Limits of the Cryptographic Realization of Dolev-Yao-style XOR

Michael Backes and Birgit Pfitzmann

IBM Zurich Research Lab
{mbc,bpf}@zurich.ibm.com

Abstract. The abstraction of cryptographic operations by term algebras, called Dolev-Yao models, is essential in almost all tool-supported methods for proving security protocols. Recently significant progress was made in proving that such abstractions can be sound with respect to actual cryptographic realizations and security definitions. The strongest results show this in the sense of reactive simulatability/UC, a notion that essentially means retention of arbitrary security properties under arbitrary active attacks and in arbitrary protocol environments, with only small changes to both abstractions and natural implementations.
However, these results are so far restricted to cryptographic systems like encryption and signatures which essentially only have constructors and destructors, but no further algebraic properties. Typical modern tools and complexity results around Dolev-Yao models also allow more algebraic operations. The first such operation considered is typically XOR because of its clear structure and cryptographic usefulness. We show that it is impossible to extend the strong soundness results to XOR, at least not with remotely the same generality and naturalness as for the core cryptographic systems. On the positive side, we show the soundness of an XOR model and realization under passive attacks.

1 Introduction

Tool-supported verification of cryptographic protocols almost always relies on abstractions of cryptographic operations by term algebras, called Dolev-Yao models after the first authors [25]. The core of these term algebras are operations like en- and decryption which ideally have very few algebraic properties. However, if one wants to benefit from such abstractions in protocols that also contain operations with more algebraic properties, those operations have to be given a similar specification. A typical such operation is the exclusive or (XOR), see, e.g., [40, 18, 19], because it is commutative and associative and has significant uses in cryptology, e.g., as the one-time pad, in modes of operation of block ciphers, and in some protocols.

Recent work has essentially bridged the original and long-standing gap between Dolev-Yao models and real cryptographic definitions: It was shown that an almost normal Dolev-Yao model of several important cryptographic system types can be implemented with real cryptographic systems secure according to standard cryptographic definitions in a way that offers reactive (blackbox) simulatability [8]. This security notion means that one system (here the cryptographic realization) can be plugged into arbitrary protocols instead of another (here the Dolev-Yao model) and retains essentially arbitrary security properties; it is also called UC for its universal composition properties. Extensions of this simulatability result to more cryptographic primitives were presented in [9, 7] and actual uses in protocol proofs in [6, 5]. Earlier results considered passive attacks only [4, 3, 30]. Later papers [39, 33, 16] consider to what extent restrictions to weaker security properties and/or less general protocol classes allow simplifications compared with [8]. All these papers have in common that they only consider core cryptographic operations, not operations with additional algebraic properties like XOR.

In this paper we study how the soundness results in the sense of reactive simulatability/UC can be extended when an XOR is added to a Dolev-Yao model and its cryptographic implementation. It turns out that this is impossible in a general way. We are quite surprised by this result, because XOR seems a relatively simple operation compared with systems like digital signatures, and it seems well described by its algebraic properties. Note that the question is not whether an XOR is a good and generally usable encryption system by itself, but just whether algebraic abstractions of it are sound. The only positive result we show is a soundness result under passive attacks; apart from this restriction the result is strong in the sense of using reactive simulatability and allowing a broad range of other operations in the Dolev-Yao model. Although early papers on bridging the gap between Dolev-Yao models and cryptography were also for passive attacks only, typical overall Dolev-Yao attackers are active. We therefore regard our negative results for the active case as the more interesting ones.

We want to show that it is not possible to cryptographically realize a Dolev-Yao model that contains XOR together with other usual cryptographic operations via real systems with actual XORs, in the sense of reactive (blackbox) simulatability. This is a meta-theorem formulation on a very high level: There is no current definition of a Dolev-Yao model independent of specific system models (like CSP, π-calculus, IO automata etc.). Nor is "an actual XOR" really well-defined. We aim at coming as close as possible to this meta-theorem with precise statements, but in the end what we show is a series of concrete impossibility results, and it is a matter of taste whether one considers these results to demonstrate the informal meta-theorem. At a minimum, these concrete impossibility results show that soundness results for Dolev-Yao models with XOR cannot be achieved with remotely the same generality and naturalness as for the core cryptographic systems.

Related Work. The XOR operation has accompanied cryptography from its beginnings, from simple ciphers in ancient and medieval times, over the one-time pad and the work of Shannon, to its widespread use in modern cryptography where it constitutes an essential component in many cryptographic protocols, e.g., [29, 13, 46]. To the best of our knowledge, the XOR operation in symbolic analysis of cryptographic protocols has first been mentioned by Meadows [35] as a possible extension of the NRL analyzer, and has since then been incorporated in many formal proof tools, e.g., NRL [36], CAPSL [41], Isabelle [43], and OFMC [10]. Recent papers on XOR in Dolev-Yao models were mainly concerned with the decidability of the insecurity of cryptographic protocols against a Dolev-Yao attack in the presence of deduction rules for the XOR operator [18, 19]. There, it was particularly shown that protocol insecurity with XOR is in NP for a certain protocol class.

This line of work typically continues with abstractions of more general Abelian groups, e.g., [20, 22, 2], and the exponentiation function as used in many cryptographic systems based on the discrete-logarithm problem, e.g., [37, 28, 42, 21, 17, 47, 1]. While we have not yet considered these extensions, we are convinced that a general use of such operations on other terms would lead to similar problems as with XOR. In the case of exponentiations, however, it may be more realistic than for XOR to make strong restrictions on the types of terms that can be exponentiated and on the use of the XOR results within larger terms, and such restrictions might help.

The first sound formal abstraction of XOR, but only in connection with pseudorandom permutations, not the typical general encryptions etc. of other Dolev-Yao models, and only for passive attacks, was presented in [31, 32]. Another sound formal abstraction of XOR was recently presented in [11], but only if XOR is restricted to terms whose corresponding bitstrings are generated according to a uniform distribution, and only for passive attacks.

The security notion of reactive simulatability, a notion of secure implementation that allows arbitrary composition, was first defined generally in [44], based on simulatability definitions for secure (one-step) function evaluation [26, 27, 12, 38, 14]. It was extended in [45, 15], called UC (universal composability) in the latter, and has since been used in many ways for proving individual cryptographic systems and general theorems. While the definitions of [45, 15] have not been rigorously mapped, for the results in this papers the differences do not matter.

As stated, our results, except for a simple first one, only consider the soundness of Dolev-Yao models in the sense of reactive simulatability/UC. We do not exclude that weaker soundness notions such as integrity-only soundness as first investigated in [39] can be extended. Nor do we exclude that certain restricted protocol classes using XOR can be secure in the sense of reactive simulatability even if the Dolev-Yao model as such is not, a direction of work started (not for XOR yet, of course), in [16]. It will be interesting to investigate the precise limits in the future. Nevertheless, we believe that our results show that one cannot achieve the general secure pluggability of a Dolev-Yao realization for a Dolev-Yao model with XOR that has been achieved for core cryptographic operations.

Overview of this Paper. As mentioned above, our major results are negative results that aim at demonstrating the informal claim that it is not possible to realize "true Dolev-Yao models" by "real XORs" in a generally composable way. In the following, we summarize our concrete impossibility results. By payload data we denote the type of non-cryptographic data that most Dolev-Yao models have. It denotes data input by the users of the Dolev-Yao model, e.g., letters to be encrypted and signed, or payment data constructed by a payment protocol using the Dolev-Yao model.

- The standard Dolev-Yao model of XOR used in the literature is insecure with respect to every moderately natural implementation when secrecy is required (not necessarily even reactive simulatability) and general terms (e.g., payload data) can be XORed.
- If payload data are used in their original form in real XORs and signatures are one of the cryptographic operations, then every system that securely abstracts from this situation (in the sense of blackbox simulatability) must be able to compute signatures from whatever cryptographic realization is used. Thus informally it is not truly Dolev-Yao. More precisely, we present a reduction proof showing that such a system can be used to build a signature oracle with minimal additional operations.

- The same result holds with a more complex counterexample if we no longer assume that arbitrary usage of the Dolev-Yao model is allowed, but only assume that certain useful-looking protocols can be built on top of it.
- The same result holds even if the payload data may be encoded in the real system before being used in XORs, but with low or well-structured redundancy such as type tags. To the best of our knowledge, all current implementations of XOR fall into this class or the previous class.
- Even if payload data may be encoded with arbitrary redundancy, a similar result holds where the system that should be a Dolev-Yao model must at least be able to test signatures. I.e., we now make a reduction proof yielding a test oracle, a notion that we first have to define because it is not usual in cryptography.

The basic underlying problem in all these cases is when an honest participant receives an XOR from an active adversary and, after some local operations, tries to convert the result into another type, in particular into payload data, thinking, e.g., that the result is a recovered plaintext. There is no general consistent way for the cryptographic realization and the Dolev-Yao abstraction to know whether such a type conversion should work, and thus to make exactly the required output when the result is really a plaintext, but not when it is not a plaintext and may thus have some cryptographic structure in the real system.

Positively we show reactive simulatability of an extension of the Dolev-Yao-style system from [8] by an XOR with a certain tolerable imperfection (needed to overcome the first negative result) with respect to a natural realization, and under the condition that users are restricted to correct type conversions, a fact that can be verified formally for protocols.

2 A Solvable Secrecy Imperfection of XOR

Before going into real impossibility results, we present one imperfection of XOR that would make a sound XOR abstraction different from the XOR abstractions in the literature. However, this imperfection alone could be taken care of by giving the Dolev-Yao adversary additional capabilities, without leaving natural Dolev-Yao models and natural XOR implementations. Thus it is similar, e.g., to the fact that cryptographic encryption cannot keep the length of arbitrary terms secret, which led to the introduction of abstract lengths into the Dolev-Yao style model of [8].

As far as we know, all Dolev-Yao abstractions with XOR allow participants to XOR arbitrary terms and to convert (typically implicitly) a result that is a term of another type back to that type. For instance, a recipient who receives a one-time pad ciphertext $c = d \oplus k$, where d is a plaintext and k a key, may ask to have c XORed with k and to obtain the plaintext d as output. The adversary has no additional capabilities in these models. For instance, if he receives an XOR of two terms that he both doesn't know, and that both did not occur in other XORs, he cannot retrieve these terms. For instance, an adversary not knowing k and d in the example above cannot retrieve k or d.

Now assume that an honest participant XORs two plaintexts written in English and sends the result to the adversary. The result can be cryptanalyzed if the texts are long enough, i.e., a real adversary can retrieve the two plaintexts, e.g., see the section on running-key ciphers in [23]. Hence we must model that an XOR leaks the underlying terms to the adversary unless we know that at least one of these terms is sufficiently

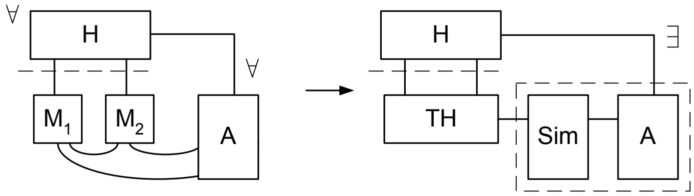

Fig. 1. Overview of blackbox simulatability. A real system is shown on the left; an ideal system plus simulator on the right. The views of H must be indistinguishable.

random. In this sense, prior Dolev-Yao models of XOR are overly optimistic. Even data types of significant entropy, like secret or public keys of public-key systems, are not sufficiently uniformly distributed given only the standard cryptographic definitions to guarantee that an XOR with them hides plaintext data or other cryptographic elements well, i.e., besides the entropy they may contain significant redundancy.

One can deal with this imperfection—and we will do so in the positive result for passive attacks—by introducing a set of random types into the Dolev-Yao model. Elements of a random type are deemed sufficiently random (often pseudo-random in reality) to restrict the adversary to standard algebraic operations on XORs. In the absence of unknown random elements in an XOR, the Dolev-Yao adversary is given the capability to parse the XOR.

3 Assumptions for Our Impossibility Results

As explained in the introduction, we want to show that it is not possible to implement *any* Dolev-Yao abstraction by *any* natural realization of XOR in a way that retains arbitrary security properties in arbitrary protocol environments. In order to turn this informal meta-theorem into concrete statements that can be verified or falsified, we need assumptions on what characterizes a Dolev-Yao model, a model of XOR in it, and a real implementation of such a model.

3.1 Reactive Simulatability

We start by surveying the notion of reactive simulatability/UC that we use for comparing a Dolev-Yao model and a cryptographic realization with respect to security. As we aim at general impossibility results, we try to avoid model-specific notation and to stay as general as possible.

Reactive simulatability is a general notion for comparing two systems, typically called real and ideal system. It relies on the notion of honest users (potential protocols) that interact with one of these systems and an adversary that mounts attacks against the system and its users. Essentially reactive simulatability states that for all attacks on the real system there exists an equivalent attack on the ideal system. More precisely, blackbox simulatability states that there exists a simulator that can use an arbitrary real adversary as a blackbox, such that arbitrary honest users cannot distinguish whether they interact with the real system and the real adversary, or with the ideal system and the simulator with its blackbox. This is illustrated in Figure 1. Here the machines M_1 and M_2 jointly denote the real system, TH (trusted host) denotes the ideal system, H

the entirety of the honest users, A the real adversary, and Sim the simulator. The combination of Sim and A is the adversary on the ideal system. The reader may regard the individual boxes as I/O automata, Turing machines, CSP or pi-calculus processes etc., whatever he or she is most familiar with. The only requirement on the underlying system model is that the notion of an execution of a system when run together with an honest user and an adversary is well-defined. The rigorous notion of equivalence of the attacks is that the honest users' views in such executions are computationally indistinguishable; this is a well-known cryptographic notion from [48].

3.2 Interface Behavior of a Dolev-Yao Model with XOR

In this section we describe the functionality that we assume a Dolev-Yao system with XOR offers. This is quite natural, but recall that there is no general definition of this in the literature yet. We also introduce some notation.

As we want to compare the Dolev-Yao model and its cryptographic realization in the sense of reactive simulatability/UC we can assume that they offer the same syntactic user interfaces, i.e., in- and output formats. In terms of Figure 1, this is the interface from TH or M_1 and M_2, respectively, to the entirety of users H. Similar concepts exist in all variants, in particular [44, 45, 15] and when extending the observational equivalence from [34] by simulators, e.g., the input and output formats of the ideal and real functionality in [15] and the free channel types in [34]. Syntactically different user interfaces would either simply prevent the same users from using the real and ideal systems or lead to trivial distinguishability.

We make the following assumptions:

- The honest users can ask for standard cryptographic terms to be constructed and sent to other users. In particulat, this includes nonces, XORs, encryptions, and signatures. We sometimes describe the commands used for this by writing down the desired result term in a quasi-algebraic notation.
- When an honest user receives a term, it gets notified by the system, and can then perform at least some cryptographic operations on this term. We sometimes designate an opaque term t in such a context by the handle notation t^{hnd}.
- The XOR operation, when considered with respect to the user inputs, fulfills the typical algebraic equations such as $(t_1 \oplus t_2) \oplus t_1 = t_2$ for all terms t_1, t_2. Note that we can require this independently of how the ideal system internally represents terms and how the real system encodes strings (e.g., with type tags or error-correcting codes).
- The users can input payload data and have them output again. They can use payload data as leaves in typical cryptographic terms (e.g., as a message to be encrypted; the examples will show where exactly we assume that payload data can be used).
- The Dolev-Yao model offers some secrecy. At a minimum, we require that it does not leak information about the payload in an encryption to the ideal adversary, except at most its term structure with types and lengths. Similarly, we require that an XOR with a fresh nonce of sufficient length does not leak information about the other XORed components to the ideal adversary, except at most their term structure and length.

Beyond these minimum assumptions, we will make more assumptions in our first examples, to get easy and natural cases, but we will relax those other assumptions later.

As discussed above, reactive simulatability/UC requies a common representation of terms at the user interface, i.e., some kind of names that designate either abstract terms or real bitstrings when a protocol uses either the ideal or the real cryptographic system. We call these names “handles” following [8]. Note, however, that we do not assume any specific implementation of these handles, in particular not that they are local names represented by successive natural numbers as in [8]. They could even be terms (with a notion of holes where the term cannot be parsed for this user) or cryptographic objects, although for many readers the latter choice would immediately disqualify the ideal system from counting as a Dolev-Yao model.

3.3 Characteristics of Dolev-Yao Models and Real Systems

Our next assumptions concern what constitutes an ideal, Dolev-Yao-style system, and what constitutes a real, cryptographic system. We obviously need such assumptions: The notion of reactive simulatability is reflexive. Thus, if an arbitrary Dolev-Yao model with XOR would also count as real, we would trivially have a secure realization of the system by itself. The same would hold if an arbitrary real cryptographic system with XOR would also count as ideal. But this is not what we want.

As we aim for impossibility results for large classes of Dolev-Yao models and realizations, we try to make only minimum requirements on ideality and reality. Note, however, that also a theorem that one particular ideal system, such as from the literature, cannot be realized, would be of some interest, and similarly for statements that specific cryptographic libraries with XOR cannot be idealized.

Dolev-Yao Models. The seemingly essential feature of Dolev-Yao models is that they work on term algebras, and not on real cryptographic bitstrings. However, this is not easy to formalize. For instance, given only a list or pairing operation, one can in principle construct terms that correspond to strings, and it may be possible to implement real cryptography on them as very complex term operations. Hence one cannot simply define the presence of terms and the absence of bitstrings as the characteristics of a Dolev-Yao model. Some other properties are easier to define, but not sufficient for our results. For instance, a Dolev-Yao model is expected to be deterministic and independent from concrete underlying cryptographic systems as long as they fulfill certain definitions. However, these properties do not help much to construct counterexamples because the simulator could give random values and algorithms to the ideal system during their first interaction. Another property, which we will use but which is not sufficient for the proof, is the secrecy of local operations. This means that during a term construction or the parsing of a received term and the potential output of some of its components to the honest user, no intermediate results are exchanged with the ideal adversary. All existing Dolev-Yao models fulfil this property since they treat the parsing of terms and the construction of response terms as monolithic transitions that do not grant the adversary access to the term structure or even the subterms.

We overcome this lack of a clear distinction of Dolev-Yao models by resorting to reduction proofs. We show essentially that every system (in particular a potential Dolev-Yao model) that can faithfully abstract from cryptography and XOR in certain situations, can be used to compute actual cryptographic operations. The reduction consists only of very few and simple operations. This intuitively means that the main work in computing the actual cryptographic operations must be done inside the system that

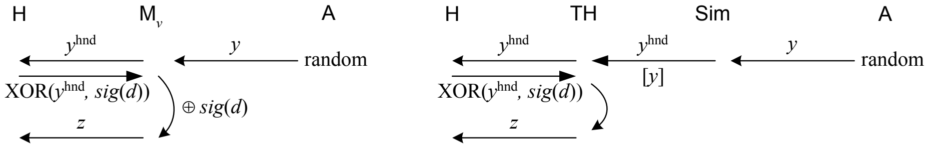

Fig. 2. Example for non-redundant data and arbitrary users.

is supposed to be a Dolev-Yao model. This leads us to conclude (again informally) that the system is not actually in Dolev-Yao style.

Real Systems. A general characteristics of real systems is that they are distributed. This means that each participant has its own machine, and that the machines are only connected by channels that offer a real adversary well-defined possibilities for observations and manipulation. Specifically for a cryptographic realization of a Dolev-Yao model, we require that the machines only exchange messages according to the user interface commands, i.e., when a term is exchanged between participants according to the protocol. Specifically for XOR we usually assume that the real system contains a real XOR on bitstrings. More precisely, we assume that message elements like signatures or keys have a representation that is fixed for their type, and a real XOR of such message elements is an actual bitstring XOR. In particular, payload data (see Section 3.2) may have an internal representation different from the external one, e.g., with a type tag, but then this fixed representation is used in all XORs. Whether the payload data are used in the XORs in their original form or in a redundant encoding is an important distinction. The first case leads to easier counterexamples and is quite natural, e.g., when one considers XORs in modes of operation like CBC, but we also consider the more complicated case.

4 Impossibility Results and their Consequences

This section contains our major results that aim at demonstrating the informal claim that it is not possible to realize true Dolev-Yao models by real XORs in a generally composable way. We first present counterexamples for simple cases, e.g., for non-redundant represenatations of payload data and arbitrary users. Then we move to more involved cases, e.g., to data of high redundancy and restricted protocol classes, that require more sophisticated reasoning to establish the unsoundness of Dolev-Yao-style XOR.

4.1 Counterexample for Non-redundant Data and Arbitrary Users

Our first example is made under the additional assumption that payload data are non-redundant in XORs, and that there is no restriction on the users H. The example is shown in Figure 2. The real situation is shown on the left, the attempt at a simulation on the right. The real adversary sends a random string y to the machine M_v of an honest participant v. The user v gets a notification that a string (or term in the ideal case) was received. The user v then asks to have this received string or term XORed with his or her signature on payload data d, and to have the result output as payload data. As there is no redundancy in payload data under the assumptions of this case, the machine M_v cannot recognize that the result is not payload data. Clearly, the user H and the adversary A together can validate in the real system that the bitstring $y \oplus z$ is a valid signature on d, assuming the public key pks for this signature is known. (Note that H and A are allowed to interact in the definitions of reactive simulatability, e.g., to model chosen-message attacks and plaintexts becoming public. Actually, in blackbox simulatability, one can simply join H and A.) Hence in the simulation TH must also output a bitstring z with this property. However, this intuitively means that TH can compute a cryptographic signature.

Reduction proof. More precisely, we show as a reduction proof that if this situation is simulated correctly, then TH can be used to compute signatures. Thus we construct a signature oracle Sig with TH as a blackbox, and where Sig only performs very few and simple operations itself in the signature computation phase. For its setup, Sig runs honest users and the simulator for the generation of a signature key pair and the publication of the public key pks. It publishes the same key. It further generates a random string y and the arrival indication that Sim would give to TH for such a message; this is denoted by y^{hnd} in Figure 2. It waits for TH to notify the user; in slight abuse of notation we again denote this with y^{hnd} because typically both these values are just some handles. Now Sig, when asked to sign a payload d, asks TH (as H would do) to XOR the term denoted by y^{hnd} with a signature on d, and to output the resulting payload data. It waits for the output z from TH and outputs $s := z \oplus y$. By the arguments above, this is indeed a signature on d valid with respect to pks. Note that in the signing phase, the only operations that Sig performs itself are the final XOR and the earlier user inputs requesting the computation of a signature and an XOR and the output of payload data.

This proof seems a clear indication that this TH can compute a signature on an arbitrary message.

4.2 Counterexamples for Non-redundant Data and Protocol Restrictions

If every permitted user H consists of a protocol $prot$ from a restricted class $Prots$ and a user H' of $prot$, then the example from Section 4.1 only still works if a permitted protocol subtracts a signature from a received string and converts the result into payload data. The simplest protocol that does this is written as follows in typical high-level notation, and where d and sks are secrets known to u and v.

$$\begin{aligned} u \to v &: \ d' \oplus sig(sks, d); \\ v &: \ \text{Output } d'. \end{aligned}$$

This protocol may or may not belong to the class $Prots$ of typical Dolev-Yao models extended by XOR. For instance, the models might not allow joint secret payload data and a joint secret key. If this simple protocol is in the class, then Figure 2 remains a counterexample, even if we know that this protocol is the only use made of the ideal or real cryptographic system: In the real system, M_v cannot know whether M_u started this protocol. Hence whenever it gets a message y supposedly from M_u, it applies its protocol step and thus acts as in Figure 2.

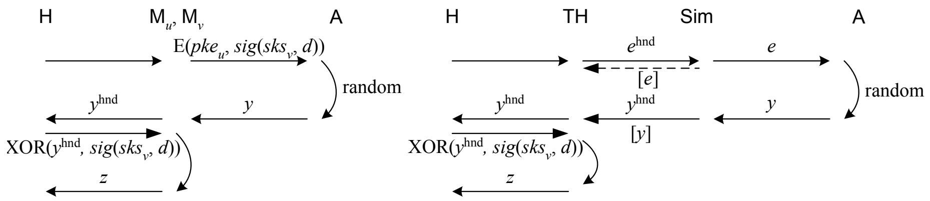

Fig. 3. Example for non-redundant data and protocol restrictions.

Otherwise, the following protocol is more likely to be permitted for arbitrary Dolev-Yao models.

$$\begin{aligned} v \to u &: \ E(pke_u, sig(sks_v, d)); \\ u \to v &: \ d' \oplus sig(sks_v, d); \\ v &: \ \text{Output } d'. \end{aligned}$$

Here pke_u denotes the public encryption key of user u and sks_v the secret signing key of user v. By our assumptions on the secrecy offered by Dolev-Yao models, the first term does not leak the encrypted plaintext to the ideal adversary, in particular neither d nor the signature. I.e., as long as TH and Sim are independent of the protocols run on top, TH only outputs an opaque handle e^{hnd} to the ideal adversary to indicate the first term sent, see Figure 3. Then when M_v receives the second message y while M_u did not really send it, the situation is essentially as in Section 4.1: M_v does not know whether M_u sent the second term, so it reacts on any y supposedly from M_u. It always outputs a result z. This result is passed through to the protocol user H'. So even H' and A alone (without observing the inner workings of the protocol $prot$) can obtain $y \oplus z$, which should be a valid signature on d, which they also know.

4.3 Counterexample for Data with Low Redundancy

The situations studied so far no longer serve as counterexamples for implementability of a Dolev-Yao XOR if we restrict the domain of payload data. Then M_v might usually output an error $\downarrow$ instead of z. For this, the real system must contain an explicit conversion between input data d and their redundant representation within the realization of terms. We sometimes omit this conversion, but sometimes write it out as data2string (with input pure payloads, and output the redundant encoding) and string2data. We write the domain of string2data as $Datastrings$ and its restriction to strings of length l as $Datastrings_l$.

A common case is that a realization implements a type system by tagging, i.e., by representing all data as pairs of a type tag and the original data, and that the original payload data can be an arbitrary string from an exponentially large domain. If the type tag in such a realization is short, the counterexample from Figure 2 still works. More generally, we assume that the family of sets $Datastrings_l$ is not negligible within the family of sets of all strings of length l. The string z in the example is uniformly random (a fresh one-time pad y XORed with a fixed string) and thus with not negligible probability we are in the situation as above, which cannot be simulated.

In some real implementations of type systems on strings, in particular XML, the overall part of a string that is fixed by a type is of considerable length, so that the use of

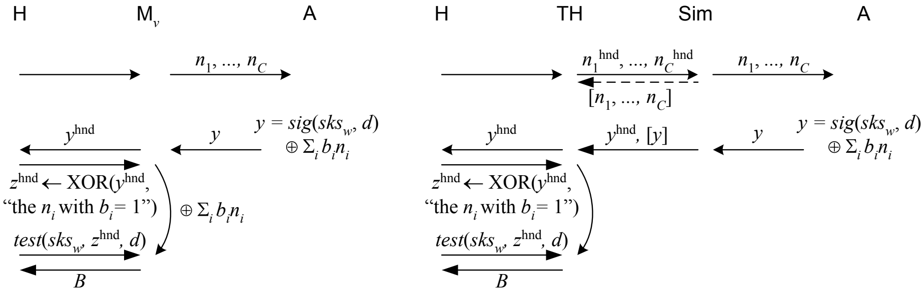

Fig. 4. Example for potentially highly redundant data.

a random y no longer works. However, a similar attack works for many realistic cases: Assume that a subset $Fixbits_{\mathsf{data},l}$ of the bits of $Datastrings_l$ is fixed (e.g., opening and closing XML tags), and similarly $Fixbits_{\mathsf{sig},l}$ for signatures. We can increase the latter set by only considering signatures made with one known algorithm and with respect to the known public key pks_v. Now if $Fixbits_{\mathsf{sig},l} \supseteq Fixbits_{\mathsf{data},l}$, the adversary can predetermine the necessary bits of z in y by XORing them with the corresponding fixed bits of a signature.

4.4 Counterexample for Highly Redundant Data and Arbitrary Users

Before delving into slightly exotic reduction proofs, let us consider one relatively normal case although it does not correspond to a specific underlying assumption: If the redundancy is of a complex form such that Sim cannot parse bitstrings with the structure $y = d' \oplus sig(d)$ where both d' and d are new, and if TH only expects from the ideal adversary and thus from Sim an opaque indication that an XOR has been received (in particular not the string y), and if the user later subtracts d' and requests a signature test on the result, then TH can obviously not react correctly, and it is too late to contact Sim because of the secrecy of local operations.

Now we return to the case without additional assumptions, i.e., Sim is arbitrary and may pass arbitrary strings to TH. We use the example from Figure 4. Participant v and the adversary share a large number C of nonces $n_1, \ldots, n_C$ of a certain length l suitable for hiding signatures. Here C is chosen such that the nonces span the whole space $\mathsf{GF}(2)^l$. This can be tested on the fly by A, and C be adapted accordingly. Now the adversary chooses a random vector $b \xleftarrow{\mathcal{R}} \mathsf{GF}(2)^C$ and computes the linear combination $N := \sum_{i=1}^{C} b_i n_i$ of the nonces; this is as good as a fresh nonce. The adversary also computes a signature s on data d with a secret key sks_w where the public key pks_w was published, and sends s hidden by the new nonce as a message $y = s \oplus N$ to participant v. Outside the system, A shares d and b with user v, represented in the entirety of users H, corresponding to a chosen-message attack. Then user v gives the commands to subtract the appropriate nonces from y and asks whether the result z is a correct signature on d. Alternatively, the adversary does the same with an incorrect signature s'. The real machine M_v will always decide this correctly. We now assume that we have an ideal system TH and a simulator Sim that correctly simulates the same situation. Intuitively, we want to show that TH must be able to test cryptographic signatures for this. Formally, we derive a test oracle Test from TH that only performs very simple, algorithm-independent operations outside its subroutine TH during the signature test phase. Intuitively, this shows that after a set-up phase, TH has learned the test algorithm.

While signature oracles are usual in the literature, test oracles are not. When trying to define one, we need to decide on an input distribution for the potential signatures. For instance, if we input random values, the oracle may be correct with overwhelming probability by always outputting false. Or, if we input either a correct signature or a random value, there may be so much trivial redundancy in the real signatures that a very simple oracle can make the distinction. We deal with this problem as follows: We allow a second, arbitrary oracle Fake that tries to fake signatures. It must always output wrong signatures. Intuitively a good oracle Fake makes its fakes as plausible as possible. For instance, for RSA signatures with additional tags, random elements, and fields for the public key and the signed data, it might set all these fields correctly, and choose the rest randomly from the correct mathematical group. We require a test oracle Test to be able to distinguish *every* fake oracle from a correct signature oracle.

Now we perform our reduction proof, i.e., we construct a test oracle Test from the given TH and Sim. We call the fake or sign oracle O. In the setup phase, Test obtains a public signature key pks_w and runs the actions of Sim on this key together with TH. This key and the corresponding pks_w are also in O. It then runs A and Sim choosing C nonces together with TH. Further, Test chooses a random string y and runs Sim upon receipt of y as a message from the adversary for participant v. This finishes the set-up phase.

Now, given input data d, it calls $s \leftarrow \mathsf{O}(d)$ and thus obtains a correct or faked (wrong but possibly plausible-looking) signature s. It computes the one-time pad N that makes s fit the previously chosen y by setting $N := s \oplus y$. It solves the equation $N = \sum_{i=1}^{C} b_i n_i$ for a vector b (parts of this can be precomputed). It then makes the user inputs for b to TH, i.e., for all i with $b_i = 1$, it asks TH to XOR the i-th nonce to the string. In this situation M_v would have the result $z = s$. Thus, when Test finally inputs the signature test command for z^{hnd}, then M_v would output $B = \mathsf{true}$ if $\mathsf{O} = \mathsf{Sig}$ and $B = \mathsf{false}$ if $\mathsf{O} = \mathsf{Fake}$. Hence TH almost always does the same. This proves that TH can be used as a test oracle for arbitrary machines Fake. Intuitively, this shows that TH must have the specific cryptographic test algorithm used in the current realization implemented after the set-up phase.

4.5 Counterexamples for Highly Redundant Data and Protocol Restrictions

Similar to Section 4.2, we may ask whether reasonable protocol restrictions allow us to exclude the situation from Section 4.4. First, if M_v's action is part of a protocol, we need that participant v is willing to execute the protocol with a corrupted participant w. This corresponds to the fact that the adversary may corrupt at least one participant, and that the other participants do not know about this. All Dolev-Yao models we know of allow this trust setting. Secondly, the protocols might not use payload data d and bit vectors b on both sides that were not exchanged using the Dolev-Yao model. For the data, we can assume a signature system with message recovery and let the user retrieve d, or we can send d explicitly with the signature, hidden by N. It is trivial to adapt the reduction proof to this.

For the bits, we may assume that they are sent as payload data, and that the protocol class is large enough that Sim has no reason to interpret these payload data in relation to specific messages. To go further towards a reasonable protocol with such behavior, assume that we are doing some group communication where every group member adds a nonce to a message, and the recipient subtracts all these nonces. In our situation, a large number of group members is dishonest, so that all these nonces are shared with the adversary. Further, the groups can vary and the current group membership is indicated by the vector b (suitably augmented by a bit for v, and bits for the possible other honest group members).

5 A Passively Sound Dolev-Yao-style Model for XOR

The main special case of Dolev-Yao-style XOR that we prove to be sound corresponds to passive attacks only, together with a type consistency requirement on the protocol above the Dolev-Yao model. In other respects the result is strong: We show reactive simulatability and need no restrictions on the other operations in the Dolev-Yao model; this distinguishes our result from that in [31, 11]. Roughly, the benefit of the restriction to passive adversaries is that all XORs are constructed bottom-up. Thus the simulator never receives bitstrings that claim to be XORs but where it does not know how to partition them. The necessity of of the type consistency requirement is shown in Section 5.1.

We then present our Dolev-Yao version of XOR in detail, in particular the extensions compared with current models in the literature that we need to overcome the problem from Section 2. Before stating these definitions, we have to repeat some facts about the overall state and knowledge representation of the model from [8] to which we add XOR terms and operations. We also add a type for nonces of variable length. Nonces in [8] are of a fixed length sufficient for cryptographic purposes, while we now want to be able to hide arbitrary other terms by XOR with a random string. Afterwards, we present the implementation of this XOR for the cryptographic realization, and sketch that this realization is as secure as the Dolev-Yao-style abstraction in the sense of reactive simulatability/UC if restricted to the above conditions.

5.1 The Necessity of Correct Type Conversions

The largest difficulty with XOR even in the passive case is typing. XORs can yield arbitrary bitstrings, while otherwise it seems necessary for achieving reactive simulatability that the Dolev-Yao model is strongly typed. The reason is that the Dolev-Yao model must make a decision what happens if a destructor is applied to a term that isn't propertly constructed, e.g., if decryption is applied to a term that is not encrypted at all or with a different key. The only result that seems consistently realizable with real cryptosystems is to prescribe that the result is an error. In other words, the terms are considered typed, and many operations (in particular destructors) yield errors when applied to wrong types. In the cryptographic implementation, this must be realized by explicit type tags.

For an XOR, however, the algebraic equations like commutativity and associativity are considered essential, and they apply to pure bitstrings, not to bitstrings with type tags. The real problem with this typing, however, occurs when converting an XOR back into the original element type. This is a standard situation when XOR is used for explicit or implicit encryption: At some time, the subterms in an XOR cancel out except for one; typically random strings cancel out and one term of another type remains, e.g., payload data. This subterm must be usable by its recipient according to its original type. This is easy to realize in the Dolev-Yao model because one can retain the knowledge of the original type of the subterm. However, in a real, distributed cryptographic system, this is not possible: When two terms are XORed, we cannot reliably decide whether the result is of an underlying type. This is obvious if we remove all type tags before XORing (which is one possibility, and comes closest to typical message formats in XORs). It is also true if we XOR base types with their type tags, e.g., data for payloads or sig for a signature, because these tags can occur by chance when XORing arbitrary strings. Then a participant in the cryptographic realization would get a result (e.g., a payload message) which is totally unpredictable in the Dolev-Yao model. One natural solution to circumvent this problem is to forbid wrong typecasts on the user layer. This may sound like a strong restriction, but actually an XOR is an operation that a cryptographic library should not offer to end users (e.g., a mail program), but only to cryptographic protocols. For protocols it is usually clear what types are expected at what times, and when one expects to be able to extract some payload from an XOR. Correct behavior in this sense can be verified on the protocol layer if we only allow passive attacks.

5.2 Notation

We first repeat important notation from [8], and then introduce additional notation for lists and matrices. We write ":=" for deterministic and "←" for probabilistic assignment, and "$\xleftarrow{\mathcal{R}}$" for uniform random choice from a set. By $x := y$++ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $\mathsf{len}(m)$, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The list operation is denoted as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$. A database D is a set of functions, called entries, each over a finite domain called attributes. For an entry $x \in D$, the value at an attribute att is written $x.att$. For a predicate $pred$ involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill $pred$. If $D[pred]$ contains only one element, we use the same notation for this element. Adding an entry x to D is abbreviated $D :\Leftarrow x$.

For lists, we define operators tail, append, and sort (with any number of arguments) in the usual way where we assume that sort proceeds according to a given standard order $<$ on list elements. As inputs to sort, we allow sets and lists. Additionally, we define an operator normalize that corresponds to the cancellation rules in an XOR, i.e., normalize first removes duplicates leaving one if there is an odd number and then applies sort to the list.

The elements of a matrix $M \in F^{m,n}$ are denoted by $M_{i,j}$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. The algorithm $\mathsf{solve}(M, v)$, given a matrix $M \in F^{m,n}$ and a vector $v \in F^m$, outputs a solution $b \in F^n$ of the equation $Mb = v$ if one exists, and otherwise $\downarrow$. It can, e.g., be built from Gaussian elimination.

5.3 Trusted-Host Machines and Overall Parameters

The underlying system model is an IO-automata model. Hence the overall Dolev-Yao-style model, with its state, is represented as a machine. It is called *trusted host* $\mathsf{TH}_\mathcal{H}$, where $\mathcal{H} = \{1, \ldots, n\}$ denotes the set of honest users. It has a port $\mathsf{in}_u?$ for inputs from and a port $\mathsf{out}_u!$ for outputs to each user $u \in \mathcal{H}$ and for $u = \mathsf{a}$, denoting the adversary.

The trusted host keeps track of the length of messages (this is needed because this length leaks to the adversary even in encryptions) using a tuple L of abstract length functions. One function from L that we need below is $\mathsf{max\_len}(k)$, which can be an arbitrary polynomial denoting the maximum length of processed messages. We extend L with two functions $\mathsf{xor\_len}$ and $\mathsf{nonce\_vl\_len}$ for computing the length of an abstract XOR from the length of its parameters, and for nonces of variable length, respectively.

5.4 States: Term Database

The main part of the state of the Dolev-Yao-style model, i.e., of the machine $\mathsf{TH}_{\mathcal{H}}$, is a database D of the existing terms. Each term is primarily given by its type (top-level operator) and top-level argument list, where the non-atomic arguments are given by pointers to the respective subterms. For this, each term contains a global index that allows us (not the participants) to refer to terms unambiguously. In addition, $\mathsf{TH}_{\mathcal{H}}$ stores the length of each term and handles that represent local names under which the different participants know the term. In particular, the handles imply the knowledge sets known from other Dolev-Yao-style models.

In detail, the database attributes of D are defined as follows; the only differences to [8] due to adding XOR are an augmented type set, the introduction of the set $randomtypes$, and the attribute $parsed$.

- $ind \in \mathcal{INDS}$, called index, consecutively numbers all entries in D. The set $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$. The index is used as a primary key attribute, i.e., one can write $D[i]$ for the selection $D[ind = i]$.
- $type \in typeset$ defines the type of the entry. We add types xor and $\mathsf{nonce\_vl}$ to $typeset$ from [8], denoting the types for exclusive or and for random strings of variable length. We let the set $randomtypes := \{\mathsf{nonce}, \mathsf{nonce\_vl}\}$ denote the set of *random types*. We say *random value* to denote an element of a type in $randomtypes$. Similarly, $secrettypes \subseteq typeset$ denotes a set of *secret types*, whose elements must not be put into messages.
- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Many values a_i are indices of other entries in D and thus in $\mathcal{INDS}$; they are sometimes distinguished by a superscript "ind".
- $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ are handles by which a user or adversary u knows this entry. The value $\downarrow$ means that u does not know this entry. The set $\mathcal{HNDS}$ is yet another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $x.len \in \mathbb{N}_0$ denotes the "length" of the term, computed using the functions from L.
- $x.parsed \in \{1, \downarrow\}$ denotes whether the ideal adversary has already parsed this term; this will be used to determine when non-random values that are no longer hidden in XORs leak.

Note that most entries will have no attribute $parsed$, then $x.parsed = \downarrow$. Initially, D is empty. As additional state parts, $\mathsf{TH}_{\mathcal{H}}$ has a counter $size \in \mathcal{INDS}$ for the current size of D, and counters $curhnd_u$ (current handle) for $u \in \mathcal{H} \cup \{\mathsf{a}\}$, denoting the most recent handle number assigned for u. They are all initialized with 0. $\mathsf{TH}_{\mathcal{H}}$ furthermore maintains explicit counters and message bounds for each port in order to ensure polynomial runtime, cf. [8]; we omit the details.

The algorithm $i^{\mathsf{hnd}} \leftarrow \mathsf{ind2hnd}_u(i)$ (with side effect) denotes that $\mathsf{TH}_{\mathcal{H}}$ determines a handle i^{hnd} for user u to an entry $D[i]$: If $i^{\mathsf{hnd}} := D[i].hnd_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].hnd_u := curhnd_u{++}$. On non-handles, it is the identity function. $\mathsf{ind2hnd}_u^*$ applies $\mathsf{ind2hnd}_u$ to each element of a list.

5.5 Derived Matrices for XORs

For the linear algebra resulting from XORs, we define matrices over $\mathsf{GF}(2)$ representing released XORs, and a list $xlist$ of the corresponding indices. Each column of a matrix corresponds to a released XOR and the rows to the indices of D, i.e., the terms. A coefficient 1 indicates that this term is a top-level parameter in the XOR. We also make a column for each released individual random value. The matrix A indicates the non-random components in each XOR, while $R^{(l)}$ for each $l \in \mathbb{N}$ indicates the random components of length at least l, and $\bar{R}^{(l)}$ those of length less than l. These matrices and lists are derived from D, but as D will always be clear from the context we do not write it as a parameter. More precisely, let $\mathcal{X} := \{i | D[i].type \in \{\mathsf{xor}\} \cup randomtypes \wedge D[i].parsed = 1\}$ and $xlist := \mathsf{sort}(\mathcal{X})$. The condition $D[i].parsed = 1$ reflects that the ideal adversary may parse XORs in arbitrary order, and the matrices only correspond to the already parsed ones. The parsing is considered in Section 5.6. The matrices A, $R^{(l)}$, and $\bar{R}^{(l)}$ are elements of $\mathsf{GF}(2)^{size,|\mathcal{X}|}$. Each element $A_{i,j}$ is defined as follows: Let $x_j := xlist[j]$. Then $A_{i,j} = 1$ iff $D[i].type \notin randomtypes$ and $D[x_j].type = \mathsf{xor}$ and $i \in D[x_j].arg$. Similarly, $R_{i,j}^{(l)} = 1$ iff $D[i].type \in randomtypes$ and $D[i].len \geq l$ and either $D[x_j].type = \mathsf{xor}$ and $i \in D[x_j].arg$, or $D[x_j].type \in randomtypes$ and $i = x_j$. The same formula, except with "$< l$", defines $\bar{R}^{(l)}$.

5.6 New Inputs and their Evaluation

As explained in Section 3.2, operations are triggered by input commands from users or the adversary into $\mathsf{TH}_{\mathcal{H}}$. In this specific model, the users refer to the terms by the handles defined in Section 5.4. The normal cryptographic operations are called basic commands. They are accepted at each input port $\mathsf{in}_u?$ and have only local effects, i.e., only an output at $\mathsf{out}_u?$ occurs and only handles for u are involved. The additional term-handling capabilities of the adversary are called local adversary commands. They are only accepted at $\mathsf{in}_\mathsf{a}?$. Finally, send commands output values to other users. The normal insecure channels actually lead to the adversary, i.e., the adversary instead of the intended recipient gets a handle to a sent message, and the adversary can send under everyone's identity.

The notation $j \leftarrow \mathsf{op}(i)$ means that $\mathsf{TH}_{\mathcal{H}}$ is scheduled with an input $\mathsf{op}(i)$ at some port $\mathsf{in}_u?$ (where we always use u as the index of that port) and returns j at $\mathsf{out}_u!$. Handle arguments are tacitly required to be in $\mathcal{HNDS}$ and existing, i.e., $\leq curhnd_u$, at the time of execution.

Basic Commands: Normal Cryptographic Operations. We first present the XOR operation. It immediately normalizes its arguments into one list, i.e., if some of its inputs are already terms of type xor, it joins their argument lists, adds the other inputs to this list, and then removes duplicates in the list. If the list is now empty, the entry corresponds to the all-zero string of the respective length. We also define the command to convert an XOR back into another type; this succeeds only if the XOR has only one argument of this type. Furthermore, we define the command to create a nonce of variable length.

Definition 1. *(Basic commands for XOR and for nonces of variable length) The trusted host* $\mathsf{TH}_{\mathcal{H}}$ *extended by XOR and nonces of variable length accepts the following additional commands at every port* $\mathsf{in}_u?$.

- *Generate XOR:* $x^{\mathsf{hnd}} \leftarrow \mathsf{xor}(m_1^{\mathsf{hnd}}, \ldots, m_j^{\mathsf{hnd}})$ *for* $0 \le j \le \mathsf{max\_len}(k)$.
  Let $m_i := D[hnd_u = m_i^{\mathsf{hnd}}].ind$ *for* $i = 1, \ldots, j$ *and* $length := \mathsf{xor\_len}(k, D[m_1].len, \ldots, D[m_j].len)$. *If* $length > \mathsf{max\_len}(k)$ *or* $m_i = \downarrow$ *or* $D[m_i].type \in secrettypes$ *for some* $i \in \{1, \ldots, j\}$*, then return* $\downarrow$.
  For $i = 1, \ldots, j$*, let* $arg_i := D[m_i].arg$ *if* $D[m_i].type = \mathsf{xor}$*, else* $arg_i := (m_i)$.
  Let $x\_arg := \mathsf{normalize}(\mathsf{append}(arg_1, \ldots, arg_j))$. *Let* $i := D[type = \mathsf{xor} \wedge arg = x\_arg].ind$. *If* $i \neq \downarrow$ *return* $x^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(i)$*, otherwise set* $x^{\mathsf{hnd}} := curhnd_u{++}$ *and*

  $$D :\Leftarrow (ind := size{++}, type := \mathsf{xor}, arg := x\_arg, hnd_u := x^{\mathsf{hnd}}, len := length).$$

- *Type conversion of XOR:* $m^{\mathsf{hnd}} \leftarrow \mathsf{conv\_xor\_to\_}type(x^{\mathsf{hnd}})$ *with* $type \in typeset \setminus (\{\mathsf{xor}\} \cup secrettypes)$.
  Let $x := D[hnd_u = x^{\mathsf{hnd}} \wedge type = \mathsf{xor}].ind$*,* $(x_1, \ldots, x_j) := D[x].arg[1]$*, and* $t := D[x_1].type$. *If* $j \neq 1$ *or* $t \neq type$ *then return* $\downarrow$*, else return* $m^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(x_1)$.
- *Generate variable-length nonce:* $n^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce\_vl}(l)$ *for* $0 \le l \le \mathsf{max\_len}(k)$.
  Let $l^* := \mathsf{nonce\_vl\_len}(l)$. *If* $l^* > \mathsf{max\_len}(k)$ *then return* $\downarrow$. *Else set* $n^{\mathsf{hnd}} := curhnd{++}$ *and*

  $$D :\Leftarrow (ind := size{++}, type := \mathsf{nonce\_vl}, arg := (l), hnd_u := n^{\mathsf{hnd}}, len := l^*).$$

$\diamond$

Local Adversary Commands. As we have excluded active attacks and any specific vulnerabilities they might add even to the Dolev-Yao-style system, we only need to adapt the command adv_parse, which allows the adversary to retrieve the arguments of an obtained term (represented by a handle) depending on whether the top-level operation of this term should ideally be invertible by the adversary. For XOR terms, the basic idea is to determine whether the part consisting of random values is linearly independent from those in previously released XORs. If yes, we consider it to hide the other components. Otherwise, we give the ideal adversary the non-random arguments of the linear combination whose random elements cancel. This is more than most real adversaries will get, but as there will usually be some information flow, it is simplest to assume the worst case where this will allow the adversary to reconstruct the entire arguments; recall Section 2.

There are two small complications: First, as we have terms of arbitrary length, we introduced nonces of arbitrary length, and thus we now have to be careful that the nonces are indeed of sufficient length. The standard case in protocols will be that all the nonces in one XOR are of the same length. In general, we consider the length l of the longest non-random element in the XOR that is not yet known to the ideal adversary. For simplicity, if the ideal adversary can cancel all nonces of length at least l by a linear combination, we give it not only the non-random elements, but also the shorter nonces in the resulting XOR. This leaves room for improvements, e.g., by only granting the adversary the XOR of those nonces, but at the cost of a more complicated proof that only serves a very restricted class of protocols. Secondly, when considering one XOR we should only use those other XORs that have already been parsed.

Furthermore, to enable the correct simulation of dependent nonces, we also determine linear dependence whenever a random type is parsed, i.e., for the new type $\mathsf{nonce\_vl}$ and the old type nonce.

Definition 2. *(Adversary parameter retrieval for XOR and nonces of variable length) The execution of the existing command* $(type, arg) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$ *always starts by setting* $m := D[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].ind$ *and* $type := D[m].type$*, while the output* arg *depends on the type. We add the definition for* $type \in \{\mathsf{xor}, \mathsf{nonce\_vl}\}$ *and extend the definition for* $type = \mathsf{nonce}$*.*

- *If* $type = \mathsf{xor}$*: Let* $(x_1, \ldots, x_h) := x\_arg := D[m].arg$*. Let* $l := \mathsf{max}(\{D[x_i].len \mid D[x_i].type \notin randomtypes \wedge D[x_i].hnd_{\mathsf{a}} = \downarrow\})$*. Let* a*,* $r^{(l)}$*, and* $\bar{r}^{(l)}$ *denote the vectors that this newly parsed XOR will add to the matrices* A*,* $R^{(l)}$*, and* $\bar{R}^{(l)}$*respectively. Let* $b \leftarrow \mathsf{solve}(R^{(l)}, r^{(l)})$*. If* $b = \downarrow$*, return* $arg := (\mathsf{independent}, D[m].len)$*. Otherwise let* $d := Ab \oplus a$ *and* $r' := \bar{R}^{(l)} b \oplus \bar{r}^{(l)}$*. These are the same linear combination of XORs, including the new one, for which we just saw that the random components of at least length* l *cancel out. Hence we let the ideal adversary learn the non-random elements designated by the vector* d *and the short random parts designated by* r'*. Let* $\mathcal{D} := \{\mathsf{ind2hnd_a}(i) \mid d_i = 1 \vee r'_i = 1\}$ *and* $d' := \mathsf{sort}(\mathcal{D})$*. Return* $arg := (\mathsf{dependent}, b, d')$*.*
- *If* $type = \mathsf{nonce\_vl}$*: Let* $l := D[m].len$*. Let* $r^{(l)}$ *denote the vector that this newly parsed nonce will add to the matrix* $R^{(l)}$*. Let* $b \leftarrow \mathsf{solve}(R^{(l)}, r^{(l)})$*. If* $b = \downarrow$*, return* $arg := (\mathsf{independent}, D[m].arg)$*. Else derive and return a result* $arg := (\mathsf{dependent}, b, d')$ *exactly as for the type* xor*.*
- *If* $type = \mathsf{nonce}$*, we similarly either add the constant* $\mathsf{independent}$ *to the normal argument, or derive and return a result* $(\mathsf{dependent}, b, d')$ *as for the type* $\mathsf{nonce\_vl}$*.*

$\diamond$

5.7 Concrete Realization of Dolev-Yao-style XOR

We only briefly sketch the concrete realization of the Dolev-Yao-style XOR presented in the previous section. A rigorous definition of the realization is given in Appendix A.

The realization of the Dolev-Yao-style model offers its users the same interface as the ideal model, i.e., honest users operate on cryptographic objects via handles. Clearly, in the real system every user has its own machine (in other terminologies protocol engine or automaton), containing only the cryptographic objects that this user knows. Sending a term on an insecure channel releases the actual bitstring to the adversary, who can do with it what he likes. The adversary can also insert arbitrary bitstrings on non-authentic channels.

As the most important part, we present the full command for generating an XOR, although some underlying notation is only defined in the appendix. It is quite natural, except that we tag XORs with a type field so that XORs cannot also be acceptable ciphertexts or keys, cf. Section 5.1. More precisely, whenever several typed bitstrings should be XORed, we first remove their type tags, XOR them, and then add an XOR tag to the resulting string. The definition uses the subroutine $(i^{\mathsf{hnd}}, D_u) :\leftarrow (i, \mathit{type}, \mathit{add\_arg})$, which determines a handle for certain given parameters in the state database D_u of M_u. Essentially, if an entry with the word i already exists, it returns the handle of that, else it generates a new entry with this word.

– *Generate XOR:* $x^{\mathsf{hnd}} \leftarrow \mathsf{xor}(m_1^{\mathsf{hnd}}, \ldots, m_j^{\mathsf{hnd}})$ for $0 \leq j \leq \mathsf{max\_len}(k)$.
  If there is an i with $D_u[m_i^{\mathsf{hnd}}].\mathit{type} \in \mathit{secrettypes} \cup \{\mathsf{garbage}\}$ or where $D_u[m_i^{\mathsf{hnd}}].\mathit{word}$ is not of the form (type, m_i') with $\mathit{type} \in \mathit{typeset}$ and $m_i' \in \{0,1\}^+$, return $\downarrow$. Else set $l := \max(\{\mathsf{len}(m_i') \mid i = 1, \ldots, j\})$ and let $x_i := 0^{l-\mathsf{len}(m_i')}||m_i'$ for $i = 1, \ldots, j$. Let $x := (\mathsf{xor}, x_1 \oplus \cdots \oplus x_j)$ and $(x^{\mathsf{hnd}}, D_u) :\leftarrow (x, \mathsf{xor}, ())$.

5.8 Soundness of the Abstraction

Our security claim is that the real cryptographic library extended with XOR is as secure as the Dolev-Yao-style cryptographic library with XOR in the sense of blackbox simulatability, provided that the adversary is restricted to passive attacks and that the surrounding protocol ensures that XORs are only converted if they consist of a single element and the conversion matches the type of this element. We call the latter property of a configuration *correct XOR conversion*, or short $\mathsf{CorrXOR}$.

To formally capture the property $\mathsf{CorrXOR}$, we need additional notation. We write an event as $p?m$ or $p!m$, meaning that message m occurs at in- or output port p. The t-th step of a trace r is written r_t; we speak of the step at time t. We further let $r_t : D$ denote the contents of database D at time t in trace r.

Definition 3. *(Correct XOR Conversion) A trace r is contained in* $\mathsf{CorrXOR}$ *if and only if for all $t \in \mathbb{N}$, $u \in \mathcal{H}$, and $i \in \mathcal{INDS}$, and with $i_u^{\mathsf{hnd}} := r_t : D[i].hnd_u$, we have*

$$\begin{aligned}
&\mathsf{in}_u?\mathsf{conv_xor_to_}x(i_u^{\mathsf{hnd}}) \in r_t && \textit{\# If an XOR is converted to type } x\\
&\quad \wedge r_t : D[i].\mathit{type} = \mathsf{xor} && \textit{\# and it really is an XOR,}\\
&\Rightarrow \exists j \in \mathcal{INDS} \colon (r_t : D[i].\mathit{arg} = (j) && \textit{\# it consists of one element}\\
&\quad \wedge r_t : D[j].\mathit{type} = x). && \textit{\# which is of the correct type.}
\end{aligned}$$

$\diamond$

Passive attacks in the underlying model can be defined as follows. First, no machines of the real cryptographic library are corrupted, i.e., we only consider the case $\mathcal{H} = \{1, \ldots, n\}$. Secondly, all channels are authentic; this is a well-defined notion in the underlying model. The proof of soundness still works if we relax the authenticity restriction by allowing message re-ordering, re-routing, and duplication, i.e., by solely requiring that the adversary only sends messages that it previously received from an honest participant.

We finally define the notion of reactive simulatability restricted to those configurations where the adversary is restricted to passive attacks and where the property $\mathsf{CorrXOR}$ holds independent of the adversary, i.e., where the honest users already guarantee the validity of the property $\mathsf{CorrXOR}$. We denote this notion by $\geq_{\mathsf{passive}}^{\mathsf{CorrXOR}}$.

Let $RPar$ be the set of valid parameter tuples for the real system, consisting of the number $n \in \mathbb{N}$ of participants, a collection of cryptographic schemes S (S may currently contain symmetric and asymmetric encryption schemes, signature schemes, and MACs) that satisfy their respective security definitions against active attacks, cf. [8, 9, 7], and length functions and bounds L'. For $(n, S, L') \in RPar$, let $Sys^{\mathsf{cry\_XOR,real}}_{n,\mathcal{S},L'}$ be the resulting real cryptographic library. Further, let the corresponding length functions and bounds of the ideal system be formalized by a function $L := \mathsf{R2Ipar}(\mathcal{S}, L')$, and let $Sys^{\mathsf{cry\_XOR,id}}_{n,L}$ be the ideal cryptographic library with parameters n and L. The extension of R2Ipar to the newly added length function for XOR and nonces of variable length is straightforward and omitted.

Theorem 1. *(Security of Cryptographic Library with XOR) For all parameters $(n, \mathcal{S}, L') \in RPar$, we have*

$$Sys^{\mathsf{cry\_XOR,real}}_{n,\mathcal{S},L'} \geq^{\mathsf{CorrXOR}}_{\mathsf{passive}} Sys^{\mathsf{cry\_XOR,id}}_{n,L},$$

where $L := \mathsf{R2Ipar}(\mathcal{S}, L')$. □

For proving this theorem for the original library without XOR, a simulator $\mathsf{Sim}_{\mathcal{H}}$ has been defined in [8] such that even the combination of arbitrary polynomial-time users H and an arbitrary polynomial-time adversary A cannot distinguish the combination of the real machines M_u from the combination of $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$. We sketch in Appendix B how we extend the simulator and the proof of correct simulation to deal with XOR.

6 Conclusion and Outlook

We have shown that Dolev-Yao models augmented by XOR, the simplest operation with algebraic equations in many formal methods and automated tools for cryptographic protocol proofs, cannot be realized by actual cryptographic libraries in a way that is at the same time natural, secure, and usable without restrictions. Our first result shows that *typical* Dolev-Yao models with XOR are not sound with respect to any secrecy definition; we only assume that the Dolev-Yao model contains at least a payload data type and allows XORs on it.

The intuitive goal of our more complex results is to show that *no* Dolev-Yao model with usual cryptographic operations and XOR can be securely implemented in the sense of reactive simulatability/UC, i.e., in the sense that the realization can be safely plugged in for the abstraction in arbitrary environments and for arbitrary goals. As there is no formal definition of what is and isn't a Dolev-Yao model, and as the result would certainly not hold if we also called cryptographic realizations Dolev-Yao models, we have approached this intuitive goal by a series of results that make certain precise assumptions about Dolev-Yao models as well as real XORs, and then show impossibility.

On the positive side, we presented a Dolev-Yao-style model with XOR that has a cryptographic realization secure against passive attacks if the surrounding protocol additionally guarantees that no incorrect conversion of XORs back into other types are attempted.

As future work, we expect that there are possibilities for positive results also under active attacks by strong restrictions on the protocol class or the security properties required, i.e., by no longer requiring reactive simulatability between the Dolev-Yao model as such and its realization. However, we believe that our impossibility results pose severe limits on the applicability of formal methods for XOR and cryptography when ultimately a cryptographically sound implementation is desired. The results certainly also prove that one cannot simply add operations with algebraic properties to a Dolev-Yao model if one aims at general secure realizations, even if the operation on its own seems simple and well characterized by its algebraic properties, as XOR is. We actually believe that the difficulties we had with XOR are not an exception, but the norm. However, this remains future work, except that the results trivially generalize to the Abelian groups $\mathbb{Z}_{2^l}$, into which bitstrings can be bijectively mapped.

References

1. M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. In *Proc. 13th European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2004.
2. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3124 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2004.
3. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
4. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
5. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.
6. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.
7. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004. Full version in IACR Cryptology ePrint Archive 2004/059, Feb. 2004, `http://eprint.iacr.org/`.
8. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.
9. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003. Extended version in IACR Cryptology ePrint Archive 2003/145, Jul. 2003, `http://eprint.iacr.org/`.
10. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.
11. M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.
12. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

13. M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology: CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1995.
14. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
15. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.
16. R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. `http://eprint.iacr.org/`.
17. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 124–135, 2003.
18. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 261–270, 2003.
19. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 271–280, 2003.
20. H. Comon-Lundh and R. Treinen. Easy intruder deductions. Research Report LSV-03-8, Laboratoire Spécification et Vérification, ENS Cachan, France, Apr. 2003.
21. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 109–125, 2003.
22. S. Delaune and F. Jacquemard. Narrowing-based constraint solving for the verification of security protocols. Research Report LSV-04-8, Laboratoire Spécification et Vérification, ENS Cachan, France, Apr. 2004.
23. D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
24. W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proc. of the IEEE*, 67(3):397–427, 1979.
25. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
26. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
27. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.
28. D. Kapur, P. Narendran, and L. Wang. An e-unification algorithm for analyzing protocols that use modular exponentiation. In *Proc. 14th International Conference on Rewriting Techniques and Applications (RTA)*, pages 165–179, 2003.
29. H. Krawczyk. LFSR-based hashing and authentication. In *Advances in Crptology: CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.
30. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
31. P. Laud. *Computationally Secure Information Flow*. PhD thesis, Universität des Saarlandes, 2002. `http://www.cs.ut.ee/~peeter_l/research/csif/lqpp.ps.gz`.
32. P. Laud. Pseudorandom permutations and equivalence of formal expressions (abstract). In *14th Nordic Workshop on Programming Theory*, pages 63–65, 2002.

33. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.
34. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
35. C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
36. C. Meadows. A model of computation for the NRL protocol analyzer. In *Proc. 7th IEEE Computer Security Foundations Workshop (CSFW)*, pages 84–89, 1994.
37. C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proc. WITS*, 2002.
38. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.
39. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.
40. J. Millen. The Interrogator model. In *Proc. 16th IEEE Symposium on Security & Privacy*, pages 251–260, 1995.
41. J. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.
42. J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 47–61, 2003.
43. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
44. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.
45. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.
46. A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, 2003.
47. V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. 13th European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 355–369. Springer, 2004.
48. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

A Formal Definition of the Real System

A.1 Machines and Parameters

The intended structure of the realization consists of n machines $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$, one for each participant. Each M_u has ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$, so that the same honest-user machines (representing applications or protocols) can connect to the ideal and the real system. Each M_u has connections to each M_v as in [8], in particular an insecure connection called $\mathsf{net}_{u,v,\mathsf{i}}$ for normal use. They are called network connections and the corresponding ports network ports. Any subset $\mathcal{H}$ of $\{1, \ldots, n\}$ can denote the indices of correct machines. The resulting actual structure consists of the correct machines with modified channels according to a channel model. In particular, each insecure channel is split so that both machines actually interact with the adversary. Similar to the tuple L of length functions in the ideal cryptographic library, there are functions defining the length of lists (based on the element lengths) and nonces. These functions are grouped in a tuple L' and can be arbitrary polynomials.

A.2 States of a Machine

Each machine M_u contains the cryptographic objects the user already knows under this user's handles. We again represent this as a database; the structure is simple here because there are only the handles, the corresponding bitstrings, and for our convenience the message types (which could be retrieved by parsing the bitstring) and a "reserve" parameter. More precisely, each such database D_u has the following attributes:

- $hnd_u \in \mathcal{HNDS}$ consecutively numbers all entries in D_u. We use it as a primary key attribute, i.e., we write $D_u[i^{\mathsf{hnd}}]$ for the selection $D_u[hnd_u = i^{\mathsf{hnd}}]$.
- $word \in \{0,1\}^+$ is the real bitstring.
- $type \in typeset \cup \{\mathsf{null}\}$ identifies the type of the entry. The value null denotes that the entry has not yet been parsed.
- add_arg is a list of ("additional") arguments. For entries of our new types it is always empty, i.e., $()$.

Initially, D_u is empty. M_u has a counter $curhnd_u \in \mathcal{HNDS}$ for the current size of D_u. The subroutine

$$(i^{\mathsf{hnd}}, D_u) :\leftarrow (i, type, add\_arg)$$

determines a handle for certain given parameters in D_u: If an entry with the word i already exists, i.e., $i^{\mathsf{hnd}} := D_u[word = i \wedge type \notin secrettypes].hnd_u \neq \downarrow$, it returns i^{hnd}, assigning the input values $type$ and add_arg to the corresponding attributes of $D_u[i^{\mathsf{hnd}}]$ only if $D_u[i^{\mathsf{hnd}}].type$ was null. Else if $\mathsf{len}(i) > \mathsf{max\_len}(k)$, it returns $i^{\mathsf{hnd}} = \downarrow$. Otherwise, it sets and returns $i^{\mathsf{hnd}} := curhnd_u{+}{+}$, $D_u :\Leftarrow (i^{\mathsf{hnd}}, i, type, add\_arg)$. Similar to Section 5.4, M_u maintains a counter $steps_{\mathsf{p?}} \in \mathbb{N}_0$ for each input port $\mathsf{p?}$, initialized with 0. All corresponding bounds $bound_{\mathsf{p?}}$ are $\mathsf{max\_in}(k)$. Length functions for inputs are tacitly defined by the domains of each input.

A.3 Inputs and their Evaluation

Now we describe how M_u evaluates the individual inputs related to XOR and for nonces of variable lengths. Clearly, there are the same basic commands for creating an exclusive or and a nonce as in the ideal system. There are no adversary local commands because a real adversary is not restricted to specific, algebraic operations, but performs arbitrary bitstring manipulations. The send commands now correspond to real sending, and receiving a message from a channel into a real machine must newly be considered.

The stateful commands are defined via functional constructors and parsing algorithms for each cryptographic type. (These stateless algorithms can be reused in the simulator and the proof, while the stateful parts are different in the simulator.) We start with the constructors; they define the exact structure of the bitstrings in the database. We represent XORs with a type tag xor, just like all other correct realizations of valid terms. Note that XORing arguments first removes their type tags and XORs the raw data.

Definition 4. *(Constructors for XOR and nonces of variable length)*

- *XOR constructor:* $x \leftarrow \mathsf{make\_xor}(m_1, \ldots, m_j)$ *for* $j \in \mathbb{N}$ *and* $m_i \in \{0,1\}^+$ *for* $i := 1, \ldots, j$.
  If for some i *we have that* m_i *is not of the form* $m_i = (type, m'_i)$ *with* $type \in typeset$ *and* $m'_i \in \{0,1\}^+$*, then return* $x := \downarrow$*. Otherwise let* $l := \max(\{\mathsf{len}(m'_i) \mid i = 1, \ldots, j\})$ *and let* $x_i := 0^{l-\mathsf{len}(m'_i)}||m'_i$ *for* $i = 1, \ldots, j$*. Let* $x := (\mathsf{xor}, x_1 \oplus \cdots \oplus x_j)$.
- *Variable-length nonce constructor:* $n \leftarrow \mathsf{make\_nonce\_vl}(l)$ *for* $l \in \mathbb{N}$*.*
  Let $n' \stackrel{\mathcal{R}}{\leftarrow} \{0,1\}^l$ *and* $n := (\mathsf{nonce\_vl}, n')$.

◇

Now we define the destructors. The term "tagged list" means a valid list of the real system. We assume that tagged lists are efficiently encoded into $\{0,1\}^+$. From the underlying Dolev-Yao-style model, we need to know the general parsing algorithm, and we add the type-specific commands:

- *General parsing:* $(type, arg) \leftarrow \mathsf{parse}(m)$.
  If m is not of the form $(type, m_1, \ldots, m_j)$ with $type \in typeset \setminus (secrettypes \cup \{\mathsf{garbage}\})$ and $j \geq 0$, return $(\mathsf{garbage}, ())$. Else call the type-specific parsing algorithm $arg' \leftarrow \mathsf{parse\_}type(m)$. If $arg = \downarrow$, then parse again outputs $(\mathsf{garbage}, ())$, else $(type, arg)$. For $type = \mathsf{xor}$, we define $arg := (x_1), \ldots, x_j)$ if x is of the form $(\mathsf{xor}, x_1, \ldots, x_j)$ with $x_i \in \{0,1\}^+$ for $i = 1, \ldots, j$, else $arg := \downarrow$. For $type = \mathsf{nonce\_vl}$, we define $arg := (l)$ if n is of the form $(\mathsf{nonce\_vl}, n')$ with $n' \in \{0,1\}^l$ for some $l \in \mathbb{N}$, else $arg := \downarrow$.

We now define how a real machine reacts on the same basic commands as the ideal Dolev-Yao-style system. We use the functional subroutines and subroutines for the state changes resulting from parsing, defined in [8]:

- "parse m^{hnd}" means that M_u calls $(type, arg) \leftarrow \mathsf{parse}(D_u[m^{\mathsf{hnd}}].word)$, assigns $D_u[m^{\mathsf{hnd}}].type := type$ if it was still null, and may then use arg.
- "parse m^{hnd} if necessary" means the same except that M_u does nothing if $D_u[m^{\mathsf{hnd}}].type \neq \mathsf{null}$.

Definition 5. *(Basic commands for XOR and nonces of variable length)*

- *Generate XOR:* $x^{\mathsf{hnd}} \leftarrow \mathsf{xor}(m_1^{\mathsf{hnd}}, \ldots, m_j^{\mathsf{hnd}})$ *for* $0 \leq j \leq \mathsf{max\_len}(k)$.
  If there is an i *with* $D_u[m_i^{\mathsf{hnd}}].type \in secrettypes \cup \{\mathsf{garbage}\}$*, return* $\downarrow$*. Else, for* $i := 1, \ldots, j$*, set* $m_i := \mathsf{tail}(D_u[m_i^{\mathsf{hnd}}].word)$*, i.e.,* m_i *is the core string without the type tag. Let* $x := \mathsf{make\_xor}(m_1, \ldots, m_j)$ *and* $(x^{\mathsf{hnd}}, D_u) :\leftarrow (x, \mathsf{xor}, ())$.
- *Type conversion of XOR:* $m^{\mathsf{hnd}} \leftarrow \mathsf{conv\_xor\_to\_}type(x^{\mathsf{hnd}})$ *with* $type \in typeset \setminus \{\mathsf{xor}\}$.
  Parse x^{hnd} *if necessary yielding* (t, arg)*. If* $D_u[x^{\mathsf{hnd}}].type \neq \mathsf{xor}$ *or* $arg \neq (x')$ *for some* $x' \in \{0,1\}^+$*, then return* $\downarrow$*, else let* $(m^{\mathsf{hnd}}, D_u) :\leftarrow ((type, x'), type, ())$.
- *Generate variable-length nonce:* $n^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce\_vl}(l)$ *for* $l \leq \mathsf{max\_len}(k)$.
  Let $n \leftarrow \mathsf{make\_nonce\_vl}(l)$*,* $n^{\mathsf{hnd}} := curhnd{+}{+}$ *and* $D_u :\Leftarrow (n^{\mathsf{hnd}}, n, \mathsf{nonce\_vl}, ())$.

◇

B Extended Simulator (Sketch)

Basically $\mathsf{Sim}_{\mathcal{H}}$ has to translate real messages from the real adversary A into handles as $\mathsf{TH}_{\mathcal{H}}$ expects them at its adversary input port $\mathsf{in_a}?$ and vice versa. In both directions, $\mathsf{Sim}_{\mathcal{H}}$ has to parse an incoming message completely because it can only construct the other version (abstract or real) bottom-up. This is done by recursive algorithms. The state of $\mathsf{Sim}_{\mathcal{H}}$ mainly consists of a database D_a that mainly stores the bitstrings the adversary knows under the adversary handles. The definition of the simulator does not contain any conceptionally interesting new techniques but is a straightforward exercise since the appropriate definition of adv_parse is already in place and since no additional cryptographic operations are involved. We hence only give an informal description of $\mathsf{Sim}_{\mathcal{H}}$.

Inputs from $\mathsf{TH}_{\mathcal{H}}$. Assume that $\mathsf{Sim}_{\mathcal{H}}$ receives an input from $\mathsf{TH}_{\mathcal{H}}$. This input is of the form $(u, v, x, l^{\mathsf{hnd}})$, denoting that u sent the term corresponding to handle l^{hnd} to v over a channel of type x, e.g., an insecure one. If a bitstring l for l^{hnd} already exists in D_a, i.e., this message is already known to the adversary, the simulator immediately outputs l to A at a corresponding network port. Otherwise, it first constructs such a bitstring l with a recursive algorithm id2real. This algorithm decomposes the abstract term using basic commands and the adversary command adv_parse. At the same time, id2real builds up a corresponding real bitstring using real cryptographic operations and enters all new message parts into D_a to recognize them when they are reused, both by $\mathsf{TH}_{\mathcal{H}}$ and by A. We sketch how the simulator is extended to deal with XOR and nonces.

If the entry corresponding to l^{hnd} is an XOR, $\mathsf{Sim}_{\mathcal{H}}$ applies adv_parse to l^{hnd}. This yields $(\mathsf{independent}, len)$ or $(\mathsf{dependent}, b, d')$. In the first case, $\mathsf{Sim}_{\mathcal{H}}$ simply chooses a random bitstring of correct length and adds the type tag xor. In the second case, $\mathsf{Sim}_{\mathcal{H}}$ first applies id2real to the handles contained in d' yielding strings $m_1, \dots, m_t$. Then $\mathsf{Sim}_{\mathcal{H}}$ determines the bitstrings corresponding to the linear combination b: If $d'_i = 1$, it takes the word entry of the i-th entry in D_a that is either a random value or of type xor. Let the result be messages $m_{t+1}, \dots, m_s$. It then constructs the XOR of the strings $m_1, \dots, m_s$ as shown in Section 5.7, i.e., $\mathsf{Sim}_{\mathcal{H}}$ first removes the type tags, pads shorter strings, computes the XOR of the padded strings, and finally adds an xor type tag to the resulting string.

If the entry corresponding to l^{hnd} is of type nonce_vl, $\mathsf{Sim}_{\mathcal{H}}$ calls $\mathsf{adv\_parse}(l^{\mathsf{hnd}})$. This yields $(\mathsf{independent}, l)$ or $(\mathsf{dependent}, b, d')$. In the first case, $\mathsf{Sim}_{\mathcal{H}}$ chooses a random string of length l and adds the type tag nonce_vl. In the second case, it constructs the XOR corresponding to the linear combination b and the learned values d' as for the type xor, except that it adds the type tag nonce_vl instead of xor. For the type nonce, the changes are similar.

Inputs from A. Now assume that $\mathsf{Sim}_{\mathcal{H}}$ receives a bitstring l from A at a network port. If l is not a valid list, $\mathsf{Sim}_{\mathcal{H}}$ aborts the transition. Otherwise it translates l into a corresponding handle l^{hnd} by an algorithm real2id, and outputs the abstract sending command $\mathsf{adv\_send\_}x(w, u, l^{\mathsf{hnd}})$ at port $\mathsf{in_a}!$. If a handle l^{hnd} for l already exists in D_a, then real2id reuses that. Note that in the case of passive adversaries, we assumed that every message coming from the adversary was already received over the network, i.e., a handle l^{hnd} for l always already exists in D_a.

In the proof of the extended cryptographic library, now including XOR, we can easily retain the proof structure from [8]. The basic structure of this proof is that a combined system $\mathsf{C}_{\mathcal{H}}$ is defined that essentially contains all aspects of both the real and the ideal system, and then bisimulations are proved between $\mathsf{C}_{\mathcal{H}}$ and the combination $\mathsf{M}_{\mathcal{H}}$ of the real machines, and between $\mathsf{C}_{\mathcal{H}}$ and the combination $\mathsf{THSim}_{\mathcal{H}}$ of the trusted host and the simulator. A bisimulation, however, cannot deal with computational indistinguishability. Hence at the beginning of the proof, the real asymmetric encryptions were replaced by simulated ones as made in the simulator. These aspects of the proof remain unchanged for our inclusion of XOR. We are left to show how the bisimulations of the original cryptographic library are extended for XOR. This part of the proof closely resembles the bisimulation for existing types except that we have to guarantee that the linear algebra for detecting dependencies among XORs is consistently performed in $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$. Overall, this is a straightforward yet tedious part of the proof without much novelty from a conceptional point of view.