the propagation graph of $\mathcal{P}$ irredundant: each variable has a unique representative node on $G_l(\mathcal{P})$, up to variable equality. This rule is applied most eagerly. Rules (L2), (L3.a)–(L3.c) and (L4.a) come next in priority, and then (L4.b). The Splitting rule (L5) is applied in the "laziest" fashion, i.e., (L5) is applied only when no other rule is applicable. The above inference rules are all "don't-care" nondeterministic. (The priority notions just mentioned serve essentially for optimizing the inference procedure.)

The validity of the rule (L4.b) ('Pushing *bc* below *cons*') results from the cancellativity of *cons* and the semi-cancellativity of *bc* (*Appendix-1*). Note that the variables Z, U', and u in the 'inferred part' of this rule (L4.b) might need to be fresh; the same is true also for the variables y and V_2 in the inferred part of the Splitting rule; but, in either case this is not obligatory, if the equations already present can be used for applying these rules. Type-inference failure is assumed to be checked implicitly; no explicit rule is given.

The following point should be kept in mind: Any given problem $\mathcal{P}$ naturally 'evolves' under the inference rules; and new variables might get added in the process, if rule (L5) or rule (L4.b) is applied; but none of the variables initially present in $\mathcal{P}$ can disappear in the process; not even under the Variable Elimination rule (L1). Thus, although the graph G_l referred to in the Occur-Check Violation rule (L6) is the graph of the 'current problem', the node it refers to might still be one corresponding to an initial variable.

We show now that such an introduction of fresh variables cannot go for ever, and that the above "don't-care" nondeterministic rules suffice, essentially, for deciding *unifiability* modulo the axioms of $\mathcal{BC}$.

Proposition 3.3. *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem, given in standard form. The system $\mathcal{INF}_l$ of list-inference rules, given above, terminates on $\mathcal{P}$ in polynomially many steps.*

Proof. Assume given a problem $\mathcal{P}$ in standard form, for which the inference process does not lead to failure on Occur-Check (L6) or Size-Conflict (L7). If $\mathcal{INF}_l$ is non-terminating on such a $\mathcal{P}$, at least one of the rules of $\mathcal{INF}_l$ must have been applied infinitely often along some inference chain; we show that this cannot be true for any of the rules in $\mathcal{INF}_l$.

Note first that an equation of the form $U =^? V$ in $\mathcal{P}$ is never handled in 'both directions' by the variable elimination rule (L1); an application of this rule means: every occurrence of the variable U in the problem is replaced by the variable V. It is easy to check then, that for this reason, (L1) cannot give rise to non-termination. On the other hand, the list-inference rules (L2) through (L4.a) eliminate a (directed) outgoing arc from some node of G_l; so their termination is easy to check. It should be clear, that for these three rules, termination is polynomial (even linear). Thus, to show the termination of the entire inference process in polynomially many steps, we have to look at how the problem evolves under the rule (L5) (*Splitting*) and the rule (L4.b) (*Pushing bc below cons*). We show that if occur-check violation (L6) does not occur, then the applications of the rule (L5) or of the rule (L4.b) cannot go on forever.

For proving this, we shall be using an equivalence relation denoted as $\sim_\beta$, on the list-variables of the given problem. It is defined as the smallest equivalence relation[1] satisfying the following conditions, on the list-variables of $\mathcal{P}$:

- If $U \sim^*_{bc} V$ then $U \sim_\beta V$.
- Let $U >_{cons} U'$ and $V >_{cons} V'$; then $U \sim_\beta V$ implies $U' \sim_\beta V'$.

Observe now that the number of *bc*-equations, i.e., list-equations of the form $U =^? bc(V, z)$, never increases. This number decreases in most cases, except for (L1), (L2) and (L5). The splitting rule (L5) does not decrease the number of *bc*-equations and may

[1]The relation $\sim_\beta$ can be viewed as a combination of the *unification closure*, a notion defined by Kanellakis and Revesz [15], and the *congruence closure* of $\sim^*_{bc}$. The difference is that here we are working with a typed system.

introduce new variables, but the number of $\sim_\beta$-equivalence classes of nodes (on the current graph) does not increase: Indeed, applying the splitting rule (L5) on a list-equation $U =^? bc(V, z)$ removes that equation, and creates a list-equation of the form $U_1 =^? bc(V_1, x)$ for some list-variables U_1 and V_1, such that $V \sim_{bc} U >_{cons} U_1 \sim_{bc} V_1$; we have: $V_1 \sim_\beta U_1$, since $V \sim_\beta U$.

Suppose now that applying the splitting rule does not terminate. Then, at some stage, the derived problem will have a sequence of variables of the form $U_0 >_{cons} U_1 >_{cons} \cdots >_{cons} U_n$, such that the length of the sequence n strictly exceeds the initial number of $\sim_\beta$-equivalence classes — which cannot increase under splitting, as we just observed above. So there must exist indices $0 \leq i < j \leq n$ such that $U_i \sim_\beta U_j$.

Let $j \leq n$ be the smallest integer for which there exists an i, $0 \leq i < j$, such that $U_i \sim_\beta U_j$. Then, by the definition of $\sim_\beta$, we must have $U_i \sim^*_{bc} U_j$. Consequently, we would then also have $[U_i] \succ_l [U_i]$; and that would have caused the inference process to terminate with FAIL, as soon as both the variables U_i and U_j appear in the problem derived under the inferences.

Termination of (L4.b) can now be proved as follows: The number of $\sim^*_{bc}$-equivalence classes may increase by 1 with each application of (L4.b), but the number of $\sim_\beta$-equivalence classes remains the same, for the same reason as above. Let m be the number of *bc*-equations in the input problem and let n be the number of variables in the input problem. We then show that the total number of applications of (L4.b) and (L5) cannot exceed mn: Indeed, whenever one of (L4.b) or (L5) is applied, some number of *bc*-equations are removed and an equal or lesser number are added, whose variables belong to $\sim_\beta$-equivalence classes at a 'lower level' as explained above, i.e., below some *cons* steps. There are at most n such equivalence classes, since the number of $\sim_\beta$ equivalence classes does not increase (and there cannot be more than n such equivalence classes, to start with). So a *bc*-equation cannot be "pushed down" more than n times. Since there are initially m *bc*-equations, the total number of applications of (L4.b) and (L5) cannot exceed mn. □

A set of equations will be said to be *L-reduced* if none of the above inference rules (L1) through (L7) is applicable. (Note: such a problem may not be in *d*-solved form: an easy example is given a couple of paragraphs below.)

Unification modulo $\mathcal{BC}$: The rules (L1) through (L7) are not enough to show the existence of a unifier modulo $\mathcal{BC}$. The subset of element-equations, $\mathcal{E}(\mathcal{P})$, may not be solvable; for example, the presence of an element-equation of the form $\{x =^? h(x, z)\}$ should lead to failure. However, we have the following:

Proposition 3.4. *If $\mathcal{L}(\mathcal{P})$ is in L-reduced form, then $\mathcal{P}$ is unifiable modulo $\mathcal{BC}$ if and only if the set $\mathcal{E}(\mathcal{P})$ of its element-equations is solvable.*

Proof. If $\mathcal{L}(\mathcal{P})$ is *L*-reduced, then setting every list-variable that is not in **nonnil** to *nil* will lead to *a unifier* for $\mathcal{L}(\mathcal{P})$, modulo $\mathcal{BC}$, provided $\mathcal{E}(\mathcal{P})$ is solvable. □

Recall that $\mathcal{BC}_0$ is the theory defined by $\mathcal{BC}$ when h is uninterpreted.

Proposition 3.5. *Let $\mathcal{P}$ be any $\mathcal{BC}_0$-unification problem, given in standard form. Unifiability of $\mathcal{P}$ modulo $\mathcal{BC}_0$ is decidable in polynomial time (wrt the size of $\mathcal{P}$).*

Proof. If the inferences of $\mathcal{INF}_l$ applied to $\mathcal{P}$ lead to failure, then $\mathcal{P}$ is not unifiable modulo $\mathcal{BC}$; so assume that this is not the case, and replace $\mathcal{P}$ by an equivalent problem which is

L-reduced, deduced in polynomially many steps by Proposition 3.3. By Proposition 3.4, the unifiability modulo $\mathcal{BC}$ of such a $\mathcal{P}$ amounts to checking if the set $\mathcal{E}(\mathcal{P})$ of its element-equations is solvable. We are in the case where h is uninterpreted, so to solve $\mathcal{E}(\mathcal{P})$ we apply the rules for standard unification, and check for their termination without failure; this can be done in polynomial time [5]. (In this case, h is fully cancellative.) □

It can be seen that while termination of the above inference rules guarantees the *existence* of a unifier (provided the element equations are syntactically solvable), the resulting L-reduced system may not lead directly to a unifier. For instance, the L-reduced system of list-equations $\{U =^? bc(V, x),\ U =^? bc(V, y)\}$ is unifiable, with the following two incomparable unifiers:

$$\{x := y,\ U := bc(V, y)\} \text{ and } \{U := nil,\ V := nil\}$$

To get a complete set of unifiers we need three more inference rules, which are "don't-know" nondeterministic, to be applied only to L-reduced systems:

(L8) *Nil-solution-Branch for bc*, at a *bc*/*bc*-peak:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil,\ W =^? nil\}}$$

(L9) *Guess a non-Nil branch for bc*, at a *bc*/*bc*-peak:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\begin{array}{c}\mathcal{EQ} \cup \{V =^? cons(v, Z),\ W =^? cons(w, Z),\ U =^? cons(u, U'),\\ U' =^? bc(Z, u),\ u =^? h(v, x),\ u =^? h(w, y)\}\end{array}}$$

(L10) *Standard Unification on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\mathcal{EQ} \cup \{U =^? bc(W, y),\ V =^? W,\ x =^? y\}}$$

Rule (L9) nondeterministically 'guesses' U to be in **nonnil**; in other words, it applies rule (L4.b) 'unconditionally'. The inference system thus extended will be referred to as $\mathcal{INF}'_l$. By the same reasonings as developed above, $\mathcal{INF}'_l$ also terminates, in polynomially many steps, on any problem given in standard form. We establish now a technical result, valid whether or not h is interpreted:

Proposition 3.6. *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem in standard form, to which none of the inferences of $\mathcal{INF}'_l$ is applicable. Then its set of list-equations is in d-solved form.*

Proof. If none of the equations in $\mathcal{P}$ involve *bc* or *cons* (i.e., all equations are equalities between list-variables), then the proposition is proved by rule (L1) (*Variable Elimination*).

Observe first that if $\mathcal{INF}_l$ is inapplicable to $\mathcal{P}$, then, on the propagation graph G_l for $\mathcal{P}$, there is *at most one outgoing directed arc* of G_l at any node U: Otherwise, suppose there are two distinct outgoing arcs at some node U on G_l; if both directed arcs bear the label $>_{cons}$, then rule (L2) of $\mathcal{INF}_l$ would apply; if both bear the label $>_{bc}$, then one of (L4.a), (L4.b), (L9), (L10) would apply; the only remaining case is where one of the outgoing arcs is labeled with $>_{cons}$ and the other has label $>_{bc}$, but then the splitting rule (L5) would apply.

Consider now any given connected component Γ of G_l. There can be no directed cycle from any node U on Γ to itself: otherwise the Occur-Check-Violation rule (L6) would have applied. It follows, from this observation and the preceding one, that there is a unique

end-node U_0 on Γ, i.e., a node from which there is *no directed outgoing arc*; and also that for any given node U on Γ, there is a unique well-defined directed path leading from U to that end-node U_0.

It follows easily from these, that the list-variables on the left hand sides of the equations in $\mathcal{P}$ (on the different connected components of G_l) can be ordered suitably, so as to satisfy the condition for $\mathcal{P}$ to be in a d-solved form. □

Example 3.7. The following $\mathcal{BC}_0$-unification problem is in standard form:

$$U =^? cons(x, W),\ U =^? bc(V, y),\ W =^? bc(V_2, y),\ x =^? h(z, y),\ y =^? a$$

We apply (L5) (*Splitting*) and write $V =^? cons(v_1, V_1)$, with v_1, V_1 fresh; this, followed by an application of rule (L2) (*Cancellation on cons*) leads to:

$$U =^? cons(x, W),\ V =^? cons(v_1, V_1),\ W =^? bc(V_1, x),\ W =^? bc(V_2, y),\\ x =^? h(v_1, y),\ x =^? h(z, y),\ y =^? a$$

We apply cancellativity of h (valid for $\mathcal{BC}_0$), and an element-variable elimination; the problem thus derived is the following:

$$U =^? cons(x, W),\ V =^? cons(z, V_1),\ W =^? bc(V_1, x),\ W =^? bc(V_2, y),\\ x =^? h(v_1, y),\ z =^? v_1,\ y =^? a$$

(i) No rule of $\mathcal{INF}_l$ is applicable: in particular, (L4.b) doesn't apply since W is not in **nonnil**; but the rule (L8) (*Nil-solution Branch for bc*) can be nondeterministically applied:

$$U =^? cons(x, W),\ W =^? nil,\ V_1 =^? nil,\ V_2 =^? nil,\ V =^? cons(z, V_1),\\ x =^? h(v_1, y),\ z =^? v_1,\ y =^? a$$

These equations, in d-solved form, give a solution to the original problem.

(ii) For the sake of completeness, we could also try the rule (L9) (*Guess a non-Nil branch*) nondeterministically, successively on the two equations for W in the problem derived above; so we write $V_1 =^? cons(v_2, V_2')$ and $V_2 =^? cons(v_3, V_3')$. These applications of (L9), followed by applications of *Variable elimination, Cancellation on cons*, and the cancellativity of h (valid for the theory $\mathcal{BC}_0$), will lead us to:

$$U =^? cons(y, W),\ V =^? cons(v_1, V_1),\ V_1 =^? cons(v_3, V_3'),\\ V_2 =^? V_1, V_2' =^? V_3',\ W =^? bc(V_1, x),\\ x =^? y,\ y =^? h(v_1, y),\ v_2 =^? v_3,\ z =^? v_1,\ y =^? a$$

The list-equations are in d-solved form, but the element-equations being unsatisfiable we are led to failure.

(iii) For the following problem (almost same as (i) above, but for an element-equation):

$$U =^? cons(x, W),\ U =^? bc(V, y),\ W =^? bc(V_2, y),\ y =^? a$$

the reasonings as developed in (ii) above would have led us to a non-nil solution for W:

$$U =^? cons(y, W),\ V =^? cons(v_1, V_1),\ V_1 =^? cons(v_2, V_3'),\ V_2 =^? V_1,\ W =^? bc(V_1, x),\\ x =^? y,\ \ y =^? a$$

where V_3' is any arbitrary list, and v_1, v_2 are any arbitrary elements. □

We turn our attention in the following section to the unification problem modulo $\mathcal{BC}$. When h is uninterpreted, we saw that this unification is decidable in polynomial time. But when h is interpreted so that $\mathcal{BC}$ models CBC, we shall see that unification modulo $\mathcal{BC}_1$ is NP-complete.

4. Solving a $\mathcal{BC}$-Unification problem

Let $\mathcal{P}$ be a $\mathcal{BC}$-Unification problem, given in standard form. We assume that $\mathcal{INF}'_l$ has terminated without failure on $\mathcal{P}$; we saw, in the preceding section (Proposition 3.6), that $\mathcal{P}$ is then in d-solved form. We also assume that we have a sound and complete procedure for solving the element-equations of $\mathcal{P}$, that we shall denote as $\mathcal{INF}_e$. For the theory $\mathcal{BC}_0$ where h is uninterpreted, we know (Proposition 3.5) that $\mathcal{INF}_e$ is standard unification, with cancellation rules for h, and failure in case of 'symbol clash'. For the theory $\mathcal{BC}_1$, where $h(x, y)$ is interpreted as $e_k(x \oplus y)$ for some fixed key k, $\mathcal{INF}_e$ will have rules for semi-cancellation on h, besides the rules for unification modulo XOR in some fixed procedure; such a procedure is assumed given once and for all.

In all cases, we shall consider $\mathcal{INF}_e$ as a black-box that either returns most general unifiers (*mgu*'s) for the element-equations of $\mathcal{P}$, or a failure message when these are not satisfiable. Note that $\mathcal{INF}_e$ is unitary for $\mathcal{BC}_0$ and finitary for $\mathcal{BC}_1$. For any problem $\mathcal{P}$ in d-solved form, satisfiable under the theory $\mathcal{BC}_0$, there is a unique mgu, as expressed by the equations of $\mathcal{P}$ themselves (cf. also [14]), that we shall denote by $\theta_{\mathcal{P}}$. Under $\mathcal{BC}_1$ there could be more than one (but finitely many) mgu's; we shall agree to denote by $\theta_{\mathcal{P}}$ any one among them. The entire procedure for solving any $\mathcal{BC}$-unification problem $\mathcal{P}$, given in standard form, can now be synthesized as a nondeterministic algorithm:

The Algorithm $\mathcal{A}$: Given a $\mathcal{BC}$-unification problem $\mathcal{P}$, in standard form.
G_l = Propagation graph for $\mathcal{P}$.
$\mathcal{INF}'_l$ = Inference procedure given above for $\mathcal{L}(\mathcal{P})$.
$\mathcal{INF}_e$ = Any given (complete) procedure for solving the equations of $\mathcal{E}(\mathcal{P})$.

(1) Compute a standard form for $\mathcal{P}$, to which the "don't-care" inferences of $\mathcal{INF}_l$ are no longer applicable. If this leads to failure, exit with FAIL. Otherwise, replace $\mathcal{P}$ by this standard form.
(2) Apply the "don't-know" nondeterministic rules (L8)–(L10), followed by the rules of $\mathcal{INF}_l$ as needed, until the equations no longer get modified by the inference rules (L1)–(L10). If this leads to failure, exit with FAIL.
(3) Apply the procedure $\mathcal{INF}_e$ for solving the residual set $\mathcal{E}(\mathcal{P})$ of element-equations; if this leads to failure, exit with FAIL.
(4) Otherwise let σ be the substitution on the variables of $\mathcal{P}$ as expressed by the resulting equations. Return σ as a solution to $\mathcal{P}$.

Proposition 4.1. *The algorithm $\mathcal{A}$ is sound and complete.*

Proof. The soundness of $\mathcal{A}$ follows from the soundness (assumed) of $\mathcal{INF}_e$ and that of $\mathcal{INF}'_l$, which is easy to check: obviously, if $\mathcal{P}'$ is any problem derived from $\mathcal{P}$ by applying any of these inference rules, then any solution for $\mathcal{P}'$ corresponds to a solution for $\mathcal{P}$. The completeness of $\mathcal{A}$ follows from the completeness (assumed) of $\mathcal{INF}_e$, and the completeness of $\mathcal{INF}'_l$ that we prove below. □

Lemma 4.2. *If σ is a solution for a given $\mathcal{BC}$-unification problem $\mathcal{P}$ in standard form, then there is a sequence of $\mathcal{INF}'_l$-inference steps that transforms $\mathcal{P}$ into a problem $\mathcal{P}'$ in d-solved form such that σ is an instance of $\theta_{\mathcal{P}'}$ (modulo $\mathcal{BC}$).*

Proof. We know that the inference rules of $\mathcal{INF}'_l$ terminate on $\mathcal{P}$; let N be the maximum number of steps needed for this termination, including along all possible "don't-know" branches of the process. We prove the lemma by induction on N, and case analysis for the possible branches.

Observe first that if $\mathcal{P}'$ is a problem derived from $\mathcal{P}$ under any inference rule of $\mathcal{INF}'_l$, then the given substitution σ, on on the variables of $\mathcal{P}$, extends naturally as a substitution on the variables of $\mathcal{P}'$, satisfying the equations of $\mathcal{P}'$. (This needs to be checked only if $\mathcal{P}'$ might involve new variables, such as when $\mathcal{P}'$ is derived from $\mathcal{P}$ under rule (L5) or rule (L4.b); the reasoning is straightforward for either of these cases.)

If $\mathcal{P}'$ is derived from $\mathcal{P}$ by applying one of the "don't-care" rules of $\mathcal{INF}_l$, then the assertion of the lemma follows from the above observation and the induction hypothesis. So we may assume wlog that the given problem $\mathcal{P}$ is already L-reduced (i.e., none of the inferences of $\mathcal{INF}_l$ is applicable). If such a $\mathcal{P}$ is already in d-solved form, then we are done, since $\sigma \preceq_{\mathcal{BC}} \theta_{\mathcal{P}}$, for some mgu $\theta_{\mathcal{P}}$. (If the theory is $\mathcal{BC}_1$, this means: there exists one among the finitely many mgus, for which this holds.)

If $\mathcal{P}$ is not in d-solved form, then several cases are possible, depending on the possible inference branches. It suffices to consider one such case – the reasoning being quite similar for all the others. Suppose there are two equations $U =^? bc(Z, v)$ and $U =^? bc(Y, w)$ in $\mathcal{P}$. If $\sigma(v) =_{\mathcal{BC}} \sigma(w)$, then we must have $\sigma(Z) =_{\mathcal{BC}} \sigma(Y)$, and σ is extendable as a solution for the problem obtained by applying the rule (L10). If $\sigma(v) \neq_{\mathcal{BC}} \sigma(w)$, then σ must be extendable as a solution to the problem derived under rule (L8) or rule (L9). The induction hypothesis (on the maximum number of inference steps needed for termination) completes then the argument to prove the lemma, in all cases. □

Proposition 4.3. *Unification modulo $\mathcal{BC}$ is finitary.*

Proof. Let $\mathcal{P}$ be a satisfiable $\mathcal{BC}$-unification problem. We can assume without loss of generality that $\mathcal{P}$ is in standard form, because any unification problem can be converted to a finite problem in standard form. Let S be the set of mgus for $\mathcal{P}$. By lemma 4.2, for each $\sigma \in S$, there is a sequence of $\mathcal{INF}'_l$-inference steps that leads to a problem $\mathcal{P}'$ in d-solved form, and an mgu $\theta_{\mathcal{P}'}$ such that σ is an instance of $\theta_{\mathcal{P}'}$. Let D be the set of all such derived problems. Because all the inference rules in $\mathcal{INF}'_l$ terminate, and because there are finitely many inference rules, D contains finitely many problems.

In the uninterpreted case $\mathcal{BC}_0$, σ is $\theta_{\mathcal{P}'}$ for some $\mathcal{P}' \in D$, so there are finitely many unifiers in S. For $\mathcal{BC}_1$, note that unification modulo XOR is finitary [16]. Therefore, there are finitely many XOR-mgus for the element problem derived from $\mathcal{P}'$, so there are finitely many unifiers in S that are instances of $\theta_{\mathcal{P}'}$. Since there are finitely many problems in D, there are finitely many unifiers in S. □

4.1. **$\mathcal{BC}_1$-Unification is NP-Complete.** Recall that $\mathcal{BC}_0$ is the theory defined by $\mathcal{BC}$ when h is uninterpreted, and $\mathcal{BC}_1$ is the theory when h is interpreted so that $\mathcal{BC}$ models the (XOR-based) cipher-block-chaining mode CBC.

Proposition 4.4. *Unifiability modulo the theory $\mathcal{BC}_1$ is NP-complete.*

Proof. NP-hardness follows from the fact that general unification modulo XOR is NP-complete [12]. We deduce the NP-upper bound from the following facts:

a) For any given $\mathcal{BC}$-unification problem, computing a standard form is in polynomial time, wrt the size of the problem.
b) Given a standard form, the propagation graph can be constructed in polynomial time (wrt its number of variables).
c) Applying (L1)-(L10) till termination takes only polynomially many steps.
d) Extracting the set of element-equations from the resulting set of equations is in P.
e) Solving the element-equations, with the procedure $\mathcal{INF}_e$, using unification modulo XOR, is in NP. □

4.2. **An Illustrative Example.**

The following public key protocol is a slight variant of one that was studied in [11] – the modification is that the namestamp of the sender of a message appears as the *first* block of the encrypted message body, and not the second as was specified in [11]:

$$A \to B : A,\ \{A, m\}_{kb}$$
$$B \to A : B,\ \{B, m\}_{ka}$$

where A, B are the participants of the protocol session, m is a message that they intend secret for others, and kb (resp. ka) is the public key of B (resp. A).

If the CBC encryption mode is assumed and the message blocks are all of the same size, then this protocol becomes insecure; here is why. Let $e_Z(x)$ stand for the encryption $e(x, kz)$ with the public key kz of any principal Z. Under the CBC encryption mode, what A sends to B is the following list, in the ML-notation:

$$A \to B : [\, A,\ [\, e_B(A \oplus v),\ e_B(m \oplus e_B(A \oplus v))\,]\,].$$

Here $\oplus$ stands for XOR and v is the initialization vector (IV) agreed upon between A and B. But then, some other agent I, entitled to open a session with B with initialization vector w, can get hold of the first encrypted block (namely: $e_B(A \oplus v)$) as well as the second encrypted block of what A sent to B, namely $e_B(m, e_B(A \oplus v))$; (s)he can then send the following as a 'bona fide' message to B:

$$I \to B : [\, I,\ [\, e_B(I \oplus w),\ e_B(m \oplus e_B(A \oplus v))\,]\,];$$

upon which B will send back to I the following:

$$B \to I : [\, B,\ [\, e_I(B \oplus w),\ e_I(\, m \oplus e_B(A \oplus v) \oplus e_B(I \oplus w) \oplus e_I(B \oplus w)\,)\,]\,].$$

It is clear now, that the intruder I can get hold of the message m intended to remain secret for him/her: By decrypting the second block of the (encrypted part of the) message received from B, (s)he first deduces: $m \oplus e_B(A \oplus v) \oplus e_B(I \oplus w) \oplus e_I(B \oplus w)$; by XOR-ing this with the first block of the message, (s)he obtains: $m \oplus e_B(A \oplus v) \oplus e_B(I \oplus w)$; from which (s)he can deduce m by XOR-ing with $e_B(I \oplus w)$ and $e_B(A \oplus v)$, both of which are known to him/her (the latter of these two terms is the first block of the message from A to B, that (s)he has intercepted).

Example 4.5. The above attack (which exploits the properties of XOR: $x \oplus x = 0,\ x \oplus 0 = x$) can be modeled as solving a certain $\mathcal{BC}_1$-unification problem. We assume that the names A, B, I, as well as the initialization vector w, are constants accessible to I. The message m and the initialization vector v, that A and B have agreed upon, are constants intended to

be secret for I. We shall interpret the function symbol h of $\mathcal{BC}$ in terms of encryption with the public key of B: i.e., $h(x, y)$ is $e_B(x \oplus y)$.

The protocol above can then be modeled as follows: We assume that the list of terms A sends to B, namely $[\, A, [h(A, v), h(m, h(A, v))]\,]$, is seen by the latter as the list of terms $[\, A, bc([A, m], v)\,]$; (s)he first recovers the namestamp A of the sender, then checks that the second argument under bc in what (s)he received is the IV agreed upon with A; subsequently (s)he sends back the appropriate list of terms to A, acknowledging receipt of the message.

Now, due to our CBC-assumption, the ground terms $h(A, v)$, $h(m, h(A, v))$ are both accessible to the intruder I. So the attack by I, mentioned above, corresponds to the fact that I *can* send to B the following list of terms: $[\, I, [\, h(I, w), h(m, h(A, v))\,]\,]$. That the attack materializes follows from the fact that B can solve the $\mathcal{BC}_1$-unification problem:

$$bc([I, z], w) =^? cons(h(I, w), [h(m, h(A, v))]),$$

for the element-variable z, i.e., B needs to solve the element-equation: $h(z, h(I, w)) =^? h(m, h(A, v))$; since h is interpreted here so that $\mathcal{BC}$ models CBC, (s)he can do so by setting: $z := m \oplus h(A, v) \oplus h(I, w)$; and that precisely leads to the attack. □

Remark 4.6. (i) The above analysis does *not* go through if the namestamp forms the *second block* of the encrypted part of the messages sent. In such a case, the protocol is 'leak-proof' even under CBC, provided we assume that an IV for a message is a secret to be shared only by the sender and the intended recipient of the message, and that it is *not* transmitted – as clear text or encrypted – as an initial 'block number zero' of the message body. Actually, by reasoning as above, one checks that the intruder I in such a case can only get hold of $m \oplus v$, where v is the (secret) IV that only A and B share. This in a sense is in accordance with [11], where the protocol was 'proved secure' under such a specification.

(ii) The considerations above lead us to conclude, implicitly, that in cryptographic protocols employing the CBC encryption mode, it is necessary to forbid free access to the IVs of the 'records' of the 'messages' sent, if information leak is to be avoided. This fact has been pointed out in the 90's, by Bellare et al ([6]), and again, in some detail, by K. G. Paterson et al in [19]; both point out that TLS 1.0 – with its predictable IVs – is inherently insecure. For more on this point, and on the relative advantages of TLS 1.1, TLS 1.2 over TLS 1.0, the reader can also consult, e.g., `http://www.educatedguesswork.org/2011/09/`

(Note: keeping IVs as shared secrets alone may not always be sufficient in general, as is shown by Example 2 above.)

5. A generic Block Chained Cipher-Decipher Scheme

In this section we extend the 2-sorted equational theory $\mathcal{BC}_0$ studied above, into one that fully models, in a simple manner and without using any AC-symbols, a 'generic' block chaining encryption-decryption scheme. This theory, that we shall refer to as $\mathcal{DBC}$, is

defined by the following set of (2-sorted) equations:

$$\begin{aligned} bc(nil,\, z) &= nil \\ bc(cons(x,Y),\, z) &= cons(h(x,z),\; bc(Y,\, h(x,z))) \\ g(h(x,y),\, y) &= x \\ db(nil,\, z) &= nil \\ db(cons(x,Y),\, z) &= cons(g(x,z),\; db(Y,\, x)) \\ db(bc(X,y),\, y) &= X \end{aligned}$$

where g is typed as $g:\ \tau_e \times \tau_e \to \tau_e$ and db is typed as $db:\ \tau_l \times \tau_e \to \tau_l$.

All these equations can be oriented from left to right under a suitable reduction ordering, to form a convergent (2-sorted) rewrite system. The 6th equation says that db is a left-inverse for bc; it is actually an inductive consequence of the first five: i.e., for any list-term X and element-term y both in ground normal form, $db(bc(X,y),y)$ reduces to X under the first five, a fact that can be easily checked by structural induction, cf. *Appendix-2*. (Its insertion as an equational axiom is for technical reasons, as will be explained in *Remark 5.8*(ii) below.)

A few words, by way of intended semantics in the context of cryptographic protocols, seem appropriate: $h(x,y)$ would in such a context stand for the encryption with the public key of an intended recipient B, of message x, *'coupled' in a sense to be defined,* with y as initialization vector (IV); and $g(h(x,y),\, y)$ would be the decryption of $h(x,y)$ with the private key of B, to be then *'decoupled', again in a sense to be defined*, with y. If an agent A wants to send a list of terms $cons(x,Y)$ to recipient B, (s)he would send out $bc(cons(x,Y),\, z)$ where z is the IV they have mutually agreed upon; and B would see it as the list of terms $cons(h(x,z),\; bc(Y,\, h(x,z)))$, from which (s)he can retrieve the individual message terms by applying the last equation for db in the system $\mathcal{DBC}$.

This generic block chained encryption-decryption scheme is a natural abstraction of the usual (XOR-based) CBC: it suffices to interpret the roles of h and g suitably, and define properly the meanings of 'coupling' and 'decoupling', to get the usual CBC mode; for that, one would *define* the 'coupling' as well as 'decoupling' of x with y as $x \oplus y$; $h(x,y)$ would then stand for $e_B(x \oplus y)$, and $g(z,y)$ would stand for $d_B(z) \oplus y$, where d_B is decryption with the private key of B. If we go back to Example 4.5 based on the usual CBC, the encrypted part of what A sends out to B (with the notation employed there) is the list of terms: $[\, h(A,v),\;\; h(m,h(A,v))\,]$, that corresponds to the term $bc([A,m],v)$. By applying the fifth equation in $\mathcal{DBC}$ to this list of terms, under the assignments: $z := v$, $x := h(A,v)$, $Y := [h(m,h(A,v)]$, B would then derive the following list:

$$[\, g(h(A,v),v),\;\; db([h(m,h(A,v))],h(A,v))\,];$$

i.e., the list $[A,m]$. In other words, the usual XOR-based CBC is indeed an 'instance' of the theory $\mathcal{DBC}$.

Remark 5.1. Other 'concrete' cipher-decipher block chaining modes can also be seen as instances of $\mathcal{DBC}$; one among them is the *Cipher FeedBack encryption mode* (CFB), which is defined as follows:

Let $M = p_1 \dots p_n$ be a message given as a list of n 'plaintext' message subblocks. Then the encryption of M with any given key k and initialization vector v is defined as the list $c_1 \dots c_n$, of ciphertext message subblocks, where:

$$c_1 = p_1 \oplus e_k(v), \quad \text{and } c_i = p_i \oplus e_k(c_{i-1}), \text{ for any } 1 < i \leq n$$

This encryption mode (also using XOR) is very similar to CBC, but works in the reverse direction (cf. e.g., `http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation`). It is an instance of $\mathcal{DBC}$, if the 'coupling' and the 'decoupling' operations of $\mathcal{DBC}$, namely $h(x, y)$ and $g(x, y)$, are both defined as $x \oplus e_k(y)$.

The theory $\mathcal{DBC}$ thus appears, indeed, as a high level equational abstraction of the block chained encryption-decryption mode; it employs no AC-symbols for this abstraction. It is easy to see, on the other hand, that the equations of $\mathcal{DBC}$ can all be oriented left-to-right under a suitable reduction ordering, to give a convergent rewrite system. We shall be showing below that unification modulo $\mathcal{DBC}$ is NP-decidable; it turns out to be actually NP-complete, due to the presence of a left-inverse for h (namely g).

Remark 5.2. : It is important to note that the function g is not semi-cancellative: $g(h(g(t, u), u), u) =_{\mathcal{DBC}} g(t, u)$, but $h(g(t, u), u)$ and t need not be equivalent modulo $\mathcal{DBC}$. However, it is easy to show that g is left-cancellative; see *Appendix-1* for the details.

5.1. **Unification modulo $\mathcal{DBC}$.**

We assume without loss of generality that any $\mathcal{DBC}$-unification problem $\mathcal{P}$ is given in a standard form, i.e., as a set of equations $\mathcal{EQ}$, each having one of the following forms:

$$U =^? V,\ U =^? bc(V, y),\ U =^? db(V, y),\ U =^? cons(v, W),\ U =^? nil,$$
$$u =^? v,\ u =^? g(w, y),\ v =^? h(w, x),\ u =^? const$$

We have to extend some of the notions and notation of Section 3.1, in order to take *db* into account. These extensions concern the propagation graph G_l of the problem and **nonnil**, the set of variables which cannot be *nil*.

(i) If $U =^? db(V, y)$ is in $\mathcal{P}$, then write $U >_{db} V$; in which case, insert a directed arc on G_l from $[U]$ to $[V]$ and label it with $>_{db}$. The graph G_l will also have then a *two-sided (undirected)* edge between $[U]$ and $[V]$, labeled with $\sim_{db}$.

(ii) The set of variables **nonnil**, defined earlier, is extended as follows: If $U =^? db(V, y)$ is in $\mathcal{P}$, then U is in **nonnil** if and only if V is in **nonnil**.

We define a new relation $>_c = >_{bc} \cup >_{db}$. Its symmetric closure is $\sim_c$ and its transitive, reflexive, and symmetric closure is $\sim_c^*$. The relations $>_c^+$, $>_{db}^+$, $>_{db}^*$ are then defined in the usual manner. If $U \sim_c V$, then U and V are related by 'chaining', i.e. by some number of *bc* and *db* operations. We refine then the partial relation $\succ_l$ on the nodes of G_l as follows:

$$\succ_l = \sim_c^* \ \circ\ >_{cons} \circ (\sim_c \cup >_{cons})^*$$

This relation can still continue to be read as: $[U] \succ_l [V]$ iff there is a directed path on G_l from $[U]$ to $[V]$, at least one arc of which has label $>_{cons}$.

We extend now the inference system $\mathcal{INF}'_l$ of Section 3.1 by adding the following list-inferences; these additional rules are essentially the *db*-counterparts of the list-inferences of $\mathcal{INF}'_l$ which only needed to consider *bc*. (There are several reasons why we have not worked with $\mathcal{DBC}$ right from the start – maybe the inference system would possibly have been more concise, if we had done so. A first reason is, that would have been at the expense of readability; a second reason is that $\mathcal{BC}$-unification is of interest on its own, especially for $\mathcal{BC}_1$, as is shown by Example 4.5 above; a third and conclusive reason is that the inference system we present below for $\mathcal{DBC}$-unification, actually reduces the problem to a problem of $\mathcal{BC}$-unification.) We first formulate the "don't-care" nondeterministic inference rules.

(DB1.a) *Nil solution-1 for db*::

$$\frac{\mathcal{EQ} \uplus \{ U =^? db(V,x),\ U =^? nil \}}{\mathcal{EQ} \cup \{ U =^? nil,\ V =^? nil \}}$$

(DB1.b) *Nil solution-2 for db*::

$$\frac{\mathcal{EQ} \uplus \{ U =^? db(V,x),\ V =^? nil \}}{\mathcal{EQ} \cup \{ U =^? nil,\ V =^? nil \}}$$

(DB1.c) *Nil solution-3 for db*::

$$\frac{\mathcal{EQ} \uplus \{ U =^? db(V,x) \}}{\mathcal{EQ} \cup \{ U =^? nil,\ V =^? nil \}} \qquad \text{if } V >^*_{db} U$$

(DB2) *Left-Cancellation on db*::

$$\frac{\mathcal{EQ} \uplus \{ U =^? db(V,x),\ U =^? db(V,y) \}}{\mathcal{EQ} \cup \{ U =^? db(V,y),\ x =^? y \}} \quad \text{if } U \in \mathbf{nonnil}$$

(DB3.a) *Push db below cons, at a* **nonnil** *db/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{ U =^? db(V,x),\ U =^? db(W,y) \}}{\begin{array}{l}\mathcal{EQ} \cup \{ V =^? cons(v,V'),\ W =^? cons(w,W'),\ U =^? cons(u,U'),\\ \qquad U' =^? db(V',v),\ U' =^? db(W',w),\ u =^? g(v,x),\ u =^? g(w,y) \}\end{array}}$$

if $U \in$ **nonnil**

(DB3.b) *Push bc and db below cons at a* **nonnil** *bc/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{ U =^? bc(V,x),\ U =^? db(W,y) \}}{\begin{array}{l}\mathcal{EQ} \cup \{ V =^? cons(v,V'),\ W =^? cons(w,W'),\ U =^? cons(u,U'),\\ \qquad U' =^? bc(V',u),\ U' =^? db(W',w),\ u =^? h(v,x),\ w =^? h(u,y) \}\end{array}}$$

if $U \in$ **nonnil**

(DB4) *Splitting for db at a cons/db-peak*::

$$\frac{\mathcal{EQ} \uplus \{ U =^? cons(x,U_1),\ U =^? db(V,z) \}}{\mathcal{EQ} \cup \{ U =^? cons(x,U_1),\ x =^? g(y,z),\ U_1 =^? db(V_1,y),\ V =^? cons(y,V_1) \}}$$

(DB5) *Flip db to bc conditionally:* :

$$\frac{\mathcal{EQ} \uplus \{U =^? db(V,x)\}}{\mathcal{EQ} \cup \{V =^? bc(U,x)\}} \quad \text{if } V >^+_c U, \text{ and } V \not>^*_{db} U$$

Rules (DB3.a), (DB3.b), (DB4) and (DB5) have the lowest priority: they are to be applied in the "laziest" fashion. The rule (DB3.b) ("*Push bc and db below cons... if* **nonnil**") is justified by the conditional left-cancellativity of *db* (cf. Lemma F, *Appendix-2*). Rule (DB5) is actually a 'narrowing' step, justified by the fact that *db* 'is a left-inverse' for *bc*.

For the completeness of the procedure, we shall also need a few more list inference rules which are "don't-know" nondeterministic; namely, the rules (DB6.a)–(DB8) below:

(DB6.a) *Guess a Nil-solution-Branch for db at a db/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{U =^? db(V,x),\ U =^? db(W,y)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil,\ W =^? nil\}}$$

(DB6.b) *Guess a Nil-solution-Branch for bc and db at a bc/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? db(W,y)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil,\ W =^? nil\}}$$

(DB7.a) *Guess a Narrowing step for db at a db/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{U =^? db(V,x),\ U =^? db(W,y)\}}{\mathcal{EQ} \cup \{V =^? bc(U,x),\ U =^? db(W,y)\}} \quad \text{if } V \not\succ^*_{db} U$$

(DB7.b) *Guess a Narrowing step for db at a bc/db-peak* ::

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? db(W,y)\}}{\mathcal{EQ} \cup \{U =^? bc(V,x),\ W =^? bc(V,y)\}} \quad \text{if } W \not\succ^*_{db} U$$

(DB8) *Standard Unification on db*::

$$\frac{\mathcal{EQ} \uplus \{U =^? db(V,x),\ U =^? db(W,y)\}}{\mathcal{EQ} \cup \{U =^? db(W,y),\ V =^? W,\ x =^? y\}}$$

We denote by $\mathcal{INF}''_l$ the inference system that extends $\mathcal{INF}'_l$ with the list-inference rules (DB1)–(DB8), given above. It is important to note that the Occur-Check Violation rule (L6) is henceforth to be applied to $\mathcal{DBC}$-unification problems in standard form, under the partial relation $\succ_l$ *as has been refined above.*

Proposition 5.3. *Let $\mathcal{P}$ be any $\mathcal{DBC}$-unification problem, given in standard form. The inference system $\mathcal{INF}''_l$ terminates on $\mathcal{P}$ in polynomially many steps.*

Proof. This is an extension of Proposition 3.3, to the inference system $\mathcal{INF}''_l$. The proof of that earlier proposition can be carried over practically verbatim: we only have to show that the new inferences that might introduce fresh variables, namely the three rules (DB3.a), (DB3.b) and (DB4), cannot lead to a non-terminating chain of inferences. To ensure this, a first observation is that the relation $\sim_\beta$, which was used in the proof of Proposition 3.3, *has to be refined now* so as to take *also* into account the relation $\sim_{db}$, the symmetric closure of $>_{db}$, as follows:

- If $U \sim^*_{db} V$ then $U \sim_\beta V$.
- Let $U >_{cons} U'$ and $V >_{cons} V'$; then $U \sim_\beta V$ implies $U' \sim_\beta V'$.

A second observation is that these three rules which might introduce fresh variables remove a $\sim_{db}$-edge at some node U, and introduce a new $\sim_{db}$-edge at a node U' such that $U >_{cons} U'$; but the number of $\sim_\beta$-equivalence classes remains the same, by the same argument as

developed in the proof of Proposition 3.3. The other details of that earlier proof carry over verbatim. □

Given any $\mathcal{DBC}$-unification problem $\mathcal{P}$ in standard form, let $\mathcal{A}''$ denote the inference procedure based on the rules of $\mathcal{INF}''_l$, given above for its list-equations; we augment the procedure $\mathcal{A}''$ with any given complete procedure for solving the residual set of element-equations in the problem, when the list-inference rules of $\mathcal{INF}''_l$ are no longer applicable. We have then the following result:

Proposition 5.4. *The procedure $\mathcal{A}''$ is sound and complete for solving $\mathcal{DBC}$-unification problems given in standard form.*

Proof. The proof uses the same lines of reasoning as for Proposition 4.1. The procedure $\mathcal{A}''$ is sound, because to any solution of a problem derived under any of its inferences, corresponds a solution for the initial problem. The completeness of $\mathcal{A}''$ is again proved, for any given problem, by induction on the maximum number of inference steps needed for the termination of the procedure $\mathcal{A}''$ on the problem; and using case analysis when necessary, based on the "don't-know" inference rules (DB6.a)–(DB8) above, for such an analysis. We leave out the details, which are straightforward. □

Proposition 5.5. *Let $\mathcal{P}$ be a $\mathcal{DBC}$-unification problem in standard form, to which none of the inferences of $\mathcal{INF}''_l$ is applicable. Then its subset of list-equations with non-nil variables on the left-hand side is in d-solved form.*

Proof. This extends Proposition 3.6 to the inference system $\mathcal{INF}''_l$. Note that we just need to show the following: From any given node $[U]$ on any given connected component Γ of the Propagation graph G_l, there is an unambiguous, cycle-free, directed path to a well-determined end-node on Γ. Now, given that any directed arc on G_l is labeled with either $>_{cons}$, or $>_{bc}$, or $>_{db}$, there can be at most one outgoing arc from $[U]$: otherwise one of the inferences (DB2)–(DB8) would have been applicable; there can be no directed $\succ_l$-cycle either at $[U]$, otherwise the Occur-Check violation rule would have been applicable. Thus, the proof of that earlier proposition carries over, essentially verbatim. □

Proposition 5.6. *Unification modulo the theory $\mathcal{DBC}$ is NP-complete.*

Proof. Given any $\mathcal{DBC}$-unification problem $\mathcal{P}$, computing a standard form can be done in polynomial time (wrt the number of variables of $\mathcal{P}$); the same holds also for constructing the propagation graph for the standard form. Applying then the inference rules of $\mathcal{INF}''_l$ till termination, on this standard form, takes only polynomially many steps, by Proposition 5.3. In case of non-failure, extracting the set of element-equations from the resulting problem can obviously be done in polynomial time.

To show that solving $\mathcal{P}$ is in NP, it suffices therefore to show that the set of its element-equations can be solved, modulo the theory defined by the single equation $g(h(x,y),y) = x$, in nondeterministic polynomial time. But this is a *collapsing* convergent system, and the unification problem for such theories is known to be decidable and finitary [13, 18]. In particular, a decision procedure can be built by using basic normalized narrowing, e.g., as given in [5]; cf. also [17]. We outline, briefly, such a procedure:

Procedure for Solving $\mathcal{E}(\mathcal{P})$: Note that every equation in $\mathcal{E}(\mathcal{P})$ is either a g-equation, i.e., an equation of the form $u =^? g(x,v)$; or an h-equation, of the form $u =^? h(x,y)$.

1. IF the set of element-equations is in d-solved form, then return that set;
   ELSE if the set contains g-equations, then go to Step 2; ELSE go to Step 3.
2. Choose nondeterministically an equation in $\mathcal{E}(\mathcal{P})$ of the form $u =^? g(x, v)$; and replace it by the h-equation $x =^? h(u, v)$.
3. If $\mathcal{E}(\mathcal{P})$ contains two different h-equations with the same lhs variable, apply standard decomposition below h on these two; and suppress one of the two equations.
4. Apply (element-)Variable Elimination to the resulting set of element-equations, if needed.
5. Go to Step 1.

(Note that Step 2 is just narrowing.) It is easy to check that this procedure is in NP on the size of $\mathcal{E}(\mathcal{P})$.

It remains to show that solving a general $\mathcal{DBC}$-unification problem is NP-hard. This follows from our Proposition 5.7 below, where we actually make a more precise statement. □

Proposition 5.7. *Unifiability modulo $g(h(x, y), y) = x$ is NP-complete.*

Proof. (cf. also [4].) We need only to prove the NP lower bound; we do that by reduction from the *Monotone 1-in-3 SAT* problem, formulated as follows:

> Given a propositional formula in CNF *without negation* such that every clause has exactly 3 literals (variables), check for its satisfiability under the condition that *exactly* one literal in each clause should evaluate to true.

This problem is known to be NP-complete [20].

Now consider the following problem of unification modulo $g(h(x, y), y) = x$, involving 3 element-variables x_1, x_2, x_3:

$$g(h(g(h(g(h(a, b), x_1), b), x_2), b), x_3) =^? g(h(a, b), c)$$

where a, b, c are ground constants.

Since $g(h(x, y), y) \to x$ is a convergent rewrite system, the unifiability problem is equivalent to finding an instance of the equation under an irreducible substitution such that both sides can be reduced to the same term. But the right-hand side term $g(h(a, b), c)$ is irreducible modulo $g(h(x, y), y) \to x$; so we need to eliminate two g symbols from the left-hand side term $g(h(g(h(g(h(a, b), x_1), b), x_2), b), x_3)$. The only way to do that is by assigning b to two of the variables, and then reduce using the rule $g(h(x, y), y) \to x$. We easily check that we obtain the following possible results: $g(h(a, b), x_1)$, $g(h(a, b), x_2)$, $g(h(a, b), x_3)$. If we assign the third 'left-out' variable – let us call it x – to b, the term obtained $g(h(a, b), b)$ would reduce to a, which is irreducible and different from $g(h(a, b), c)$. If we assign this left-out variable to some irreducible term t different from b and c, then $g(h(a, b), t)$ would be irreducible, again different from $g(h(a, b), c)$. Hence, the only way to reduce both sides of the given problem to become equal, is to assign c to the left-out variable. In other words: solving this problem amounts to assigning the term c to exactly one of the three variables x_1, x_2, x_3, and assigning b to the other two.

Now let us consider a (finite) set of clauses, each with three positive literals. To each clause $L_1 \vee L_2 \vee L_3$ in this set, we associate 3 element-variables x_1, x_2, x_3, and the element-equation $g(h(g(h(g(h(a, b), x_1), b), x_2), b), x_3) =^? g(h(a, b), c)$ on these variables. From the discussion above, the system of derived equations has a solution modulo $g(h(x, y), y) = x$ if and only if the set of clauses is 1-in-3 satisfiable. □

Remark 5.8. (i) It can be shown that $\mathcal{DBC}$-unification is finitary, along the same lines of reasoning as for the proof of Proposition 4.3.

(ii) The inference rules (DB5), (DB7.a) and (DB7.b) of $\mathcal{INF}''_l$ – which are justified by the last equation of $\mathcal{DBC}$ – play the role of reducing unification modulo $\mathcal{DBC}$, in fine, to unification modulo $\mathcal{BC}$.

Example 5.9. (i) The following problem: $U =^? db(V, x)$, $V =^? cons(y, W)$, $W =^? bc(U, z)$ is unsatisfiable. Our procedure exits with failure: we have an Occur-Check Violation: $U >_{db} V >_{cons} W >_{bc} U$.

(ii) The following problem $\mathcal{P}$ is in standard form:

$$U =^? db(V, y),\ U =^? cons(x, U_1),\ V =^? cons(y, V_1)$$

We have a $cons/db$-peak at $[U]$ on the graph of $\mathcal{P}$, and the only "don't-care" rule applicable is the Splitting rule (DB4); we can use the equation $V =^? cons(y, V_1)$ for that splitting. After cancellation on *cons* and a variable elimination step, the problem derived is:

$$U =^? cons(x, U_1),\ x =^? g(y, y),\ U_1 =^? db(V_1, y),\ V =^? cons(y, V_1)$$

which is in d-solved form, and gives a solution. □

Example 5.10. (i) The following problem: $U =^? db(V, y),\ V =^? db(U, z)$ is in standard form, but is not in a *d*-solved form. Rule (DB1.c) is applicable, and gives the "nil" solution to U and V, with y, z arbitrary.

(ii) The following problem $\mathcal{P}$ is in standard form: $U =^? bc(V, x),\ V =^? db(U, y)$, but not in a d-solved form; the only applicable inference rule is (DB5) (*Flip db to bc* conditionally), and the problem becomes:

$$U =^? bc(V, x), \qquad U =^? bc(V, y)$$

This is a $\mathcal{BC}$-unification problem which is L-reduced, but not in a d-solved form. None of the list-variables U, V is in **nonnil**; so, an obvious easy solution is $U := nil$, $V := nil$, the element-variables x, y being arbitrary; this corresponds to applying rule (L8). We could also nondeterministically apply the rule (L10) (*Standard unification on bc*); to deduce then the most general solution solution, namely: $U := bc(V, x), x := y$. □

Example 5.11. The following problem $\mathcal{P}$ is in standard (but not in a d-solved) form:

$$U =^? bc(V, x),\ V =^? db(W, y),\ W =^? db(T, z),\ T =^? bc(U, t),\ U =^? cons(u, U_1)$$

Observe that $T >_c^+ W$ but $T \not>_{db}^* W$, so the rule (DB5) (*Flip db to bc conditionally*) is applicable to the equation on W; and that gives:

$$U =^? bc(V, x),\ V =^? db(W, y),\ T =^? bc(W, z),\ T =^? bc(U, t),\ U =^? cons(u, U_1)$$

The problem now presents a bc/bc-peak at T which is in **nonnil**, so rule (L4.b) can be applied, by writing $W =^? cons(w, W_1)$; this, followed by Cancellation on *cons*, and a Standard unification step on h, leads us to deduce: $w =^? u, t =^? z, W_1 =^? U_1$, and subsequently $W =^? U$; the problem is thus transformed (after some Variable Elimination steps) into:

$$U =^? bc(V, x),\ V =^? db(U, y),\ T =^? bc(U, z),\ U =^? cons(u, U_1),\ W =^? U,\ t =^? z$$

The rule (DB5) (*Flip db to bc conditionally*) is again applicable, now to the equation on V; we thus get:

$$U =^? bc(V, x),\ U =^? bc(V, y),\ T =^? bc(U, z),\ U =^? cons(u, U_1),\ W =^? U,\ t =^? z$$

The rule (L4.a) (*Semi-Cancellation on bc at a bc/bc-peak*) is now applicable, and we deduce: $y =^? x$; after Variable Elimination, the problem transforms to:

$$U =^? bc(V, x),\ T =^? bc(U, z),\ U =^? cons(u, U_1),\ W =^? U,\ y =^? x,\ t =^? z$$

which presents a *cons*/*bc*-peak on U, so the Splitting rule (L5) is applicable; we write $V =^? cons(v, V_1)$, and the problem evolves (after Variable Elimination) to:

$$U =^? cons(u, U_1),\ V =^? cons(v, V_1),\ U_1 =^? bc(V_1, h(v, x)),\ T =^? bc(U, z),\ W =^? U,$$
$$u =^? h(v, x),\ y =^? x,\ t =^? z$$

The list-equations, as well as the element-equations, are now in d-solved form; and they do give a solution to the problem we started with (as can be easily checked). □

6. Conclusion

We first addressed the unification problem modulo a convergent 2-sorted rewrite system $\mathcal{BC}$, that models, in particular, the (usual, XOR-based) CBC encryption mode of cryptography, by interpreting suitably the function h in $\mathcal{BC}$. A procedure is given for deciding unification modulo $\mathcal{BC}$, which has been shown to be sound and complete (and finitary) when h is either uninterpreted, or interpreted in such a manner. In the uninterpreted case, the procedure is a combination of the inference procedure $\mathcal{INF}_l'$ presented in this paper, with syntactic unification; it turns out to be of polynomial complexity, essentially for this reason. In the case where h is interpreted as mentioned above, the unification procedure is a combination of $\mathcal{INF}_l'$ with any complete procedure for deciding unification modulo the associative-commutative theory for XOR; and it turns out to be NP-complete for this reason. The second part of the work extends $\mathcal{BC}$ into a theory $\mathcal{DBC}$ that models, at an abstract level, a cipher-decipher block chaining scheme. Unifiability modulo $\mathcal{DBC}$ is shown to be decidable by an inference procedure, which essentially 'reduces' any $\mathcal{DBC}$-unification problem in fine into one over $\mathcal{BC}$. Unification modulo $\mathcal{DBC}$ is also (finitary and) NP-complete.

A point that seems worth mentioning here concerns the binary function symbol *cons* in $\mathcal{DBC}$. We have implicitly assumed that in practical situations (such as in Example 2 above) the two arguments of *cons* are 'accessible'; this can be made more explicit by adding two 'projection' equations to $\mathcal{DBC}$, using *car* and *cdr* on *cons*, to get the following set of 8

equations:

$$
\begin{aligned}
car(cons(x, Y)) &= x && (6.1)\\
cdr(cons(x, Y)) &= Y && (6.2)\\
bc(nil,\ z) &= nil && (6.3)\\
bc(cons(x, Y),\ z) &= cons(h(x, z),\ bc(Y,\ h(x, z))) && (6.4)\\
g(h(x, y),\ y) &= x && (6.5)\\
db(nil,\ z) &= nil && (6.6)\\
db(cons(x, Y),\ z) &= cons(g(x, z),\ db(Y,\ x)) && (6.7)\\
db(bc(X, y),\ y) &= X && (6.8)
\end{aligned}
$$

with *car* typed as $\tau_l \to \tau_e$, and *cdr* as $\tau_l \to \tau_l$. All these equations can be oriented left-to-right under a suitable simplification ordering, and the resulting rewrite system remains convergent. It is not difficult to check that, even after the addition of these two projection rules, unification problems – with some very minor restrictions on the form of equations involving *car* and *cdr* – can still be assumed in a standard form, and solved by the inference procedure $\mathcal{INF}''_l$ given above. In other words, the results of Section 5 remain valid for this enlarged 2-sorted convergent rewrite system – that we shall again refer to as $\mathcal{DBC}$, since no confusion seems likely.

The rewrite system $\mathcal{DBC}$ thus enlarged can actually been shown to be Δ-strong in the sense of [3], under a suitable precedence based (lpo- or rpo- like) simplification ordering, by taking Δ to be the subsystem formed of the two rules (6.1) and (6.2). It would then follow from Proposition 11 of [3], that the so-called 'passive deduction' problem, for an intruder, is decidable, if the intruder capabilities are modeled by this theory $\mathcal{DBC}$. This would yield, to our knowledge, the first purely rewrite/unification based approach for analyzing cryptographic protocols employing the CBC encryption mode. The details will be given elsewhere, where we also hope to present decision procedures for a couple of other security problems, where an intruder eavesdrops or guesses some low-entropy data in the context of block ciphers.

Finally, observe that unification modulo equational theories often serves as an auxiliary procedure in several formal protocol analysis tools, such as Maude-NPA, CL-Atse, ..., for handling algebraic properties of cryptoprimitives. The work we have presented in this paper could be of use in these tools, as a first step towards the automation of attack detection in cryptographic protocols employing CBC.

References

[1] M. Abadi, V. Cortier. "Deciding Knowledge in Security Protocols Under Equational Theories". *Theoretical Comp. Science* 367(1-2):2–32, 2006.

[2] S. Anantharaman, C. Bouchard, P. Narendran, M. Rusinowitch. "Unification modulo Chaining". In *Proc. of 6th Int. Conference on Language and Automata Theory and Applications - LATA 2012*, LNCS 7183, pp. 70–82, Springer-Verlag, 2012.

[3] S. Anantharaman, P. Narendran, M. Rusinowitch. "Intruders with Caps". In *Proc. of the Int. Conference RTA'07*, LNCS 4533, pp. 20–35, Springer-Verlag, 2007.

[4] S. Anantharaman, H. Lin, C. Lynch, P. Narendran, M. Rusinowitch. "Unification modulo Homomorphic Encryption". *Journal of Automated Reasoning* 48(2):135–158 (2012)

[5] F. Baader, W. Snyder. "Unification Theory". In *Handbook of Automated Reasoning*, pp. 440–526, Elsevier Sc. Publishers B.V., 2001.

[6] M. Bellare, R. Guérin, P. Rogaway. "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Function" In *Proc. of the Int. Conference* CRYPT0 '95, LNCS 963, pp. 15–28, Springer-Verlag, 1995

[7] M. Baudet. "Deciding security of protocols against off-line guessing attacks". In *Proc. of the* 12*th ACM Conf. on Computer and Comm. Security*, CCS'05, pp. 16–25, 2005.

[8] H. Comon-Lundh, R. Treinen. "Easy Intruder Deductions." Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday (N. Dershowitz, ed.). In *LNCS* 2772, pp. 225–242, Springer-Verlag, 2003.

[9] H. Comon-Lundh, V. Shmatikov. "Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive-Or." In *Proc. of the Logic In Computer Science Conference, LICS'03,* pp. 271–280, 2003.

[10] N. Dershowitz. "Termination of Rewriting." *Journal of Symbolic Computation* 3(1/2): 69–116 (1987).

[11] D. Dolev, S. Even, R. Karp, "On the Security of Ping-Pong Protocols". *Information and Control* 55:57-68 (1982).

[12] Q. Guo, P. Narendran, D.A. Wolfram. "Unification and Matching Modulo Nilpotence." In *Proc. of the 13th Int. Conf. on Automated Deduction,* (CADE-13), LNCS 1104, pp. 261–274, Springer, 1996.

[13] J.-M. Hullot. "Canonical forms and Unification." In *Proc. of the 5th Int. Conf. on Automated Deduction,* (CADE-5), LNCS 87, pp. 318–334, Springer, July 1980.

[14] J.-P. Jouannaud, and C. Kirchner. "Solving Equations in Abstract Algebras: a Rule-Based Survey of Unification." In *Computational Logic: Essays in Honor of Alan Robinson,* 360–394, MIT Press, Boston, 1991.

[15] P. C. Kanellakis, and P. Z. Revesz. "On the Relationship of Congruence Closure and Unification." *J. Symbolic Computation* 7: 427-444 (1989).

[16] C. Lynch, Z. Liu, "Efficient General Unification for XOR with Homomorphism." In em Proc. of the 23rd Int. Conference on Automated Seduction, (CADE-23), LNCS 6803, pp. 407–421, Springer-Verlag, 2011.

[17] C. Lynch, B. Morawska, "Basic Syntactic Mutation." In em Proc. of the 18th Int. Conference on Automated Deduction, (CADE-18), LNAI 2392, pp. 471–485, Springer-Verlag, 2002.

[18] J. Millen, H.-P. Ko. "Narrowing Terminates for Encryption." In *Proc. of the Ninth IEEE Computer Security Foundations Workshop (CSFW)*, pp. 39–44, 1996.

[19] K. G. Paterson, T. Ristenpart, T. Shrimpton. "Tag Size *Does* Matter: Attacks and Proofs for the TLS Record Protocol" In *Proc. of Int. Conference* ASIACRYPT 2011, LNCS 2073, pp. 372–389, Springer-Verlag, 2011.

[20] T. J. Schaefer. "The complexity of satisfiability problems." In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pp. 216–226, 1978.

Appendix-1: On the Cancellativity properties of bc, g and db

Lemma A. For all terms T_1, T_2, t, we have:

$$bc(T_1, t) \approx_{\mathcal{BC}} bc(T_2, t) \quad \text{if and only if} \quad T_1 \approx_{\mathcal{BC}} T_2.$$

Proof. The proof is by structural induction on the terms, based on the semi-cancellativity of h and the cancellativity of $cons$. If either T_1 or T_2 is nil, then the other has to be nil too, and the assertion of the Lemma is trivial. So suppose that T_1 and T_2 are not nil. Then $T_1 = cons(u_1, T_1')$ and $T_2 = cons(u_2, T_2')$, for some terms u_1, u_2, T_1', T_2'. Substituting back into the original equation and applying the second axiom of $\mathcal{BC}$, we deduce that:

$$cons(\, h(u_1, t), bc(T_1', h(u_1, t))\,) \approx_{\mathcal{BC}} cons(\, h(u_2, t), bc(T_2', h(u_2, t))\,)$$

Since $cons$ is cancellative, we get:

$$h(u_1, t) \approx_{\mathcal{BC}} h(u_2, t), \quad \text{and} \quad bc(T_1', h(u_1, t)) \approx_{\mathcal{BC}} bc(T_2', h(u_2, t)).$$

From the semi-cancellativity of h, we then deduce that:

$$u_1 \approx_{\mathcal{BC}} u_2, \quad \text{and} \quad bc(T_1', h(u_1, t)) \approx_{\mathcal{BC}} bc(T_2', h(u_1, t)).$$

Therefore, by structural induction, we deduce that $T_1' \approx_{\mathcal{BC}} T_2'$, and the result follows. □

Lemma B. For all terms T, t_1, t_2, we have:

$$bc(T, t_1) \approx_{\mathcal{BC}} bc(T, t_2) \quad \text{if and only if} \quad T \approx_{\mathcal{BC}} nil \quad \text{or} \quad t_1 \approx_{\mathcal{BC}} t_2.$$

Proof. The proof is by exactly the same reasonings as for proving the previous lemma. □

We shall paraphrase these two lemmas together by saying that bc is "conditionally" semi-cancellative.

Lemma C. For all terms $u_1, T_1, u_2, T_2, u_3, u_4$: If $bc(cons(u_1, T_1), u_3) \approx_{\mathcal{BC}} bc(cons(u_2, T_2), u_4)$ then $h(u_1, u_3) \approx_{\mathcal{BC}} h(u_2, u_4)$ and $T_1 \approx_{\mathcal{BC}} T_2$.

Proof. By applying the second axiom of $\mathcal{BC}$, we get:

$$cons(\, h(u_1, u_3), bc(T_1, h(u_1, u_3))\,) \approx_{\mathcal{BC}} cons(\, h(u_2, u_4), bc(T_2, h(u_2, u_4))\,)$$

Cancellation on $cons$ gives:

$$h(u_1, u_3) \approx_{\mathcal{BC}} h(u_2, u_4) \quad \text{and} \quad bc(T_1, h(u_1, u_3)) \approx_{\mathcal{BC}} bc(T_2, h(u_2, u_4))$$

By Lemma A above, this implies that $T_1 \approx_{\mathcal{BC}} T_2$. □

In what follows, by $\mathcal{DBC}$ we shall mean the equational theory $\mathcal{DBC}$ of Section 5, and the rewrite system it defines.

As for the analogs of the above results for the operator db of $\mathcal{DBC}$, we first observe that the function g is not semi-cancellative – more precisely, it is not right-cancellative: indeed, we have $g(h(g(t, u), u), u) =_{\mathcal{DBC}} g(t, u)$, although $h(g(t, u), u) \neq_{\mathcal{DBC}} t$, in general. But left-cancellativity holds for g.

Lemma D. If $g(s,t_1) =_{\mathcal{DBC}} g(s,t_2)$ then $t_1 =_{\mathcal{DBC}} t_2$.

Proof. We can assume wlog that the terms s, t_1, and t_2 are in normal form. If $t_1 \neq_{\mathcal{DBC}} t_2$, then both $g(s,t_1)$ and $g(s,t_2)$ must be redexes, or, in other words, $s = h(s',t_1) = h(s',t_2)$ for some s'. Since h is semi-cancellative this leads to a contradiction. □

Corollary E. If $g(s_1,t_1) =_{\mathcal{DBC}} g(s_2,t_2)$, and $t_1 \neq_{\mathcal{DBC}} t_2$, then $s_1 \neq_{\mathcal{DBC}} s_2$.

So, the analog of Lemma A for *db* does not hold in general. However, *db* is 'conditionally' left-cancellative:

Lemma F. For all terms T, x, y, we have:

$$db(T,x) \approx_{\mathcal{DBC}} db(T,y) \quad \text{if and only if} \quad T \approx_{\mathcal{DBC}} nil \quad \text{or} \quad x \approx_{\mathcal{DBC}} y.$$

Proof. We just need to prove the "only if" assertion. If T is not nil, then $T = cons(t,T_1)$ for some t, T_1. Applying the last axiom of $\mathcal{DBC}$, we get:

$$cons(g(t,x), db(T_1,t)) \approx_{\mathcal{DBC}} cons(g(t,y), db(T_1,t)).$$

The assertion follows then from the cancellativity of *cons* and the left-cancellativity of g. □

Appendix-2: *db* as inductive left-inverse for *bc*

Lemma G. Let $\mathcal{DBC}'$ be the convergent rewrite system formed of the first five rules in the system $\mathcal{DBC}$ of Section 5. For any list-term U and element-term x both in $\mathcal{DBC}'$-normal form, we have: $db(bc(U,x),x) =_{\mathcal{DBC}'} U$.

Proof. The proof is by structural induction on U. The base case when U is nil is trivial; so suppose $U = cons(u,U_1)$ for some element-term u, and list-term U_1. Substituting for U and using first the 2nd equational axiom of $\mathcal{DBC}'$, the left-hand side of the assertion becomes:

$$db(cons(h(u,x), bc(U_1, h(u,x)), x).$$

To which we can apply the 5th equational axiom of $\mathcal{DBC}'$ to get:

$$cons(g(h(u,x),x), db(bc(U_1,h(u,x)), h(u,x));$$

By applying now the 3rd axiom of $\mathcal{DBC}'$, and the induction hypothesis, this reduces (modulo $\mathcal{DBC}'$) to $cons(u,U_1)$, that is to say U. □