Limits of the Reactive Simulatability/UC of Dolev-Yao Models with Hashes

Michael Backes[1], Birgit Pfitzmann[2], Michael Waidner[2]

[1] Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de
[2] IBM Zurich Research Lab, Switzerland, {bpf,wmi}@zurich.ibm.com

Abstract. Automated tools such as model checkers and theorem provers for the analysis of security protocols typically abstract from cryptography by Dolev-Yao models, i.e., abstract term algebras replace the real cryptographic operations. Recently it was shown that in essence this approach is cryptographically sound for certain operations like signing and encryption. The strongest results show this in the sense of blackbox reactive simulatability (BRSIM)/UC with only small changes to both Dolev-Yao models and natural implementations. This notion essentially means the preservation of arbitrary security properties under active attacks in arbitrary protocol environments.
We show that it is impossible to extend the strong BRSIM/UC results to usual Dolev-Yao models of hash functions in the general case. These models treat hash functions as free operators of the term algebra. In contrast, we show that these models are sound in the same strict sense in the random oracle model of cryptography. For the standard model of cryptography, we also discuss several conceivable restrictions to the Dolev-Yao models and classify them into possible and impossible cases.

1 Introduction

Tools for proving security protocols typically abstract from cryptography by deterministic operations on abstract terms and simple cancellation rules. An example term is $\mathsf{E}_{pke_w}(\mathsf{hash}(\mathsf{sign}_{sks_u}(m, N_1), N_2))$, where m denotes a payload message and N_1, N_2 two nonces, i.e., representations of fresh random numbers. We wrote the keys as indices only for readability; formally they are normal operands in the term. A typical cancellation rule is $\mathsf{D}_{ske}(\mathsf{E}_{pke}(m)) = m$ for corresponding keys. 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. Such abstractions, although different in details, are collectively called Dolev-Yao models after their first authors [24].

It is not obvious that a proof in a Dolev-Yao model implies security with respect to real cryptographic definitions. Recently, this long-standing gap was essentially closed by proving 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/UC [9]. We abbreviate blackbox reactive simulatability by BRSIM in the following. This security notion means that one system (here the cryptographic realization) can be plugged into arbitrary protocols instead of another system (here the Dolev-Yao model) and retains essentially arbitrary security properties; it is also called UC for its universal composition properties [38, 39, 17, 23]. In other words, this result shows that the Dolev-Yao model as such can serve as an ideal functionality that is correctly implemented by a real functionality given by actual cryptographic systems. Extensions of this simulatability result to more cryptographic primitives were presented in [10, 6], uses in protocol proofs in [5, 3, 4, 7], stronger links to conventional Dolev-Yao-style type systems in [31], and an integration into the Isabelle theorem prover in [41]. Earlier results on relating Dolev-Yao models and real cryptography considered passive attacks only [2, 1, 29]. Later papers [34, 30, 19] consider to what extent restrictions to weaker security properties, such as integrity only, and/or less general protocol classes, e.g., a specific class of key exchange protocols, allow simplifications compared with [9]. A BRSIM/UC result for a specific protocol class means that simulatability is not shown for the Dolev-Yao model as such (and thus by the composition theorems for all protocols using it), but only for the specific compositions of the Dolev-Yao model with this protocol class.[1]

No paper relating Dolev-Yao models and cryptography considers hashing or one-way functions although they are important operators in automated proof tools based on Dolev-Yao models, e.g., [32, 37, 40, 14, 11]. The typical model is that hash is a free operator in the term algebra, i.e., there is no inverse operator, nor any other cancellation rule with operators like E and D. Only a party who knows or guesses a potentially hashed term t can test whether hashing t equals a given hash term h, at least if the surrounding formal protocol language contains equality tests.

The goal of our paper is to close this gap, and to study how the soundness results in the sense of BRSIM/UC can be extended when hash or one-way functions are added to a Dolev-Yao model and its cryptographic implementation. In the following, we only speak of hash functions since the standard Dolev-Yao model for the two classes is the same.

1.1 Our Contributions

It turns out that proving BRSIM/UC for Dolev-Yao models with hash functions is impossible in a general way. Note that the question is not whether a hash function is a good and generally usable cryptographic primitive by itself, but only whether its idealization as a free operator in a term algebra, or a similar plausible idealization, is sound. Prior work showed that certain (classes of) ideal functionalities are not securely realizable, e.g., for bit commitments [18], coin tossing, zero knowledge, and oblivious

[1] There is also work on formulating syntactic calculi for dealing with probabilism and polynomial-time considerations and encoding them into proof tools, in particular [35, 36, 28, 22, 15]. This is orthogonal to the work of justifying Dolev-Yao models, which offer a higher level of abstractions and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

transfer [17], classes of secure multi-party computation [20] and certain game-based definitions [21]. However, none of these works investigated a Dolev-Yao model. Impossibility of a Dolev-Yao model with XOR was shown in [8]. For our case of hashes, the reasons for impossibility and thus the proofs are quite different. Furthermore, the proofs in [8] are all reduction proofs, essentially saying that if an idealization of XOR and other cryptographic operations is soundly implementable in the sense of BRSIM/UC, it can be used to compute cryptographic algorithms and is therefore not intuitively Dolev-Yao. In contrast, we obtain absolute impossibility results. We achieve this by making stronger definitions on what makes an ideal functionality of hashing and other cryptographic operations a Dolev-Yao model.

It is important to note that there is so far no rigorous definition of "any Dolev-Yao model" in the literature that is independent of specific underlying system models such as CSP, π-calculus, IO automata, or strand spaces. For positive results, this is not a problem. However, an impossibility result that only holds for one of these models would not be very convincing. To come as close as possible to capturing the desired generality, we will not prove impossibility for one specific Dolev-Yao model, but only make certain assumptions on the Dolev-Yao model, which we believe are fulfilled by all such models existing so far. Essentially we only assume that the hash functions are abstracted as free operators as informally explained above, that they are applicable to arbitrary terms, and that the model contains some other typical operators and base types.

One reason (but not the only one) for the impossibility in the general case is that hash functions, at least those with a one-way property, are by nature committing, i.e., if one first gets $h = \mathsf{hash}(m)$ and later m one can validate whether indeed $h = \mathsf{hash}(m)$. It is well known that such a commitment property often causes problems in proofs of BRSIM/UC: If the simulator has to simulate a bitstring for h before knowing m, then whatever it picks will most likely not match m. Thus the simulation fails if m is later revealed. In some cases, the commitment problem can be circumvented by using non-standard models of cryptography, e.g., the random oracle model [13] or the common random string model, cf. [18]. Indeed we can show BRSIM/UC for the standard Dolev-Yao model of hashes if the cryptographic realization of the hash function is treated as a random oracle.

For the standard model of cryptography, the next question is whether certain restrictions on the use of hash functions enable a BRSIM/UC soundness result. One option is to restrict the types of terms that can be hashed, in particular to forbid the hashing of payloads. By "payload" we mean an application message, e.g., an email text or the amount and currency of a payment in a payment protocol that uses the cryptographic functionality. Technically, payloads play a special role (as m in the example of the commitment problem shows) because they are known outside the cryptographic system and thus can typically not be modified by the simulator, in contrast to nonces, keys, etc. As to practical usage, this restriction is serious but not unreasonable; e.g., key exchange protocols typically do not use payloads. However, we still obtain an impossibility result if excluding payloads is the only restriction on hashable terms. The basic idea in that case is that by constructing large enough terms, the users of the cryptographic system can simulate payloads. Another conceivable restriction is therefore on the size of hashable terms. Again this restriction seems serious (e.g., general hash chains and trees

are now excluded) but not unreasonable because many protocols only use rather small terms, e.g., one-time signature schemes only use hashes of single nonces. In fact, for the restriction to hashing single nonces, we obtain a positive result.

Another restriction is to give up the ideal secrecy property of the hash functions, i.e., to give at least the ideal adversary an operator that inverts hash. The technical motivation is that this clearly gets rid of the commitment problem.[2] On the application side, this restriction excludes all protocols where one-wayness is the core property for which a hash function is used. However, we can use such a model in protocols where shortening of messages with collision resistance is the core property desired. Note that in the Dolev-Yao model, we can have collision freeness, i.e., no equality between hashes of different terms, without secrecy. The realization of such an operator by a real cryptographic function would of course still have to be collision-resistant and thus one-way if it is sufficiently shortening. Anyway, we still obtain an impossibility result in this case: No shortening hash function exists that enables a secure realization of a Dolev-Yao model with hashes even without secrecy. If we combine giving up secrecy with not hashing payloads and only terms of constant size, then we obtain a positive result.

Of course the restrictions that we considered are not the only conceivable ones; in particular it may be interesting to find other positive cases in the standard model of cryptography than the two that we prove.

2 Summary of Reactive Simulatability/UC, also with Random Oracle

As our results are for the security definitions of BRSIM/UC, we first briefly review this notion. Reactive simulatability/UC is used for comparing an ideal and a real system with respect to security. It was first generally defined in [38] and is based on simulatability definitions for secure (one-step) function evaluation [25, 26, 12, 33, 16]. It was extended in [39, 17] and has since been used in many ways for proving individual cryptographic systems and general theorems.

We believe that all our following results are independent of the small differences between the definition styles and therefore write "BRSIM/UC" and similar term pairs like "ideal system/functionality". However, we have to use a specific formalism for the actual results, and we use that from [39]. 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, often denoted by A, who is often given more power than the honest users; in particular in real systems A typically controls the network. A and H can interact; this corresponds to known- and chosen-message attacks etc.

[2] Furthermore, hash functions are known not to offer cryptographic secrecy in the same strong sense as encryption schemes because they are deterministic, but this can be addressed by allowing tests of hash values also in the ideal system.

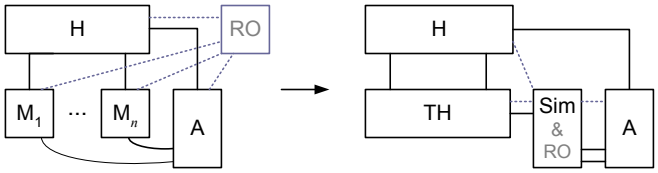


Fig. 1. Overview of blackbox reactive simulatability with a real system on the left and an ideal system on the right, and a potential random oracle. The views of H must be indistinguishable.

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 as a blackbox, such that arbitrary honest users cannot distinguish whether they interact with the real system and the real adversary, or with the ideal system and the simulator with its blackbox. Indistinguishability, here applied to the two families of views of the honest users, is the well-known notion from [42]. We always assume that all parties are polynomial-time.

A formal representation of random oracles in the UC framework was given in [27]. This can be used one-to-one in the BRSIM terminology. In a real system, each machine has distinguished connections for querying the random oracle RO, which is the usual stateful machine from [13] that generates a random string as "hash value" for each message m when it is first queried about m. In the ideal system with a simulator, these distinguished connections connect to the simulator, i.e., the simulator learns every query to the random oracle and can give arbitrary answers. This is also shown in Figure 1.[3]

3 Informal Overview of the Impossibility Proofs

In this section, we present our results as proof sketches with a minimum amount of notation. Later we define more notation and precise assumptions, and then extend the proof sketches to full proofs.

In the following, messages may occur in several representations, which we distinguish by superscripts. We write terms in the Dolev-Yao sense without superscipt, e.g., $h := \mathsf{hash}(m)$ for a hash term. The real cryptographic versions get a superscript $^{\mathsf{r}}$, e.g., $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$ for the corresponding real bitstring, computed by applying a real hash function $\mathsf{hash}^{\mathsf{r}}$ to the real representation m^{r} of m. The users and adversaries may concretely address the terms/bitstrings in yet another way when interacting with the real or ideal functionality (hopefully indistinguishably, hence we need only one notation); we write these representations with superscripts ^{u} for honest user u and $^{\mathsf{a}}$ for the adversary. (Using the actual terms as these representations is a special case.)

In the figures we write H_u for the actual user u, which is a part of the global H in Figure 1.

[3] Alternatively, one could use a correct random oracle also in the ideal system and only allow the simulator to eavesdrop the queries of some or all parties. However, this weakens the power of the simulator considerably, and most of our impossibility proofs for the standard model of cryptography would hold for this model with only minor changes.

3.1 Scenarios with Payloads

Our first scenario in Figure 2 demonstrates that the real hash function $\mathsf{hash}^{\mathsf{r}}$ must at least be collision-resistant in order to offer a sound implementation of a Dolev-Yao model with hashes and payloads. This is not surprising, but we need the collision resistance in the next proofs. The proof idea is that otherwise the adversary can find two colliding

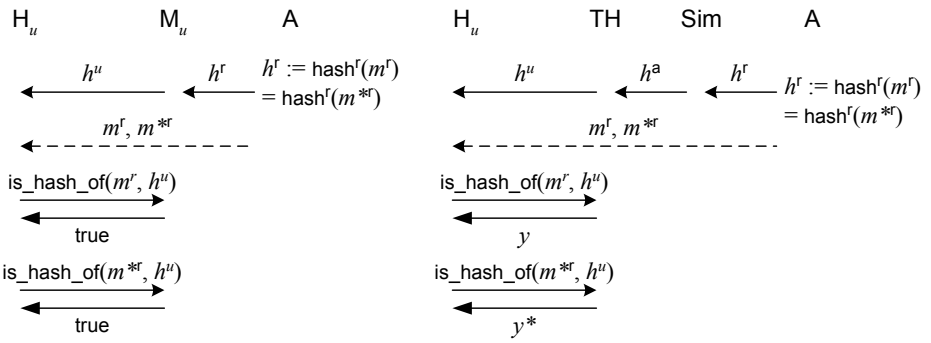


Fig. 2. Counterexample with payload hashing for not collision-resistant hash function.

payloads m^{r} and $m^{*\mathsf{r}}$ and send their hash h^{r} to an honest party u. It also exchanges m^{r} and $m^{*\mathsf{r}}$ with the user u outside the system. Recall that all BRSIM/UC variants allow such outside exchanges for chosen-message attacks etc.; we denote them by dashed arrows in the interaction figures. Then user u uses the ideal or real functionality to check whether the message received through the system is the hash of both payloads. In the real system (on the left in all our interaction figures), the answer is true both times by the choice of h^{r}. However, the ideal collision freeness of the ideal system (on the right in all our interaction figures) does not allow this; hence the ideal and the real system are distinguishable.

The major challenge in formalizing this proof sketch is in the treatment of payloads, because most Dolev-Yao models do not put the actual payloads into the terms. Below we make precise assumptions on this treatment and define the ideal collision freeness, and then turn this sketch into a proof.

Our second scenario in Figure 3 demonstrates that even with a collision-resistant function $\mathsf{hash}^{\mathsf{r}}$, a sound implementation of a Dolev-Yao model with hashes and payloads is impossible if the ideal Dolev-Yao functionality offers ideal secrecy. By ideal secrecy we mean that an adversary who obtains the hash of an otherwise unknown term cannot do better than comparing this hash with self-made hashes of guessed terms. The scenario is that an honest party u selects a random payload m^{r} of length $2k$ (where k is the security parameter), sends it to the adversary outside the system, and sends the hash of this payload to the adversary through the ideal or real system. From the real system, the adversary gets the real hash $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$, tests whether this is indeed the correct hash value of the payload, and tells the result to u outside the system. By the ideal secrecy, the simulator Sim for the ideal system cannot find out more about m^{r} than excluding polynomially many guesses. Using the collision resistance of the real hash

function, we show that Sim can consequently only guess h^{r} with negligible probability, and thus cannot simulate this scenario correctly.

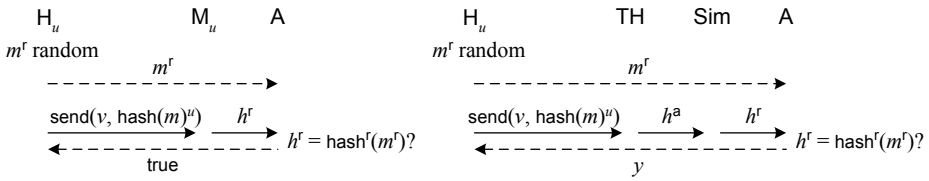


Fig. 3. Counterexample with payload hashing and ideal secrecy for collision-resistant hash function.

The technical difficulties with the full proof for this scenario lie in an appropriate formulation of the ideal secrecy, independent of one specific Dolev-Yao model.

Our third scenario in Figure 4 demonstrates that omitting the ideal secrecy requirement does not help as long as the real hash function is shortening as well as collision-resistant, and thus one-way. Here the real adversary agrees on a random payload m^{r} with an honest user u outside the system, and sends its hash h^{r} to u through the system. The user tests, using the ideal or real functionality, whether the obtained message is indeed a hash of m^{r}. In the real system, the output is clearly true. The best way for the simulator to cause the same output from the ideal functionality would be to send the term $h = \mathsf{hash}(m)$ via TH in the first step. However, this would require guessing m^{r} and thus breaking the one-way property of $\mathsf{hash}^{\mathsf{r}}$.

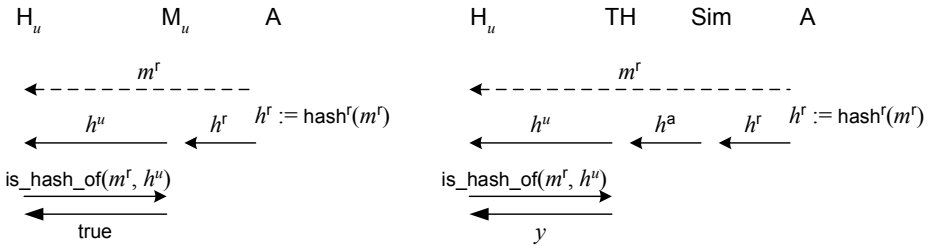


Fig. 4. Counterexample with payload hashing for shortening hash function.

The difficulty with the full proof for this scenario, besides the question of payload representations as in the first scenario, is that $\mathsf{hash}(m)$ is not the only term that causes the output true. For instance, $\mathsf{D}(\mathsf{E}(\mathsf{hash}(m)))$ or $\mathsf{hash}(\mathsf{D}(\mathsf{E}(m)))$ for en- and decryption operators E and D are other such terms. We therefore have to be careful in how we can argue that every successful strategy for the simulator really leads to a successful algorithm that extracts m^{r} and thus breaks the one-way property.

3.2 Scenarios without Payloads

After showing that payloads and hashes in Dolev-Yao models lead to comprehensive impossibility results for secure realizations in the sense of BRSIM/UC, we consider restricted Dolev-Yao models without payloads. However, as long as this is the only restriction, we still prove impossibility. The basic proof idea is to let the users and the adversary simulate payloads by encoding them into the structure of long terms. Concretely, we use a list of $2k$ nonces and encode a payload as a bit vector $\vec{b}$ that selects a sublist of these nonces. Instead of nonces, any other type can be used of which one can generate $2k$ instances that are ideally different, e.g., keys.

With a scenario similar to Figure 2, only adding the random choice of the nonces for the encoding, we show that a hash function must be collision-resistant on these bit vectors in order to offer a sound implementation of a Dolev-Yao model with hashes and lists of nonces. Then with a scenario similar to Figure 3, we show that if ideal secrecy is offered no sound implementation exists at all. With a scenario similar to Figure 4, we show that even without ideal secrecy, no sufficiently shortening hash function, in particular one whose output length depends only on the security parameter, yields a sound implementation.

4 Assumptions on Dolev-Yao Models for our Impossibility Results

As explained in the introduction, we would like to work out the impossibility proofs sketched in Section 3 not only for one specific Dolev-Yao model, but for all of them. However, "all Dolev-Yao models" is not a notion that anyone tried to formalize before. Hence we will now characterize Dolev-Yao models and their realizations by rather weak rigorous requirements in order to make our impossibility results as strong as possible.

4.1 Minimum Assumptions on a Dolev-Yao Model with Hashes

In this section we describe the functionality that we assume a Dolev-Yao system with hashes offers. We start with the basic notions of terms, including a hash operator. Recall that hash is essentially a free operator in the term algebra of typical Dolev-Yao models. However, we do not define this strong freeness, but only a weaker property, ideal collision freeness (both because this makes our results stronger, and because not all Dolev-Yao models are actually defined as initial models of an equational specification).

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

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.*[4]

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

b. *A unary operator* hash*, which fulfils* ideal collision freeness*, i.e.,* $\mathsf{hash}(t) \equiv \mathsf{hash}(t') \Rightarrow t \equiv t'$ *for all* $t, t' \in Terms$.
c. *A set* $Hashable\_Terms \subseteq Terms$ *of the terms that are valid operands of the operator* hash*. We speak of a model with* unrestricted hashing *if* $Hashable\_Terms = Terms$.
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 probably nobody doubted).

While our notion of term representations t^u for individual users 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 user 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 simply use t itself in the protocol representations (e.g., "$\mathsf{hash}(m)$" even when someone who does not know m forwards this term). More detailed representations, e.g., in CSP or π-calculus, typically use the concepts of variables inherent to these calculi, usually by matching received messages with a pattern describing the expected message format, and then using the pattern variables in subsequent message constructions. The syntax explicitly made for BRSIM/UC of the Dolev-Yao-style model in [9] 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 do not need a full definition of how a user acquires term representations. However, we define that terms can be sent and that the ideal adversary controls the network as usual in Dolev-Yao models. Furthermore we define that users can hash terms and compare hashes. In some, but not all, Dolev-Yao models this comparison can be made by using a general equality operator corresponding to the term equivalence $\equiv$.

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

a. *If an honest user* 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*).*
c. *If a user* u *(honest or the adversary represented by* $u = \mathsf{a}$*) has a term representation* t^u*, then it also has a representation for the term* $\mathsf{hash}(t)$*. (Typically this is something like the string "*$\mathsf{hash}(t^u)$*".)*
d. *An input* $\mathsf{is\_hash\_of}(t^u, h^u)$ *by a user* u *(honest or* a*) leads to a Boolean output* y *for user* u *with* $y = \mathsf{true}$ *iff* $h \equiv \mathsf{hash}(t)$.

◇

4.2 Payload Assumptions

All Dolev-Yao models in real proof tools have at least payload messages, nonces, and keys as atoms. However, as payloads are particularly problematic in simulations, we define Dolev-Yao models with and without them. A payload m models an application message, i.e., its cryptographic realization m^{r} can be an arbitrary bitstring; examples are emails, payment messages, and digital pictures. In this sense, our scenarios in Section 3 are perfectly natural: The users and the adversary select payloads as arbitrary bitstrings. However, the internal representation of payloads in the terms in Dolev-Yao proof tools is usually a constant supply of payload names or a nonce-like construction of fresh names. We therefore assume that the full ideal functionality maintains a translation table between the real payloads that occur in a system execution and their internal representations.[5]

Definition 3 (Payloads in Dolev-Yao Models in the BRSIM/UC Setting). *A* Dolev-Yao model with payloads *allows us to derive a type (subset)* payload *in the set* $Terms$. *In every execution, every occurring payload term has a fixed realization* m^{r}, *and* $m^{\mathsf{r}} = m'^{\mathsf{r}}$ *implies* $m \equiv m'$. *The range of payload realizations* m^{r} *is at least* $\{0,1\}^{2k}$. *A real payload* m^{r} *can always be used as an input representation* m^u *by every user.* $\diamond$

We now consider how secret a hashed term is in a Dolev-Yao model when the adversary learns its hash. We only need this in our second scenario, where we want to show that an adversary receiving a (representation of) a term $t = \mathsf{hash}(m)$ containing a payload m cannot get significant information about the real payload m^{r} and thus its real hash. In normal Dolev-Yao models, the hash operator is free, and thus there is no inverse operator that the adversary can use to extract m, nor a sequence of such operators. In addition, in many Dolev-Yao models one would represent the initial situation where the adversary does not know m by not giving the adversary any representation of m, thus excluding any possibility that the adversary guesses m. With such a strong assumption the impossibility proof would be easy. However, we allow the more realistic case that the adversary might guess payloads (as, e.g., in [9]). Furthermore, we only make the minimum assumption that payloads are secret in hash terms except for this guessing.

Definition 4 (Ideal Secrecy of New Payloads). *A Dolev-Yao model with hashes offers* ideal secrecy of new payloads *iff the following holds: If user* u *inputs* $\mathsf{send}(v, h^u)$ *where* $h = \mathsf{hash}(m)$ *for a newly chosen payload* m^{r}, *then the ideal adversary, from its output* $\mathsf{receive}(u, v, h^{\mathsf{a}})$ *and without further interaction with the user* u, *cannot obtain more information about* m^{r} *than by learning for* x *bitstrings* m'^{r} *whether* $m'^{\mathsf{r}} = m^{\mathsf{r}}$ *(in addition to its a-priori information), if it interacts at most* x *times with the ideal system and thus in particular if it runs in time* x. $\diamond$

[5] As an additional motivation for this assumption, recall that we want to compare the Dolev-Yao model and its cryptographic realization in the sense of BRSIM/UC. Thus they must offer the same syntactic user interfaces, i.e., in- and output formats. This holds for all definition variants of BRSIM/UC. In particular, in Figure 1 this is the interface between TH or $\mathsf{M}_1, \ldots, \mathsf{M}_n$, respectively, and the entirety of honest users H. In [17], it is the input and output formats of the ideal and real functionality. 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.

Finally, we define the weak freeness property of Dolev-Yao hashes that we need for the third scenario. Essentially this is that without knowing a payload m or its real representation m^{r} one cannot construct a term equivalent to $\mathsf{hash}(m)$. Like other definitions of "knowledge" in cryptography, this is done by defining that the capability to construct such a term implies the capability to find out m^{r}. This reduction is done constructively by an extractor algorithm.

Definition 5 (Minimum Non-Constructibility of Unknown Payload Hashes). *A Dolev-Yao model with hashes offers* minimum non-constructibility of unknown payload hashes *if there exists a polynomial-time algorithm* Ext*, called extractor, such that the following holds: If the ideal adversary (for simplicity at the system start) makes a sequence of inputs and then sends a term t to an honest user such that $t \equiv \mathsf{hash}(m)$ for a payload m, then the extractor, given the transcript of the ideal adversary's in- and outputs, outputs m^{r}.* ◇

For Dolev-Yao models with well-defined and constructible normalizations of terms, the extractor is essentially this normalization: It constructs t and the relation of payload terms and their representations from the transcript (typically the transcript is simply of the form "$\mathsf{send}(v, t^{\mathsf{a}})$" where payloads in t^{a} are in their real representation) and normalizes t; the result is $\mathsf{hash}(m)$, from which m^{r} can be looked up. This clearly holds for typical Dolev-Yao models that only have constructors and destructors like encryption and decryption. It gets more complex in Dolev-Yao models with algebraic operations like XOR; however, specifically XOR is known not to be realizable in BRSIM/UC [8].

4.3 Nonce List Assumptions

For the case without payloads, our scenarios use lists of nonces. We therefore define what we assume about nonces (lists are already in Definition 1). The first assumption is extremely simple and normal, except that some basic Dolev-Yao models only allow a fixed number of nonces, while we need at least $2k$ (as does every Dolev-Yao model suitable for arguing about an unbounded number of sessions).

Definition 6 (Nonces in Dolev-Yao Models). *A* Dolev-Yao model with nonce lists *allows us to derive a type (subset)* nonce *in the set $Terms$. Every participant can use, or explicitly generate, at least $2k$ new nonces (we do not need a fixed syntax for this generation); such nonces are pairwise not equivalent.* ◇

The next definition extends the ideal secrecy of hashed terms, which we earlier defined only for new payloads, to new lists of nonces. More precisely, we define that an ideal hash term does not divulge which of the many potential sublists of a list of nonces was hashed. (We make these weak special assumptions to strengthen the impossibility results, and to avoid complex considerations about prior knowledge in the general case.)

Definition 7 (Ideal Secrecy of New Nonce Lists). *A Dolev-Yao model with hashes offers* ideal secrecy of new nonce lists *iff the following holds: Let user u generate $2k$ new nonces $\vec{n} = (n_1, \ldots, n_{2k})$ and potentially send them to the adversary, select a random bit vector $\vec{b} = (b_1, \ldots, b_{2k}) \xleftarrow{R} \{0,1\}^{2k}$, and input* $\mathsf{send}(v, (\mathsf{hash}(\vec{b} \odot \vec{n}))^u)$

where $\vec{b} \odot \vec{n}$ denotes the sublist consisting of the nonces n_i with $b_i = 1$. Then the ideal adversary, from its output $\mathsf{receive}(u, v, h^{\mathsf{a}})$ *and without further interaction with the user u, cannot obtain more information about $\vec{b}$ than by learning for x bit vectors $\vec{b'}$ whether $\vec{b'} = \vec{b}$, if it interacts at most x times with the ideal system and thus in particular if it runs in time x.* ◇

Finally, we define that the ideal adversary cannot construct a hash over a sublist of nonces without knowing which sublist of the nonces it is using.

Definition 8 (Minimum Non-Constructibility of Unknown Nonce-list Hashes). *A Dolev-Yao model with hashes offers* minimum non-constructibility of unknown nonce-list hashes *if there exists a polynomial-time algorithm* Ext*, called extractor, such that the following holds: If the ideal adversary (for simplicity at the system start) makes a sequence of inputs and then sends a list $\vec{n}$ of $2k$ pairwise different nonces and a term h to an honest user such that $h \equiv \mathsf{hash}(\vec{b} \odot \vec{n})$, then the extractor, given the transcript of the ideal adversary's in- and outputs, outputs $\vec{b}$.* ◇

Again, the existence of such an extractor is clear for Dolev-Yao models with a normalization algorithm because the term $\mathsf{hash}(\vec{b} \odot \vec{n})$ cannot be further reduced and is thus the normal form of every equivalent term. Given the overall list of nonces $\vec{n}$, which the ideal adversary sent separately, the selection of nonces in this term and thus $\vec{b}$ can be read off.

4.4 Minimum Assumptions on a Cryptographic Realization

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 hashes, we make the following (natural) minimum assumptions in the standard model of cryptography: Real channels are insecure; the input to send a term t leads to actual sending of a bitstring t^{r}; and hash terms are realized by applying a fixed (hash) function to the realization of the contained terms.

Definition 9 (Realization of a Dolev-Yao Model with Hashes). *In a* realization of a Dolev-Yao model with hashes in the standard model of cryptography*, 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'. There must be a deterministic, polynomial-time function* $\mathsf{hash}^{\mathsf{r}}$ *such that* $(\mathsf{hash}(t))^{\mathsf{r}} = \mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}})$ *for all $t \in Hashable\_Terms$. An input* $\mathsf{is\_hash\_of}(t^u, h^u)$ *to a machine* M_u *leads to the output* true *iff* $\mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}}) = h^{\mathsf{r}}$.

For nonces, there must be a probabilistic polynomial-time algorithm $\mathsf{G_n}$ *that is used to generate n^r when it is needed for a new nonce n, and $2k$ executions of* $\mathsf{G_n}$ *must yield pairwise different results $n^{\mathsf{r}}_1, \ldots, n^{\mathsf{r}}_{2k}$ with overwhelming probability.* ◇

In realizations with type tagging we can consider an original cryptographic hash function together with the type tag as $\mathsf{hash}^{\mathsf{r}}$. Note that we made no assumptions on the

cryptographic properties of $\mathsf{hash}^{\mathsf{r}}$ and only a weak one on G_{n}; we will show that neither "good" nor "bad" realizations lead to soundness in the sense of BRSIM/UC. [6]

5 Details of the Impossibility Proofs

We now present the missing details for the impossibility proof sketches in Section 3, using the definitions from Section 4.

5.1 Unsoundness of Dolev-Yao Models with Payloads

The first scenario from Section 3.1 becomes the following lemma.

Lemma 1. *(Collision Resistance of the Real Hash Function) If a Dolev-Yao model with hashes and payloads has a realization in the standard model of cryptography that is secure in the sense of BRSIM/UC, then the hash function* $\mathsf{hash}^{\mathsf{r}}$ *in this realization is collision-resistant. For simplicity we define here that a collision for security parameter* k *consists of two messages of length* $2k$*.* □

Proof. We elaborate our first scenario in Figure 2. Assume that $\mathsf{hash}^{\mathsf{r}}$ is not collision-resistant. Then an adversary A can find a collision $m^{\mathsf{r}} \neq m^{*\mathsf{r}}$ with not negligible probability. By Definitions 2 and 3, the adversary and the user can indeed act as described in Section 3.1, and by Definition 9, the output for H_u in the real system is indeed true for both messages. In the ideal system, by Definition 2 the user will only get these outputs if $h \equiv \mathsf{hash}(m)$ and $h \equiv \mathsf{hash}(m^*)$. With the ideal collision freeness (Definition 1) this implies $m \equiv m^*$ and thus $m^{\mathsf{r}} = m^{*\mathsf{r}}$ (with Definition 9), in contradiction to the choice of m^{r} and $m^{*\mathsf{r}}$. ■

The second scenario from Section 3.1 together with this lemma gives us the following theorem.

Theorem 1. *(Unsoundness of Dolev-Yao Models with Hashes and Ideal Secrecy of New Payloads) No Dolev-Yao model with hashes and ideal secrecy of new payloads has a realization in the standard model of cryptography that is secure in the sense of BRSIM/UC.* □

Proof. Assume that a Dolev-Yao model and a realization as specified in the theorem exist. By Lemma 1, the hash function $\mathsf{hash}^{\mathsf{r}}$ in the realization must be collision-resistant. Then $\delta(k) := \max_{h^{\mathsf{r}} \in \{0,1\}^*}(\mathsf{Pr}[\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}}) = h^{\mathsf{r}} :: m^{\mathsf{r}} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}])$ is negligible (as a function of k), because otherwise two random messages of length $2k$ are a collision with not negligible probability. We elaborate our second scenario in Figure 3: The user H_u chooses the payload as $m^{\mathsf{r}} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$. By Definitions 2 and 3, the adversary and the user can indeed act as described in Section 3.1, and by Definition 9, the output

[6] In computational considerations about $\mathsf{hash}^{\mathsf{r}}$ we allow $\mathsf{hash}^{\mathsf{r}}$ to depend on the security parameter k, which is fixed in each system execution. To allow collision-resistance in the sense of the typical cryptographic definition, it should even depend on a key pk chosen at the beginning of each system execution; our proofs could easily be adapted to this case.

for A in the real system is $\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$. In the ideal system, the simulator Sim gets an output $\mathsf{receive}(u, v, h^{\mathsf{a}})$ from the Dolev-Yao model TH and has to produce a string h^{r} for the adversary. For indistinguishability, this string must fulfill $h^{\mathsf{r}} = \mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$ with overwhelming probability.

Definition 4 is applicable and implies that Sim (which acts as the ideal adversary here), with x calls to TH, cannot obtain more information about m^{r} than by learning for x bitstrings m'^{r} whether $m'^{\mathsf{r}} = m^{\mathsf{r}}$. As m^{r} is uniformly random, the probability that Sim hits $m'^{\mathsf{r}} = m^{\mathsf{r}}$ in this process is $x/2^{2k}$, where x is polynomial because Sim is polynomial-time. Thus this probability is negligible. In the other case, the optimal choice of h^{r} for Sim is the most likely hash value over the remaining $2^{2k} - x$ possible payloads. The probability that this value is correct is at most $\delta(k)2^{2k}/(2^{2k} - x)$. This is negligible because x is polynomial. ■

Next we consider the case without secrecy of hashed terms, but with the additional assumption that the output length of the real hash function depends only on the security parameter, not on the input length. (Weaker definitions of significantly shortening hash functions would also suffice.) The third scenario from Section 3.1 together with Lemma 1 gives us the following theorem.

Theorem 2. *(Unsoundness of Dolev-Yao Models with Hashes and Payloads without Secrecy) No Dolev-Yao model with hashes and payloads, even without ideal secrecy, has a realization in the standard model of cryptography that is secure in the sense of BRSIM/UC and where the real hash function is shortening. For simplicity, we require that the range of a shortening hash function is* $\{0,1\}^k$. □

Proof. Assume that a Dolev-Yao model and a realization as specified in the theorem exist. By Lemma 1, the hash function $\mathsf{hash}^{\mathsf{r}}$ in the realization must be collision-resistant. As $\mathsf{hash}^{\mathsf{r}}$ is also shortening, it is one-way. (Otherwise the following algorithm finds a collision with not negligible probability: Select a random payload $m^{\mathsf{r}} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$, use the assumed inversion algorithm $\mathsf{A}_{\mathsf{owf}}$ to find a preimage m'^{r} of $\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$, and output m^{r} and m'^{r} if they are unequal. This holds because all payloads, except less than 2^k and thus negligibly many, collide with another one. If $\mathsf{A}_{\mathsf{owf}}$ succeeds for such a payload m^{r}, then with probability at least $1/2$ we have $m'^{\mathsf{r}} \neq m^{\mathsf{r}}$.)

We now elaborate our third scenario in Figure 4. By Definitions 2 and 3, the adversary and the user can indeed act as described in Section 3.1, and by Definition 9 the output for H_u in the real system is indeed true. By the assumption of the proof, the simulator can achieve the same result in the ideal system with overwhelming probability. Hence it makes an input $\mathsf{send}(v, u, h^{\mathsf{a}})$ where, by Definition 2, $h \equiv \mathsf{hash}(m)$ for the term m that is realized as m^{r}. Definition 5 is applicable to our scenario and essentially states that Sim, which acts as the ideal adversary, must know m^{r} for this. More precisely, we use the postulated extractor Ext to extract m^{r} from the transcript of Sim whenever Sim is successful. This gives us an inversion algorithm $\mathsf{A}_{\mathsf{owf}}$ for the function $\mathsf{hash}^{\mathsf{r}}$ that succeeds with not negligible probability, in contradiction to the one-wayness of $\mathsf{hash}^{\mathsf{r}}$. Concretely, $\mathsf{A}_{\mathsf{owf}}$ is the combination of TH, Sim and Ext. This is indeed a non-interactive algorithm as required in the definition of one-wayness: Sim initially gets one input h^{r} and TH has no input so far. Then Sim and TH interact with each other, but interaction with H or A would be distinguishable from the real system. ■

5.2 Unsoundness without Payloads

Now we work out the scenarios for restricted Dolev-Yao models without payloads. As sketched in Section 3.2, we proceed similar to the scenarios with payloads, letting the users and the adversary replace payloads by bit vectors that select sublists of nonces. For this, we first define collision resistance and one-wayness with respect to the bit vectors.

Definition 10 (Bit-vector Collision Resistance and One-Wayness). *Let a Dolev-Yao model with hashes and a realization in the standard model of cryptography with the hash function* hash^r *be given. We say that* hash^r *is* bit-vector collision-resistant *if every polynomial-time adversary can only find a list* $\vec{n}^\mathsf{r} = (n^\mathsf{r}_1, \ldots, n^\mathsf{r}_{2k})$ *of* $2k$ *pairwise different real nonces and bit vectors* $\vec{b} \neq \vec{b^*} \in \{0,1\}^{2k}$ *with* $\mathsf{hash}^\mathsf{r}(\vec{b} \odot \vec{n}^\mathsf{r}) = \mathsf{hash}^\mathsf{r}(\vec{b^*} \odot \vec{n}^\mathsf{r})$ *with negligible probability, where* $\vec{b} \odot \vec{n}^\mathsf{r}$ *for a bit vector* $\vec{b} = b_1, \ldots, b_{2k}$ *denotes the sublist consisting of the nonces* n^r_i *with* $b_i = 1$.

We say that hash^r *is* bit-vector one-way *if every polynomial-time algorithm* $\mathsf{A}_{\mathsf{owf}}$*, on input* $h^\mathsf{r} := \mathsf{hash}^\mathsf{r}(\vec{b} \odot \vec{n}^\mathsf{r})$ *for random* $\vec{b} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$ *and real nonces generated with* G_n*, can only output a bit vector* $\vec{b^*} \in \{0,1\}^{2k}$ *with* $h^\mathsf{r} = \mathsf{hash}^\mathsf{r}(\vec{b^*} \odot \vec{n}^\mathsf{r})$ *with negligible probability.* ◇

Lemma 2. *(Bit-vector Collision Resistance of the Real Hash Function) If a Dolev-Yao model with hashes where* $Hashable\_Terms$ *contains at least all lists of up to* $2k$ *nonces has a realization in the standard model of cryptography that is secure in the sense of BRSIM/UC, then the hash function* hash^r *in this realization is bit-vector collision-resistant.* □

Proof. Assume that hash^r is not bit-vector collision-resistant. We then use the scenario in Figure 5 (adapted from Figure 2): A real adversary A can find, with not negligible probablity, a list $\vec{n}^\mathsf{r}$ of $2k$ pairwise different nonces and bit vectors $\vec{b} \neq \vec{b^*} \in \{0,1\}^{2k}$ with $h^\mathsf{r} := \mathsf{hash}^\mathsf{r}(\vec{b} \odot \vec{n}^\mathsf{r}) = \mathsf{hash}^\mathsf{r}(\vec{b^*} \odot \vec{n}^\mathsf{r})$. It sends the individual nonces and h^r to user H_u through the system, and $\vec{b}$ and $\vec{b^*}$ outside the system. User H_u thus obtains representations of the nonces and the hash term. It now inputs $\mathsf{is\_hash\_of}(\vec{b} \odot \vec{n}^u, h^u)$, where $\vec{b} \odot \vec{n}^u$ denotes the representation of the corresponding sublist of the nonces, and $\mathsf{is\_hash\_of}(\vec{b^*} \odot \vec{n}^u, h^u)$. In the real system, the result is true both times by Definition 9. In the ideal system, this result would imply $\vec{b} \odot \vec{n} \equiv \vec{b^*} \odot \vec{n}$ with the ideal collision freeness (Definitions 1 and 2). However, as the real nonces n^r_i are pairwise different, their term versions n_i are pairwise non-equivalent (Definition 9). The selection of sublists by $\vec{b}$ and $\vec{b^*}$ therefore yields non-equivalent sublists (Definition 1). This is the desired contradition. ■

Theorem 3. *(Unsoundness of Dolev-Yao Models with Hashes and Ideal Secrecy of New Nonce Lists) No Dolev-Yao model with hashes and ideal secrecy of new nonce lists has a realization in the standard model of cryptography that is secure in the sense of BRSIM/UC.* □

Proof. We use the scenario in Figure 6. The user H_u selects a bit vector $\vec{b} \leftarrow \{0,1\}^{2k}$ and sends $2k$ new nonces to the adversary via the system. It also sends the hash of the

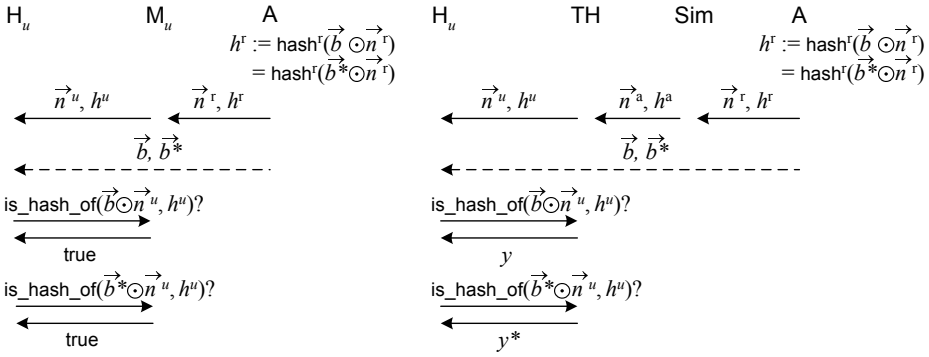


Fig. 5. Counterexample with nonce lists for not bit-vector collision-resistant hash function.

sublist of these nonces selected by $\vec{b}$ through the system, and $\vec{b}$ on an external channel. Thus in the real system, the adversary A obtains the real nonces $\vec{n}^{\mathsf{r}}$, the hash value $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(\vec{b} \odot \vec{n}^{\mathsf{r}})$, and $\vec{b}$. It tests whether the received h^{r} is correct and tells the result to the user H_u; in the real system this is always true. Hence in the ideal system, the simulator has to produce real nonces $\vec{n}^{\mathsf{r}}$ with an indistinguishable probability distribution and the hash value $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(\vec{b} \odot \vec{n}^{\mathsf{r}})$, without initially knowing $\vec{b}$. With overwhelming probability, it must make the nonces n_i^{r} pairwise different because otherwise the situation is distinguishable from the real system (Definition 9); we continue with this case.

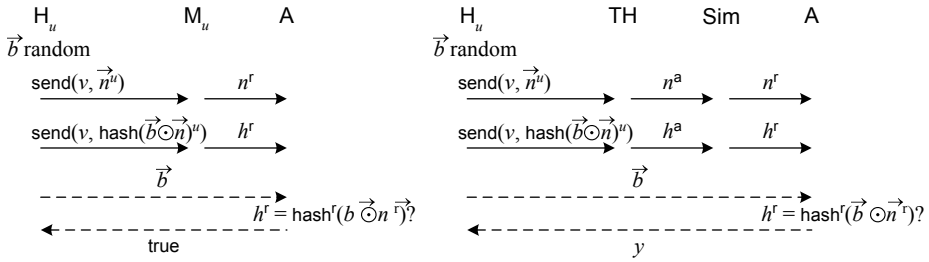


Fig. 6. Counterexample with nonce lists and ideal secrecy for bit-vector collision-resistant hash function.

By the ideal secrecy (Definition 7), Sim cannot learn more about $\vec{b}$ with x interactions with TH than learning for x vectors $\vec{b'}$ whether $\vec{b'} = \vec{b}$. The probability that it hits $\vec{b'} = \vec{b}$ in this process is negligible. We now consider the other case.

We first show that it is hard to choose a correct h^{r} without the extra knowledge about $\vec{b}$: We claim that for every polynomial-time algorithm S, the probability

$$P_{\mathsf{S}}(k) := \mathsf{Pr}[\mathsf{hash}^{\mathsf{r}}(\vec{b} \odot \vec{n}^{\mathsf{r}}) = h^{\mathsf{r}} \wedge \mathsf{pwd}(\vec{n}^{\mathsf{r}}) :: (\vec{n}^{\mathsf{r}}, h^{\mathsf{r}}) \leftarrow \mathsf{S}(k); \vec{b} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}]$$

is negligible, where pwd denotes the predicate that the elements of a list are pairwise different. Assume it were not. Then the algorithm $(\vec{n}^{\mathsf{r}}, h^{\mathsf{r}}) \leftarrow \mathsf{S}(k); \vec{b}, \vec{b^*} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$

finds bit-vector collisions with not negligible probability, in contradiction to the bit-vector collision resistance of $\mathsf{hash}^{\mathsf{r}}$.[7] Similar to the proof of Theorem 1, the probability that Sim succeeds, given that it excluded x bit vectors, is only negligibly larger (by a factor of at most $2^{2k}/(2^{2k}-x)$). This is the desired contradiction. ∎

Theorem 4. *(Unsoundness of Dolev-Yao Models with Hashes and Nonce Lists without Secrecy) Let a Dolev-Yao model with hashes be given whose set $Hashable\_Terms$ contains at least all lists of up to $2k$ nonces. Then there is no sound cryptographic implementation in the sense of BRSIM/UC in the standard model of cryptography with a shortening real hash function.* □

Proof. Assume a Dolev-Yao model and a realization as described in the theorem exist, and let $\mathsf{hash}^{\mathsf{r}}$ be the real hash function used. Recall that for simplicity we defined that the output length of a shortening hash function is k. By Lemma 2, $\mathsf{hash}^{\mathsf{r}}$ is bit-vector collision-resistant. We first see that $\mathsf{hash}^{\mathsf{r}}$ is bit-vector one-way similar to the start of the proof of Theorem 2: Assume there were a successful inversion algorithm $\mathsf{A}_{\mathsf{owf}}$. Then we generate a collision with the following algorithm: $\vec{n}^{\mathsf{r}} \xleftarrow{\mathcal{R}} (\mathsf{G}_{\mathsf{n}}(k))^{2k}$; $\vec{b} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$; $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(\vec{b} \odot \vec{n}^{\mathsf{r}})$; $\vec{b^*} \leftarrow \mathsf{A}_{\mathsf{owf}}(h^{\mathsf{r}})$; if $\vec{b^*} \neq \vec{b}$ output $(\vec{b}, \vec{b^*})$. This succeeds with not negligible probability by the same arguments as in the proof of Theorem 2.

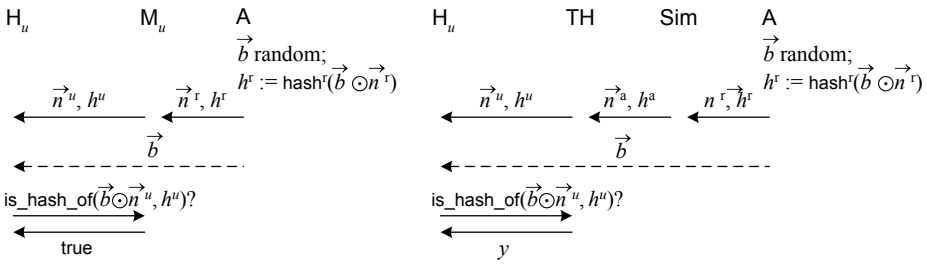


Fig. 7. Counterexample with nonce lists for shortening hash function.

Using the bit-vector one-wayness, we argue similar to the main part of the proof of Theorem 2, see Figure 7: The adversary chooses a random bit vector $\vec{b} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$ and $2k$ nonces $\vec{n}^{\mathsf{r}}$ with the correct generation algorithm G_{n}. With overwhelming probability the real nonces are pairwise different, and thus also the corresponding nonce terms (Definition 9). We continue with this case. It sends the nonces and $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(\vec{b} \odot \vec{n}^{\mathsf{r}})$ to user u through the system and $\vec{b}$ on an external channel. The user tests, using the real or ideal functionality, whether it received the hash of the correct sublist of the nonces. In the real system the result is always true by Definition 9. In the ideal system, the simulator Sim has to enter a command $\mathsf{send}(v, u, h^{\mathsf{a}})$ such that H_u also obtains $y =$

[7] This can be seen by considering the set X of pairs $(\vec{n}^{\mathsf{r}}, h^{\mathsf{r}})$ such that a bit vector $\vec{b}$ fits with probability at least $P_{\mathsf{S}}(k)/2$. The probability of X must be at least $P_{\mathsf{S}}(k)/2$. For a pair $(\vec{n}^{\mathsf{r}}, h^{\mathsf{r}}) \in X$, the probability that two bit vectors both fit (and thus in particular are a collision) is at least $P_{\mathsf{S}}(k)^2/4$.

true with overwhelming probability. By Definition 2 this implies $h \equiv \mathsf{hash}(\vec{b} \odot \vec{n})$. Hence the simulator has to construct a representation h^{a} of $\mathsf{hash}(\vec{b} \odot \vec{n})$. Definition 8 is applicable to our scenario and essentially states that Sim, which acts as the ideal adversary, must know $\vec{b}$ for this. More precisely, we use the postulated extractor Ext to extract $\vec{b}$ from the transcript of Sim whenever Sim is successful. This gives us a bit-vector inversion algorithm for the function $\mathsf{hash}^{\mathsf{r}}$ that succeeds with not negligible probability, in contradiction to the bit-vector one-wayness of $\mathsf{hash}^{\mathsf{r}}$. ■

6 Soundness Results

In this section we show that Dolev-Yao-style hashes can be proven sound in the random oracle model, and under specific restrictions on the usage of hash functions or their properties in the ideal system even in the standard model of cryptography.

6.1 Soundness of Dolev-Yao Models with Hashes in the Random Oracle Setting

The first soundness result states that normal Dolev-Yao models without specific restrictions can be proven sound in the random oracle model. As an overall result for an operator-rich Dolev-Yao model with hashes, this requires 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 that of [9]. However, what happens specifically with the hashes can be explained well without specific notation from [9]. We sketch this in this section, leaving more details to Appendix A.

The Dolev-Yao functionality is that of a free hash operator with unrestricted hashing and with ideal secrecy in the typical sense that the adversary, upon learning a hash value, has no deconstruction operator or other ways to obtain information about the contained term except by the input is_hash_of for comparing a message and a potential hash. The only additional power that the ideal adversary gets compared with honest users is to make hashes with unknown preimages, i.e., terms that could be written $\mathsf{hash}(?)$. The preimages will remain unknown forever; that this works in the realization is a consequence of the random oracle model.

In the cryptographic realization, the operator hash is essentially realized by the random oracle. The only addition is that the bitstrings are typed, i.e., the actual realization $\mathsf{hash}(t)^{\mathsf{r}}$ of a hash term is the pair $(\text{'hash'}, \mathsf{RO}(t^{\mathsf{r}}))$ where 'hash' is a fixed string and RO abbreviates the result of a (in reality stateful) random oracle call.

Our security claim is that this realization is as secure as this ideal Dolev-Yao-style system in the sense of BRSIM/UC in the random oracle model; see Section 2.

Theorem 5. *(Soundness of a Dolev-Yao Model with Hashes in the Random Oracle Model) A Dolev-Yao model with unrestricted hashing and secrecy of hashed terms, defined in detail in Appendix A.1, can be securely implemented by a canonical cryptographic realization, defined in detail in Appendix A.2, in the sense of BRSIM/UC in the random oracle model.* □

For proving BRSIM/UC for the underlying Dolev-Yao-style model without hashes and its realization, a simulator Sim has been defined in [9]. It maintains a table of term representations m^{a} and corresponding bitstrings m^{r}, reuses old terms or bitstrings when possible when forwarding messages between TH and A, and otherwise constructs a new correspondence. Essentially, we extend this simulator as follows (details are given in Appendix A.4): For each hash term h^{a}, our Sim also stores the contained term m^{a} if it knows it (i.e., the ideal adversary would know it).

If Sim gets a new hash term h^{a} from TH, it constructs h^{r} as a new type-tagged random string.

If Sim gets a new type-tagged hash string h^{r} from A, it forwards a hash with unknown preimage to TH, i.e., $h = \mathsf{hash}(?)$. We see later that this only happens when A did not construct h^{r} by calling the random oracle (which is simulated by Sim); hence no preimage will ever be found.

If Sim gets a random oracle query for message m^{r} from A or H, it checks whether it already has a hash string h^{r} for m^{r}. For this, Sim constructs the corresponding term representation m^{a} and looks whether it knows a hash representation h^{a} with m^{a} as the contained term. Otherwise, it checks whether m^{a} is the preimage of some hash term h^{a} for which it did not know the preimage yet (but the ideal functionality TH does), by inputting $\mathsf{is\_hash\_of}(m^{\mathsf{a}}, h^{\mathsf{a}})$ for all the possible term representations h^{a}. If yes, it stores this contained-term relation and uses the corresponding real hash h^{r}. Otherwise it constructs h^{r} as a new type-tagged random string, inputs a new term $h = \mathsf{hash}(m)$ into TH, and stores the contained-term relation of h^{a} and m^{a} as well as h^{r}.

The correctness of this simulator is proved in Appendix A.5.

6.2 Soundness Results in the Standard Model

Finally, we briefly present two restricted but still practically useful types of Dolev-Yao models with hashes that have secure realizations even in the standard model of cryptography. Both models allow the ideal adversary to construct hash terms with unknown preimages, i.e., terms $\mathsf{hash}(?)$. In contrast to the model in Section 6.1, the adversary can later provide a preimage for such a term. Both realizations require a collision-resistant hash function; in the first case the hash function must also be one-way.

The first type of Dolev-Yao model gives up the ideal secrecy, and can then work with a significant class of hashable terms. By the size of a term t we mean the number of nodes in the tree representation of t.

Theorem 6. *(Soundness Without Payloads or Secrecy for Constant-Sized Terms) A Dolev-Yao model without secrecy of hashed terms where terms in $Hashable\_Terms$ do not contain payloads and are at most of a constant size l can be securely realized in the sense of BRSIM/UC with arbitrary collision-resistant, one-way hash functions in the standard model of cryptography.* □

We believe that this theorem can be extended to terms that contain payloads, but only together with fresh nonces that remain secret, but the practical usefulness does not seem to justify the overhead of such a condition that must be defined over an overall system execution.

The second theorem offers the ideal secrecy of typical Dolev-Yao models of hashing, but only individual nonces can be hashed, as for instance in one-time signatures.

Theorem 7. *(Soundness With Secrecy for Nonce Hashing) A Dolev-Yao model with secrecy of hashed terms and where the set $Hashable\_Terms$ contains only individual nonces can be securely realized in the sense of BRSIM/UC with arbitrary collision-resistant hash functions in the standard model of cryptography.* □

Sketches of both proofs are postponed to Appendix B. Similar to the random oracle case, for the precise model we rely on the existing Dolev-Yao model of [9] and extend it with hashes. The detailed models are therefore very similar to Appendix A.1 and A.2.

7 Conclusion

We have investigated whether Dolev-Yao models with hashes or one-way functions can be realized in the sense of BRSIM/UC, i.e., with the Dolev-Yao model as the ideal functionality. We showed that this is not possible for the standard type of such Dolev-Yao models. This impossibility result holds for all polynomial-time computable functions in the role of the real hash function. No such absolute impossibility proof for a Dolev-Yao model was previously known in the literature – the proofs of impossibility for Dolev-Yao models with XOR are reductions.

In contrast, we showed that the realization is possible in the random oracle model.

We then considered several restrictions of the Dolev-Yao model or its ideal properties that have a potential to simplify simulations. For these, we obtained the following additional impossibility results: First, even if no payloads can be hashed, but only cryptographic terms (in fact, lists of nonces are sufficient), we can still show impossibility. Secondly, even if we give up the ideal secrecy property of hashes (retaining the ideal collision freeness so that the model is still reasonable), we obtain impossibility for all polynomial-time computable functions with message-independent output length (and thus all typical hash functions). In particular, this is the first impossibility proof for a Dolev-Yao model that does not assume any ideal secrecy property.

On the positive side, we obtain BRSIM/UC in the standard model of cryptography for two cases: One includes ideal secrecy, but only allows hashing of single nonces, e.g., for the use in one-time signatures. The other gives up ideal secrecy, but allows hashing of arbitrary cryptographic terms, i.e., terms without payloads, up to an arbitrary but constant size.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th TACS*, pages 82–94, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP TCS*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.
3. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th ESORICS*, volume 3193 of *LNCS*, pages 89–108. Springer, 2004.

4. M. Backes and M. Dürmuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proc. 18th IEEE CSFW*, pages 78–93, 2005.
5. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.
6. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE CSFW*, pages 204–218, 2004.
7. M. Backes and B. Pfitzmann. Cryptographic key secrecy of the strengthened Yahalom protocol via a symbolic security proof. Research Report 3601, IBM Research, 2005.
8. M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proc. 10th ESORICS*, volume 3679 of *LNCS*, pages 178–196. Springer, 2005.
9. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM CCS*, pages 220–230, 2003.
10. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th ESORICS*, volume 2808 of *LNCS*, pages 271–290. Springer, 2003.
11. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.
12. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
13. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM CCS*, pages 62–73, 1993.
14. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE CSFW*, pages 82–96, 2001.
15. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006. To appear.
16. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
17. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE FOCS*, pages 136–145, 2001.
18. R. Canetti and M. Fischlin. Universally composable commitments. In *Proc. CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.
19. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*. Springer, 2006. To appear.
20. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Proc. EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002.
21. A. Datta, A. Derek, J. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Proc. 3rd Theory of Cryptography Conference (TCC)*. Springer, 2006. To appear.
22. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. 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.
23. A. Datta, R. Küsters, J. C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. In *Proc. 2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *LNCS*, pages 476–494. Springer, 2005.
24. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

25. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th ACM STOC*, pages 218–229, 1987.
26. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Proc. CRYPTO '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1990.
27. D. Hofheinz and J. Müller-Quade. Universally composable commitments using random oracles. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 58–76. Springer, 2004.
28. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.
29. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th ESOP*, pages 77–91, 2001.
30. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.
31. P. Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM CCS*, 2005.
32. M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.
33. S. Micali and P. Rogaway. Secure computation. In *Proc. CRYPTO '91*, volume 576 of *LNCS*, pages 392–404. Springer, 1991.
34. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
35. 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.
36. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
37. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
38. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000.
39. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.
40. P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley, 2001.
41. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. Submitted to CSFW 2006.
42. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE FOCS*, pages 80–91, 1982.

Appendix

In this appendix, we present the omitted proofs from Section 6. We show the soundness proof of a general Dolev-Yao model in the random oracle model in detail, and sketch proofs of the two soundness results for restricted Dolev-Yao models in the standard model of cryptography.

A A Sound Dolev-Yao Model with Hashes in the Random Oracle Setting

This section contains the proof that Dolev-Yao-style hash functions with ideal secrecy of hashed terms can be proven sound in the random oracle model, as sketched in Section 6.1. We first present the additions needed for hashing to the ideal and real functionality of [9], then additions to the simulator, and finally the new parts of the proof of correctness of this simulator. To make it credible that we really can add the functionality we sketched above to the model of [9], we now use the notation from [9].

We first repeat some general notation from [9]. By $x := y$++ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $\mathsf{len}(m)$, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The elements of a list $l := (x_1, \ldots, x_j)$ are 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 $D :\Leftarrow x$.

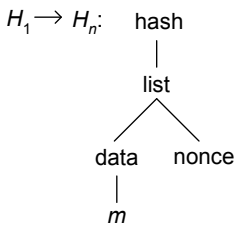
A.1 Hash Additions to the Ideal Dolev-Yao-style System

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

The trusted host keeps track of the length of messages (this is needed because this length leaks to the adversary even in encryptions) using a tuple L of abstract length functions. One function from L that we need below is $\mathsf{max\_len}(k)$, which can be an arbitrary polynomial denoting the maximum length of processed messages. We extend L with a function $\mathsf{hash\_len}^*(k)$ denoting the length of an abstract hash m, i.e., we assume that the hash length is independent of the message length.

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

An example is shown in Figure 8, where user H_1 sends a hash to user H_n. The left side indicates the main action that has happened so far: a list consisting of a payload message m and a nonce N has been hashed. The database contains the payload message (of type data), the nonce, the list, and the hash. The figure shows that this message has arrived safely so that H_n has a handle to the hash, but due to the ideal secrecy H_u has not obtained handles to the subterms.



ind	type	arg	hnd_1	...	hnd_n	len
1	data	(*m*)	1	...	↓	1000
2	nonce	()	2	...	↓	128
3	list	(1,2)	3	...	↓	1156
4	hash	(3)	4	...	1	160

Fig. 8. Example of the database representation of terms.

In detail, the database attributes of D are defined as follows; the only differences to [9] due to adding hashes are an augmented type set.

- $ind \in \mathcal{IND}$, called index, consecutively numbers all entries in D. The set $\mathcal{IND}$ is isomorphic to $\mathbb{N}$. The index is used as a primary key attribute of the database, i.e., one can write $D[i]$ for the selection $D[ind = i]$.
- $type \in typeset$ defines the type of the entry. We add the type 'hash' to $typeset$ from [9].
- $arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Many values a_i are indices of other entries in D and thus in $\mathcal{IND}$; they are sometimes distinguished by a superscript "ind".
- $hnd_u \in \mathcal{HND} \cup \{\downarrow\}$ 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 $\mathcal{HND}$ is yet another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $len \in \mathbb{N}_0$ denotes the "length" of the term, computed using the functions from L.

Initially, D is empty. As additional state parts, $\mathsf{TH}_{\mathcal{H}}$ has a counter $size \in \mathcal{IND}$ for the current size of D, and counters $curhnd_u$ (current handle) for $u \in \mathcal{H} \cup \{\mathsf{a}\}$, denoting the most recent handle number assigned for u. They are all initialized with 0. $\mathsf{TH}_{\mathcal{H}}$ furthermore maintains explicit counters and message bounds for each port in order to ensure polynomial runtime, cf. [9]; we omit the details.

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

Hash-Related Inputs. In this model, users build up terms in the ideal functionality step by step, and they refer to the terms by the handles defined in Section A.1. The new commands we have to add to this model are therefore hash for hashing a term and is_hash_of. Such cryptographic operations are called basic commands. They are accepted at each input port $\mathsf{in}_u?$ and have only local effects, i.e., only an output at $\mathsf{out}_u?$ occurs. For such commands we use the notation $j \leftarrow \mathsf{op}(i)$, and we always use u as the index of the concerned ports.

The following command definitions look rather complex because type checking is explicit and so is the test that the resulting term does not exceed the length bound needed for polynomial time, but in principle they just construct or test the top level of a

hash term as one would expect. Handle arguments are tacitly required to be in $\mathcal{HNDS}$ and existing, i.e., $\leq curhnd_u$, at the time of execution. By a general convention in [9], cryptographic operations—here hashing—are only applied to lists.

Definition 11 (Basic Commands for Hashes). *The trusted host* $\mathsf{TH}_{\mathcal{H}}$ *extended by hashes accepts the following additional commands at every port* $\mathsf{in}_u?$*:*

- *Hashing:* $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(l^{\mathsf{hnd}})$. *Let* $l := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$, $length := \mathsf{hash\_len}^*(k)$ *and return* $\downarrow$ *if* $l = \downarrow$ *or* $length > \mathsf{max\_len}(k)$. *Let* $h := D[type = \text{'}\mathsf{hash}\text{'} \wedge arg = (l)].ind$. *If* $h \neq \downarrow$ *set* $h^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(h)$, *else set* $h^{\mathsf{hnd}} := curhnd_u\text{++}$ *and* $D :\Leftarrow (ind := size\text{++}, type := \text{"}\mathsf{hash}\text{"}, arg := (l), hnd_u := h^{\mathsf{hnd}}, len := length)$.
- *preimage test:* $b \leftarrow \mathsf{is\_hash\_of}(l^{\mathsf{hnd}}, h^{\mathsf{hnd}})$. *Let* $l := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$, $h := D[hnd_u = h^{\mathsf{hnd}} \wedge type = \text{'}\mathsf{hash}\text{'}].ind$ *and return* $\downarrow$ *if* $l = \downarrow$ *or if* $h = \downarrow$. *If* $D[h].arg = (l)$ *return* $b := \mathsf{true}$ *else* $b := \mathsf{false}$. $\diamond$

As explained in Section 6.1, the ideal adversary is given one capability that the honest users do not have: It can construct hash terms with unknown content. This is done by a so-called local adversary command, only accepted at port $\mathsf{in}_\mathsf{a}?$. For consistency, we also have to extend the existing command adv_parse to hashes. This command allows the adversary to retrieve all information about subterms that is not explicitly required to be hidden, but for hashes it simply returns the empty list because of the ideal secrecy.

Definition 12 (Local Adversary Commands for Hashes). *The trusted host* $\mathsf{TH}_{\mathcal{H}}$ *extended by hashes accepts the following additional commands at the port* $\mathsf{in}_\mathsf{a}?$.

- *Generate unknown hash:* $h^{\mathsf{hnd}} \leftarrow \mathsf{unknown\_hash}()$. *Set* $h^{\mathsf{hnd}} := curhnd_u\text{++}$ *and* $D :\Leftarrow (ind := size\text{++}, type := \text{"}\mathsf{hash}\text{"}, arg := (), hnd_u := h^{\mathsf{hnd}}, len := \mathsf{hash\_len}^*(k))$.
- *Parameter retrieval:* $(type, arg) \leftarrow \mathsf{adv\_parse}(h^{\mathsf{hnd}})$. *This existing command always sets* $type := D[hnd_\mathsf{u} = m^{\mathsf{hnd}}].type$. *For hashes, we set* $arg = ()$. $\diamond$

A.2 Realization of the Dolev-Yao-style System with Hashes

The realization of the Dolev-Yao model follows the overall description in Section 4.4, i.e., each user has a machine M_u, and the machines are connected by insecure channels. Each machine has one port pair $\mathsf{in}_u?$ and $\mathsf{out}_u!$ and accept the same inputs from honest users there as the ideal system. Essentially, Machine M_u contains the projections of the Dolev-Yao-style database to the objects for which user u has handles, with terms replaced by bitstrings. Figure 9 shows the real situation for the example from Figure 8. We assume that the random oracle outputs strings of length $\mathsf{hash\_len}(k)$, polynomial in k. The corresponding ideal length function is $\mathsf{hash\_len}^*(k) := \mathsf{list\_len}(\mathsf{len}(\text{'}\mathsf{hash}\text{'}), \mathsf{hash\_len}(k))$.

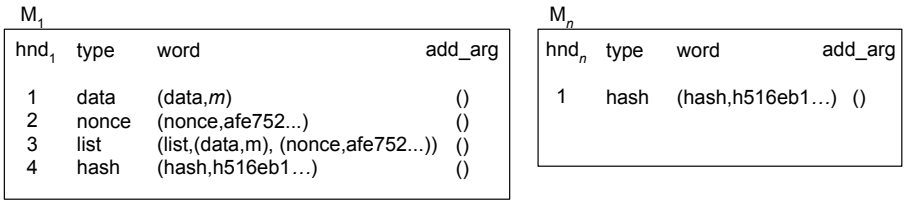


Fig. 9. Real situation for the same example as above.

States. Each entry x in the database D_u of machine M_u has the following attributes:

- $x.hnd_u \in \mathcal{HNDS}$ consecutively numbers the entries in D_u. We use it as a primary key attribute, i.e., we write $D_u[i^{\mathsf{hnd}}]$ for the selection $D_u[hnd_u = i^{\mathsf{hnd}}]$.
- $x.word \in \{0,1\}^+$ is the real representation of x.
- $x.type \in typeset \cup \{\mathsf{null}\}$ identifies the type of x, where the value null denotes an unparsed entry. Recall that we added the type 'hash' to $typeset$.
- $x.add\_arg$ is a list of ("additional") arguments. For entries of type 'hash' it is always $()$.

Initially, D_u is empty. M_u has a counter $curhnd_u \in \mathcal{HNDS}$ for the current size of D_u. The subroutine $(i^{\mathsf{hnd}}, D_u) :\leftarrow (i, type, add\_arg)$ determines a handle for certain given parameters in D_u: If an entry with the word i already exists, i.e., $i^{\mathsf{hnd}} := D_u[word = i \wedge type \notin secrettypes].hnd_u \neq \downarrow$,[8] it returns i^{hnd}, assigning the input values $type$ and add_arg to the corresponding attributes of $D_u[i^{\mathsf{hnd}}]$ only if $D_u[i^{\mathsf{hnd}}].type$ was null. Else if $\mathsf{len}(i) > \mathsf{max\_len}(k)$, it returns $i^{\mathsf{hnd}} = \downarrow$. Otherwise, it sets and returns $i^{\mathsf{hnd}} := curhnd_u\mathsf{++}$, $D_u :\Leftarrow (i^{\mathsf{hnd}}, i, type, add\_arg)$.

Similar to the machine $\mathsf{TH}_{\mathcal{H}}$, M_u maintains bounds on the length of messages and number of activations to achieve polynomial runtime. We omit further details.

Hash-Related Inputs. Now we describe how M_u evaluates individual new inputs. The stateful commands in [9] are defined via functional constructors and parsing algorithms for each cryptographic type. (These stateless algorithms can be reused in the simulator and the proof, while the stateful parts are different in the simulator.)

Definition 13 (Constructors and Destructors for Hashes).

- *Hash constructor:* $h^* \leftarrow \mathsf{make\_hash}(l)$*, for* $l \in \{0,1\}^+$*. Let* $h \leftarrow \mathsf{RO}(l)$ *and return* $h^* := (\text{'hash'}, h)$.
- *Hash parsing:* $arg \leftarrow \mathsf{parse\_hash}(h^*)$*. If* h^* *is of the form* $(\text{'hash'}, h)$ *with* $h \in \{0,1\}^{\mathsf{hash\_len}(k)}$*, return* $()$*, else* $\downarrow$. ◇

From the underlying model from [9], we use the general parsing algorithm and stateful parsing routines:

[8] The restriction $type \notin secrettypes$ is included for compatibility to the original library. Similar statements will occur some more times, but no further knowledge of such types is needed for understanding the new work.

– *General parsing:* $(type, arg) \leftarrow \mathsf{parse}(m)$. If m is not of the form $(type, m_1, \ldots, m_j)$ with $type \in typeset \setminus secrecttypes$ and $j \geq 0$, returns $(\mathsf{garbage}, ())$. Else call the type-specific parsing algorithm $arg' \leftarrow \mathsf{parse\_}type(m)$. If $arg = \downarrow$, then parse again outputs $(\mathsf{garbage}, ())$, else $(type, arg)$.
– "parse m^{hnd}" means that M_u calls $(type, arg) \leftarrow \mathsf{parse}(D_u[m^{\mathsf{hnd}}].word)$, assigns $D_u[m^{\mathsf{hnd}}].type := type$ if it was still null, and may then use arg.
– "parse m^{hnd} if necessary" means the same except that M_u does nothing if $D_u[m^{\mathsf{hnd}}].type \neq \mathsf{null}$.

We can define how a real machine reacts on the same basic commands as the ideal Dolev-Yao-style system. They are again local; in the real system this means that they produce no outputs onto the network.

Definition 14 (Basic Commands for Hashes).

– *Hashing:* $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(l^{\mathsf{hnd}})$. *Parse* l^{hnd} *if necessary. If* $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$, *return* $\downarrow$. *Otherwise set* $l := D_u[l^{\mathsf{hnd}}].word$ *and* $h^* \leftarrow \mathsf{make\_hash}(l)$. *If* $|h^*| > \mathsf{max\_len}(k)$, *return* $\downarrow$, *else* $(h^{\mathsf{hnd}}, D_u) :\leftarrow (h^*, \mathsf{hash}, ())$.
– *Preimage test:* $b \leftarrow \mathsf{is\_hash\_of}(l^{\mathsf{hnd}}, h^{\mathsf{hnd}})$. *If* $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$ *or* $D_u[h^{\mathsf{hnd}}].type \neq$'hash', *return* $\downarrow$. *Otherwise set* $l := D_u[l^{\mathsf{hnd}}].word$, $h := D_u[h^{\mathsf{hnd}}].word$, *and* $h^* \leftarrow \mathsf{make\_hash}(l)$. *If* $h = h^*$ *return* true, *else* false. ◇

A.3 The Security Theorem

Our security claim is that the Dolev-Yao-style system with hashes defined in Appendix A.1 is securely implemented by the realization defined in Appendix A.2 in the sense of BRSIM/UC in the random oracle model, as described in Section 2. We denote the resulting BRSIM/UC notion by $\geq^{\mathsf{ROM}}$.

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 satisfying respective security definitions against active attacks, cf. [9, 10, 6], and length functions and bounds L'. (Currently $\mathcal{S}$ may contain symmetric and asymmetric encryption schemes, signature schemes, MACs, and of course nonces.) For $(n, \mathcal{S}, L') \in RPar$, let $Sys^{\mathsf{cry\_Hash,real}}_{n,\mathcal{S},\mathsf{hash}^{\mathsf{r}},L'}$ be the resulting realization. Further, let the corresponding length functions and bounds of the ideal system be formalized by a function $L := \mathsf{R2Ipar}(S, L')$, and let $Sys^{\mathsf{cry\_Hash,id}}_{n,L}$ be the ideal Dolev-Yao-style system with parameters n and L.

Theorem 8. *(Soundness of the Dolev-Yao-style System with Hashes in the Random Oracle Model) For all parameters* $(n, \mathcal{S}, \mathsf{hash}, L') \in RPar$ *and* $L := \mathsf{R2Ipar}(\mathcal{S}, L')$, *we have*

$$Sys^{\mathsf{cry\_Hash,real}}_{n,\mathcal{S},\mathsf{hash}^{\mathsf{r}},L'} \geq^{\mathsf{ROM}} Sys^{\mathsf{cry\_Hash,id}}_{n,L}.$$ □

A.4 Simulator

For proving Theorem 8 for the underlying sytem without hashes (and without a random oracle), a simulator $\mathsf{Sim}_{\mathcal{H}}$ was defined in [9]. Basically $\mathsf{Sim}_{\mathcal{H}}$ has to translate real

messages from the real adversary A into handles (i.e., term representations) as $\mathsf{TH}_{\mathcal{H}}$ expects them at its adversary input port $\mathsf{in}_{\mathsf{a}}?$ and vice versa. In both directions, $\mathsf{Sim}_{\mathcal{H}}$ has to parse an incoming message completely because it can only construct the other version (abstract or real) bottom-up. This is done by recursive algorithms real2id and id2real, respectively. The state of $\mathsf{Sim}_{\mathcal{H}}$ mainly consists of a database D_{a}, similar to the databases D_u, but storing the knowledge of the adversary. Each entry contains a handle, a bitstring, a type, and possible an additional argument add_arg. We now define the extension of the simulator for hash functions, i.e., we consider the case that in the recursive functions real2id and id2real the simulator is confronted with a hash string or term, respectively. Furthermore, the simulator now has to answer random oracle queries from A and H.

Inputs from $\mathsf{TH}_{\mathcal{H}}$. Assume that in the recursion of id2real, the simulator $\mathsf{Sim}_{\mathcal{H}}$ obtained a handle h^{hnd} denoting a hash term. It first looks in its database whether h^{hnd} has already been assigned a hash string h; if yes, it outputs h. Otherwise, it chooses a new random value h' and adds the type tag. The main part of id2real automatically inserts an entry $(h^{\mathsf{hnd}}, h, \text{'hash'}, ())$ into D_{a}. The fourth element $add\_arg = ()$ means that no preimage of the hash is known yet. More formally, the simulator does the following:

- Let $x := D_{\mathsf{a}}[hnd_{\mathsf{a}} = h^{\mathsf{hnd}}]$.
- If $x \neq \downarrow$, let $h := x.word$. (We postulate here, and show in the proof, that $h \neq \downarrow$ whenever $x \neq \downarrow$.)
- Otherwise let $h' \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{hash\_len}(k)}$, $h := (\text{'hash'}, h')$ and $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h, \text{'hash'}, ())$.

Inputs from A. Now assume that in the recursion of real2id, the simulator $\mathsf{Sim}_{\mathcal{H}}$ has to translate a hash string h (recognized by its type tag) into a handle h^{hnd}. It first looks in its database whether h is already present. If yes, it outputs the handle stored for that bitstring. We later show that the only case where no such entry is present is that h was not correctly generated with a random oracle query; thus nobody "knows" a preimage of this hash string. Hence the simulator enters a command that generates an unknown hash term into the ideal system and makes a new entry in D_{a}. More formally:

- Let $x := D_{\mathsf{a}}[word = h]$.
- If $x \neq \downarrow$, let $h^{\mathsf{hnd}} := x.hnd_{\mathsf{a}}$. (We postulate here, and show in the proof, that $h^{\mathsf{hnd}} \neq \downarrow$ whenever $x \neq \downarrow$.)
- Else enter $h^{\mathsf{hnd}} \leftarrow \mathsf{unknown\_hash}()$ and $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h, \text{'hash'}, ())$.

Random Oracle Queries. If the simulator $\mathsf{Sim}_{\mathcal{H}}$ gets a random oracle query for a bitstring m, it first constructs the term corresponding to m using the routine real2id; this yields a handle m^{hnd}. It then checks whether m is the designated preimage of an already existing hash according to the attribute add_arg of the hash handles in its database. If yes, it outputs the existing hash string from that hash entry (without the type tag). Otherwise, $\mathsf{Sim}_{\mathcal{H}}$ checks whether m^{hnd} is the so far undesignated preimage of an existing hash by applying the command is_hash_of to the given message handle m^{hnd} and the handles of the hash terms it knows. If the result is true for a hash term z, $\mathsf{Sim}_{\mathcal{H}}$

stores that m^{hnd} is the preimage of z^{hnd} and outputs $z.word$. If no such term z exists, it chooses a new random value h', adds the type tag, outputs the result, and makes a new entry in D_{a}. More formally:

- Call $m^{\mathsf{hnd}} \leftarrow \mathsf{real2id}(m)$.
- Let $y := D_{\mathsf{a}}[type = \text{'hash'} \wedge add\_arg = (\mathsf{preimage}, m^{\mathsf{hnd}})]$. If $y \neq \downarrow$, output $h := y.word[2]$. (We postulate and later prove that at most one such y exists, and that $y.word \neq \downarrow$ if $y \neq \downarrow$.)
- Otherwise let $Z := \{z \in D_{\mathsf{a}} \mid z.type = \text{'hash'}\}$. For each $z \in Z$, call $b_z \leftarrow \mathsf{is\_hash\_of}(m^{\mathsf{hnd}}, z.hnd_{\mathsf{a}})$.
  If $b_z = \mathsf{true}$ for some z (we postulate that at most one such element exists), set $z.add\_arg := (\mathsf{preimage}, m^{\mathsf{hnd}})$ and output $z.word$.
  Otherwise, let $h' \xleftarrow{R} \{0,1\}^{\mathsf{hash\_len}(k)}$, $h := (\text{'hash'}, h')$ call $\mathsf{TH}_{\mathcal{H}}$ with $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(m^{\mathsf{hnd}})$, set $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h, \text{'hash'}, (\mathsf{preimage}, m^{\mathsf{hnd}}))$, and output h'.

A.5 Proof of Correct Simulation

We now prove that the simulator $\mathsf{Sim}_{\mathcal{H}}$ yields a correct simulation. We start by defining invariants of the database D_{a} of $\mathsf{Sim}_{\mathcal{H}}$ and proving that they hold with overwhelming probability. In particular, these invariants prove the postulations we made in the construction of the simulator.

Lemma 3 (Invariants of D_{a} in $\mathsf{Sim}_{\mathcal{H}}$ for Hashes). *The following statements are invariants of $\mathsf{Sim}_{\mathcal{H}}$, i.e., they hold for all traces when $\mathsf{Sim}_{\mathcal{H}}$ is run with $\mathsf{TH}_{\mathcal{H}}$ and arbitrary (even computationally unbounded) user and adversary machines, up to a negligible error probability in k.*

a. For every $x \in D_{\mathsf{a}}$ with $x.type = \text{'hash'}$, we have $x.hnd_{\mathsf{a}} \neq \downarrow$ and $x.word \neq \downarrow$.
b. For every $x \in D_{\mathsf{a}}$ with $x.type = \text{'hash'}$, we have $x.add\_arg \in \{()\} \cup \{(\mathsf{preimage}, m^{\mathsf{hnd}}) \mid \exists y \in D_{\mathsf{a}} \setminus \{x\} \colon y.hnd_{\mathsf{a}} = m^{\mathsf{hnd}}\}$.
c. For every $x \in D_{\mathsf{a}}$ with $x.type = \text{'hash'}$, if $x.add\_arg = (\mathsf{preimage}, m^{\mathsf{hnd}})$ then $D_{\mathsf{a}}[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].word \neq \downarrow$.
d. If h was the output of an oracle query m, then there exist $x, y \in D_{\mathsf{a}}$ with $x.word = m$, $y.word[2] = h$, $y.type = \text{'hash'}$, and $y.add\_arg = (\mathsf{preimage}, x.hnd_{\mathsf{a}})$. □

Proof. Parts a. to c. can be easily proved by inspection of the commands of $\mathsf{Sim}_{\mathcal{H}}$. For Part d., let h denote the output of the oracle query on message m. Within this oracle query, $\mathsf{Sim}_{\mathcal{H}}$ calls $m^{\mathsf{hnd}} \leftarrow \mathsf{real2id}(m)$. This ensures that the claimed entry x exists and that $x.hnd_{\mathsf{a}} = m^{\mathsf{hnd}}$. Then it sets $y := D_{\mathsf{a}}[type = \text{'hash'} \wedge add\_arg = (\mathsf{preimage}, m^{\mathsf{hnd}})]$. If $y \neq \downarrow$, we have $h = y.word[2]$ by construction. If $y = \downarrow$, an assignment $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h'', \text{'hash'}, (\mathsf{preimage}, m^{\mathsf{hnd}}))$ with $h = h''[2]$ immediately preceeds the output of h. ■

The correctness of the simulator in [9] was proven using a *cryptographic bisimulation*, which is a probabilistic bisimulation with reduction proofs and static information-flow analysis. In our case, the invariants of D_{a} and the use of a random oracle significantly simplify the extension of this bisimulation to hashes.

In the following, we denote the look-up table of the random oracle RO in the real system by D_{RO}, ranging over attributes $query$ and $hash$.

Definition of the Combined System. The cryptographic bisimulation makes use of a combined system which essentially contains the joint information of the databases of the ideal and real system. We briefly define this combined system. Possible ambiguities will disappear below, where we compare the effects of the hash-related inputs in the three systems.

The main part of $\mathsf{C}_{\mathcal{H}}$ is a database D^* structured like D in $\mathsf{TH}_{\mathcal{H}}$. An entry x may have the following additional attributes:

- $x.word \in \{0,1\}^*$ is always defined and contains real data as in $\mathsf{M}_{\mathcal{H}}$ or $\mathsf{Sim}_{\mathcal{H}}$ under the same handle(s). For $x.type \in \{\mathsf{sks}, \mathsf{ske}\}$, it is ϵ for adversary keys, i.e., if $\mathsf{owner}(x) = \mathsf{a}$ else a real secret key. For all other types, the word is non-empty.
- $x.parsed_u \in \{\mathsf{true}, \mathsf{false}\}$ for $u \in \mathcal{H}$ is $\downarrow$ if $x.hnd_u = \downarrow$; otherwise true indicates that the entry would be parsed in D_u, and false that it would still be of type null.
- $x.query$ for hashes is (l) if $x.word$ was the output of a previous random oracle query l; otherwise it is $()$. Note that by definition of the random oracle, $x.query = y.query = (l)$ for some entries x, y and string l implies $x = y$ up to a negligible error probability.
- $x.queried\_by\_env$ for hashes is true if $x.word$ was the output of at least one previous random oracle query by the user or by the adversary (the "environment"); otherwise it is false.
- $x.owner$ for ciphertexts with honest-user keys is adv if the ciphertext was received from the adversary, otherwise honest. For other ciphertexts it is $\downarrow$.
- $x.ec$ for secret encryption keys corresponds to the encryption counter in $ciphers$ of the encryption machines.

When evaluating inputs, $\mathsf{C}_{\mathcal{H}}$ acts on the D-part of its database D^*, the variables $size$ and $curhnd_u$, and the ideal secure channels treated exactly like $\mathsf{TH}_{\mathcal{H}}$. An entry whose first handle $x.hnd_u$ is for $u \in \mathcal{H}$ gets the word that M_u would contain under this handle; an entry whose first handle is for the adversary gets the word from $\mathsf{Sim}_{\mathcal{H}}$. Thus, essentially, entries created due to basic commands from H get the words that M_u would construct (possibly by querying the random oracle), while words received in network inputs from A as well as random oracle queries by H and A are entered as by $\mathsf{Sim}_{\mathcal{H}}$.

Derivations. We now define the derivations of the original systems from the combined system. They are the mappings that we will show to be bisimulations. We now assume that a state of $\mathsf{C}_{\mathcal{H}}$ is given and define derived states corresponding to the original systems.

$\mathsf{TH}_{\mathcal{H}}$**:** D**:** This is the restriction of D^* to all attributes except $word$ and $parsed_u$.

$\mathsf{M}_{\mathcal{H}}{}^*$**:** D_u^***:** (For every $u \in \mathcal{H}$.) We derive D_u^* as follows, starting with an empty database: For every $x^{\mathsf{hnd}} \leq curhnd_u$, let $x := D^*[hnd_u = x^{\mathsf{hnd}}].ind$, $type := D^*[x].type$, and $m := D^*[x].word$. Then
  - If $D^*[x].parsed_u = \mathsf{false}$, then $D_u^* :\Leftarrow (x^{\mathsf{hnd}}, m, \mathsf{null}, ())$.
  - Else if $type =$'hash', then $D_u^* :\Leftarrow (x^{\mathsf{hnd}}, m, type, ())$.

$\mathsf{Sim}_{\mathcal{H}}{}^*$**:** D_{a}^***:** We derive D_{a}^* as follows, starting with an empty database: For all $x^{\mathsf{hnd}} \leq curhnd_{\mathsf{a}}$, let $x := D^*[hnd_{\mathsf{a}} = x^{\mathsf{hnd}}].ind$, $type := D^*[x].type$, and $m := D^*[x].word$.

– If $type$ ='hash' and $D^*[x].queried\_by\_env$ = false, then $D^*_{\mathsf{a}} :\Leftarrow (x^{\mathsf{hnd}}, m, \text{'hash'}, ())$.
– If $type$ ='hash', $D^*[x].queried\_by\_env$ = true, and $D^*[x].query = (l)$, then let $l^{\mathsf{hnd}}_{\mathsf{a}} := D^*[word = l].hnd_{\mathsf{a}}$ and $D^*_{\mathsf{a}} :\Leftarrow (x^{\mathsf{hnd}}, m, \text{'hash'}, (\mathsf{preimage}, l^{\mathsf{hnd}}_{\mathsf{a}}))$.

RO: D_{RO}: We derive D_{RO} as follows, starting with an empty database: For every $x \leq size$ with $m := D^*[x].word$:
– If $type$ ='hash', $D^*[x].query = (l)$, and $m = (\text{'hash'}, m')$ for some $m' \in \{0,1\}^{\mathsf{hash\_len}(k)}$, let $D_{\mathsf{RO}} :\Leftarrow (l, m')$

Invariants in the Combined System $\mathsf{C}_{\mathcal{H}}$ and RO. For the bisimulation, we need invariants about the combined system $\mathsf{C}_{\mathcal{H}}$ and RO.

– *Correct arguments.* For all $i \leq size$, the real message $m := D^*[i].word$ and the abstract type and arguments, $type^{\mathsf{id}} := D^*[i].type$ and $arg^{\mathsf{ind}} := D^*[i].arg$, are compatible. More precisely, let $arg^{\mathsf{real}} := \omega^*(arg^{\mathsf{ind}})$. Then we require:
  • If $type^{\mathsf{id}} \notin \{\mathsf{sks}, \mathsf{ske}\}$, let $(type, arg^{\mathsf{parse}}) := \mathsf{parse}(m)$. Then $type = type^{\mathsf{id}}$, and:
    ∗ If $type$ ='hash' then $D^*[i].query = arg^{\mathsf{real}}$.
– *Strongly correct arguments if* $\mathsf{a} \notin \mathsf{owners}(D^*[i])$ *or* $D^*[i].owner = \mathsf{honest}$. Let $type := D^*[i].type$, $arg^{\mathsf{ind}} := D^*[i].arg$ and $arg^{\mathsf{real}} := \omega^*(arg^{\mathsf{ind}})$. Then $type \neq \mathsf{garbage}$ and $m := D^*[i].word$ has the following probability distribution:[9]
  • If $type$ ='hash', then $m \leftarrow \mathsf{make\_hash}(arg^{\mathsf{real}})$.

In the rest of the proof, we show that all input types from the users H and the adversary A retain the invariants and lead to the same outputs in the different systems.

Input $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(l^{\mathsf{hnd}})$ by $u \in U$. Let $l^{\mathsf{ind}} := D^*[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. $\mathsf{TH}_{\mathcal{H}}$ returns $\downarrow$ if $l^{\mathsf{ind}} = \downarrow$ while M_u outputs $\downarrow$ if $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$. This is equivalent by "correct derivation". $\mathsf{TH}_{\mathcal{H}}$ furthermore outputs $\downarrow$ if $\mathsf{hash\_len}^*(k) \geq \mathsf{max\_len}(k)$. Else it sets $h^{\mathsf{ind}} := D^*[type = \text{'hash'} \wedge arg = (l^{\mathsf{ind}})].ind$. If $h^{\mathsf{ind}} \neq \downarrow$, it sets $h^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(h^{\mathsf{ind}})$; otherwise it sets $h^{\mathsf{hnd}} := curhnd_u{+}{+}$ and $D^* :\Leftarrow (ind := size{+}{+}, type := \text{'hash'}, arg := (l^{\mathsf{ind}}), hnd_u := h^{\mathsf{hnd}}, len := \mathsf{hash\_len}^*(k))$.

M_u sets $l := D_u[l^{\mathsf{hnd}}].word$ and $h^* \leftarrow \mathsf{make\_hash}(l)$, i.e., $h^* := (\text{'hash'}, h')$ for $h' := \mathsf{RO}(l) \in \{0,1\}^{\mathsf{hash\_len}(k)}$. If $|h^*| \geq \mathsf{max\_len}(k)$, it outputs $\downarrow$. This length test equals that in $\mathsf{TH}_{\mathcal{H}}$ since by definition of $\mathsf{hash\_len}^*$ we have $|h^*| = \mathsf{list\_len}(|\text{'hash'}|, \mathsf{hash\_len}(k)) = \mathsf{hash\_len}^*(k)$. If $h^{\mathsf{ind}} \neq \downarrow$ both $\mathsf{TH}_{\mathcal{H}}$ and M_u together with the random oracle produce a consistent output by "correct derivation" without changing their state. Hence either all three of them do not change their state, or all make the prescribed state updates. Otherwise M_u sets $h^{\mathsf{hnd}} := curhnd{+}{+}$ and makes an entry $D_u :\Leftarrow (h^{\mathsf{hnd}}, h^*, \text{'hash'}, ())$.

Now we consider the new hash entry: "Correct derivation" is clear. If "word uniqueness" is not fulfilled, i.e., if h' within h^* equals an old value in the same place in

[9] Here one sees that the bisimulation is probabilistic, i.e., we actually consider distributions of states before and after a transition. This invariant says that in such a state distribution, and given the mentioned arguments, m is distributed as described independent of other state parts.

a word, we put the run into a so-called error set $Nonce\_Coll$, which contained runs where collisions of randomly chosen strings (such as h') occured. One later easily shows that the joint probabilities of runs in this set is negligible, cf. [9]. "Correct arguments" follows immediately since $D^*[h^{\mathsf{ind}}].query = (l) = (D^*[l^{\mathsf{ind}}].word)$ with $D^*[h^{\mathsf{ind}}].arg = (l^{\mathsf{ind}})$. "Strongly correct arguments" holds by construction.

Input is_hash_of($l^{\mathsf{hnd}}, h^{\mathsf{hnd}}$) by $u \in U$. Let $l^{\mathsf{ind}} := D^*[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$ and $h^{\mathsf{ind}} := D^*[hnd_u = h^{\mathsf{hnd}} \wedge type = \text{'}\mathsf{hash}\text{'}].ind$. $\mathsf{TH}_{\mathcal{H}}$ returns $\downarrow$ if $l^{\mathsf{ind}} = \downarrow$ or $h^{\mathsf{ind}} = \downarrow$ while M_u outputs $\downarrow$ if $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$ or $D_u[h^{\mathsf{hnd}}].type \neq \mathsf{hash}$. This is equivalent by "correct derivation".

$\mathsf{TH}_{\mathcal{H}}$ outputs true iff $D^*[h^{\mathsf{ind}}].arg = (l^{\mathsf{ind}})$, and false otherwise. M_u sets $l := D_u[l^{\mathsf{hnd}}].word$, $h := D_u[h^{\mathsf{hnd}}].word$, and $h^* := (\text{'}\mathsf{hash}\text{'}, h') \leftarrow \mathsf{make\_hash}(l)$. It outputs true if $h^* = h$, and false otherwise.

First assume that $D^*[h^{\mathsf{ind}}].query = ()$. "Correct arguments for $D^*[h^{\mathsf{ind}}]$ implies that $\omega^*(D^*[h^{\mathsf{ind}}].arg) = ()$, and hence $D^*[h^{\mathsf{ind}}].arg = ()$. Thus TH outputs false in this case. $D^*[h^{\mathsf{ind}}].query = ()$ furthermore implies that there was no previous oracle query with resulting output h, i.e., we in particular have $(l, h) \notin D_{\mathsf{RO}}$. If there exists $(l, h'') \in D_{\mathsf{RO}}$ for some $h'' \neq h'$, the random oracle outputs h'' and this we have $h^* \neq h$, yielding an output false by M_u. If $(l, h'') \notin D_{\mathsf{RO}}$ for all h'', then h' is chosen as $h' \xleftarrow{R} \{0,1\}^{\mathsf{hash\_len}(k)}$ by the random oracle. Thus $h^* = h$ only holds with exponentially small probability, hence M_u will also output false with overwhelming probability. (More formally, runs in which $h^* = h$ holds would again be put in the error set $Nonce\_Coll$ which then later can be shown to have negligible probability.)

Now assume that $D^*[h^{\mathsf{ind}}].query = (i)$ for some i, and let $i^{\mathsf{ind}} := D^*[word = i].ind$. Then "correct derivation" for $D^*[h^{\mathsf{ind}}]$ implies $(i, j) \in D_{\mathsf{RO}}$ where j is defined by $(\text{'}\mathsf{hash}\text{'}, j) := h$. "Correct arguments" furthermore implies $\omega^*(D^*[h^{\mathsf{ind}}].arg) = (i)$. Thus "word uniqueness" implies $D^*[h^{\mathsf{ind}}].arg = (i^{\mathsf{ind}})$. Hence $\mathsf{TH}_{\mathcal{H}}$ outputs true iff $i^{\mathsf{ind}} = l^{\mathsf{ind}}$. If $i^{\mathsf{ind}} = l^{\mathsf{ind}}$, then $(l, h') \in D_{\mathsf{RO}}$, thus $h^* = h$ and M_u also outputs true. If $i^{\mathsf{ind}} \neq l^{\mathsf{ind}}$, then "correct derivation" implies that $(l, h') \notin D_{\mathsf{RO}}$ up to an exponentially small error probability that a randomly chosen h' satisfies $(\text{'}\mathsf{hash}\text{'}, h') = i$, cf. the previous case. Thus M_u also outputs false with overwhelming probability. The invariants are clearly retained.

Inputs from TH. We now consider the simulation of real hash strings based on received hash terms, i.e., the simulator receives a handle h^{hnd} to a hash term h^{ind} from $u \in \mathcal{H}$ in the routine id2real and has to constructs a suitable hash string for it. Intuitively, this part shows that the adversary does not get any information in the real system that it cannot get in the ideal system, because any real information can be simulated indistinguishably given only the outputs of $\mathsf{TH}_{\mathcal{H}}$.

Let $h^{\mathsf{ind}} := D^*[hnd_u = h^{\mathsf{hnd}}].ind$. Now M_u always outputs $h := D^*[h^{\mathsf{ind}}].word$. An inductive proof is used that the overall translating routine id2real from ideal terms to concrete strings retains all invariants and produces the right outputs. id2real starts with three steps that are essentially independent of the type of the considered entry (up to domain checks which are fulfilled by construction when interacting with $\mathsf{TH}_{\mathcal{H}}$). The fourth step proceeds depending on the $type$ of the term. Each of these variants ends with an assignment to h, which is then output, and $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h, type, add\_arg)$ for certain arguments add_arg.

In [9], it has been proven (in Lemma 7.6) that it is sufficient to show:

- a correct result $h = h^*$, where h^* is the word the M_u produces, i.e., $h* := D^*[h^{\mathsf{ind}}].word$. We further can assume "strongly correct arguments" for h^*.
- "correct derivation" of add_arg in the new entry;
- "word secrecy" for h, i.e., no flow of secret information into h, where arguments h_i are not secret information. This invariant is used to later show that the adversary cannot guess any information about, e.g., nonces that it has not learned "ideally" yet, i.e., that it does not have a handle for.

For our new hash type, these conditions are also sufficient. This can be proven analogously to the original proof. Since the proof mainly relies on a thorough investigation of the first three steps of id2real, we omit the details here.

For type 'hash', the subroutine of id2real sets $h' \xleftarrow{R} \{0,1\}^{\mathsf{hash\_len}(k)}$, $h := (\text{'hash'}, h')$, and $add\_arg = ()$. "Strongly correct arguments" for h^* means that h^* has the probability distribution $m^* \leftarrow \mathsf{make\_}type(arg^{\mathsf{real}})$, where arg^{real} is defined for $D^*[h^{\mathsf{ind}}]$ as in "strongly correct arguments". By definition of the random oracle, h^* is distributed as $h'' \xleftarrow{R} \{0,1\}^{\mathsf{hash\_len}(k)}$ and $h^* := (\text{'hash'}, h'')$. Thus derivation of D^*_{a} gives the same distribution as we get in D_{a}. "Word secrecy" clearly holds for h hashes since each generation of a new hash does not depend on prior information.

Inputs from A. We now consider the simulation of ideal hash terms based on received hash strings in the routine real2id, i.e., the simulator receives a string h and has to constructs a term h^{hnd}. An inductive proof is again used that the overall translating routine real2id from concrete strings to ideal terms retains all invariants and produces the right outputs. Using a lemma from [9], we only have to show the following properties of each call $h^{\mathsf{hnd}}_{\mathsf{a}} \leftarrow \mathsf{real2id}(h)$ with $0 < |h| \leq \mathsf{max\_len}(k)$:

- At the end, $D^*[hnd_{\mathsf{a}} = h^{\mathsf{hnd}}_{\mathsf{a}}].word = h$ and $D^*[hnd_{\mathsf{a}} = h^{\mathsf{hnd}}_{\mathsf{a}}].type \notin secrettypes$.
- "Correct derivation" of D_{a} and $curhnd_{\mathsf{a}}$.
- The invariants within D^* are retained, where "strongly correct arguments" is already clear and "word secrecy" need only be shown for the outermost call (without subcalls) if more entries than $D^*[hnd_{\mathsf{a}} = h^{\mathsf{hnd}}_{\mathsf{a}}]$ are made or updated there.

The lemma carries over to our new type 'hash' with marginal extensions of the proof.

If there is already a handle h^{hnd} with $D_{\mathsf{a}}[h^{\mathsf{hnd}}].word = h$, real2id returns that. The postulated output condition is fulfilled by "correct derivation", and the others because no state changes are made. Otherwise, the word h is not yet present in D_{a}. Then id2real sets $(type, arg) := \mathsf{parse}(h)$. This yields $type \in typeset \setminus secrettypes$. As parse is a functional algorithm, no invariants are affected. Then id2real calls an type-specific subroutine $add\_arg \leftarrow \mathsf{real2id\_}type(h, arg)$ with side-effects. Finally it sets $h^{\mathsf{hnd}} := curhnd_{\mathsf{a}}{+}{+}$ and $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h, add\_arg)$.

We therefore have to show the postulated properties for our new type-specific algorithms together with those last two assignments.

For type 'hash', the subroutine $\mathsf{real2id\_hash}(h, ())$ calls $h^{\mathsf{hnd}} \leftarrow \mathsf{unknown\_hash}()$ at in_{a}! and sets $add\_arg := ()$. Hence $\mathsf{TH}_{\mathcal{H}}$ makes a new entry with $h^{\mathsf{hnd}} := curhnd_{\mathsf{a}}{+}{+}$. Together with the new entry in D_{a}, this results in $D^* :\Leftarrow (ind :=$

$size$++, $type$:=‘hash’, arg := (), hnd_{a} := h^{hnd}, len := $\mathsf{hash\_len}^*(k)$, $word$:= h, $query$ = (), $queried\_by\_env$ = false). It fulfills the postulated output condition. “Correct derivation” is clear, and “word secrecy” is clear as no other entries are involved. For “word uniqueness”, assume there is a prior entry $x \in D^*$ with $x.word = h$. We then put this run in an error set $Nonce\_Guess$, which contains those runs in which the adversary guessed a random string that he had no information about. One can later show that this error set has negligible probability by exploiting the invariant “word secrecy” to show that indeed no information in the Shannon sense has flowed from variables that the adversary knows into the considered random strings, cf. [9].

Answering Oracle Queries. Finally assume that a user u or A asks the random oracle to hash a message l.

If there exists $(l, h) \in D_{\mathsf{RO}}$ in the real system for some h, then the random oracle outputs h; otherwise, it chooses $h \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{hash\_len}(k)}$, sets $D_{\mathsf{RO}} :\Leftarrow (l, h)$ and outputs h. Let $l^{\mathsf{ind}} := D^*[word = l].ind$, $h^{\mathsf{ind}} := D^*[word = (\text{‘hash’}, h)].ind$, and $h_{\mathsf{a}}^{\mathsf{hnd}} := D^*[h^{\mathsf{ind}}].hnd_{\mathsf{a}}$. The simulator calls $l_{\mathsf{a}}^{\mathsf{hnd}} := algoreal2id(l)$.

Assume first that there exists $(l, h) \in D_{\mathsf{RO}}$. Then $D^*[h^{\mathsf{ind}}].query = (l)$. If $D^*[h^{\mathsf{ind}}].queried\_by\_env = \mathsf{true}$ “correct derivation” implies $D_{\mathsf{a}}^*[h^{\mathsf{ind}}].arg = (\mathsf{preimage}, l_{\mathsf{a}}^{\mathsf{hnd}})$. There is no state change in this case and $\mathsf{Sim}_{\mathcal{H}}$ outputs $D_{\mathsf{a}}^*[h^{\mathsf{ind}}].word[2]$ while the random oracle outputs h. This is equivalent by “correct derivation”. If $D^*[h^{\mathsf{ind}}].queried\_by\_env = \mathsf{false}$, we have $D_{\mathsf{a}}[type =\text{‘hash’} \wedge add\_arg = (\mathsf{preimage}, l_{\mathsf{a}}^{\mathsf{hnd}})].ind = \downarrow$ by “correct derivation”. Let $Z := \{z \in D_{\mathsf{a}} \mid z.type =\text{‘hash’}\}$ and $b_z \leftarrow \mathsf{is\_hash\_of}(l_{\mathsf{a}}^{\mathsf{hnd}}, z.hnd_{\mathsf{a}})$ for $z \in Z$. If there is one z with $b_z = \mathsf{true}$, the simulator outputs $z.word$. “Correct arguments” implies $D^*[h^{\mathsf{ind}}].arg = (l^{\mathsf{ind}})$, thus there is one z with $b_z = \mathsf{true}$ iff $h_{\mathsf{a}}^{\mathsf{hnd}} := D^*[h^{\mathsf{ind}}].hnd_{\mathsf{a}} \neq \downarrow$. If $h_{\mathsf{a}}^{\mathsf{hnd}} \neq \downarrow$, the simulator outputs $z.word$ which is correct by “correct derivation”. If $h_{\mathsf{a}}^{\mathsf{hnd}} = \downarrow$ $\mathsf{Sim}_{\mathcal{H}}$ sets $h' \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{hash\_len}(k)}$ and $h^* := (\text{‘hash’}, h')$, calls $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(l_{\mathsf{a}}^{\mathsf{hnd}})$, $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h^*, \text{‘hash’}, (\mathsf{preimage}, l_{\mathsf{a}}^{\mathsf{hnd}}))$, and outputs h'. Because of $D^*[h^{\mathsf{ind}}].arg = (l^{\mathsf{ind}})$, we have $h^{\mathsf{hnd}} = h_{\mathsf{a}}^{\mathsf{hnd}}$, i.e., there is no state change in $\mathsf{TH}_{\mathcal{H}}$ except for setting the handle hnd_{a}. The string h that the random oracle computes has the same probability distribution as h' by “strongly correct arguments” for the entry $D^*[h^{\mathsf{ind}}]$, yielding a correctly distributed output. We finally consider the new hash entry: “Correct derivation” is clear. If “word uniqueness” is not fulfilled, i.e., if h or h^* equal an existing word, we put the run into the error set $Nonce\_Coll$, which can later shown to occur with negligible probability only. “Correct arguments” follows immediately since $D^*[h^{\mathsf{ind}}].query = (l) = (D^*[l^{\mathsf{ind}}].word)$. “Strongly correct arguments” holds by construction.

Now assume that $(l, h) \notin D_{\mathsf{RO}}$ for all h. $\mathsf{Sim}_{\mathcal{H}}$ calls $l^{\mathsf{hnd}} \leftarrow \mathsf{real2id}(l)$. Let $Z := \{z \in D_{\mathsf{a}} \mid z.type =\text{‘hash’}\}$ and $b_z \leftarrow \mathsf{is\_hash\_of}(l^{\mathsf{hnd}}, z.hnd_{\mathsf{a}})$ for $z \in Z$. Now $(l, h) \notin D_{\mathsf{RO}}$ for all h implies $D^*[h^{\mathsf{ind}}].query \neq (l)$ for all h^{ind}, thus $D^*[h^{\mathsf{ind}}].arg \neq (l^{\mathsf{ind}})$ for all h^{ind} by “correct arguments” and “word uniqueness’. This implies $b_z = \mathsf{false}$ for all $z \in Z$. Hence $\mathsf{Sim}_{\mathcal{H}}$ sets $h' \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{hash\_len}(k)}$ and $h^* := (\text{‘hash’}, h')$, calls $h^{\mathsf{hnd}} \leftarrow \mathsf{hash}(l_a^{\mathsf{hnd}})$ and $D_{\mathsf{a}} :\Leftarrow (h^{\mathsf{hnd}}, h^*, \text{‘hash’}, (\mathsf{preimage}, l_{\mathsf{a}}^{\mathsf{hnd}}))$, and outputs. The string h that the random oracle computes has the same probability distribution as h' by

definition of the random oracle, yielding a correctly distributed output. The invariants for the new hash entry can be shown as in the previous case.

B Proof Sketches of Soundness in the Standard Model

We only briefly sketch the proofs of Theorems 6 and 7. Hence we mainly retain the notation from the main part of this paper, instead of switching to the notation of [9] as in Appendix A.

Both proofs require extending the simulator Sim from [9] to cope with hash terms and hash strings, similar to the more detailed proof in the random oracle model. In particular, recall that the simulator maintains a table (database) of corresponding term representations (handles) and bitstrings, as well as the preimages of the hash term representations where it knows them.

Proof (Theorem 6, Sketch). In the Dolev-Yao model without secrecy of hashed terms, the ideal adversary can derive (a representation of) the contained term t whenever it learns the hash $\mathsf{hash}(t)$. Hence the simulator can translate every ideal term $\mathsf{hash}(t)$ to a bitstring $\mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}})$ in a straightforward (recursive) manner by first constructing t^{r} and then applying the hash function.

Translating a real hash string h^{r} into an ideal hash term h is simulated as follows. First, the simulator constructs the set T_l of all terms of size at most l using the available operators over the terms in its current table of term representations. In the underlying model of [9], terms are immediately normalized, hence each term in T_l can be constructed with at most l operator applications. Since T_l is of polynomial size, this takes polynomial time. The simulator then checks for every term $t \in T_l$ whether hashing the corresponding string t^{r} obtained or computed from the table equals the given string h^{r}. Let $I \subseteq T_l$ denote the set of terms that pass this test. We distinguish three cases:

1. If $I = \{t\}$ for some term t, the simulator sets $h := \mathsf{hash}(t)$ and enters this into the ideal system as well as its own table.
2. If $|I| = \emptyset$, the simulator enters an unknown hash h, i.e., a hash with unknown preimage, into the ideal system and its table.
3. If $|I| \geq 2$, the simulator gives up the simulation. (A collision of the function $\mathsf{hash}^{\mathsf{r}}$ has been found, because in this Dolev-Yao model and realization unequal terms have unequal realizations with overwhelming probability.)

Whenever the simulator receives a new string m^{r} of arbitrary type and produces a corresponding term m, it additionally checks for each unknown hash h whether the contained term has now become known. For this, it constructs all terms t of size at most l containing m using the available operators over the terms in its current table of term representations, and checks whether $\mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}}) = h^{\mathsf{r}}$. Let T_h be the set of terms that pass this test. If $T_h = \{t\}$ for a term t, the simulator enters t as a preimage of h into its table, as well as into the ideal system using the special input for providing initially unknown preimages later. If $T_h = \emptyset$, the simulator does nothing, and if $|T_h| \geq 2$, it gives up the simulation. (A collision of the function $\mathsf{hash}^{\mathsf{r}}$ has been found.)

Clearly, the simulator Sim only gives up with negligible probability, as otherwise the combination of Sim, TH, A and H is an algorithm that contradicts the collision resistance of $\mathsf{hash}^{\mathsf{r}}$. For the other cases one can see, ultimately again by a cryptographic bisimulation, that the ideal system with the simulator is indistinguishable from the real system. ■

Proof (Theorem 7, Sketch). The simulator for this case can translate real hash strings into ideal terms as a simpler case of the procedure in the proof of Theorem 6, since individual nonces are a special case of terms of at most constant size.

As we allow hashing of individual nonces only, we can apply a similar technique for translating an ideal hash term h to a real hash string h^{r} although ideal secrecy of hashed terms is required: The simulator checks for all already known nonce terms n whether $\mathsf{is\_hash\_of}(n^{\mathsf{a}}, h^{\mathsf{a}}) = \mathsf{true}$. Since the number of known nonces is polynomial, this takes polynomial time. If such a nonce term n exists, the simulator sets $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(n^{\mathsf{r}})$ for the corresponding real nonce n^{r} in its table. At most one such term exists since preimages are unique on the term level. If no such n exists, the simulator chooses a new nonce $n^{*\mathsf{r}}$ with the algorithm $\mathsf{G_n}$ and sets $h^{\mathsf{r}} := \mathsf{hash}^{\mathsf{r}}(n^{*\mathsf{r}})$. It stores $n^{*\mathsf{r}}$ and its relation to the term h. Furthermore, whenever the simulator receives a new nonce term n, it first checks whether $\mathsf{is\_hash\_of}(n^{\mathsf{a}}, h^{\mathsf{a}}) = \mathsf{true}$ for some hash term h that it knows, but for which it does not yet know a preimage on the term level. (Again there is at most one.) If such a term h exists, then Sim outputs the stored real nonce $n^{*\mathsf{r}}$ and stores in its table that $n^{*\mathsf{r}}$ is now the realization of n. This simulation of the construction of nonces and their hashes is always correct. ■