Technical Report: Limits of the BRSIM/UC Soundness of Dolev-Yao-style XOR*

Michael Backes[1] and Birgit Pfitzmann[2]

[1] Saarland University & MPI-SWS
[2] IBM Zurich Research Lab

January 17, 2009

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 Dolev-Yao models can be sound with respect to actual cryptographic realizations and security definitions. The strongest results show this in the sense of blackbox reactive simulatability (BRSIM)/UC, a notion that essentially means the preservation of arbitrary security properties under arbitrary active attacks and in arbitrary protocol environments, with only small changes to the Dolev-Yao models and natural implementations.
However, these results are so far restricted to core cryptographic systems like encryption and signatures. Typical modern tools and complexity results around Dolev-Yao models also allow operations with more algebraic properties, in particular XOR because of its clear structure and cryptographic usefulness. We show that it is not possible to extend the strong BRSIM/UC results to XOR, at least not with remotely the same generality and naturalness as for the core cryptographic systems. We also show that for every potential soundness result for XOR with secrecy implications, one significant change to typical Dolev-Yao models must be made. On the positive side, we show the soundness of a rather general Dolev-Yao model with XOR and its realization in the sense of BRSIM/UC under passive attacks.

1 Introduction

Tool-supported verification of cryptographic protocols almost always relies on abstractions of cryptographic operations by term algebras with cancellation rules, called *Dolev-Yao models* after the first authors [3]. An example term is $\mathsf{D}_{ske}(\mathsf{E}_{pke}(\mathsf{E}_{pke}(m)))$, where E and D denote public-key encryption and decryption, *ske* and *pke* a corresponding secret and public encryption key, and m a payload message. By *payloads* we denote the type of non-cryptographic data that most Dolev-Yao models have. It is used for data input by the users of the Dolev-Yao model, e.g., emails to be encrypted and signed or payment data constructed by a payment protocol using the Dolev-Yao model. We wrote the keys as indices for readability; formally they are normal operands in the term. A typical cancellation rule is $\mathsf{D}_{ske}(\mathsf{E}_{pke}(t)) = t$ for all corresponding keys and terms t, thus the above term is equivalent to $\mathsf{E}_{pke}(m)$. The proof tools handle these terms symbolically, i.e., they never evaluate them to bitstrings. In other words, the tools perform abstract algebraic manipulations on trees consisting of operators and base messages, using only the cancellation rules, the message-construction rules of a particular protocol, and abstract models of networks and adversaries. The core of these term algebras are operations like encryption and decryption which ideally have very few algebraic properties; essentially there are only constructors and destructors. 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 so that they fit into the overall term algebra. A typical such operation is the exclusive or (XOR), see, e.g., [4–6], 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.

1.1 Dolev-Yao Soundness and XOR

It is not at all clear from the outset whether Dolev-Yao models are a sound abstraction from real cryptography with its computational security definitions. In particular, the tools assume that *only* the

* A preliminary version of this paper appeared in [1, 2].

modeled operations and cancellation rules are possible manipulations on terms, and that terms that cannot be constructed with these rules are completely secret. For instance, if an adversary (also called intruder) only saw the example term above and only the mentioned cancellation rule was given, then m would be considered secret. In term-algebra terminology, this corresponds to considering only the *initial model* of the given equational specification. While it is usually clear that real cryptography is *some* model of the specification, it is not clear that it is exactly the initial model. For instance, if arbitrarily long payloads m are allowed, real cryptography cannot prevent that the term above leaks information about the length of m. Hence there is already a soundness problem.

Recent work has essentially bridged this 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 blackbox reactive simulatability (BRSIM) [7]. This security (or soundness) notion essentially means that one system, here the cryptographic realization, can be plugged into arbitrary protocols instead of another system, here the Dolev-Yao model, without any noticeable difference [8–10]. Essentially the same notion is also called UC for its universal composition properties [11].[1] This BRSIM/UC result was extended to more cryptographic primitives [12–14] and used in protocol proofs [15–20]. General theorems on property preservation through the BRSIM notion imply that the same Dolev-Yao model and realization also fulfill some other soundness notions [8, 21–23, 21, 24, 25], and further soundness results specific to this Dolev-Yao model and realization were proved in [26]. Stronger links of this Dolev-Yao model to conventional Dolev-Yao type systems were provided in [27], and an integration into the Isabelle theorem prover in [28]. Earlier soundness results considered passive attacks only [29–31]. Later papers such as [32–34] define weaker soundness notions, such as integrity only or offline mappings between runs of the two systems, and/or allow less general protocol classes, e.g., only a specific class of key exchange protocols. For these cases, they can use simpler Dolev-Yao models and/or realizations than [7]. Since computational soundness has become a highly active line of research, we exemplarily list further recent results in this area without going into further details [35–40].

All these Dolev-Yao soundness papers consider only core cryptographic systems like encryption and signatures, not operations with additional algebraic properties like XOR. The first sound formal abstraction of XOR was presented in [41, 42] in a calculus for pseudorandomness that can replace more standard Dolev-Yao calculi if encryptions (e.g., block ciphers) are treated as cryptographic pseudorandom permutations. It only treats passive attacks.

In this paper we first show that one change to Dolev-Yao models of XOR is necessary to obtain any soundness result that implies secrecy: the adversary must be allowed to parse XORs unless one component is uniformly random; we call this the *XOR-parsing need*. Types that are not uniformly random are payloads, but also keys and ciphertexts for standard cryptographic realizations. Then we study whether the soundness results in the sense of BRSIM/UC can be extended to Dolev-Yao models with XOR. It turns out that this is impossible in a general way. We are quite surprised by this result, because XOR seems a simple operation compared with systems like digital signatures, and it seems well described by its algebraic properties. We have also not found reasonable restrictions to the protocol class considered, or reasonable modifications to the Dolev-Yao model or the realization of XOR that would make a BRSIM/UC result possible. The precise range of Dolev-Yao models, protocol classes, and realizations, for which we show such *BRSIM/UC impossibility results* is discussed in more detail below. The only positive result we show is restricted to passive attacks. Otherwise this result is strong: It shows soundness in the sense of BRSIM/UC, allows a broad range of other operations in the Dolev-Yao model, and correctly handles situations where some components in an XOR are uniformly random and others are not. We call it *passive BRSIM/UC XOR soundness*. 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, and indeed most security protocols are intended for scenarios with wide-ranging active attacks. We therefore consider our negative results for the active case more important.[2]

[1] The 2005 revision of the long version of [11] also contains an explicit blackbox version of UC, which is proven to be equivalent to UC. A similar equivalence was first shown in the long version of [8] for universal and blackbox synchronous reactive simulatability.

[2] Another soundness result for XOR in the passive case was recently obtained in [43]. Here XOR is restricted to terms whose corresponding bitstrings have a uniformly random distribution. Thus, e.g., one cannot even

A considerable technical problem in the BRSIM/UC impossibility results is that we would like to show that *no* Dolev-Yao model with XOR has *any* realization sound in the sense of BRSIM/UC. However, this is a meta-theorem formulation: There is no current definition of a Dolev-Yao model independent of specific system models such as CSP, π-calculus, I/O automata etc. For positive results, this is not a problem. However, an impossibility result that only holds for one such model would not be very convincing. (In particular, the closest model to build on would be the BPW-DY model from [7], because it already avoids "smaller" impossibilities for BRSIM/UC soundness; however, due to syntax idiosyncrasies many people find it hard to transfer basic ideas from that model to others. An equivalent less idiosyncratic version was published too late to influence the present results [28].) Not even the notion of an XOR realization is completely well-defined, e.g., if one considers adding XOR to a Dolev-Yao realization where other operators are type-tagged. Hence, instead of proving impossibility for one specific Dolev-Yao model, we will only make certain assumptions on the Dolev-Yao model; we believe they are fulfilled by all such models existing so far.

Clearly, the BRSIM/UC impossibility results that we show for Dolev-Yao models with XOR leave room for considering weaker notions of soundness. It is also not excluded that even a BRSIM/UC soundness result holds for certain restricted protocol classes to which none of our impossibility results applies. Furthermore, certain protocols built with XORs may be secure with respect to certain specifications (ideal systems) in the sense of BRSIM/UC even when the underlying cryptographic operations altogether cannot be abstracted by a BRSIM/UC-sound Dolev-Yao model. It will be interesting to investigate the precise limits in the future. Nevertheless, these results show that the general secure pluggability of a cryptographic realization for a Dolev-Yao model given by the BRSIM/UC results for core cryptographic operations cannot be extended to XOR.

1.2 Further 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., [45–47]. To the best of our knowledge, the XOR operation in the symbolic analysis of cryptographic protocols was first mentioned by Meadows as a possible extension of the NRL analyzer [48]. It has been incorporated into many formal proof tools, e.g., NRL [49], CAPSL [50], Isabelle [51], and OFMC [52]. Recent papers on XOR in Dolev-Yao models mainly investigate the decidability and complexity of the security of certain protocol classes against a Dolev-Yao attack in the presence of deduction rules for the XOR operator [5, 6].

The line of work on Dolev-Yao models with XOR typically continues with abstractions of more general Abelian groups, e.g., [53–55], and the exponentiation function as used in many cryptographic systems based on the discrete-logarithm problem, e.g., [56–58]. While we have not yet considered the soundness of these extensions, we are convinced that a general use of such operations on other terms would lead to similar problems as with XOR. For exponentiations, however, it may be more realistic than for XOR to make strong restrictions on the types of terms that can be exponentiated and the use of the results within larger terms, and such restrictions might help.

Reactive simulatability (RSIM) could, in terms of the semantics community, be called an implementation or refinement relation, with a particular emphasis on also retaining secrecy properties, in contrast to typical implementation relations. It was first defined generally in [8], based on simulatability definitions for secure (one-step) function evaluation [59–63]. On the side of formal methods, it is also highly related to the observational equivalence notions for a probabilistic π-calculus from [64]. Reactive definitions of simulatability for asynchronous systems were presented in [9, 11], called UC (universal composability) and with somewhat different details in the latter. Since then, these definitions have been used in many ways for proving individual cryptographic systems and general theorems. While the definitions of [9, 11] have not been rigorously mapped, we believe that for the results in this paper the differences do not matter, in particular if one thinks of the equivalent blackbox version of UC. Similarly, we believe that the results would hold in the formalism started in [64].

model a one-time pad combining a random string and an arbitrary plaintext. Moreover, we recently showed a similar results for Dolev-Yao style representations of hash functions [44].

In the wider field of linking formal methods and cryptography, there is also work on formulating syntactic calculi for dealing with probabilism and polynomial-time considerations directly and encoding them into proof tools, in particular [65–69]. This is orthogonal to the work of justifying Dolev-Yao models: In situations where Dolev-Yao models are applicable and sound, they are likely to remain important because of the strong simplification they offer to the tools, which enables the tools to treat larger overall systems automatically than with the more detailed models of cryptography.

1.3 More Details on Our Results

We now summarize our results in a bit more detail.

XOR-parsing need. In Section 2, we show that the standard Dolev-Yao model of XOR used in the literature is not sound with respect to every moderately natural implementation if secrecy is required (not necessarily BRSIM/UC) and arbitrary terms such as payloads can be XORed. Instead, the adversary must be allowed to parse XORs unless one component is uniformly random. In realizations based only upon standard cryptographic security definitions, the only sufficiently random types are nonces, i.e., explicitly generated fresh random or pseudorandom strings, while payloads, keys, ciphertexts, and signatures are not automatically sufficiently random. However, in restricted situations more random types may be possible, e.g., if the payload distribution is known and strong compression is used, or if one restricts symmetric encryption or authentication schemes to those with uniformly random keys. While this result is not very surprising from the cryptographic point of view, the consequences such as special operators for the adversary would already be significant changes in some Dolev-Yao models in the literature.

BRSIM/UC impossibility results. 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.

- If we assume a Dolev-Yao model with XOR and payloads and a realization where payloads are arbitrary bitstrings and used in their original form, and if we postulate that the Dolev-Yao model treats XORs of payloads in a certain natural way, then it is not sound in the sense of BRSIM/UC. This is sketched in Section 4.1 and rigorously shown in Section 6.1. We only need XORs of two payloads and the protocol could be fixed and generally known. Hence this is a rather strong result. We also discuss why one might nevertheless not be satisfied with the “certain natural way”, and thus be interested in the later results also (Section 4.2).
- If we assume a Dolev-Yao model with XOR and signatures, and a realization where payloads are arbitrary and used in their original form, then soundness in the sense of BRSIM/UC implies that the Dolev-Yao model can compute actual signatures according to the real algorithms used in the assumed cryptographic realization. Informally this contradicts the assumption that the given system is a Dolev-Yao model. More precisely, we present a reduction proof showing that under the given assumptions, the (supposed) Dolev-Yao model can be used to build a signing algorithm with minimal additional operations. This is sketched in Section 4.3 and rigorously shown in Section 6.2.
- The same result holds even if the payloads may be encoded in the real system before being used in XORs, but with low or well-structured redundancy such as type tags (Section 7.1). To the best of our knowledge, all current implementations of XOR fall into this class or the previous class.
- The same result holds with more complex counterexamples 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. For our main counterexample in this case we additionally assume that public-key encryption with the standard secrecy features is available in the Dolev-Yao model (Section 7.2).
- Even if there are no payloads at all, 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 that yields an approximate test algorithm, a notion that we first define rigorously (Section 7.3).

The basic underlying problem in all these cases is when an honest participant receives an XOR from an active adversary, and the simulator of the BRSIM/UC definition cannot know how to parse it, and will thus either parse it wrong with high probability, or leave real work to the supposed Dolev-Yao model.

Positive BRSIM/UC result for passive attacks. In Section 8, we show BRSIM/UC soundness for an extension of the Dolev-Yao model from [7] and its realization, where both restrict the adversary to passive attacks. The adversary capabilities in the Dolev-Yao model are extended compared with standard models as necessary according to the result on XOR-parsing need. Additionally, we need a restriction that the users, i.e., typically the protocols that use the Dolev-Yao systems, only request correct type conversions from XORs back to underlying term types. The condition is of a class that can be verified formally for the protocols. A third slightly unusual feature, but natural in the context, is that we allow nonce types of different lengths, so that the corresponding real nonces can be used to hide terms of arbitrary length.

2 The Need for Special XOR Parsing by the Dolev-Yao Adversary

We first show why every XOR abstraction that is sound with respect to secrecy, i.e., messages that are symbolically secret in the Dolev-Yao model are also cryptographically secret in the realization, must be different from the standard Dolev-Yao XOR in the literature. Thus this negative result is broader than only for the strong BRSIM/UC notion of soundness. However, it only concerns the standard Dolev-Yao models of XOR, and can be circumvented by adding a feature to these Dolev-Yao models which we consider well within the spirit of Dolev-Yao models. Thus we do not call it an impossibility result. In particular, our passively sound model in Section 8 has this additional feature.

As far as we know, all Dolev-Yao models with XOR that are used in tools or in decidability and complexity results 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 $c = \mathsf{XOR}(d, k)$, 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 it receives an XOR of two terms that it both doesn't know, and that both did not occur in other XORs, it cannot derive 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 [70]. 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 random. In this sense, prior Dolev-Yao models of XOR are overly optimistic. (The pseudorandomness calculus from [41] of course recognizes that XORs are only pseudorandom if at least one contained term is.) 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 detail in the positive result for passive attacks—by distinguishing a set of random types among the types of a Dolev-Yao model. Elements of a random type are deemed sufficiently random (often pseudo-random in reality) to restrict the adversary to the standard algebraic operations on XORs. However, if there are no unknown random elements in an XOR, the Dolev-Yao adversary is given the capability to parse this XOR, i.e., if he learns the XOR, then he also learns the XORed terms.

3 Blackbox Reactive Simulatability

Our remaining positive and negative results about the soundness of Dolev-Yao models with XOR concern the soundness notion of blackbox reactive simulatability (BRSIM)/UC. Hence we start by surveying this notion. Reactive simulatability is a general notion for comparing two systems, typically called real and ideal system. In terms of the semantics community one might call RSIM an implementation or refinement relation, specifically geared towards the preservation of what one might call secrecy properties compared with functional properties. We believe that all our following results are independent of the small differences between the definition styles of [64, 8, 11], and therefore write "BRSIM/UC". However, we have to use a specific formalism for the actual results, and we use that from [9]. Here one speaks of ideal and real systems (the functionalities and protocols of UC). The ideal system is often called

TH for "trusted host", see Figure 1, and the protocol machines of the real system are often called M_u, where u is a user index. The ideal or real system interacts with arbitrary so-called honest users, often collectively denoted by a machine H; this corresponds to potential protocols or human users to whom the functionality is offered. Furthermore, the ideal or real system interacts with an adversary, who is often given more power than the honest users; in particular in real systems A typically controls the network and can manipulate messages on the bitstring level. Adversaries are often denoted by A. They are allowed to interact directly with H; this corresponds to known-message and chosen-message attacks.

Reactive simulatability between the real and ideal system essentially means that for every attack on the real system there exists an equivalent attack on the ideal system. More specifically, blackbox reactive simulatability (BRSIM) states that there exists a simulator Sim that can use an arbitrary real adversary A as a blackbox, such that arbitrary honest users H cannot distinguish whether they interact with the real system and the real adversary, or with the ideal system and the simulator with its blackbox. Indistinguishability of families of random variables, here applied to the two families of views of the honest users, is a well-known cryptographic notion from [71]. As Sim is chosen before A and H, the BRSIM definition allows A and H that communicate directly. Our first counterexample, however, will not make use of this feature.

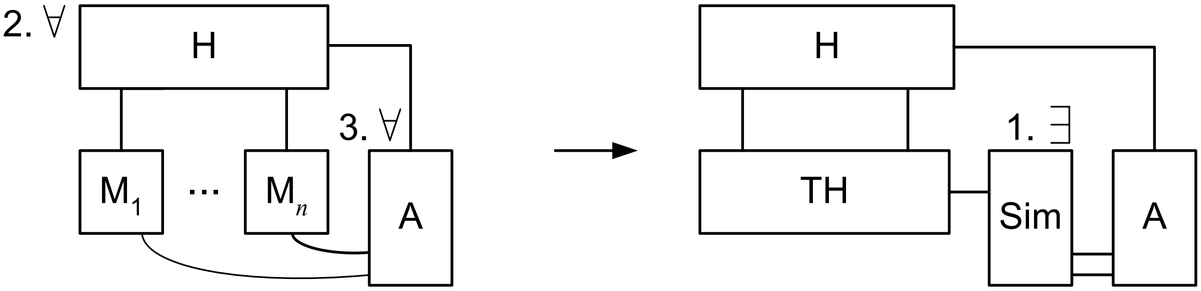

Fig. 1. Overview of blackbox reactive simulatability (BRSIM). A real system is on the left; an ideal system plus simulator on the right. The views of H must be indistinguishable. The quantifiers are numbered to show their order.

The reader may regard the machines, i.e., the individual boxes in Figure 1, as (possibly probabilistic) I/O automata, Turing machines, CSP or pi-calculus processes etc. 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. In [9], the machines are a type of probabilistic I/O automata. We always assume that all parties are polynomial-time.

In the following, the ideal system TH will always be a Dolev-Yao model with XOR and the real system its distributed realization with bitstring XORs. The question is whether the BRSIM relation can be fulfilled between such systems.

4 Main Scenarios for BRSIM/UC Impossibility Results

In this section, we informally describe two scenarios that demonstrate the impossibility of BRSIM/UC soundness for Dolev-Yao models with XOR. We start with a simple scenario, then discuss the assumptions about the Dolev-Yao model and its realization needed in this scenario, and then provide a more complex scenario that needs weaker assumptions. In Section 5 we make the assumptions more precise. In Section 6 we prove the impossilibity of these scenarios with the precise assumptions, and in Section 7 we sketch further scenario extensions to broaden the impossibility results. Readers with a specific Dolev-Yao model in mind should be able to see already in the current section how the scenarios would be expressed in that model, and thus how the impossibility results apply there.

4.1 Scenario with Payload XOR

We first study the scenario in Figure 2. In all our interaction figures, we show a real scenario on the left and an attempted simulation on the right. We write H_u for the actual user with index u, which is a part of the global H in Figure 1.

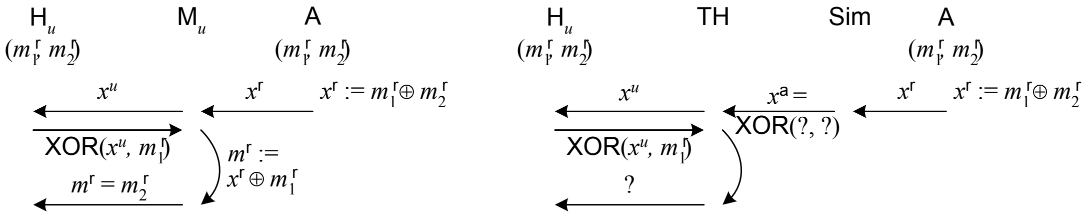
H_u
M_u
A
$(m_1^\mathsf{r}, m_2^\mathsf{r})$
$(m_1^\mathsf{r}, m_2^\mathsf{r})$
x^u
x^r
$x^\mathsf{r} := m_1^\mathsf{r} \oplus m_2^\mathsf{r}$
$\mathsf{XOR}(x^u, m_1^\mathsf{r})$
$m^\mathsf{r} :=$
$x^\mathsf{r} \oplus m_1^\mathsf{r}$
$m^\mathsf{r} = m_2^\mathsf{r}$
H_u
TH
Sim
A
$(m_1^\mathsf{r}, m_2^\mathsf{r})$
$(m_1^\mathsf{r}, m_2^\mathsf{r})$
x^u
$x^\mathsf{a} =$
$\mathsf{XOR}(?, ?)$
x^r
$x^\mathsf{r} := m_1^\mathsf{r} \oplus m_2^\mathsf{r}$
$\mathsf{XOR}(x^u, m_1^\mathsf{r})$
?

Fig. 2. Scenario with payload XOR.

In this scenario, the adversary and the honest user are parametrized with two payloads m_1^r and m_2^r. The superscript r denotes that these are real bitstrings, not abstract terms; we write terms without superscript. The real adversary XORs these two bitstrings and sends the result x^r to the machine M_u, which notifies its user H_u that a message was received, and possibly that it is an XOR. We write the representations of terms/real messages to user H_u with a superscript u; we discuss this below. The user H_u now asks its machine M_u to XOR the received message with the payload m_1^r and to output the resulting payload. Hence H_u obtains m_2^r.

When the simulator Sim tries to simulate this with the ideal system TH, i.e., with the Dolev-Yao model, it also obtains the real XOR x^r from the real adversary, and it is supposed to send a corresponding XOR term using the Dolev-Yao model; we denote this by x^a. (The a means that this is the ideal adversary's representation of the term.) However, at this moment there are many possibilities of what m_1^r and m_2^r could be, even if the simulator knows that this is an XOR of precisely two payloads. Hence if Sim has to select some, with overwhelming probability it will select another pair m_3^r and m_4^r with $m_3^\mathsf{r} \oplus m_4^\mathsf{r} = x^\mathsf{r}$. Now TH, like M_u, notifies user H_u that a message was received, and possibly that it is an XOR. In our scenario, H_u asks TH to XOR the received message with the payload m_1^r and to output the resulting payload. If TH acts purely symbolically on the term $\mathsf{XOR}(m_3, m_4)$, it obtains $\mathsf{XOR}(m_3, m_4, m_1)$. For indistinguishability between the real and ideal system, the actual output should, however, be m_2^r. This clearly requires that TH evaluates the real, non-algebraic XOR on the bitstrings corresponding to the payload terms m_3, m_4, and m_1. This is not what a standard Dolev-Yao model would do.

If the simulator, instead of inputting the term x as the XOR of two guessed payloads, can input it as an "unknown XOR", as indicated by the question marks in Figure 2, the simulation fails even more clearly: Now TH has no information at all that the XOR of the terms x and m_1 is equivalent to m_2.

4.2 Discussion of the Scenario with Payload XOR

First note that the scenario with payload XOR in Section 4.1 is valid even if the "protocol" used on top of the Dolev-Yao model is fixed and known to the simulator: The protocol in this case is simply that H_u knows two payloads (or the protocol machine H_u accepts them from a "real user" H'_u, who in turn knows them from somewhere), receives a message from the adversary, XORs it with one of the payloads and tests that the result is the other payload. The knowledge of this does not help Sim above.

Secondly, we assumed that payloads can be random bitstrings and are used without additional redundancy in real XORs. Very strong redundancy might allow Sim to parse x^r uniquely into the two payloads m_1^r and m_2^r. However, this would be a highly unusual class or encoding of payloads. As long as there exist four different payloads such that, in their encoding before the application of XOR, we have $m_1^\mathsf{r} \oplus m_2^\mathsf{r} = m_3^\mathsf{r} \oplus m_4^\mathsf{r}$, the scenario is still a valid counterexample because the simulator will still be wrong with significant probability, unless we allow the simulator to input a term representation x^a that contains all possible parsings, and assume it is feasible to find them.

Finally, the relation between real payloads and payload terms needs a discussion. In typical Dolev-Yao models, different payloads are a priori abstracted to atoms of the term algebra, e.g., two emails m_1^r and

m_2^{r} become two atoms m_1 and m_2. When we plug such a Dolev-Yao model into an overall real system, where payloads may have complex application semantics so that we cannot simply make them atoms, we therefore assume that the full ideal functionality TH maintains a translation table between the real payloads that occur in a system execution and the corresponding Dolev-Yao terms. This is why we wrote the user inputs and outputs as m_1^{r} and m_2^{r}. If they were just atoms m_1 and m_2, the impossibility in the scenario above would be even clearer: Either Sim would not even have the atoms m_1 and m_2, or with probability at least one half it would select a wrong pair (m_3, m_4) of atoms, and then a later derivation that $\mathsf{XOR}(m_3, m_4, m_1)$ equals m_2 is impossible.

When considering soundness in the sense of BRSIM/UC, we have an even stronger motivation for also considering real payloads even if the core Dolev-Yao model abstracts from them: In the realization, the real payload bitstrings from the users are encrypted, XORed, etc. Hence they must be input and output. For indistinguishability, the inputs and outputs of the ideal system TH must be syntactically the same. Syntactically different user interfaces would either simply prevent the same users from using alternatively the real or the ideal system, or lead to trivial distinguishability. Hence if the Dolev-Yao model has a different "core" representation of payloads, there must be a translation between the input/output representation and this "core" representation. This holds for all definition variants of BRSIM/UC.

However, once we assume such a translation for real payloads, one might argue that augmenting it from pure table lookup to performing bitstring XORs, such as of m_3^{r}, m_4^{r} and m_1^{r} above, is not a huge addition and not "impossible" for Dolev-Yao models. Hence we show more complex scenarios in the following where TH would not only have to evaluate bitstring XORs, but also real cryptographic operations to make a simulation possible. Thus TH would even more obviously not be a Dolev-Yao model.

4.3 Scenario with Signature Computation

The scenario in Figure 3 additionally exploits a signature operator S as a cryptographic operation that a Dolev-Yao model should only evaluate symbolically. We assume that the honest party u already has a pair (sks, pks) of a secret signing key and a public test key, and that the test key was published, so that the real adversary knows the real public key pks^{r}. We assume that payloads have no redundant encoding.

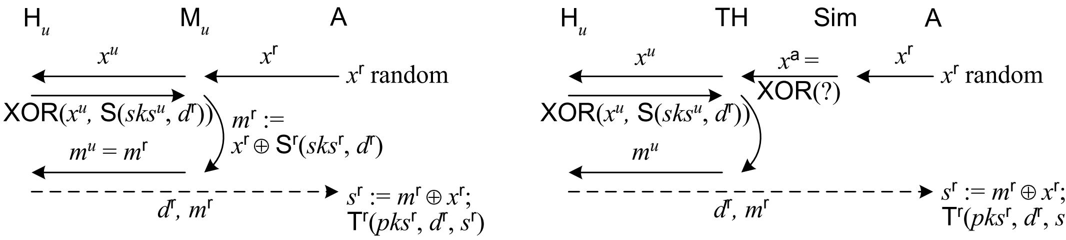
H_u
M_u
A
x^u
x^{r}
x^{r} random
XOR(x^u, S(sks^u, d^{r}))
$m^{\mathsf{r}} :=$
$x^{\mathsf{r}} \oplus \mathsf{S}^{\mathsf{r}}(sks^{\mathsf{r}}, d^{\mathsf{r}})$
$m^u = m^{\mathsf{r}}$
d^{r}, m^{r}
$s^{\mathsf{r}} := m^{\mathsf{r}} \oplus x^{\mathsf{r}}$;
$\mathsf{T}^{\mathsf{r}}(pks^{\mathsf{r}}, d^{\mathsf{r}}, s^{\mathsf{r}})$
H_u
TH
Sim
A
x^u
$x^{\mathsf{a}} =$
XOR(?)
x^{r}
x^{r} random
XOR(x^u, S(sks^u, d^{r}))
m^u
d^{r}, m^{r}
$s^{\mathsf{r}} := m^{\mathsf{r}} \oplus x^{\mathsf{r}}$;
$\mathsf{T}^{\mathsf{r}}(pks^{\mathsf{r}}, d^{\mathsf{r}}, s^{\mathsf{r}})$

Fig. 3. Scenario with signature computation.

The real adversary sends a random string x^{r} to the machine M_u, which notifies its user H_u that a message x^u has arrived. Then H_u asks its machine M_u to XOR the received message x^u with his or her signature on a payload d^{r} and to output the resulting payload m^{r}. As we assume that payloads have no redundant encoding, the machine M_u does not recognize that m^{r} is not an "original" payload. The user H_u and the adversary A use the real signature test algorithm T^{r} to verify that the bitstring $s^{\mathsf{r}} := m^{\mathsf{r}} \oplus x^{\mathsf{r}}$ is a valid signature on d^{r} with respect to the user's public key pks^{r}.

For indistinguishability, in the simulation TH must also output a bitstring m^{r} with this property. However, this intuitively means that TH can compute a cryptographic signature and is thus not a Dolev-Yao model. More precisely, the simulator, upon getting x^{r} from A, can essentially only input to TH that an unknown XOR is sent. If this is the only input, then TH does not even have enough information to compute a correct m^{r} (but if we assume this we could take the simpler scenario from Section 4.1). If Sim additionally inputs the real bitstring x^{r} somehow in the representation x^{a}, e.g., by claiming that the term x is the XOR of just this one payload x^{r}, then we show by a reduction proof that it is actually

TH that does the main work in computing the signature s^{r}. We will do this after making more precise assumptions.

5 Common Assumptions for our Main Scenarios

As explained in the introduction, we want to show that it is not possible to securely implement *any* Dolev-Yao model by *any* natural realization of XOR in the sense of BRSIM/UC. In order to turn this informal meta-theorem into real theorems, we need assumptions on what characterizes a Dolev-Yao model, a model of XOR in it, and a real implementation of such a model. We obviously need *some* such assumptions: The notion of BRSIM/UC is reflexive. Thus, if an arbitrary Dolev-Yao model with XOR also counted 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 also counted as a Dolev-Yao model. But this is not what we want. To make our results as strong as possible, we only make minimal assumptions. We start with the basic notions of terms, including an XOR operator.

Definition 1 (Terms of a Dolev-Yao Model with XOR). *We require that we can derive definitions of the following concepts from a Dolev-Yao model with XOR:*

a. *A set Terms denoting the overall set of valid terms. We speak of* atoms *and* operators *denoting the potential leaves and inner nodes, respectively, of the terms considered as trees. The terms, atoms and operators may be typed. There is an equivalence relation "$\equiv$" on Terms. We call $(Terms, \equiv)$ the term algebra.*[3]
b. *An operator* XOR *that is commutative, associative, and where each element is an inverse of itself. More precisely, we require* $\mathsf{XOR}(t_1, t_2) \equiv \mathsf{XOR}(t_2, t_1)$ *and* $\mathsf{XOR}(t_1, \mathsf{XOR}(t_2, t_3)) \equiv \mathsf{XOR}(\mathsf{XOR}(t_1, t_2), t_3)$ *and that* $\mathsf{XOR}(t_1, t_1)$ *is a neutral element (for XOR) for all* $t_1, t_2, t_3 \in Terms$*. (The neutral elements correspond to all-zero strings; a Dolev-Yao model might represent only one all-zero string, or one of every possible length.)*
c. *A set XORable_Terms $\subseteq$ Terms of the terms that are valid operands of the operator* XOR*.*
d. *A list operator (possibly implemented by repeated pairing in the original syntax). Two lists are equivalent iff all their corresponding elements are.*

◇

Next we define some minimum actions that the users and the adversary can carry out on the terms, and the results of these actions. In our context, this is the basis for showing that our impossibility scenarios are at least executable in every Dolev-Yao model (which was hopefully intuitively clear).

We already used the notation t^u for the representations of a term t for a user with index u in the informal scenarios. While this notation is certainly more general than notions that may be familiar to some readers, and thus can only strengthen our impossibility results, let us briefly motivate how it relates to such notions: An important concept in Dolev-Yao models is that of terms t constructible for some participant u or the adversary (by applying operators and cancellation rules to previously known messages); however, the syntax for this concept varies considerably. Some high-level representations, such as the typical arrow pictures, simply use t itself in the protocol representations, e.g., "$\mathsf{E}(pke, m)$" even when someone who does not know m forwards this encryption term. More detailed representations typically use the concepts of variables inherent to the underlying formal protocol languages. A usual case is to match received messages with a pattern describing the expected message format, and then to use the pattern variables in subsequent message constructions. To the best of our knowledge, the first explicit such protocol representation was the PROLOG message derivation in the Interrogator [72], while pattern matching of an existing calculus was first used for CSP and FDR in [73]. The syntax of the Dolev-Yao model with BRSIM/UC soundness in [7] uses local variables called handles and explicit parsing of received messages. The syntax from all these models can easily be mapped to that in our following definition.

We say that a user u "has" a term representation if it has learned or constructed it. We do not need a full definition of how this learning and constructing is done; only the obvious parts that users

[3] Clearly syntactic term equality "=" implies equivalence. Typically "$\equiv$" is constructed from cancellation rules.

learns received terms can XOR them. Furthermore, we define that terms can be sent and that the ideal adversary controls the network as usual in Dolev-Yao models. Furthermore we require that users can XOR terms.

Definition 2 (Actions on a Dolev-Yao Model with XOR). *Users and the ideal adversary can make at least the following inputs into the ideal functionality of a Dolev-Yao model with XOR, with the described results.*

a. If an honest user H_u *inputs* $\mathsf{send}(v, t^u)$ *for a term representation* t^u*, this leads to an output* $\mathsf{receive}(u, v, t^{\mathsf{a}})$ *for the adversary.*

b. If the adversary inputs $\mathsf{send}(u, v, t^{\mathsf{a}})$ *for a term representation* t^{a}*, this leads to an output* $\mathsf{receive}(u, t^v)$ *for user* v *(i.e., the adversary impersonates* u*), and outputs of this format only occur upon such inputs. After such an output, user* v *"has" the term representation* t^v*.*

c. If a user with index u *(honest or the adversary represented by* $u = \mathsf{a}$*) has term representations* t_1^u *and* t_2^u *for* $t_1, t_2 \in$ *XORable_Terms, then it also has a representation for the term* $\mathsf{XOR}(t_1, t_2)$*. (Typically this is something like the string "*$\mathsf{XOR}(t_1^u, t_2^u)$*".)* ◇

We already discussed in Section 4.2 that payloads are application data, and that the interface between the users and the real or ideal system must be able to pass these bitstrings through so that the real system can encrypt real bitstrings etc. We make the following minimum assumption about this setting.

Definition 3 (Payloads in Dolev-Yao Models). *A* Dolev-Yao model with payloads *allows us to derive a type (subset)* $\mathsf{payload}$ *in the set Terms. In every execution, every occurring payload term* m *has a fixed realization* m^{r}*, and* $m^{\mathsf{r}} = m'^{\mathsf{r}}$ *implies* $m \equiv m'$*. For an arbitrary but fixed polynomial* plen *we can assume that the range of payload realizations* m^{r} *contains at least all bitstrings up to the length* $\mathsf{plen}(k)$*, where* k *is the cryptographic security parameter. A real payload* m^{r} *can always be used as an input representation* m^u *by user* u*, and a user* u *who has any payload representation* m^u *can ask to have the corresponding payload* m^{r} *output; then this happens without outputs to other parties, in particular the ideal adversary.* ◇

A general characteristics of real systems is that they are distributed. This means that each participant u has its own machine, here called M_u, and the machines are only connected by channels that offer well-defined possibilities for observations and manipulations by a real adversary. Specifically for the realization of Dolev-Yao models with XOR, we make the following (natural) minimum assumptions: Real channels are insecure; the input to send a term t leads to the actual sending of a bitstring t^{r}; and XOR terms are realized by applying an actual XOR to the realization of the contained terms.

Definition 4 (Pure Realization of a Dolev-Yao Model with XOR). *In a* pure realization of a Dolev-Yao model with XOR*, an input* $\mathsf{send}(v, t^u)$ *to a machine* M_u *releases a bitstring* t^{r} *to the real adversary such that within one execution of the system* $t \equiv t' \Rightarrow t^{\mathsf{r}} = t'^{\mathsf{r}}$ *for all terms* t, t'*. We have* $(\mathsf{XOR}(t_1, t_2))^{\mathsf{r}} = t_1^{\mathsf{r}} \oplus t_2^{\mathsf{r}}$ *for all* $t_1, t_2 \in$ *XORable_Terms.* ◇

We wrote "pure" in this definition as Dolev-Yao model realizations profit from type tags, see [74, 7]. For readability we left this option out of the core definition, as with XORs one must be careful how to apply type tags so that the desired algebraic properties still hold. Essentially one has to normalize before type tagging, see Section 8.7 for a concrete version. We sketch in Section 7.1 how the impossibility proofs extend to this case.

We did not define any ideal secrecy of XORs here, i.e., that an ideal adversary learning certain XORs cannot derive the individual xored terms. While one would expect this for positive results, our impossibility results do not rely on such ideal secrecy.

6 Rigorous Impossibility Proofs

In this section we reconsider the scenarios from Section 4 under the rigorous definitions. We start with the scenario with payload XOR from Figure 2. Here we need more assumptions for proving impossibility than the common assumptions defined in Section 5, in particular because the common assumptions do not exclude the discussed possibility that the lookup-translation between real payloads and opaque payload terms is augmented by an actual XOR computation in the ideal system. In contrast, the proof for the more complex scenario with signatures will not need such additional assumptions (only a simple assumption that the Dolev-Yao model contains a signature operator).

6.1 Additional Assumptions and Proof for the Scenario with Payload XOR

Intuitively, in the scenario with payload XOR in Section 4.1 we postulated that the Dolev-Yao model performs no bitstring computations that would allow it to recognize the term x as the XOR of m_1 and m_2 at the end if the ideal adversary, here the simulator, has not guessed these messages a priori. We define this as follows.

Definition 5 (Dolev-Yao Model without Bitstring XOR). *A Dolev-Yao model with XOR and payloads is called* without bitstring XOR *if the following holds: If an input* $\mathsf{send}(u, v, x^{\mathsf{a}})$ *by the ideal adversary at a time T leads to an output* $\mathsf{receive}(u, x^v)$ *for user* H_v *with a term* $x \equiv \mathsf{XOR}(m_1, m_2)$ *such that* $m_1, m_2 \in \mathsf{payload}$ *and* $m_1 \neq m_2$*, and if the terms m_1 and m_2 were not present by themselves or as subterms in* TH *before time T, then the input representation x^{a} explicitly contains the payloads* m_1^{r} *and* m_2^{r}*.* ◇

This is quite normal for Dolev-Yao models if one has accepted that TH, in addition to the core Dolev-Yao model, contains the translation table between real payloads and their representations: Sim might enter terms like $\mathsf{D}(\mathsf{E}(\mathsf{XOR}(m_1, m_2)))$ or $\mathsf{XOR}(\mathsf{XOR}(m_1, m_3), \mathsf{XOR}(m_2, m_3))$ instead of $\mathsf{XOR}(m_1, m_2)$, but m_1 and m_2 have to occur somewhere explicitly as long as no prior terms such as $t = \mathsf{XOR}(m_1, m_3)$ are available where a representation t^{a} might simply be a local variable "t". We could also extend this definition to a multi-step interaction between the ideal adversary and TH if we allowed inputs other than $\mathsf{send}(u, v, x^{\mathsf{a}})$ by the ideal adversary; we omit this extension for readability.

Theorem 1. *A Dolev-Yao model without bitstring XOR (Definition 5) and with* $\mathsf{payload} \subseteq$ *XORable_Terms does not have a pure realization (Definition 4) that is sound in the sense of BRSIM/UC.* □

Proof. Assume the contrary for a Dolev-Yao model TH and a realization. Definitions 1 to 4 imply that the users and the real adversary can carry out the scenario from Figure 2 with these systems. More precisely, the statement that A and H_u are parametrized with messages m_1^{r} and m_2^{r} means that we consider a family of honest users $\mathsf{H}_{u,m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$ and adversaries $\mathsf{A}_{m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$, where m_1^{r} and m_2^{r} are arbitrary payloads of a fixed length $l \geq 1$ with $m_1^{\mathsf{r}} \neq m_2^{\mathsf{r}}$. (This is possible by Definition 3.) Furthermore, these definitions imply that in the real system, the result from M_u for $\mathsf{H}_{u,m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$ is indeed $m^{\mathsf{r}} = m_2^{\mathsf{r}}$.

Thus for BRSIM/UC soundness, the simulator Sim has to achieve that the ideal system TH also outputs $m^u = m^{\mathsf{r}} = m_2^{\mathsf{r}}$ with overwhelming probability, because every other output would be distinguishable for $\mathsf{H}_{u,m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$. The corresponding term m is derived in TH as $m := \mathsf{XOR}(x, m_1)$. Hence $m^{\mathsf{r}} = m_2^{\mathsf{r}}$ implies $m_2 \equiv m \equiv \mathsf{XOR}(x, m_1)$ with Definition 3, and thus $x \equiv \mathsf{XOR}(m_1, m_2)$ with the algebraic properties of XOR (Definition 1). Here x is the term sent by the ideal adversary; by Definition 2 there must be exactly one such sending action. Furthermore, $m_1^{\mathsf{r}} \neq m_2^{\mathsf{r}}$ implies $m_1 \neq m_2$ with Definition 4. Hence by Definition 5, the ideal adversary cannot achieve this equivalence without inputting the actual payloads m_1^{r} and m_2^{r} within the representation x^{a}. The definition is applicable in this scenario because no other term containing m_1 and m_2 is available in TH yet.

However, Sim is fixed while we consider the family of users $\mathsf{H}_{u,m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$ and adversaries $\mathsf{A}_{m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$, and it only obtains the input $x^{\mathsf{r}} = m_1^{\mathsf{r}} \oplus m_2^{\mathsf{r}}$ from $\mathsf{A}_{m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$. Hence for every x^{r} and security parameter k, Sim has to choose its guess at m_1^{r} and m_2^{r} with a fixed distribution $D_{x^{\mathsf{r}},k}$. For every such distribution, at least one pair $(m_1^{\mathsf{r}}, m_2^{\mathsf{r}})$ has probability at most 2^{-l}, because by Definitions 3 and 4, all bitstrings of length l are possible payloads. Hence there exists a pair $(m_1^{\mathsf{r}}, m_2^{\mathsf{r}})$ that has probability at most $2^{-l} \leq 1/2$ for infinitely many values of k (recall that l is a constant, and can actually be very small). Thus for the corresponding user $\mathsf{H}_{u,m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$ and adversary $\mathsf{A}_{m_1^{\mathsf{r}},m_2^{\mathsf{r}}}$, the probability that the simulation is correct is at most $1/2$ for infinitely many k. This is not overwhelming. Therefore BRSIM/UC soundness is not possible. ■

6.2 Reduction Proof for the Scenario with Signatures

Our second scenario uses signatures. However, we do not need any cryptographic properties, but simply exploit signing as an operator whose realization is typically of significant complexity, so that we can argue that an ideal system TH that evaluates the *realization* of this operator is not a Dolev-Yao model. We therefore make a minimal definition without even mentioning the test operator. For notational simplicity, we also make this definition only for deterministic memory-less signature schemes. This is no significant restriction in particular as our scenario needs only one signature; moreover we sketch afterwards why the proof also holds in the general case.

Definition 6 (Minimal Signatures in a Dolev-Yao Model). *A Dolev-Yao model with XOR and a realization according to Definitions 1, 2 and 4 is called* with simple signatures *if it has an operator* S *with two parameters where the first denotes the secret key and the second the signed term such that* $(\mathsf{S}(sks, t))^{\mathsf{r}} = \mathsf{S}^{\mathsf{r}}(sks^{\mathsf{r}}, t^{\mathsf{r}})$ *for the signing algorithm* S^{r} *of a secure cryptographic signature system whenever the term* $\mathsf{S}(sks, t)$ *is well-defined. Furthermore, it must be possible to publish corresponding public keys* pks *in the Dolev-Yao model such that in the realization* $(sks^{\mathsf{r}}, pks^{\mathsf{r}})$ *are a key pair of the same signature system.* ◇

By a secure cryptographic signature scheme we mean one according to the definition from [75]. For the following theorem we only need the simplest part of this definition: Signatures correctly made with a secret key sks^{r} pass the test with the corresponding public key pks^{r}; we call this "valid with respect to pks^{r}". We only use the security in the complexity arguments after the theorem. We assume without loss of generality that for a given security parameter k and a given message length l, all signatures have a fixed length $\mathsf{slen}(k, l)$.

We now state precisely what we prove with the scenario from Figure 3.

Theorem 2. *Let* TH *be a Dolev-Yao model with XOR, payloads, and simple signatures and with a realization that is secure in the sense of BRSIM/UC (Definitions 1 to 4 and 6), and let payloads and signatures be in the set XORable_Terms. Then* TH *can be used to compute a real signature on an arbitrary message of length* k *with respect to the signature scheme used in the given realization. The reduction algorithm* Sig *that computes a signature using* TH *as a subprogram only needs time linear in* $\mathsf{slen}(k, k)$*, the signature length for messages of length* k*. (Here we do not count the time that* TH *needs.)* □

This theorem shows that a machine TH that offers the external functionality of a Dolev-Yao model with XOR and has a secure cryptographic realization cannot be what one would intuitively call a Dolev-Yao model. For instance, if it has *any* secure realization with a signature scheme where the signatures are reasonably short, concretely where $\mathsf{slen}(k, k)$ is at most linear in k, but signing takes time of the order of at least k^2, then TH performs the bulk of the signature computation in the reduction. We chose the message length k for simplicity and because typically the bulk of a signature computation only happens on one hash value; we could argue similarly with other message lengths.

Proof. (Of Theorem 2.) Let the preconditions of the theorem be true. The definitions imply that the honest user H_u and the real adversary A can carry out the scenario from Figure 3 with the given Dolev-Yao model and realization, including the initial key generation and publication that is only described in the text in Section 4.3. More precisely, we assume that H_u consists of a fixed protocol component H'_u that receives its payload input d^{r} of length k from a second component H^*_u (typically a higher level protocol or a human), and that A chooses x^{r} randomly among the bitstrings of length $\mathsf{slen}(k, k)$. The definitions imply that in the real system, the bitstring $s^{\mathsf{r}} := m^{\mathsf{r}} \oplus x^{\mathsf{r}}$ (computed with the output m^{r} of M_u) is indeed a signature on d^{r} valid with respect to pks^{r}.

Thus the assumed successful simulator Sim must also achieve that TH outputs a value m^{r} to H_u such that the resulting value s^{r} is a valid signature on d^{r} with overwhelming probability, because otherwise A and H_u together (via external communication) can distinguish the interaction with TH and Sim from the interaction with the real machine M_u.

We now construct a machine Sig with TH as a blackbox that carries out key generation and signs one message, i.e., Sig is our reduction algorithm. While Sig also uses the other participants of the ideal system as blackboxes, the steps of all those have to be counted within the complexity of what Sig does itself.

- *Key generation.* Initially Sig runs the honest user component H'_u and the simulator Sim for generating a real signature key pair $(sks^{\mathsf{r}}, pks^{\mathsf{r}})$ and publishing pks^{r} to A. Our machine Sig publishes this key pks^{r} as the key for which it will make a signature. It further generates a random string x^{r} as A

would and resumes running Sim to produce the ideal version of this message that it passes to TH in a message $\mathsf{send}(v, u, x^{\mathsf{a}})$. By Definition 2 there must be exactly one such sending action. In response TH gives Sig (here in the role of the user component H'_u) an output $\mathsf{receive}(v, x^u)$.

- *Computing one signature.* When Sig is asked to sign a payload d^{r} (with respect to its only keypair), it asks TH (as H'_u would do) to XOR the term represented by x^u with a signature on the payload d^{r} and to output the resulting payload. It waits for the output m^{r} from TH and outputs $s^{\mathsf{r}} := m^{\mathsf{r}} \oplus x^{\mathsf{r}}$ as its signature.

It follows immediately from the initial discussion about the result of Sim and TH that the output s^{r} of Sig is indeed a signature on d^{r} valid with respect to pks^{r} with overwhelming probability.

In the signing phase (the only phase for which we claim a highly efficient reduction), Sig does only two things itself: First it makes the user input that requests the computation of a signature, an XOR, and the output of the resulting payload. This is the input of a fixed small term or program-like string, where even the representation x^u is already fixed at least since the key generation. Later Sig computes a real XOR. This needs time linear in the signature length $\mathsf{slen}(k, k)$, as x^{r} was chosen of the same length and consequently m^{r} is also of this length.[4] This finishes the reduction proof that using TH plus only computation linear in the length of a signature, we can compute a cryptographic signature (of whatever signature system is used in the realization). ■

In this proof, the signature scheme in the realization could also be probabilistic and/or with memory: Only one signature is ever computed, and we only argue that this signature passes the test. To rigorously cover this case, we would have to adapt Definition 6 such that the signature terms in the ideal system have an additional tag, e.g., a counter as in [7].

7 Scenario Extensions

We have now proved, using certain scenarios as counterexamples, that under reasonable, common assumptions about Dolev-Yao models and their realizations, the realizations cannot be BRSIM/UC sound. Even if we stick to the strong BRSIM/UC soundness with its general composability, this still leaves some options for positive results if we give up some of the assumptions. Three possibilities in particular come to mind: First, change the XOR realization or the representation of payloads within the realization to include some additional redundancy, at least type tags. Secondly, restrict the users H to certain protocol classes. Thirdly, consider Dolev-Yao models without payloads, as our two main scenarios were based on payloads, and there are protocol classes without payloads, or at least without general payloads from large real domains. We now present extended scenarios that are still impossible after certain concrete instantiations of these changes.

7.1 Extensions for Low Redundancy in Payloads or XORs

So far we assumed that payloads can be arbitrary bitstrings, and that they are used in the realization in their original form, and that the XOR operator is implemented directly as the real $\oplus$. However, at least type tags are not unknown in realizations of Dolev-Yao models, and there might be other forms of structured low redundancy, such as payloads encoded in ASCII or XML.

The scenario with payload XOR (Figure 2) is essentially unaffected by such changes: In the real system, x^{r} has an xor tag. XORing it with m_1 means that the real operation $\oplus$ is applied to the untagged part of x^{r} and the full m_1^{r}. In general, the result would get an xor tag again, but here the result is output as a payload, and thus without the xor tag. As to the payloads, tags in a fixed place make no difference to the proof. The effect of redundancy in the payloads themselves was already discussed in Section 4.2.

For the scenario with signature computation (Figure 3), we have to be more careful with redundancy, because the real machine M_u might usually output an error $\downarrow$ instead of m^{r} if m^{r} is not of the correct

[4] By making x^{r} shorter, we could achieve that Sig computes the XOR of even fewer bits, but Sig still at least has to output the entire signature. Therefore we chose a scenario that remains valid in protocol classes that only allow XORs of strings of equal length.

payload format. Nevertheless, as long as the valid payload strings are not negligible within the family of sets of all strings of length l (where l is the index of the family), then m^{r} is still output with not negligible probability, because it is uniformly random (a fresh one-time pad x^{r} XORed with a fixed string). Thus TH can still be used in the reduction proof to compute actual signatures with not negligible probability, which, intuitively, contradicts the Dolev-Yao property.

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. Then the scenario with its random choice of x^{r} no longer works. However, a similar attack works for many realistic cases: Assume that a subset $\mathit{Fixbits}_{\mathsf{payload},l}$ of the bits of the encoded payloads is fixed (e.g., the opening and closing XML tags), and similarly $\mathit{Fixbits}_{\mathsf{sig},l}$ 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 $\mathit{pks}^{\mathsf{r}}$. Now if $\mathit{Fixbits}_{\mathsf{sig},l} \supseteq \mathit{Fixbits}_{\mathsf{payload},l}$, the adversary A can predetermine the necessary bits of m^{r} in x^{r} by XORing them with the corresponding fixed bits of a signature.

7.2 Extensions for Protocol Restrictions

If every permitted global H (representing the entirety of all users) consists of a protocol *prot* from a restricted class *Prots* and a user H' of *prot*, then a scenario only remains a valid counterexample if there is a protocol in *Prots* where one honest party acts like H_u in the scenario.

For the scenario with payload XOR (Figure 2) it is easy to see for a given class *Prots* whether this is true, or can be adapted. Essentially, if XORs contain at most one payload in this protocol class, the scenario cannot be carried out. If the protocol class allows XORs where two components are payloads, it seems reasonable that the protocol class also allows users to subtract one payload from such an XOR and to retrieve the remaining payload; then the scenario works.

For the scenario with signature computation (Figure 3), a simple protocol where party u acts as in the scenario can be written as follows in the typical high-level arrow notation for simple security protocols, and where d and *sks* are secrets known to parties u and v.

$$\begin{aligned} v \to u :&\ m \oplus \mathsf{S}(\mathit{sks}, d);\\ u :&\ \text{Output } m. \end{aligned}$$

This protocol only makes sense for deterministic signature schemes so that party u subtracts the same real signature that v added. As party u (both the user and the machine acting for it) cannot know whether party v really started this protocol, it applies its protocol step whenever it gets a message x^u supposedly from v. Thus it acts as in Figure 3. If a protocol class does not allow secret signing keys to be known to two parties, one alternative is to use a symmetric primitive instead of the signature. For making this extension rigorous, one needs a definition of the symmetric primitive used similar to Definition 6. Another alternative is to add an initial step where the signature is exchanged in encrypted form; now we can also use a probabilistic signature scheme again because v reuses a signature computed by u.

$$\begin{aligned} u \to v :&\ \mathsf{E}(\mathit{pke}_v, \mathsf{S}(\mathit{sks}_u, d));\\ v \to u :&\ m \oplus \mathsf{S}(\mathit{sks}_u, d);\\ u :&\ \text{Output } m. \end{aligned}$$

Here pke_u denotes the public encryption key of party v and sks_v the secret signing key of party u. For making this extension rigorous, one needs a definition of of public-key encryption including the requirement that an encrypted message is secret in the ideal system, except possibly its structure and length. Then the ideal adversary in this extension does not learn the signature and the signed message d. Hence nor does the simulator, and thus, when the simulator constructs x^{a}, it is essentially in the same position as before.

7.3 Scenario without Payloads

Finally, we consider a Dolev-Yao model without any payloads, or a realization where payloads have or are encoded with significant redundancy, so that none of the prior scenarios works. In particular, we

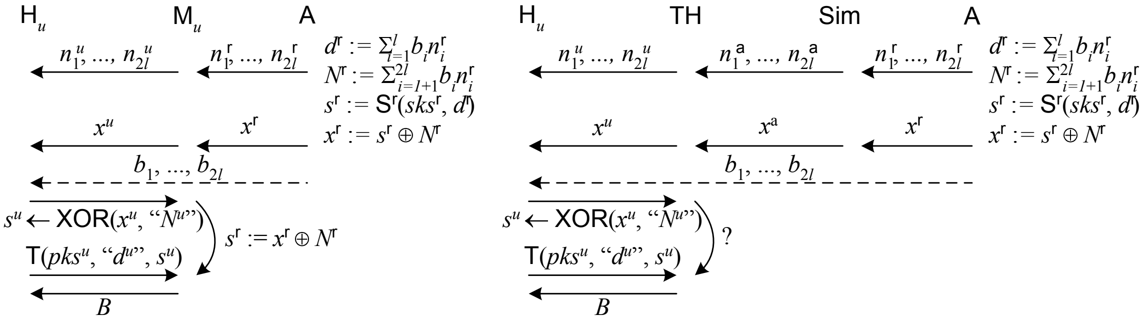

Fig. 4. Scenario without payloads.

can no longer cause M_u to output a significant bitstring to H_u, like the former m^{r}. Hence we only prove a reduction where TH essentially tests a signature instead of computing one; the reduction is also less efficient. The scenario is shown in Figure 4. We use nonces, another typical data type in Dolev-Yao models. They correspond to random or pseudorandom values in typical realizations. We assume that the nonce realizations can be arbitrary bitstrings of a length l suitable to hide a signature in an XOR. (In a tagged version the nonce tag should not extend beyond the signature tag; we now argue with an untagged version for readability.)

In the real system, we assume that the adversary A already has a secret signature key sks^{r} and has published the corresponding public key pks^{r}. Now A randomly chooses nonces n_i^{r} for $i = 1, \ldots, 2l$, repeating each choice until the first l nonces are linearly independent, and so are the second l nonces. Next A sends the nonces to the machine M_u, which notifies its user H_u with representations n_i^u of these nonces. The adversary also chooses random bits $b_1, \ldots, b_{2l}$ and computes the linear combinations $d^{\mathsf{r}} := \sum_{i=1}^{l} b_i n_i^{\mathsf{r}}$ and $N^{\mathsf{r}} := \sum_{i=l+1}^{2l} b_i n_i^{\mathsf{r}}$ of the nonces. Both d^{r} and N^{r} are random values if one does not know the bits b_i. The adversary computes a signature s^{r} on d^{r} with the secret key sks^{r} and sends $x^{\mathsf{r}} := s^{\mathsf{r}} \oplus N^{\mathsf{r}}$ to M_u, which notifies its user H_u with a representation x^u of this message.

Now the adversary sends the bits $b_1, \ldots, b_{2l}$ to the user H_u outside the system, corresponding to a chosen-message attack. User H_u asks its machine M_u to subtract the appropriate nonces from x^u, i.e., each n_i^u with $i > l$ and $b_i = 1$. This is abbreviated by "N^u" in Figure 4. User H_u then asks M_u to test whether the result s^u is a correct signature on the message obtained by XORing the other appropriate set of nonces, i.e., each n_i^u with $i \leq l$ and $b_i = 1$. This is abbreviated by "d^u" in Figure 4. Alternatively, the adversary does the same with an incorrect signature s'^{r}. The real machine M_u always decides this correctly; we denote its output by a Boolean value B.

Now assume a simulator Sim correctly simulates this scenario. Thus Sim must achieve that the Dolev-Yao model TH makes the same correct output B to H_u after the same interactions with A and H_u with overwhelming probability.

Intuitively, we show that TH must be able to test cryptographic signatures for this. However, we are not aware of a prior definition of what it means that an algorithm tests signatures correctly with overwhelming probability (in contrast to always), i.e., in what probability space over messages and signatures this must be true. For instance, if we input random values as potential signatures, the algorithm may be correct with overwhelming probability by always outputting false because the signatures may be sparse. 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 algorithm can usually make the distinction. We deal with this problem as follows: We allow a second, arbitrary (probabilistic polynomial-time) algorithm F (for "fake") that tries to fake signatures, given a public key. It must always output invalid signatures. Let $\mathcal{F}$ be the set of such algorithms. Intuitively a good algorithm F makes its fakes as plausible as possible. For instance, for RSA signatures with additional tags and a field for the signed data, it might set these tags and the signed data correctly, and choose the rest randomly from the correct mathematical group.

Definition 7 (Signature Test Approximation). *Given a signature system* $(\mathsf{G}^{\mathsf{r}}, \mathsf{S}^{\mathsf{r}}, \mathsf{T}^{\mathsf{r}})$*, a pair of algorithms* $\mathsf{TA} = (\mathsf{TA}_1, \mathsf{TA}_2)$ *is a* test approximation *if the following holds for all signature-faking algorithms*

$\mathsf{F} \in \mathcal{F}$:

$$\begin{aligned}\mathsf{Pr}[b^* \neq b :: (sks^\mathsf{r}, pks^\mathsf{r}) \leftarrow \mathsf{G}^\mathsf{r}(1^k); v^\mathsf{r} \leftarrow \mathsf{TA}_1(pks^\mathsf{r});\\ (m^\mathsf{r}, s_0^\mathsf{r}) \leftarrow \mathsf{F}(pks^\mathsf{r}); s_1^\mathsf{r} \leftarrow \mathsf{S}^\mathsf{r}(sks^\mathsf{r}, m^\mathsf{r});\\ b \in_\mathcal{R} \{0,1\}; b^* \leftarrow \mathsf{TA}_2(pks^\mathsf{r}, m^\mathsf{r}, s_b^\mathsf{r}, v^\mathsf{r})]\\ \in NEGL.\end{aligned}$$

Here NEGL is the set of negligible functions and $\in_\mathcal{R}$ is the random uniform choice from a set. The notation $\mathsf{Pr}[E :: A]$ means the probability of the event E in the probability space defined by the probabilistic algorithm A. ◇

Splitting TA into two algorithms TA_1 and TA_2 allows us to reason separately about the complexity of pre-computations given only the public key pks^r, and of the algorithm for distinguishing a fake and a real signature. The precomputations are done by TA_1, which outputs its result as an intermediary value v^r. Later TA_2 tries to distinguish a real or fake signature, using v^r as an additional input.

As a reduction proof, we construct a test approximation TA with the given TH as a blackbox. It also uses Sim as a blackbox and simulates the actions of H_u and A, but those steps count among the additional complexity of TA. In the key distribution phase, TA_1 obtains a public signature key pks^r from A and runs the actions of Sim on this key together with TH. It then runs A and Sim choosing $2l$ nonces together with TH. It also computes the inverses M_1^{-1} and M_2^{-1} of the matrices M_1 and M_2 constructed from the nonces $n_1, \ldots, n_l$ and $n_{l+1}, \ldots, n_{2l}$, respectively. This will later allow for solving equations of the form $d^\mathsf{r} = \sum_{i=1}^{l} b_i n_i^\mathsf{r}$ and $N^\mathsf{r} = \sum_{i=l+1}^{2l} b_i n_i^\mathsf{r}$ for a vector b more quickly. Finally, TA_1 chooses a random string x^r and runs Sim upon receipt of x^r as a message from the adversary for participant u. All the values computed by TA_1 become part of a tuple v^r.

Later TA_2 is given the public key pks^r, a message m^r, a supposed signature s_b^r, and the tuple v^r of values precomputed by TA_1. It first computes the one-time pad N^r that makes s_b^r fit the previously chosen x^r from v^r by setting $N^\mathsf{r} := s_b^\mathsf{r} \oplus x^\mathsf{r}$. It then solves the equations $N^\mathsf{r} = \sum_{i=l+1}^{2l} b_i n_i^\mathsf{r}$ and $m^\mathsf{r} = \sum_{i=1}^{l} b_i n_i^\mathsf{r}$ for a vector b using M_2^{-1} and M_1^{-1} from v^r. Next, TA_2 causes TH to subtract the nonce denoted by N^u from x^u: For all $i \in \{l+1, \ldots, 2l\}$ with $b_i = 1$, it asks TH to XOR the i-th nonce to the term represented by x^u. In this situation M_u would obtain the result $s^\mathsf{r} = s_b^\mathsf{r}$. Then TA_2 causes TH to construct d^u, i.e., it asks TH to XOR the i-th nonce for all $i \in \{1, \ldots, l\}$ with $b_i = 1$. In this situation M_u would obtain the result d^r. Thus, when TA_2 finally inputs the signature test command for s^u, then M_u would output $B = \mathsf{true}$ if $b = 1$ and $B = \mathsf{false}$ if $b = 0$. Hence TH does the same with overwhelming probability, which shows that $\mathsf{TA} = (\mathsf{TA}_1, \mathsf{TA}_2)$ is a valid test approximation.

The algorithm TA_2 for testing the validity of a signature does only two things itself: First it makes the user inputs requesting the computation of two XORs (one for subtracting the nonces from the signature, the other for constructing the message represented as d^u) and the testing of a signature. This is the input of two terms or program-like string of length linear in l. Secondly, it solves two linear equation systems given the respective inverse matrices. This is standard matrix-vector multiplication; it can be done in time quadratic in l, where l was the length of a signature. This reduction again seems a clear indication that TH can test the validity of signatures and is thus not what one would intuitively call a Dolev-Yao model. Even though the complexity of the reduction is not as convincing as for the scenario with signature computation, we also know the exact reduction algorithm TA_2. If any realization existed where TH would not do the major part of the signature testing, then the major part of this testing would have to be the matrix-vector multiplication of TA_2. This is a very serious restriction on potential realizations.

8 A Passively BRSIM/UC-Sound Dolev-Yao Model of XOR

The special Dolev-Yao model of XOR that we prove to be sound corresponds to passive attacks only, together with a type consistency requirement on the protocol expressed in the Dolev-Yao model. In other respects the result is strong: We show BRSIM/UC soundness and need no restrictions on the other operations in the Dolev-Yao model; this distinguishes our result from those in [41, 43]. Roughly, the benefit of the restriction to passive adversaries is that all XORs are constructed bottom-up by honest parties. Thus the simulator never receives bitstrings that seem to be XORs but where the simulator does not know how to partition the XOR into its components, as in the impossibility scenarios in the previous sections. The need for the type consistency requirement is shown in Section 8.1.

In Sections 8.2 to 8.6, we present our Dolev-Yao version of XOR in detail, in particular the extended adversary parsing capabilities that we need according to Section 2. As we aim at an overall, operator-rich Dolev-Yao model with XOR, we need an underlying Dolev-Yao model with the other usual cryptographic operators and a realization secure in the sense of BRSIM/UC. Hence we have to use the Dolev-Yao model of [7]. Essentially we only add XOR terms and operations to this model. We also add a type for nonces of variable length. Typically, Dolev-Yao models have only one nonce type, and realizations use random bitstrings of a fixed length sufficient for unguessability. However, we now want to hide arbitrary other terms by XORing them with a nonce of suitable length. To make these additions to the existing model rigorous, we have to use the notation from [7] instead of our generic notation from Section 5. We will mention the links between the notations.

In Section 8.7, we present the cryptographic realization of this Dolev-Yao model of XOR; in Section 8.8 we sketch that this realization is as secure as the Dolev-Yao model in the sense of BRSIM/UC if restricted to passive adversaries and type consistency.

8.1 The Necessity of Correct Type Conversions by the Users or Protocols

The largest difficulty with XOR even in the passive case is typing. XORs can yield arbitrary bitstrings, while otherwise it seems necessary for achieving BRSIM/UC 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 is not properly constructed, e.g., if decryption is applied to a term that is not encrypted at all or with a different key. The only decision 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 XOR, however, algebraic equations like commutativity and associativity are essential, and they apply to pure bitstrings, not to bitstrings with type tags. The main problem with this typing 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 all the random subterms cancel out and one term of another type remains, e.g., a payload. 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 a machine XORs two bitstrings, it cannot reliably decide whether the result is of an underlying type. This is obvious if all type tags are removed before XORing (which is one possibility, and comes closest to typical message formats in XORs). It is also true if we XOR base types including 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) that 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 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 a given protocol it is usually clear what types are expected in what messages, and thus, e.g., in which step an XOR operation should yield a payload. Whether a protocol is correctly typed in this sense can be verified on the protocol layer if we only allow passive attacks.

For simplicity, we treat the length of the resulting term in a type conversion from an XOR back to another type in the same way, i.e., the user enters the correct length of the desired term. An advantage of this solution is that it allows a simple realization. In particular, for the most common case of protocols that only XOR terms of equal length, these realizations correspond to plain XORs without additional length fields or leakages of the term structure.

8.2 Notation

We first repeat important notation from [7], and then introduce additional notation for lists and matrices. As before, we write “:=” for deterministic and “$\leftarrow$” for probabilistic assignment, and we write “$\overset{\mathcal{R}}{\leftarrow}$” for uniform random choice from a set. By $x := \text{++}y$ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m^{r} is denoted as $\mathsf{len}(m^{\mathsf{r}})$, 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 $\mathsf{normalize}$ that corresponds to the cancellation rules in an XOR, i.e., $\mathsf{normalize}$ first removes duplicates from the input list, leaving one element whenever there is an odd number of equal elements, and then applies sort to the resulting list.

The elements of a matrix $M \in F^{m,n}$ over a field F are denoted by $M_{i,j}$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. By $\mathsf{solve}(M, v)$ we denote an algorithm that, 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 a solution exists, and otherwise $\downarrow$. Such an algorithm can, e.g., be built from Gaussian elimination.

8.3 Trusted-Host Machine and Overall Parameters

The underlying system model in [7] is an I/O-automata model. Hence the overall Dolev-Yao model, with its state, is represented as a machine TH, called trusted host, as in Figure 1. Let $\mathcal{H} = \{1, \ldots, n\}$ denote the set of honest users. As we consider passive attacks only, there are no corrupted users. For each $u \in \mathcal{H}$, the trusted host TH has a port $\mathsf{in}_u?$ for inputs from H_u and a port $\mathsf{out}_u!$ for outputs to H_u, and it has analogous ports $\mathsf{in}_\mathsf{a}?$ and $\mathsf{out}_\mathsf{a}!$ for the adversary. The trusted host keeps track of the length of messages (this is needed because this length may be leaked to the adversary when honest parties send encrypted messages) using a tuple L of length functions of abstract terms; these length functions can be arbitrary polynomials. One function from L that we mention below is $\mathsf{max\_len}(k)$, which denotes the maximum length of processed messages. We extend L by two functions $\mathsf{xor\_len}((l_1, t_1), \ldots, (l_j, t_j))$ for computing the length of an abstract XOR from the lengths l_i and types t_i of its parameters, and $\mathsf{nonce\_vl\_len}(l)$ for the overall length of a nonce of variable length l (where l corresponds to the desired entropy).

8.4 States: Term Database

The main part of the state of the Dolev-Yao model, i.e., of the machine TH, is a *term database* D. Each term is primarily given by its type (top-level operator) and top-level argument list. The non-atomic arguments in this list are pointers to the respective subterms. For this, each term entry in D contains a global index that allows us (not the participants) to refer to terms unambiguously. In addition, the term database D contains the length of each term and handles that represent local names under which the different participants know the term, if they do know it. In particular, the handles imply the knowledge sets known from other Dolev-Yao models. The handles are a specific instantiation of the term representations from Definition 2. Recall that a mapping from the database representation to a more standard term representation is now available in [28].

In detail, the attributes of the term database D are defined as follows; the only differences to [7] due to adding XOR are an augmented type set, the introduction of the set *randomtypes*, and a new attribute *parsed* that we need within the treatment of the parsing of terms that contain several XORs.

- *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* is the type of the entry. We add types xor and $\mathsf{nonce\_vl}$ to *typeset* from [7], denoting the types for XOR and for nonces 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* from [7] denotes a set of *secret types*, whose elements must not be put into messages.

	n_1	n_2	n_3	d_1	d_2	x_1	x_2
n_1	0	0	0	0	0	1	0
n_2	0	0	0	0	0	0	1
n_3	0	0	1	0	0	1	1
d_1	0	0	0	0	0	1	1
d_2	0	0	0	0	0	0	1
x_1	0	0	0	0	0	0	0
x_2	0	0	0	0	0	0	0

Fig. 5. Example of the matrix representation of XORs. The XORs are $x_1 = n_1 \oplus n_3 \oplus d_1$ and $x_2 = n_2 \oplus n_3 \oplus d_1 \oplus d_2$.

- $x.arg = (a_1, a_2, \ldots, a_j)$ 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 isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $x.len \in \mathbb{N}_0$ denotes the "length" of the term. It is computed using the functions from the parameter tuple L.
- $x.parsed \in \{1, \downarrow\}$ denotes whether the ideal adversary has already parsed this term. Most entries have no attribute *parsed*; so $x.parsed = \downarrow$.

Initially, D is empty. As additional state parts, TH 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. These counters are all initialized with 0. Moreover, TH maintains explicit counters and message bounds for each port in order to ensure polynomial runtime; see [7] for the details.

The algorithm $i^{\mathsf{hnd}} \leftarrow \mathsf{ind2hnd}_u(i)$ for $i \in \mathcal{INDS}$ (with side effect) denotes that TH determines a handle i^{hnd} for user u to the database entry (term) $D[i]$: If $i^{\mathsf{hnd}} := D[i].hnd_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].hnd_u := \text{++}curhnd_u$. The algorithm is extended for $i \notin \mathcal{INDS}$ by the identity function. The algorithm $\mathsf{ind2hnd}^*_u$ for lists applies $\mathsf{ind2hnd}_u$ to each element of its input list.

8.5 Derived Matrices for XORs

For the linear algebra resulting from XORs, we define several matrices over $\mathsf{GF}(2)$ representing released XORs, i.e., XORs that the adversary learned. The rows and columns correspond to indices in the term database D. Roughly, coefficient 1 in row i and column j indicates that the i-th term in D is a top-level XOR-component in the j-th term and that the adversary has learned the j-th term. We also make such matrix entries for each released individual random value. The matrix A indicates the non-random components in each released 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 a given term database D, but as D will always be clear from the context we do not write it as a parameter.

More precisely, we define three matrices A, $R^{(l)}$, and $\bar{R}^{(l)}$ in $\mathsf{GF}(2)^{size,size}$. Here $A_{i,j} = 1$ iff $D[i].type \notin randomtypes$ and $D[j].type = \mathsf{xor}$ and $D[j].parsed = 1$ and $i \in D[j].arg$. Similarly, $R^{(l)}_{i,j} = 1$ iff $D[i].type \in randomtypes$ and $D[i].len \geq l$ and $D[j].type = \mathsf{xor}$ and either $D[j].type = \mathsf{xor}$ and $i \in D[j].arg$, or $D[j].type \in randomtypes$ and $i = j$. The same formula, except with "$D[i].len < l$", defines $\bar{R}^{(l)}$. The condition $D[i].parsed = 1$ reflects that an XOR only gets a column when it has been parsed. (Recall that we will need this for treating terms with several, possibly nested, XORs.)

Figure 5 gives an example of this matrix representation: The terms in D are three nonces n_i of length l, two payloads d_i, and two XORs x_i. The adversary has learned and parsed the two XORs and the third nonce. The matrix in the figure is the OR of A and $R^{(l)}$: The matrix $R^{(l)}$ has the depicted rows for the nonces, but only zeroes in the rows for payloads, and vice versa for A. The matrix $\bar{R}^{(l)}$ contains only zeroes.

8.6 New Inputs and their Evaluation

Operations are triggered by so-called input commands from users or the adversary into TH. In these commands, the users refer to the terms by the handles defined in Section 8.4. The normal cryptographic operations are called basic commands. They are accepted at each input port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$ 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 model the transfer of terms to other users. In our case with passive adversaries only, we only use the command for sending a term in an authentic way. Its effect is that the adversary immediately gets a handle to the sent term, and if the channel is later scheduled, the intended recipient also gets a handle to the sent term.

The notation $j \leftarrow \mathsf{op}(i)$ means that TH 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., $\leq \mathit{curhnd}_u$, at the time of the command execution; else the command execution immediately aborts.

The XOR operation 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 a command to convert an XOR back into another type; this conversion succeeds only if the XOR has only one argument of this type and the desired length. Furthermore, we define a command that creates a nonce of variable length.

Definition 8. *(Basic commands for XOR and for nonces of variable length) The trusted host* TH *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* $0 \leq j \leq \mathsf{max\_len}(k)$*.*
  For $i = 1, \ldots, j$*, let* $m_i := D[\mathit{hnd}_u = m_i^{\mathsf{hnd}}].\mathit{ind}$ *and* $l_i := D[m_i].\mathit{len}$ *and* $t_i := D[m_i].\mathit{type}$*. Let* $l := \mathsf{xor\_len}((l_1, t_1), \ldots, (l_j, t_j))$*. If* $l > \mathsf{max\_len}(k)$ *or* $t_i \in \mathit{secrettypes}$ *for some* $i \in \{1, \ldots, j\}$*, then return* $\downarrow$*.*
  For $i = 1, \ldots, j$*, let* $\mathit{arg}_i := D[m_i].\mathit{arg}$ *if* $t_i = \mathsf{xor}$*, else* $\mathit{arg}_i := (m_i)$*. Let* $x\_\mathit{arg} := \mathsf{normalize}(\mathsf{append}(\mathit{arg}_1, \ldots, \mathit{arg}_j))$*. Set* $x^{\mathsf{hnd}} := \text{++}\mathit{curhnd}_u$ *and make a new entry in the term database as* $D :\Leftarrow (\mathit{ind} := \text{++}\mathit{size}, \mathit{type} := \mathsf{xor}, \mathit{arg} := x\_\mathit{arg}, \mathit{hnd}_u := x^{\mathsf{hnd}}, \mathit{len} := l)$*.*
- *Type conversion of XOR:* $m^{\mathsf{hnd}} \leftarrow \mathsf{conv\_xor\_to\_}\mathit{type}(x^{\mathsf{hnd}}, l)$ *with* $\mathit{type} \in \mathit{typeset} \setminus (\{\mathsf{xor}\} \cup \mathit{secrettypes})$ *and* $0 \leq l \leq \mathsf{max\_len}(k)$*.*
  Let $x := D[\mathit{hnd}_u = x^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{xor}].\mathit{ind}$*,* $(x_1, \ldots, x_j) := D[x].\mathit{arg}$*, and* $t := D[x_1].\mathit{type}$*. If* $j \neq 1$ *or* $t \neq \mathit{type}$ *or* $D[x_1].\mathit{len} \neq l$ *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* $\mathsf{nonce\_len}(k) \leq l \leq \mathsf{max\_len}(k)$*.*
  Let $l^* := \mathsf{nonce\_vl\_len}(l)$*. If* $l^* > \mathsf{max\_len}(k)$ *then return* $\downarrow$*. Else set* $n^{\mathsf{hnd}} := \text{++}\mathit{curhnd}$ *and make a new entry in the term database as* $D :\Leftarrow (\mathit{ind} := \text{++}\mathit{size}, \mathit{type} := \mathsf{nonce\_vl}, \mathit{arg} := (), \mathit{hnd}_u := n^{\mathsf{hnd}}, \mathit{len} := l^*)$*.* ◇

As we have excluded active attacks and any specific vulnerabilities they might add even to the Dolev-Yao model, we do not need to introduce new local adversary commands. We only extend the command $\mathsf{adv\_parse}$, which allows the ideal 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 an XOR term, the basic idea is to determine whether the part consisting of random values is linearly independent from the corresponding parts in previously released XORs and individually released nonces. If yes, we consider this part to hide the other components of the XOR. Otherwise, we obey the XOR-parsing need from Section 2 by giving the ideal adversary the non-random arguments of the linear combination of the new XOR and previous XORs whose random elements cancel each other.

There are two small complications: First, as we have terms of arbitrary length, we introduced nonces of arbitrary length, and we now have to be careful that the nonces are indeed of sufficient length. In most protocols, all the nonces in one XOR are simply 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 the nonces of length at least l by a linear combination, we let the ideal adversary learn not only the non-random elements, but also the shorter nonces in that linear combination. This leaves room for improvements, e.g., by only granting the ideal adversary the XOR of those nonces, but at the cost of a more complicated proof that only serves a very rare class of protocols. Secondly, when considering one XOR we only use those other XORs that have already been parsed.

Whenever the adversary learns a term of a random type, we also determine whether it is linearly dependent from XORs and nonces released earlier. This concerns the new type $\mathsf{nonce\_vl}$ and the old type nonce.

Definition 9. *(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* $type = \mathsf{xor}$*: Set* $D[m].parsed := 1$*. Let* $(x_1, \ldots, x_h) := D[m].arg$ *and* $l := \mathsf{max}(\{D[x_i].len \mid D[x_i].type \notin randomtypes \wedge D[x_i].hnd_{\mathsf{a}} = \downarrow\})$*. Let* a*,* $r^{(l)}$*, and* $\bar{r}^{(l)}$ *denote the vectors that this newly parsed XOR will add (as columns) 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* $r' := \bar{R}^{(l)}b \oplus \bar{r}^{(l)}$*.*[5] *Let* $\mathcal{B} := \{\mathsf{ind2hnd_a}(i) \mid b_i = 1\}$ *and* $\mathcal{D} := \{\mathsf{ind2hnd_a}(i) \mid d_i = 1 \vee r'_i = 1\}$ *and* $arg := (\mathsf{dependent}, \mathcal{B}, \mathcal{D}, D[m].len)$*.*
- *If* $type = \mathsf{nonce\_vl}$*: Set* $D[m].parsed := 1$*. 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* $arg := (\mathsf{independent}, l)$*. Else derive a result* $arg := (\mathsf{dependent}, \mathcal{B}, \mathcal{D}, l)$ *exactly as for the type* xor*.*
- *If* $type = \mathsf{nonce}$*: Set* $D[m].parsed := 1$ *and similarly define* $arg := (\mathsf{independent})$ *or derive a result* $arg = (\mathsf{dependent}, \mathcal{B}, \mathcal{D})$ *as for the type* $\mathsf{nonce\_vl}$*; the parameter* l *is not needed since elements of type* nonce *have a fixed length. (In the system without XOR, the result for nonces was simply* $arg = ()$*.)* ◇

In the example from Figure 5, if the adversary next learns a term $x_3 = n_1 \oplus d_2$, then the random part is n_1, which is the XOR of the random parts of the previously released terms n_3 and x_1. Thus the result of $\mathsf{adv\_parse}$ for this new term x_3 is is of the form $(\mathsf{xor}, (\mathsf{dependent}, \mathcal{B}, \mathcal{D}, l))$ where $\mathcal{B}$ is derived from $b = 0010010$; the XOR where these random parts cancel out is $d = n_3 \oplus x_1 \oplus x_3 = d_1 \oplus d_2$, and thus $\mathcal{D}$ consists of the adversary handles of d_1 and d_2.

8.7 Realization of this Dolev-Yao Model of XOR

We now present the core parts of the concrete realization of the Dolev-Yao model of XOR presented in the previous section. As in Figure 1, every user u has its own machine called M_u. This machine contains the cryptographic objects that user u knows. It offers its user the same interface as the Dolev-Yao model, i.e., it has ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ and accepts the same commands there, in particular xor, $\mathsf{gen\_nonce\_vl}$, and $\mathsf{conv\_xor\_to\_}type$ with the same parameters as in Definition 8. In the real system, sending a term on an authentic channel (recall that we consider only passive attacks) releases the actual bitstring to the adversary, and once the channel is scheduled, also to the intended recipient machine M_v.

As a specialization of the underlying system from [7] we assume that all the real type tags are, when the abstract syntax is encoded into bitstrings, of equal length tlen and attached at the left side of the bitstring. As abstract syntax we still write lists and use a larger alphabet, e.g., we write $(\mathsf{nonce}, 1100111)$ for a tagged nonce where nonce is the type tag.

As the most important part of the realization, we present the functional parts of the basic commands, i.e., the core operations on bitstrings (with type tags etc.) without the state-keeping part of the commands. These operations are quite natural given the prior discussions about typing and lengths: Whenever several typed bitstrings are XORed, we remove their type tags, pad the remainders with zeros on the left to the maximum occurring length, XOR them, and finally add an XOR tag to the resulting

[5] 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', as well as which previously learned terms are used in this linear combination.

string. In the conversion back from an XOR to a base type, the XOR tag is replaced by the target tag and padding is removed according to the target length input by the user. This target length is the overall length with the type tag. In contrast, for the nonce constructor we assume that the input length designates the number of real random bits. In the notation of the impossibility sections, all these bitstrings would get a superscript $^{\mathsf{r}}$. We omit this superscript in the following to stay close to the notation of the underlying system from [7].

Definition 10. *(Functional part of the realization of the basic commands 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* $i := 1, \ldots, j$.
  Parse each parameter as $m_i = (type, m'_i)$ *with* $type \in typeset$ *and* $m'_i \in \{0,1\}^+$. *Then if* $type = \mathsf{xor}$, *let* m''_i *equal* m'_i *without the right-most* $\mathsf{nonce\_len}(k)$ *bits, else* $m''_i := m'_i$. *Let* $l := \max(\{\mathsf{len}(m''_1), \ldots, \mathsf{len}(m''_j)\})$ *and* $x_i := 0^{l-\mathsf{len}(m''_i)} || m''_i$ *for* $i = 1, \ldots, j$. *Let* $r \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{nonce\_len}(k)}$ *and* $x := (\mathsf{xor}, x_1 \oplus \cdots \oplus x_j \ || \ r)$.
- *Type conversion from XOR:* $m \leftarrow \mathsf{func\_conv\_xor\_to\_}type(x, l)$ *for* $x \in \{0,1\}^+$ *and* $l \in \mathbb{N}$.
  Parse x *as* $x = (\mathsf{xor}, x' \ || \ r)$ *with* $\mathsf{len}(r) = \mathsf{nonce\_len}(k)$ *and let* x^* *be the* $l - \mathsf{tlen}$ *rightmost bits of* x'. *(When this conversion is called, there will indeed always be at least* $l - \mathsf{tlen}$ *bits, and the deleted bits on the left will be former zero-paddings added by the XOR constructor.) Return* $m := (type, x^*)$.
- *Variable-length nonce constructor:* $n \leftarrow \mathsf{make\_nonce\_vl}(l)$ *for* $l \in \mathbb{N}$.
  Let $n' \xleftarrow{\mathcal{R}} \{0,1\}^l$ *and* $n := (\mathsf{nonce\_vl}, n')$. ◇

The non-functional parts of the commands add length and type tests corresponding to those in TH, assign new handles where needed, and store new words in a database of M_u.

8.8 Soundness Theorem

Our security claim is that the Dolev-Yao model with XOR defined in Sections 8.2 to 8.6 is soundly implemented by the realization sketched in Section 8.7 in the sense of BRSIM/UC, provided that the surrounding protocol ensures that an XOR is only converted to another type if the XOR has only one argument, and the type and length of this argument equal the target type and length of the conversion. We call this precondition *correct XOR conversion*, or short $\mathsf{CorrXOR}$. We have already built a restriction to passive attacks into the definitions of the real and ideal system by only allowing uncorrupted participants and authentic channels. The soundness proof would still work if we relaxed the authenticity restriction by allowing message re-ordering, re-routing, and duplication, i.e., if we solely required that the adversary only sends messages that were constructed by the correct machines.

To formally capture the property $\mathsf{CorrXOR}$, we need additional notation. The underlying system model from [9] has a well-defined notion of traces that applies to the combination of our trusted host TH, honest users H, and an ideal adversary A. Essentially, a trace is a sequence of events that occur when the given machines interact. The t-th step of a trace r is written r_t; we speak of the step at time t. By $p?m \in r_t$ we mean that message m is input at port $\mathsf{p}?$ in step r_t, and $r_t : D$ denotes the contents of the term database D in step r_t. The formula in the following definition can be read as follows: If a term i is converted to type x, and the term really is an XOR, then it is an XOR of only one argument j, and this argument is of the correct type and length.

Definition 11. *(Correct XOR Conversion) A trace* r *is contained in* $\mathsf{CorrXOR}$ *if and only if for all* $t \in \mathbb{N}$, $u \in \mathcal{H}$, $x \in typeset$, $l \in \mathbb{N}$, *and* $i \in \mathcal{INDS}$, *and with* $i^{\mathsf{hnd}} := r_t : D[i].hnd_u$, *we have*

$$\begin{aligned} &\mathsf{in}_u?\mathsf{conv\_xor\_to\_}x(i^{\mathsf{hnd}}, l) \in r_t \wedge r_t : D[i].type = \mathsf{xor} \\ &\Rightarrow \exists j \in \mathcal{INDS} \colon (r_t : D[i].arg = (j) \\ &\qquad \wedge r_t : D[j].type = x \wedge r_t : D[j].len = l). \end{aligned}$$

◇

We finally define the notion of blackbox reactive simulatability restricted to those users that guarantee the property $\mathsf{CorrXOR}$ (independent of the adversary). We repeat in the notation that we only consider passive attacks, although this is not a restriction in the following definition, but built into the systems.

Definition 12. *(Reactive Simulatability with Correct XOR Conversion and Passive Attacks) A user* H *uses* correct XOR conversion *with respect to the machine* TH *if for all ideal adversaries* A*, the property* $\mathsf{CorrXOR}$ *from Definition 11 holds for all possible traces of the machine set* $\{\mathsf{TH}, \mathsf{H}, \mathsf{A}\}$*. We denote the restriction of blackbox reactive simulatability for the machine* TH *to users with correct XOR conversion by* $\geq^{\mathsf{CorrXOR}}_{b,\mathsf{passive}}$. ◇

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 $\mathcal{S}$ of cryptographic schemes (currently containing symmetric and asymmetric encryption schemes, signature schemes, and MACs) that satisfy their respective security definitions against active attacks, see [7, 12, 13], and length functions and bounds L'. For $(n, \mathcal{S}, L') \in RPar$, let $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}_{n,\mathcal{S},L'}$ be the resulting realization of the Dolev-Yao model. The derivation of suitable length functions and bounds for the Dolev-Yao model from the real parameters is given by a function $L := \mathsf{R2Ipar}(\mathcal{S}, L')$. This function is extended by the new length functions for XOR and nonces of variable length. We have $\mathsf{nonce\_vl\_len}(l) := l + \mathsf{tlen}$. For defining $l := \mathsf{xor\_len}((l_1, t_1), \ldots, (l_i, t_i))$, let $l'_j := l_j - \mathsf{nonce\_len}(k)$ if $t_j = \mathsf{xor}$, else $l'_j := l_j$. Then $l := \max(l'_1, \ldots, l'_i) + \mathsf{nonce\_len}(k)$. Let $\{\mathsf{TH}\}_{n,L}$ be the Dolev-Yao model with parameters n and L.

Theorem 3. *(Soundness of the Dolev-Yao Model with XOR) For all parameters* $(n, \mathcal{S}, L') \in RPar$ *and for* $L := \mathsf{R2Ipar}(\mathcal{S}, L')$*, we have*

$$\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}_{n,\mathcal{S},L'} \geq^{\mathsf{CorrXOR}}_{b,\mathsf{passive}} \{\mathsf{TH}\}_{n,L}.$$

□

For proving BRSIM soundness for the underlying Dolev-Yao model without XOR, a simulator Sim was defined in [7] (recall Figure 1) and the required indistinguishability between the combination of TH and Sim, and the combination of the real machines M_u was shown. We now show how we extend this simulator to deal with XOR, and we sketch how to extend the indistinguishability proof.

8.9 Simulator Extensions for XOR and Nonces of Variable Length

Basically Sim translates handles (which represent terms) that it receives from the Dolev-Yao model TH into real bitstrings as the real adversary expects them and vice versa. In our case with authentic channels only, there are no messages from the real adversary (only scheduling signals that designate when existing messages are delivered). Hence we only need the translation from terms to bitstrings, where we extend the existing procedure by the treatment of XORs and nonces of variable length, and modify the treatment of normal nonces.

The state of Sim mainly consists of a database D_a that stores the bitstrings the adversary knows under the adversary handles; these attributes are denoted by hnd_a and $word$ (similar to the databases D_u in the real machines).

A sent term is indicated by TH to the ideal adversary, and thus here to the simulator, in the form $(u, v, \mathsf{a}, m^{\mathsf{hnd}})$, meaning that user u is sending the term corresponding to handle m^{hnd} to user v over an authentic channel. If Sim already has a bitstring m for m^{hnd} in D_a, i.e., this message is already known to the adversary, then Sim immediately outputs m to A at the corresponding network port $\mathsf{net}_{u,v,\mathsf{a}}$. Otherwise, it first constructs such a bitstring m with a recursive algorithm $\mathsf{id2real}(m^{\mathsf{hnd}})$. This algorithm decomposes the abstract term using the adversary command $\mathsf{adv\_parse}$ and basic commands. At the same time, $\mathsf{id2real}$ builds up a corresponding real bitstring using real cryptographic operations and enters all subterms and corresponding bitstrings into D_a.

Each execution of $\mathsf{id2real}$ with an input m^{hnd} starts with a call $(type, arg) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$. If this call yields $type = \mathsf{XOR}$, then either $arg = (\mathsf{independent}, l)$ with $l \in \mathbb{N}$ or $arg = (\mathsf{dependent}, \mathcal{B}, \mathcal{D}, l)$ where $\mathcal{B}$ and $\mathcal{D}$ are sets of handles (by Definition 9) and $l \in \mathbb{N}$. In the first case, where the new XOR is linearly independent from previously known terms, $\mathsf{id2real}$ chooses a random bitstring m' of length $l - \mathsf{tlen}$ and returns $m := (\mathsf{xor}, m', r)$ with $r \xleftarrow{\mathcal{R}} \{0,1\}^n$. In the second case, the generic part of $\mathsf{id2real}$ from [7] makes recursive calls to ensure that bitstrings are constructed and entered in the database D_a for all new handles (subterms) in $\mathcal{B}$ and $\mathcal{D}$. Recall that $\mathcal{B}$ are the handles of prior XORs and random values such that, when they are XORed together with the new term designated by m^{hnd}, all random elements of sufficient length cancel out, and that $\mathcal{D}$ contains the handles of all non-random elements and too-short random values in the resulting XOR. Hence id2real simply XORs all the corresponding bitstrings to obtain the desired, correctly simulated bitstring for the new term: Let $\{m_1, \ldots, m_s\}$ denote the set of strings in D_{a} that correspond to the handles in $\mathcal{B} \cup \mathcal{D}$. Then $m := \mathsf{make\_xor}(m_1, \ldots, m_s)$.

If $\mathit{type} = \mathsf{nonce\_vl}$, we also have $\mathit{arg} = (\mathsf{independent}, l)$ or $(\mathsf{dependent}, \mathcal{B}, \mathcal{D}, l)$. In the first case, id2real chooses a random bitstring m' of length $l - \mathsf{tlen}$ and adds the type tag as $m := (\mathsf{nonce\_vl}, m')$. In the second case, it constructs the XOR corresponding to the linear combination $\mathcal{B}$ and the learned values $\mathcal{D}$ as for the type xor; let us call this result m' instead of m now. Then it converts this back to a nonce of variable length as $m \leftarrow \mathsf{func\_conv\_xor\_to\_nonce\_vl}(m', l)$.

For the type nonce, the procedure is similar, except that the random bitstring m' in the first case is chosen with the fixed length for these standard nonces.

8.10 Proof of Indistinguishability (Sketch)

The proof of the extended Dolev-Yao model, now including XOR but restricted to passive attacks, is an add-on to the proof from [7]. The basic structure of this proof is that a combined system C is defined that essentially contains all aspects of both the real and the ideal system, and then bisimulations are proved between C and the combination of the real machines, and between $\mathsf{C}_{\mathcal{H}}$ and the combination 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. It remains to be shown how the bisimulations are extended for XOR. This is done by considering each input in corresponding states of the three systems: One shows that it leads to equally distributed outputs in the three systems, retains certain invariants, and leads to corresponding states again. The essential parts of these proofs for the different input are the following: For the XOR command, one shows that the normalization in the ideal system is correct. For the type conversion of XOR, one shows that the padding and padding removal are consistent. For the construction of variable-length nonces, only standard technical details need to be shown. (Furthermore, as in some other operations, the bisimulation can fail here if nonces happen to collide, but there is a standard error set mechanism by which one shows at the end that this only happens with negligible probability.) Sending an XOR is the operation where the correctness of the matrix operations in the ideal system is proved, i.e., essentially that if the ideal system does not let the ideal adversary learn arguments of an XOR, then the real adversary obtains no Shannon information from about these arguments. The same kind of arguments apply to sending a nonce of constant or variable length. Receiving messages from the network needs no special consideration because of the restriction to passive attacks.

9 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 type and allows XORs of payloads.

The intuitive goal of our more complex results is to show that *no* Dolev-Yao model with the usual cryptographic operations and XOR can be securely implemented in the sense of BRSIM/UC, i.e., in the sense that the realization can be safely plugged in for the abstraction in arbitrary environments and if arbitrary security goals may be required. As there was no prior formal definition of what is and isn't a Dolev-Yao model, we have approached this intuitive goal by a set of concrete impossibility results under different precise assumptions about Dolev-Yao models and about the implementation of XORs.

On the positive side, we presented a Dolev-Yao 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. Except for the restrictions to passive attacks and correct type conversions, this result is strong: It uses a BRSIM/UC-style definition, allows a broad range of other operations in the Dolev-Yao model, and correctly handles situations where some components in an XOR are uniformly random and others are not.

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, and when the Dolev-Yao model is extended compared with typical ones at least as in our passive result. 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.

Acknowledgments. We thank the anonymous reviewers and Martín Abadi, Véronique Cortier, Anupam Datta, Ante Derek, Jonathan Herzog, Cathy Meadows, John Mitchell, Andre Scedrov, Michael Waidner, and Bogdan Warinschi for interesting discussions.

References

1. Michael Backes and Birgit Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.
2. Michael Backes and Birgit Pfitzmann. Limits of the BRSIM/UC soundness of Dolev-Yao-style xor. *International Journal of Information Security*, 7(1):33–54, 2008.
3. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
4. Jonathan K. Millen. The interrogator model. In *Proc. 16th IEEE Symposium on Security & Privacy*, pages 251–260, 1995.
5. Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu 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.
6. Hubert Comon-Lundh and Vitaly 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.
7. Michael Backes, Birgit Pfitzmann, and Michael 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/`.
8. Birgit Pfitzmann and Michael 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`.
9. Birgit Pfitzmann and Michael 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/`.
10. Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability framework for asynchronous systems. *Information and Computation*, pages 1685–1720, 2007.
11. Ran 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/`.
12. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *LNCS*, pages 271–290. Springer, 2003.
13. Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

14. Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks - BRSIM/UC-soundness of symbolic encryption with key cycles. In *Proceedings of 20th IEEE Computer Security Foundation Symposium (CSF)*, 2007. Preprint on IACR ePrint 2005/421.
15. Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2003. Full version in IACR Cryptology ePrint Archive 2003/121, Jun. 2003, `http://eprint.iacr.org/`.
16. Michael Backes. A cryptographically sound dolev-yao style security proof of the Otway-Rees protocol. In *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.
17. Michael Backes and Markus Duermuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.
18. Michael Backes and Birgit Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *Proceedings of 21st IFIP International Information Security Conference (SEC)*, pages 233–245, 2006.
19. Michael Backes, Sebastian Moedersheim, Birgit Pfitzmann, and Luca Vigano. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging Scenario. In *Proceedings of Foundations of Software Science and Computational Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2006.
20. Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In *Proceedings of 11th European Symposium on Research in Computer Security(ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 362–383. Springer, 2006. Preprint on IACR ePrint 2006/219.
21. Michael Backes and Birgit Pfitzmann. Computational probabilistic non-interference. In *Proceedings of 7th European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2002.
22. Michael Backes and Christian Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *LNCS*, pages 675–686. Springer, 2003.
23. Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.
24. Michael Backes and Birgit Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proc. 24th IEEE Symposium on Security & Privacy*, pages 140–152, 2003.
25. Michael Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2005.
26. Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(2):109–123, 2005.
27. Peeter Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 26–35, 2005.
28. Christoph Sprenger, Michael Backes, David Basin, Birgit Pfitzmann, and Michael Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.
29. Martín Abadi and Phillip 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 *LNCS*, pages 3–22. Springer, 2000.
30. Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
31. Peeter Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
32. Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
33. Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.
34. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 380–403. Springer, 2006.

35. Michael Backes, Christian Jacobi, and Birgit Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proceedings of 11th International Symposium on Formal Methods Europe (FME)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.
36. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Low-level ideal signatures and general integrity idealization. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2004.
37. Michael Backes, Markus Duermuth, Dennis Hofheinz, and Ralf Kuesters. Conditional reactive simulatability. In *Proceedings of 11th European Symposium on Research in Computer Security (ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2006. Preprint on IACR ePrint 2006/132.
38. Michael Backes and Peeter Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS)*, pages 370–379, 2006.
39. Michael Backes, Stefan Lorenz, Matteo Maffei, and Kim Pecina. Caspa: Causality-based abstraction for security protocol analysis. In *Proceedings of 20th International Conference on Computer Aided Verification (CAV)*, 2008.
40. Michael Backes, Dennis Hofheinz, and Dominique Unruh. Cosp: a general framework for computational soundness proofs. In *Proceedings of the 16th ACM conference on Computer and communications security (CCS)*, pages 66–78, New York, NY, USA, 2009. ACM.
41. Peeter Laud. *Computationally Secure Information Flow.* PhD thesis, Universität des Saarlandes, 2002. `http://www.cs.ut.ee/~peeter_l/research/csif/lqpp.ps.gz`.
42. Peeter Laud. Pseudorandom permutations and equivalence of formal expressions (abstract). In *14th Nordic Workshop on Programming Theory*, pages 63–65, 2002.
43. 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 *LNCS*, pages 652–663. Springer, 2005.
44. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the reactive simulatability/UC of Dolev-Yao models with hashes. In *Proceedings of 11th European Symposium on Research in Computer Security(ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 404–423. Springer, 2006.
45. Hugo Krawczyk. LFSR-based hashing and authentication. In *Advances in Crptology: CRYPTO '94*, volume 839 of *LNCS*, pages 129–139. Springer, 1994.
46. Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology: CRYPTO '95*, volume 963 of *LNCS*, pages 15–28. Springer, 1995.
47. Alan T. Sherman and David A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, 2003.
48. Catherine Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
49. Catherine Meadows. A model of computation for the NRL protocol analyzer. In *Proc. 7th IEEE Computer Security Foundations Workshop (CSFW)*, pages 84–89, 1994.
50. Jonathan Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.
51. Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
52. David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.
53. Hubert Comon-Lundh and R. Treinen. Easy intruder deductions. Research Report LSV-03-8, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2003.
54. Stéphanie Delaune and Florent Jacquemard. Narrowing-based constraint solving for the verification of security protocols. Research Report LSV-04-8, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2004.
55. Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3124 of *LNCS*, pages 46–58. Springer, 2004.
56. Jon Millen and Vitaly Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 47–61, 2003.
57. Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu 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.

58. Vitaly Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. 13th European Symposium on Programming (ESOP)*, volume 2986 of *LNCS*, pages 355–369. Springer, 2004.
59. Oded Goldreich, Silvio Micali, and Avi 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.
60. Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1990.
61. Donald Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
62. Silvio Micali and Phillip Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 392–404. Springer, 1991.
63. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
64. 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.
65. J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.
66. John Mitchell, Mark Mitchell, Andre Scedrov, and Vanessa Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
67. Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.
68. Anupam Datta, Ante Derek, John Mitchell, Vitalij Shmatikov, and Matthieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 16–29. Springer, 2005.
69. Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 140–154, 2006.
70. Dorothy Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.
71. Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.
72. Jonathan K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
73. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, 1995.
74. James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 255–268, 2000.
75. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.