# Computational Soundness of Symbolic Zero-Knowledge Proofs Against Active Attackers*

Michael Backes[1,2] and Dominique Unruh[1]
[1] Saarland University, Germany
[2] MPI-SWS

April 17, 2008

## Abstract

*The abstraction of cryptographic operations by term algebras, called Dolev-Yao models, is essential in almost all tool-supported methods for proving security protocols. Recently significant progress was made in proving that Dolev-Yao models offering the core cryptographic operations such as encryption and digital signatures can be sound with respect to actual cryptographic realizations and security definitions. Recent work, however, has started to extend Dolev-Yao models to more sophisticated operations with unique security features. Zero-knowledge proofs arguably constitute the most amazing such extension.*

*In this paper, we first identify which additional properties a cryptographic zero-knowledge proof needs to fulfill in order to serve as a computationally sound implementation of symbolic (Dolev-Yao style) zero-knowledge proofs; this leads to the novel definition of a* symbolically-sound zero-knowledge proof system. *We prove that even in the presence of arbitrary active adversaries, such proof systems constitute computationally sound implementations of symbolic zero-knowledge proofs. This yields the first computational soundness result for symbolic zero-knowledge proofs and the first such result against fully active adversaries of Dolev-Yao models that go beyond the core cryptographic operations.*

## 1 Introduction

Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward for humans to make. In fact, vulnerabilities have accompanied the design of such protocols such as early authentication protocols like Needham-Schroeder [DS81, NS78], carefully designed de-facto standards like SSL and PKCS [WS96, Ble98], and current widely deployed products like Microsoft Passport [Fis03] and Kerberos [BCJ+06]. Hence work towards the automation of such proofs started soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following [DY83, EG83, Mer83], e.g., see [KMM94, Sch96, AG97, Low96, Pau98, BMV04]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. It was not at all clear from the outset whether Dolev-Yao models are a sound abstraction from real cryptography with its computational security definitions. Recent work has largely bridged this gap for Dolev-Yao models offering the core cryptographic operations such as encryption and digital signatures, e.g., see [AR00, Lau01, BPW03, BP04, Lau04, MW04, CW05, CH06].

While Dolev-Yao models traditionally comprised only basic cryptographic operations such as encryption and digital signatures, recent work has started to extend them to more sophisticated primitives with unique security features that go far beyond the traditional goal of cryptography to solely offer secrecy and authenticity of communication.

Zero-knowledge proofs constitute the most prominent and arguably most amazing such primitive. A zero-knowledge proof consists of a message or a sequence of messages that combines two seemingly contradictory properties: First, it constitutes a proof of a statement $x$ (e.g, $x =$ "the message within this ciphertext begins with $0$") that cannot be forged, i.e., it is impossible, or at least computationally infeasible, to produce a zero-knowledge proof of a wrong statement. Second, a zero-knowledge proof does not reveal any information besides the bare fact that $x$ constitutes a valid statement. Zero-knowledge proofs were introduced in [GMR89], they were proven to exist for virtually all statements [GMW91], and they in particular serve

*A short version of this paper appears at CSF 2008 [BU08].

as the central ingredient of modern e-voting and attestation protocols such as the Direct Anonymous Attestation (DAA) protocol [BCC04].

A Dolev-Yao style (symbolic) abstraction of zero-knowledge proofs has recently been put forward in [BMU08]. The proposed abstraction is suitable for mechanized proofs and was already successfully used to produce the first fully mechanized proof of central properties of the DAA protocol. However, no computational soundness guarantee for this abstraction has been established yet, i.e., it is not clear if security guarantees established using the symbolic abstraction of zero-knowledge will carry over to protocol implementations relying on cryptographic zero-knowledge proofs, or which of the various standard or nonstandard additional properties of zero-knowledge proofs would be required to achieve this computational soundness result.

In this paper, we first identify which standard and which more sophisticated properties a cryptographic zero-knowledge proof needs to fulfill in order to serve as a computationally sound implementation of symbolic zero-knowledge proofs. This process culminates in the novel definition of a *symbolically-sound zero-knowledge proof system*; we remark that protocols already exist that satisfy this definition. Our main result will then show that symbolically-sound zero-knowledge proof systems constitute computationally sound implementations of symbolic zero-knowledge proofs. This in particular yields the first computational soundness result against fully active attackers of Dolev-Yao models that go beyond the core cryptographic operations, and it constitutes the first soundness result for symbolic zero-knowledge proofs. Our soundness result applies to trace-properties like authentication and weak secrecy.

**Outline of the Paper.** In Section 2, we briefly review the modeling of abstract zero-knowledge proofs, and we identify the properties a cryptographic zero-knowledge proof should fulfill to serve as a computationally sound implementation of the abstraction. Section 3 and 4 contain the abstract and concrete execution model. Section 5 contains our computational soundness result for symbolic zero-knowledge.

**Notation.** By $[n]$ we denote $\{1, \ldots, n\}$. We abbreviate $x_1, \ldots, x_n$ by $\underline{x}$ where $n$ is implicit. We will sometimes use sets and non-terminals interchangeably. E.g., given a grammar $\mathsf{A} = \mathsf{B}|(\mathsf{C}, \mathsf{A})$ we might write $x \in (\mathsf{C}, \mathsf{A})$, or we might say "$x$ has the form $(\mathsf{C}, \mathsf{A})$". We might also write "$x$ has the form $(c, \mathsf{A})$" for a given $c \in \mathsf{C}$.

## 2 Zero-Knowledge Proofs

In this section, we first introduce our modeling of abstract (symbolic) zero-knowledge proofs in an intuitive manner to familiarize the reader with our notation and to prepare the ground for the examples discussed below. A formal semantics will be given to these expressions in Section 3. We afterwards review concrete zero-knowledge proofs, i.e., zero-knowledge proofs in the cryptographic setting. Our particular focus is on identifying which standard and more sophisticated properties such a proof needs to fulfill in order to serve as a cryptographically sound implementation of abstract zero-knowledge proofs.

### 2.1 Abstract Zero-Knowledge Proofs

We start with an example that involves a zero-knowledge proof of medium complexity. Assume that an agent $B$ expects a message $m$ and is supposed to answer with an encryption $c := \{\{\langle\langle m, n\rangle, m'\rangle\}^{R_1}_{\mathsf{ek}(A)}\}^{R_2}_{\mathsf{ek}(S)}$ for a random nonce $n$, a value $m' \in \{m_1, m_2, m_3\}$ and for some agents $A$ and $S$. Here $\mathsf{ek}(X)$ denotes the public key of $X$, and $R_1, R_2$ denote the abstract randomness used to build the encryptions. The protocol under consideration now aims at convincing the recipient $C$ of $c$ that $c$ is of the right form, i.e., the inner plaintext should contain $m$ and some value $m' \in \{m_1, m_2, m_3\}$. In addition, the protocol aims at hiding from $C$ the nonce $n$ and the precise selection of the message $m'$. Zero-knowledge proofs constitute salient tools to achieve these seemingly contradictory properties in that they allow $B$ to prove that it knows some terms that satisfy the desired properties without revealing those terms.

In the example we consider, $B$ intends to prove that it knows some abstract randomness $\rho_1, \rho_2$ and some values $\alpha_1, \alpha_2$ such that with $\beta_1 := m_1$, $\beta_2 := m_2$, $\beta_3 := m_3$, $\beta_4 := S$, $\beta_5 := D$, $\beta_6 := c$, and $\beta_7 := m$ the following formula $F$ evaluates to true:

$$\beta_6 = \left\{\{\langle\langle\beta_7, \alpha_1\rangle, \alpha_2\rangle\}^{\rho_1}_{\mathsf{ek}(\beta_5)}\right\}^{\rho_2}_{\mathsf{ek}(\beta_4)} \wedge (\alpha_2 = \beta_1 \vee \alpha_2 = \beta_2 \vee \alpha_2 = \beta_3).$$

Immediately including the values of $\beta_i$ in the formula $F$ would arguably have increased the readability of the formula; our language defined in Section 3, however, will require a strict separation of the actual formula $F$ and the public parameters that are determined at runtime, resulting in this slightly more complicated notation.

Granting $B$ the ability to produce such a proof is modeled by introducing an abstract constructor $\mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b})$, called a *zero-knowledge proof*. (Recall that we abbreviate tuples $r_1, \ldots, r_n$ by $\underline{r}$, similarly for $\underline{a}$ and $\underline{b}$). Its arguments are an abstract randomness $R$, a formula $F$, as well as values $r_i, a_i, b_i$ that will serve as substitutes for the variables

$\rho_i, \alpha_i, \beta_i$ in $F$. In our example, the agent $B$ will send the proof $z := \mathsf{ZK}_F^R(R_1, R_2; n, m; m_1, m_2, m_3, S, D, c, m)$. The semantics of this constructor (formally defined in Section 3) will guarantee two properties: First, a zero-knowledge proof can only be constructed by providing suitable instantiations $\underline{r}, \underline{a}, \underline{b}$ for $\underline{\rho}, \underline{\alpha}, \underline{\beta}$ so that the formula $F$ yields true. Second, while the formula $F$ and the values $\underline{b}$ can be retrieved from a zero-knowledge proof, the values $\underline{a}$ and the randomness $\underline{r}$ are kept secret. These properties imply that the proof $z$ indeed guarantees that $c$ has the right form without revealing any additional information about $n, m$. In an abstract zero-knowledge proof, we call $\underline{r}; \underline{a}$ the *witness* and $\underline{b}$ the *public part*.

For more elaborate examples on how zero-knowledge proofs can be used in an abstract setting, comprising a larger set of base constructors such as blind signatures, we refer the interested reader to [BMU08].

## 2.2 Concrete Zero-Knowledge Proofs

We now move to concrete zero-knowledge proofs, i.e., zero-knowledge proofs in the cryptographic setting. A central contribution of this paper is to identify which standard and more sophisticated additional properties of zero-knowledge proofs are required to establish the desired computational soundness result. Hence we now explain in some detail why each such property is needed. In the end, this task will culminate in the novel definition of a *symbolically-sound zero-knowledge proof system*. We first state the properties in an informal way and give the exact definitions in Definition 1.

**Completeness, Soundness, and Zero-knowledge.** We start with the basic definition of a non-interactive zero-knowledge proof. We need to focus on non-interactive proofs since the abstract model considers a proof as a single message that can processed further locally, e.g., it can be encrypted. This would not be meaningful if the zero-knowledge proof was allowed to be interactive.

A zero-knowledge proof consists of four algorithms $K, P, V, S$, called the CRS-generator, the prover, the verifier, and the simulator, respectively. The CRS-generator outputs a random bitstring called the *common reference string* (CRS) that can be seen as a public key for the encryption scheme. The prover $P$ expects as inputs the CRS, a circuit $C$, and a witness $w$ such that $C(w) = 1$ and outputs a corresponding proof $z$ (intuitively denoting that $C$ is a satisfiable circuit). The verifier expects the CRS, a circuit $C$, and a proof $z$ and checks whether $z$ is indeed a proof for the satisfiability of $C$. (It is sufficient to consider satisfiability of circuits since every NP-language can be reduced to this problem.)

Three properties are expected from a zero-knowledge proof: It should be possible to prove correct statements (*completeness*), it should not be possible to prove incorrect statements (*soundness*), and the verifier should not learn anything about the witness, beyond what can be deduced from the fact that $C$ is satisfiable (*zero-knowledge*):

- *Completeness:* For any $C$ and $w$ with $C(w) = 1$, if $z$ is the proof produced by $P$, then $V$ accepts $z$.
- *Soundness:* For any $C$ and $w$ with $C(w) = 0$, and for any polynomial-time adversary $\mathcal{A}$ that outputs a proof $z$, the verifier does not accept $z$.
- *Zero-knowledge:* When computing the CRS, the algorithm $K$ additionally outputs a *simulation trapdoor* $simtd$ (that can be seen as a secret key for the CRS) such that the following holds: Fix $C$ and $w$ with $C(w) = 1$. Let $z$ be the proof produced by prover $P$. Let $z'$ be the proof produced by $S$ on input $simtd$ and $C$ (but not $w$). Then $z$ and $z'$ are computationally indistinguishable.

In this section, we omit certain details such as the fact that the conditions are allowed to be broken with negligible probability. Similarly, we implicitly assume that $P$ and $V$ use the circuit $C$ and the witness $w$. The final Definition 1 below will contain all these details.

A scheme satisfying these three properties is referred to as a non-interactive zero-knowledge proof system. Readers that are familiar with interactive zero-knowledge proof may notice that the definition of interactive zero-knowledge proofs does not include a CRS. In the non-interactive case, however, zero-knowledge proofs without a CRS are impossible unless $\mathrm{NP} \subseteq \mathrm{BPP}$ [Gol01, Thm. 4.4.12].

**Extractability.** While the three properties of completeness, soundness, and zero-knowledge are sufficient for many applications, they do not suffice to offer a cryptographically sound implementation of abstract zero-knowledge proofs. This can be seen by inspecting the following example: Assume that we are using an encryption scheme that permits to efficiently check that a given ciphertext $c$ constitutes a valid encryption of some message (without having to know this message or the secret key). Then let $c := \{m\}^R_{\mathsf{ek}(a)}$ and consider the proof $z := \mathsf{ZK}_F^{R'}(R; m; c, a)$ with $F := (\beta_1 = \{\alpha_1\}^{\rho_1}_{\mathsf{ek}(\beta_2)})$, i.e., a proof that one knows the message and the randomness contained in $c$. In the cryptographic setting, this proof would be performed by first constructing a circuit $C$ such that $C(R, m) = 1$ iff $m$ is encrypted with randomness $R$ and the public key of $a$ yields the ciphertext $c$. Since one can efficiently check if $c$ is the encryption of some message, one can hence efficiently check as well if $C$ has a satisfying input. Thus, one can prove the satisfiability of $C$ without having to use $R, m$. Such a proof trivially conforms to the zero-knowledge property, since the proof does not exploit

the witness, and it satisfies soundness since it only proves valid statements (if $C$ was not satisfiable, the proof would not succeed). In the abstract model, however, it is obvious that one needs to know $m$ in order to produce $z$. What went wrong? The soundness condition only guarantees the existence of a witness, but it does not require the prover to actually know this witness. We introduce an additional algorithm $E$ (besides $K, P, V, S$, called the extraction algorithm) to capture this requirement and define the stronger condition of extractability. A proof system with extractability is called a *proof of knowledge*.

- *Extractability:* When computing the CRS, the algorithm $K$ additionally outputs an *extraction trapdoor* $extd$ such that: Fix $C$ (where $C$ may or may not be satisfiable). For a polynomial-time adversary $\mathcal{A}$ that outputs a proof $proof$, we have that if $V$ accepts $proof$, then $E(C, proof, extd)$ outputs a witness $w$ with $C(w) = 1$.

With this definition, our example no longer causes any problems: An extractable proof system that allows to prove the satisfiability of $C$ without using $w$ would lead to a contradiction since the machines $\mathcal{A}$ and $E$ together could then compute $w$. (Technically, this is only a contradiction if $w$ is not easy to compute from $C$ in the first place.) We stress that extractability already implies soundness: If $C$ is not satisfiable, then $E(C, proof, extd)$ cannot output a witness $w$ with $C(w) = 1$, thus by contraposition we have that $V$ does not accept $proof$.

**Extraction zero-knowledge.** Even complementing the properties completeness, soundness, and zero-knowledge with the extractability property is still not sufficient for the desired computational soundness result. Consider a proof system with the following property: If $proof_1$ constitutes a proof for the circuit $C_1$ and $proof_2$ constitutes a proof for the circuit $C_2$, then $(proof_1, proof_2)$ constitutes a proof for the circuit $C_1 \wedge C_2$ (with $(C_1 \wedge C_2)(w_1, w_2) := C_1(w_1) \wedge C_2(w_2)$). This property is not unrealistic, and for circuits that are of this conjunctive form, concatenating proofs for the individual circuits indeed often constitutes the most efficient way to produce a proof for the combined circuit. Furthermore, allowing to prove these sub-circuit individually does not contradict the properties we have discussed so far. In the abstract model, however, given $\mathsf{ZK}_F^R(r; a; b)$ and $\mathsf{ZK}_{F'}^{R'}(r'; a'; b')$, it is not possible to construct a proof $\mathsf{ZK}_{\overline{F \wedge F'}}(\_; \_; b, b')$ without knowing $r, r', a, a'$ (where _ matches everything, and where in the formula $F'$ the $\rho_1, \alpha_1, \beta_1$ are renamed to $\rho_2, \alpha_2, \beta_2$). We hence have to exclude the possibility of concatenating proofs to generate new proofs. More precisely, we have to ensure that given a proof for some statement $x_1$, it is not possible to construct a proof for another statement $x_2$, even if $x_2$ is logically related to $x_1$. This property is called *non-malleability* and closely resembles the notion of non-malleability of encryption schemes. In the context of zero-knowledge, several properties are known to imply non-malleability. We will exploit the *extraction zero-knowledge* property from [GO07]. Although this is a rather strong property and weaker definitions of non-malleability exist, our proof relies on this particular property; we leave it as an open problem if our computational soundness result can be proven using a weaker formalization of non-malleability.

- *Extraction zero-knowledge:* Let $simtd, extd$ be the simulation and the extraction trapdoor as output by $K$, respectively. Consider a polynomial-time adversary $\mathcal{A}$ that has access to the simulation trapdoor $simtd$ and to an extraction oracle $E(C, \cdot, extd)$, i.e., when invoking the oracle $E(C, \cdot, extd)$ with input $proof$, it returns $E(C, proof, extd)$ where $E$ is the extraction algorithm. The adversary may output $C, w$ with $C(w) = 1$. Then the adversary gets either (a) a real proof $proof$ produced by $P$, or (b) a simulated proof $proof$ produced by the simulator $S$ (which has no access to $w$). We then require that $\mathcal{A}$ cannot distinguish the cases (a) and (b) as long as it does not query $proof$ from the extraction oracle.

We stress that extraction zero-knowledge implies the zero-knowledge property since it implies that for any $C, w$ with $C(w) = 1$ the proofs produced by the prover and the simulator are indistinguishable.

Why does extraction zero-knowledge indeed imply non-malleability? Assume that from a proof $proof_1$ for the satisfiability of $C_1$, some algorithm $M$ can produce a proof $proof_2$ for the satisfiability of $C_2$ where the satisfiability of $C_2$ follows from that of $C_1$ (in the sense that a witness $w_1$ for $C_1$ can be converted into a witness $w_2$ for $C_2$). Then an adversary $\mathcal{A}$ could break the extraction zero-knowledge property roughly as follows: First, it outputs $C_1, w_1$ and gets a proof $proof_1$ (this might be a fake proof). It applies $M$ to $proof_1$ and gets a proof $proof_2$ for $C_2$. Since $proof_2 \neq proof_1$, the adversary may give $proof_2$ to the extraction oracle. In case (a), the extraction oracle will output a witness $w_2$. In case (b), however, the proof $proof_1$ has been produced without exploiting the witness $w_1$, and so has $proof_2$. Since the extraction oracle cannot produce a witness, it will fail to produce a witness for $proof_2$ in case (b). Thus the adversary can distinguish the two cases, contradicting the extraction zero-knowledge property.

**Unpredictability.** We are lacking one last property for soundly implementing abstract zero-knowledge proofs. If a proof using the same witness and the same public part is produced by two different agents in the abstract model, it will always lead to two different terms because the two terms will have different randomness. A proof system with the properties described in this section, however, does not nec-

essarily ensure that two proofs produced with the same witness and circuit will always be different (with overwhelming probability). Indeed, it is possible to construct proofs that are deterministic for at least some inputs. We hence additionally require that any two independently produced proofs are different, or equivalently:

- *Unpredictability.* Let a polynomial-time adversary $\mathcal{A}$ output $C, w, proof'$ with $C(w) = 1$. Let $proof$ be produced by $P$. Then with overwhelming probability, $proof \neq proof'$.

Unpredictability is an easily achievable property, e.g., by letting the prover include some randomness in the proof.

**Symbolically-sound zero-knowledge.** Finally, we further require that the verifier $V$ and the extraction algorithm $E$ are deterministic. This is not strictly necessary but it will simplify the proof of soundness, and we are not aware of a non-interactive zero-knowledge proof system where this condition is not fulfilled. The full name of a zero-knowledge scheme satisfying all the above properties would be *unpredictable non-interactive multi-theorem adaptive extraction zero-knowledge argument of knowledge with deterministic verification and extraction*. Since this is somewhat unwieldy, we coin the term *symbolically-sound zero-knowledge proof system*. The following definition formally defined the properties we informally discussed above.

**Definition 1** (Symbolically-sound zero-knowledge proof system)**.** *A* symbolically-sound zero-knowledge proof system *is a tuple of polynomial-time algorithms* $(K, P, V, S, E)$ *with the following properties:*

- Completeness: *Let a nonuniform polynomial-time adversary $\mathcal{A}$ be given. Let* $(crs, simtd, extd) \leftarrow K(1^\eta)$. *Let* $(C, w) \leftarrow \mathcal{A}(1^\eta, crs)$. *Let* $proof \leftarrow P(C, w, crs)$. *Then with overwhelming probability in* $\eta$, $C(w) = 0$ *or* $V(C, proof, crs) = 1$.
- Extractability: *Let a nonuniform polynomial-time adversary $\mathcal{A}$ be given. Let* $(crs, simtd, extd) \leftarrow K(1^\eta)$. *Let* $(C, proof) \leftarrow \mathcal{A}(1^\eta, crs)$. *Let* $w \leftarrow E(proof, extd)$. *Then with overwhelming probability, if* $V(C, proof, crs) = 1$ *then* $C(w) = 1$.
- Unpredictability: *Let a nonuniform polynomial-time adversary $\mathcal{A}$ be given. Let* $(crs, simtd, extd) \leftarrow K(1^\eta)$. *Let* $(C, w, proof') \leftarrow \mathcal{A}(1^\eta, crs, simtd, extd)$. *Then with overwhelming probability, we have* $proof' \neq P(C, w, crs)$.
- Extraction Zero-Knowledge: *Let a nonuniform polynomial-time adversary $\mathcal{A}$ be given. Consider the following experiment parametrized by a bit $b$: Let* $(crs, simtd, extd) \leftarrow K(1^\eta)$. *Let* $(C, w, state) \leftarrow \mathcal{A}^{E(\cdot, extd)}(1^\eta, crs, simtd)$. *Then let* $proof \leftarrow P(C, w, crs)$ *if* $b = 0$ *and* $proof \leftarrow S(C, crs, simtd)$ *if* $b = 1$. *Let* $guess = \mathcal{A}^{E(\cdot, extd)}(1^\eta, crs, simtd, state, proof)$. *Let* $P_b(\eta)$ *denote the following probability:*

$$P_b(\eta) := \Pr\big[guess = 1 \text{ and } C(w) = 1 \text{ and } proof \text{ has not been queried from } E(\cdot, extd)\big].$$

  *Then* $|P_0(\eta) - P_1(\eta)|$ *is negligible.*
- Deterministic verification and extraction. *The algorithms $V$ and $E$ are deterministic.*

We stress that protocols already exist that satisfy this notion, e.g., the one proposed in [GO07, Sect. 3] under the assumption that enhanced trapdoor permutations [Gol04, Def. C.1.1] exist. The latter exist, e.g., under the assumption that factoring is hard.

We have formulated symbolically-sound zero-knowledge proof systems against nonuniform adversaries. We believe that our results easily carry over to the uniform case.

## 3 The Abstract Model

In the following we define the abstract model in which the execution of a symbolic protocol involving zero-knowledge proofs takes place. The basic ideas follow the framework presented in [CKKW06]. However, to incorporate zero-knowledge proofs, we have to make various non-trivial modifications to the abstract model. In the following sections, we try to highlight and explain the design choices made in our modeling.

**ZK-proofs and messages.** First, we fix several countably infinite sets. By $\mathsf{A}$ we denote the set of agent identifiers, by $\mathsf{Nonce}$ the set of nonces. We use elements from $\mathsf{Garbage}$ to represent ill-formed messages (corresponding to unparseable bitstrings in the concrete model). Finally, elements of $\mathsf{Rand}$ denote abstract randomness used in the construction of ciphertexts and zero-knowledge proofs. We assume that $\mathsf{Nonce}$ is partitioned into $\mathsf{Nonce}_{ag}$ and $\mathsf{Nonce}_{adv}$, representing the nonces of honest agents and the nonces of the adversary. Similarly, $\mathsf{Rand}$ is partitioned into $\mathsf{Rand}_{ag}$ and $\mathsf{Rand}_{adv}$.

We proceed by defining the syntax of messages that can be sent in a protocol execution. Since such messages can contain zero-knowledge proofs, and these are parametrized over a statement that is to be proven, we first have to define the syntax of these formulas. Let

$$\mathsf{ZKTerm} = \mathsf{ek}(\beta_i) \mid \alpha_i \mid \beta_i \mid \langle \mathsf{ZKTerm}, \mathsf{ZKTerm} \rangle \mid \{\mathsf{ZKTerm}\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$$

where $i = 1, 2, \ldots$. We call terms produced by this grammar ZK-terms. On the intuitive level, $\mathsf{ek}(a)$ denotes a public

encryption key of agent $a$, $\langle\cdot,\cdot\rangle$ a pair, and $\{t\}^R_{\mathsf{ek}(a)}$ an encryption of $t$ with the public key of $a$ using randomness $R$.

Then a ZK-Formula $F$ is a Boolean formula over terms of the form $\mathsf{ZKTerm} = \mathsf{ZKTerm}$ satisfying the following conditions: If $\alpha_i$ occurs in $F$, then $\alpha_1, \ldots, \alpha_i$ occur all in $F$. An analogous condition holds for $\rho_i$ and $\beta_i$.[1] We denote the set of ZK-Formulas with $\mathsf{Formula}$. The $\alpha$-arity of a ZK-Formula $F$ is the largest index of an $\alpha_i$ occurring in $F$. The $\rho$-arity and the $\beta$-arity are defined analogously.

The intuitive interpretation of a ZK-formula is that it is a term with free variables $\underline{\rho}$, $\underline{\alpha}$, and $\underline{\beta}$. The $\underline{\rho}$ will be substituted with randomness, the $\underline{\alpha}$ and $\underline{\beta}$ with messages. A zero-knowledge proof $\mathsf{ZK}^R_F(\underline{r};\underline{a};\underline{b})$ then represents a proof that when substituting $\underline{r},\underline{a},\underline{b}$ for $\underline{\rho},\underline{\alpha},\underline{\beta}$, the resulting expression $F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ is a true statement. The randomness $\underline{r}$ and the messages $\underline{a}$ will be considered secret, while the messages $\underline{b}$ will be contained in the proof in clear (one can think of the formula as being parametrized in the values $\underline{b}$).

Note the following interesting asymmetry: We allow $\mathsf{ek}(\beta_i)$ to appear in a formula, but not $\mathsf{ek}(\alpha_i)$. This is due to the fact that a proof with a formula containing $\mathsf{ek}(\alpha_i)$ would not be easily realizable computationally: In order to perform the zero-knowledge proof, we need to build a circuit accepting only satisfying values $\underline{a}$ for the $\underline{\alpha}$. To build such a circuit for a formula with $\mathsf{ek}(\alpha_i)$, one would have to encode a list of *all* public keys in this circuit. On the other hand, in the case of $\mathsf{ek}(\beta_i)$, the value $b_i$ substituted for $\beta_i$ is known while constructing the circuit, thus we can directly hard-code $\mathsf{ek}(b_i)$ into the circuit.

Given the syntax for ZK-formulas, we can define the set $\mathsf{M}$ of messages as

$$\begin{aligned}\mathsf{M} = \mathsf{A} \mid \mathsf{Nonce} \mid \mathsf{ek}(\mathsf{A}) \mid \mathsf{ek}(\mathsf{Garbage}) \mid \mathsf{dk}(\mathsf{A}) \mid \langle \mathsf{M}, \mathsf{M}\rangle \\ \mid \{\mathsf{M}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{A})} \mid \{\mathsf{Garbage}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{Garbage})} \mid \mathsf{Garbage} \\ \mid \mathsf{ZK}^{\mathsf{Rand}}_{\mathsf{Formula}}(\mathsf{Rand}^*;\mathsf{M}^*;\mathsf{M}^*).\end{aligned}$$

with the following additional condition: For each subterm $\mathsf{ZK}^{\mathsf{Rand}}_F(\underline{r};\underline{a};\underline{b})$, we have that $|\underline{r}|, |\underline{a}|, |\underline{b}|$ are the $\rho$-arity, $\alpha$-arity, and $\beta$-arity of $F$, respectively. Here $\mathsf{ek}(a)$ and $\mathsf{dk}(a)$ represent encryption and decryption keys for the agent $a$, $\langle\cdot,\cdot\rangle$ means pairing, $\{t\}^R_{\mathsf{ek}(a)}$ is the encryption of message $t$ under the key $\mathsf{ek}(a)$ with randomness $R$, and $\mathsf{ZK}^R_F(\underline{r};\underline{a};\underline{b})$ denotes a zero-knowledge proof for the formula $F$ produced using the randomness $R$ where the (secret) witness consists of the randomness $\underline{r}$ and the messages $\underline{a}$, and the public part consists of the messages $\underline{b}$.

Since both honest agents and the adversary should only send ZK-proofs that actually correspond to true statements, we will need the following definition that characterizes the messages that do not contain wrong proofs.

[1] We actually use this condition in the proof only for the $\underline{\rho}$. However, we included the condition also for $\underline{\alpha}$ and $\underline{\beta}$ for symmetry.

**Definition 2** (True ZK-Proofs)**.** *Let a message of the form $Z := \mathsf{ZK}^R_F(\underline{r};\underline{a};\underline{b})$ be given.*

*Let $Z_1 := F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$. Replace all subterms of $Z_1$ that are of the form $t = t$ by* True *and all subterms of $Z_1$ that are of the form $t = u$ with $t \neq u$ (where $\neq$ is meant as being syntactically different as terms, no equational theory is involved) by* False*. Call the result $Z_2$. Note that $Z_2$ is a Boolean formula without variables.*

*We say that $Z$ is* a true proof*, if all subterms of $Z_1$ are messages*[2] *and $Z_2$ evaluates to* True*.*

*We say a message $M$* contains true proofs *if every subterm of $M$ of the form $\mathsf{ZK}^{\mathsf{Rand}}_{\mathsf{Formula}}(\ldots)$ is a true proof.*

**Deduction rules.** In order to restrict the actions the adversary may perform during a protocol execution, we have to introduce a deduction relation $\vdash$ which is given by the rules in Figure 1. The rules for the deduction are standard, only the rules concerning zero-knowledge proofs merit additional comment. The rule $\varphi \vdash \mathsf{ZK}^r_F(\underline{r};\underline{a};\underline{b}) \Longrightarrow \varphi \vdash \underline{b}$ represents the zero-knowledge property; from a zero-knowledge proof $\mathsf{ZK}^r_F(\underline{r};\underline{a};\underline{b})$, all that can be extracted is the public part $\underline{b}$, but not the witness $\underline{r},\underline{a}$.

More interesting and involved is the last rule in Figure 1. This rule states under which conditions the adversary may construct a zero-knowledge proof $\mathsf{ZK}^r_F(\underline{r};\underline{a};\underline{b})$. First, of course, the resulting proof must be a proof of a true statement. This is represented by the condition that $F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ is a true proof (as in Definition 2). Furthermore, we have to require that the adversary actually knows the witness $\underline{r},\underline{a}$ and the public part $\underline{b}$, corresponding to the fact that we model proofs of *knowledge*. For $\underline{a}$ and $\underline{b}$ this condition is modeled by requiring $\varphi \vdash \underline{a}$ and $\varphi \vdash \underline{b}$. For the randomness $\underline{r}$, however, the condition is more involved. The adversary may know some randomness $r_i$ in two cases. First, if it is its own randomness $r_i \in \mathsf{Rand}_{adv}$. Second, it may be able to extract that randomness from an encryption produced by some honest party. Namely, the condition that a cryptosystem is IND-CCA secure does not imply that one cannot retrieve the randomness used in an encryption *provided one can decrypt that message.* For example, in the popular RSA-OAEP cryptosystem [BR95, FOPS04], the randomness used for encrypting a message is actually computed during an honest decryption. Thus we have to allow the adversary to use randomness $r_i$ from messages it was able to decrypt. As an example why this condition is actually needed for computational soundness, consider the following simple protocol. Agent $a$ sends $c := \{m\}^R_{\mathsf{ek}(b)}$ and if it receives a proof matching $\mathsf{ZK}^-_F(\_;;b,m,c)$ with $F := (\beta_3 = \{\beta_2\}^{\rho_1}_{\mathsf{ek}(\beta_1)})$, the protocol fails (here the symbol _ matches everything). If we would only allow the adversary to use $r_i \in \mathsf{Rand}_{adv}$

[2] This would be violated, e.g., by $\mathsf{ZK}^R_F(R';;c,m,n)$ with $F := (\beta_1 = \{\beta_2\}^{\rho_1}_{\mathsf{ek}(\beta_3)})$ where $n$ is a nonce.

$$\frac{m \in \varphi}{\varphi \vdash m} \qquad \frac{g, g' \in \mathsf{Garbage} \qquad r \in \mathsf{Rand}_{adv}}{\varphi \vdash g \qquad \varphi \vdash \mathsf{ek}(g) \qquad \varphi \vdash \{g'\}^r_{\mathsf{ek}(g)}} \qquad \frac{b \in \mathsf{A}}{\varphi \vdash \mathsf{ek}(b) \qquad \varphi \vdash b} \qquad \frac{\varphi \vdash m_1 \qquad \varphi \vdash m_2}{\varphi \vdash \langle m_1, m_2 \rangle}$$

$$\frac{\varphi \vdash \langle m_1, m_2 \rangle}{\varphi \vdash m_1 \qquad \varphi \vdash m_2} \qquad \frac{\varphi \vdash \mathsf{ek}(b) \qquad \varphi \vdash m \qquad r \in \mathsf{Rand}_{adv}}{\varphi \vdash \{m\}^r_{\mathsf{ek}(b)}} \qquad \frac{\varphi \vdash \{m\}^r_{\mathsf{ek}(b)} \qquad \varphi \vdash \mathsf{dk}(b)}{\varphi \vdash m}$$

$$\frac{\varphi \vdash \mathsf{ZK}^r_F(\underline{r}; \underline{a}; \underline{b})}{\varphi \vdash \underline{b}} \qquad \frac{\begin{array}{c}\varphi \vdash \underline{a} \qquad \varphi \vdash \underline{b} \qquad F \in \mathsf{Formula} \qquad F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\} \text{ is a true proof} \\ r \in \mathsf{Rand}_{adv} \qquad \forall i : r_i \in \mathsf{Rand}_{adv} \vee (\exists t, a : \varphi \vdash \{t\}^{r_i}_{\mathsf{ek}(a)} \wedge \varphi \vdash \mathsf{dk}(a))\end{array}}{\varphi \vdash \mathsf{ZK}^r_F(\underline{r}; \underline{a}; \underline{b})}$$

**Figure 1. Deduction rules for the adversary.**

in the witness, the protocol would be secure abstractly, even if the adversary knows the secret key $sk(b)$. Yet a concrete adversary could possibly (depending on the encryption scheme) extract the randomness from $c$ and produce such a proof.[3]

**Patterns.** In order to conveniently define the notion of a protocol, we need a way to succinctly describe how messages are parsed and constructed by honest agents. To this aim, we define the concept of a pattern.[4]

Let $\mathsf{X}.a$, $\mathsf{X}.n$, $\mathsf{X}.p$, $\mathsf{X}.c$, $\mathsf{X}.z$ be countably infinite sets (variables of sort agent, nonce, pair, ciphertext, ZK-proof, respectively). Let $\mathsf{X} := \mathsf{X}.a|\mathsf{X}.n|\mathsf{X}.e|\mathsf{X}.p|\mathsf{X}.c|\mathsf{X}.z$. In the following, when considering mappings from variables $\mathsf{X}$ to messages $\mathsf{M}$, we will always assume that a variable is mapped to a message of corresponding type. We then define the set $\mathsf{Pat}$ of patterns as

$$\mathsf{Pat} = \mathsf{X} \mid \mathsf{ek}(\mathsf{X}.a) \mid \langle \mathsf{Pat}, \mathsf{Pat} \rangle \mid \{\mathsf{Pat}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{X}.a)} \mid \{\mathsf{Pat}\}^{-}_{\mathsf{ek}(\mathsf{X}.a)} \mid$$
$$\mathsf{ZK}^{\mathsf{Rand}}_{\mathsf{Formula}}(\mathsf{Rand}^*; \mathsf{Pat}^*; \mathsf{Pat}^*) \mid \mathsf{ZK}^{=}_{\mathsf{Formula}}(\_^*; \_^*; \mathsf{Pat}^*)$$

with the following additional conditions: For each subterm $\mathsf{ZK}^{\mathsf{Rand}}_F(\underline{r}; \underline{a}; \underline{b})$, we have that $|\underline{r}|, |\underline{a}|, |\underline{b}|$ are the $\rho$-arity, $\alpha$-arity, and $\beta$-arity of $F$, respectively, and if $\mathsf{ek}(\beta_i)$ occurs in $F$, then $b_i$ has the form $\mathsf{ek}(A)$ or $\mathsf{ek}(\mathsf{X}.a)$. These conditions are needed to ensure that a pattern (containing no _) becomes a valid message when the variables are instantiated.

The symbol _ is supposed to match anything. More exactly, we say a message $m \in \mathsf{M}$ matches a pattern $p \in \mathsf{Pat}$ if there is a substitution $\theta : \mathsf{X} \to \mathsf{M}$ such that $p\theta$ equals $m$ up to occurrences of _ in $p\theta$ (where distinct occurrences of _ may correspond to different subterms in $m$). We call $\theta$ the matcher of $m$ and $p$. Thus intuitively _ in a pattern corresponds to a value we do not care about and that we do not intend to (and cannot) extract, e.g., the randomness used in a ciphertext[5] or the witness of a zero-knowledge proof.

Note that patterns do not contain explicit nonces, agent identifiers, or garbage. We omit garbage because we do not want the protocol to explicitly construct ill-formed messages. Nonces and agents are not needed since the protocol execution (see below) will provide pre-initialized variables for the nonces used by an agent and for the ids of the communication partners in a given protocol session. We disallow patterns of the form $\mathsf{dk}(a)$ since we do not allow protocols to explicitly use their private keys (except for decrypting). This is to ensure that no key cycles occur; it is known that the IND-CCA property does not guarantee security in the presence of key cycles.

**Roles and protocols.** We are now ready to define what a protocol is. For space reasons, we only give an informal description and postpone exact definitions to Appendix A. A *$k$-party protocol* $\Pi$ is a mapping that assigns each $i \in [k]$ a role $\Pi(i)$. In our setting, a *role* is modeled as an ordered edge-labeled finite tree. The nodes of the role tree correspond to states of an agent executing that role, and the edges correspond to transitions caused by incoming messages. We assume only insecure channels between agents, therefore all

[3] After these explanations, the reader might wonder why similar adjustments are not necessary for, e.g., the rule for deducing ciphertexts $\{m\}^r_{\mathsf{ek}(b)}$ since a concrete adversary could use extracted randomness also in this case. The rough reason why this is not necessary is that if the concrete randomness $r$ is used to encrypt another message or under another key, we can consider it to be another abstract randomness. Only if the *same* $m$ is encrypted under the *same* $\mathsf{ek}(b)$, we will have to consider the concrete randomness to be the same. However, in this case the resulting ciphertext will be also be the same, thus the adversary has just produced a message it already knew.

[4] Note that one could be tempted to define a pattern just as a message with variables in it. However, this definition would leave several points open, e.g., variables of what type might occur in which position, etc. Thus we give an explicit grammar and use this opportunity to reduce the set of patterns to such that make sense in protocols (e.g., a protocol may not explicitly send Garbage.

[5] On the preceding page we said that it is possible to extract randomness from ciphertexts. However, at that point we were talking about the adversary and had to assume the worst case. When defining protocols, we may only include capabilities that may be implemented with any encryption scheme. E.g., in the Cramer-Shoup cryptosystem [CS98], extracting the randomness implies breaking the discrete logarithm problem, even when given the private key.

messages are sent to the adversary and received from the adversary. What messages a role sends, and the state the role enters upon receipt of a given message is specified by the labels on the edges of the role tree. More concretely, an edge is labeled with a pair $(l, r)$ of patterns. Here $l$ represents the pattern for matching incoming messages, and $r$ the pattern for constructing the answers to these messages. More exactly, the state of a role consists of a node in its tree, and a partial mapping $\sigma : \mathsf{X} \to \mathsf{M}$ representing (fragments of) messages parsed so far. Given an incoming message $m$, a state $\sigma$, and an edge $(l, r)$, the following steps take place:[6]

- First, in the pattern $l$, the variables that have already been assigned are instantiated. Formally, the pattern $l\sigma$ is computed.
- Then $m$ is matched against $l\sigma$. If this succeeds, let $\theta$ be the matcher. Otherwise, the transition corresponding to the edge $(l, r)$ will not be taken.
- Now all variables in the outgoing pattern $r$ are instantiated, either with variables assigned previously in $\sigma$, or in the previous step ($\theta$). More formally, the message $m' := r\sigma\theta$ is computed. If $m'$ does not contain true proofs (Definition 2), the transition will not be taken. Otherwise follow the edge, send message $m'$ and let the new state be $\sigma \cup \theta$.

A node may have several outgoing (ordered) edges, in this case the first one will be chosen that matches and results in a message $m'$ containing true proofs. If no such edge is found, the role will ignore the message $m$ (i.e., the state is unmodified). A role may access the agent id of the $i$-th communication partners in its session via the pre-initialized variable $A_i \in \mathsf{X}.a$, and accesses its own nonces via the pre-initialized variables $X^j_{A_i} \in \mathsf{X}.n$ (accessing the nonces via variables enables us to model that each session has different nonces).

This model of a role is very similar to that presented in [CKKW06] with the exception of the additional check whether the outgoing message $m'$ contains true proofs. This check is necessary, since we have no syntactic condition that guarantees that a role can only generate true proofs. In particular, if a role produces proofs that depend on incoming messages, and if these messages happen to be modified by the adversary, it may happen that the proofs are instantiated with the wrong values. Thus we have to make a design choice. We can restrict the patterns such that no matter how the variables are instantiated, no incorrect proofs can be produced. We can impose a static condition on the roles that guarantees that for no sequence of incoming messages, an incorrect proof can be produced. Or we can perform a runtime check to avoid incorrect proofs. The first method seems very restrictive, the second might make the definition of a role unnecessarily complicated, thus we have opted for the third variant. Note that not all current tools for protocol verification are able to handle such runtime checks and might need to be extended.

Of course, not all trees with edges labeled by patterns represent valid protocol roles. Instead, we have to impose a variety of sanity conditions, e.g., we have to require that the pattern $r$ (for constructing messages) does not contain free variables, or that the pattern matching an incoming message does not imply decrypting with someone else's secret key. Most conditions are of this kind and just guarantee that the abstract protocol can indeed be implemented as a concrete protocol. Complete details are given in Definition 5 in Appendix A. At this point, we only mention two conditions that are of particular importance.

First, as already discussed on the previous page, a pattern cannot explicitly contain secret keys. Thus a role cannot sent these keys over the network. (Note that the adversary can however get access to secret keys by corrupting parties and can then send them.) This condition is not related to the introduction of zero-knowledge proofs; it is also present, e.g., in [CKKW06].

Since both encryption and zero-knowledge proofs are probabilistic, we have to ensure that each randomness is used only once. In a model without zero-knowledge proofs (as, e.g., [CKKW06]) this can be done by requiring that for any randomness $R$, there is at most one subterm containing $R$ (but the same subterm may occur several times to allow for sending several copies of a single ciphertext). In the presence of zero-knowledge proofs, however, such a rule would be too restrictive. For example, an agent might want to send a ciphertext $c := \{t\}^R_{\mathsf{ek}(a)}$ and then prove that $t$ satisfies some property $P(t)$, i.e., it sends $z := \mathsf{ZK}^{R'}_F(R; t; c, a)$ with $F := (\beta_1 = \{\alpha_1\}^{\rho_1}_{\mathsf{ek}(\beta_2)} \wedge P(\alpha_1))$. Since both $c$ and $z$ contain $R$, such an agent would be disallowed. To relax this restriction, we have to allow the use of a given randomness $R$ in the ($\rho$-part of the) witness of a zero-knowledge proof. However, allowing completely unrestricted use of $R$ in the witness could also lead to problems. For example, consider an agent creating and sending a ciphertext $c$ using a given randomness $R$, and then trying to produce a zero-knowledge proof $z$ proving a statement about *another* ciphertext $c'$ using the *same* randomness $R$. In this case, the adversary learns the ciphertext $c$ and whether the proof $z$ is true (since the further actions of the agent depends on whether it succeeded in constructing the proof or not). It is not clear that the information whether the proof $z$ is true might not already leak up to one bit of information about $c$. We therefore have to ensure that a given randomness $R$ occurs only in a single subterm $t$ plus additionally in the witness of zero-knowledge proofs *as long as it is used in the formula to produce the same term* $t$. To capture this more formally, we introduce the notion of an effective $R$-

[6] Actually, this is not part of the definition of a role, but of the protocol execution. However, we describe it here so that the intended behavior of a role becomes clear.

subpattern. Roughly, an effective $R$-subpattern of a pattern $P$ is either a subterm of $P$, or a subterm that results from substituting the arguments of a zero-knowledge proof in $P$ into its formula. Formally, we get the following definition:

**Definition 3** (Effective subpatterns)**.** *Let $P$ be a pattern. We say that a pattern $S$ is an* effective subpattern *of $P$ if*

- *$S$ is a subterm of $P$, or*
- *there is a subterm $\mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$, and a ZK-Term $z$ in the ZK-formula $F$, such that $S$ is a subpattern of $z\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.*

*We call $S$ an* effective $R$-subpattern *if it is of the form $\{\mathsf{Pat}\}_{\mathsf{Pat}}^R$ or $\mathsf{ZK}_{\mathsf{Formula}}^R(\mathsf{Rand}^*;\mathsf{Pat}^*;\mathsf{Pat}^*)$.*

We can now formulate the condition that randomness may not be reused: For any randomness $R$, there is at most one effective $R$-subpattern in the role tree $\Pi(k)$ (but that subpattern may occur in several places).

**Protocol execution.** The definition of the execution of an abstract protocol $\Pi$ closely resembles the one in [CKKW06], so we only quickly mention the main points. A detailed definition is postponed to Appendix A. An abstract trace for a $k$-party protocol $\Pi$ is a sequence of global states with some restrictions on the possible transitions (detailed below). A *global state* is a triple $(\mathsf{Sid}, f, \varphi)$ where $\varphi$ is the set of messages the adversary learned so far (the *adversary knowledge*, initially set to $\mathsf{Nonce}_{adv}$), the set $\mathsf{Sid}$ contains the ids of all sessions currently running, and the function $f$ maps every session id $sid$ in $\mathsf{Sid}$ to the local state of that session. A session contains exactly one agent $a$ executing one role. However, since the intended protocol execution always involves $k$ parties, a session additionally specifies what other agents the agent $a$ is (supposedly) communicating with. The *local state* of a given session is a tuple $(i, \sigma, p, (a_1, \ldots, a_k))$. Here $i$ is the number of the role the agent $a$ executes in this session. The tuple $(a_1, \ldots, a_k)$ specifies the indented communication partners for that session, in particular, $a = a_i$ is the agent executing this session. The state of the agent $a$ is given by the current node $p$ of the role tree $\Pi(i)$ and the mapping $\sigma$ that maps variables to (fragments of) messages received by the agent $a$ in that session. See the discussion of the role tree on page 7.

We allow three kinds of transitions between global states, namely $\mathbf{corrupt}(a_1, \ldots, a_l)$, $\mathbf{new}(i, a_1, \ldots, a_k)$, and $\mathbf{send}(sid, m)$. In a $\mathbf{corrupt}(a_1, \ldots, a_l)$ transition, the adversary specifies an list of agents $a_1, \ldots, a_l$ whom it wants to corrupt. In consequence, the adversary's knowledge $\varphi$ in the next global state will be extended by $\{\mathsf{dk}(a_1), \ldots, \mathsf{dk}(a_l)\}$, i.e., the adversary learns all secrets of the corrupted parties. Only the first transition is allowed to be of this type, i.e., we consider static corruptions. In a $\mathbf{new}(i, a_1, \ldots, a_k)$ transition, a new session id $sid$ is allocated and added to $\mathsf{Sid}$. The local state of $sid$ is initialized as $(i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where $\varepsilon$ is the root of the role tree $\Pi(i)$ and $\sigma$ maps the variables $A_j$ to $a_j$ and the variables $X_{A_i}^j$ to fresh nonces. In other words, a new session is initialized in which agent $a_i$ runs role $\Pi(i)$ together with $(a_1, \ldots, a_k)$. The most important transition is $\mathbf{send}(sid, m)$. Here, the agent $a$ executing $sid$ is given the message $m$ and its answer $m'$ is added to the adversaries knowledge $\varphi$. Assume that agent $a$ has the local state $(i, \sigma, p, (a_1, \ldots, a_k))$. Then to compute the answer $m'$, the first outgoing edge from $p$ is searched such that its label $(l, r)$ matches $m$ and produces an answer $m'$ that contains true proofs. Details on how this is done have already been given in the discussion of roles on page 7. If no such edge is found, both the local state as well as the knowledge of the adversary are unmodified. Note that the only change with respect to the modeling in [CKKW06] is that we have introduced the additional condition for taking an edge that the answer should contain true proofs.

We call sequences of global states satisfying these rules *symbolic execution traces* or *Dolev-Yao traces*. The set of Dolev-Yao traces for $\Pi$ is denoted $\mathsf{Exec}^s(\Pi)$.

## 4 The Concrete Model

We now proceed to define the concrete execution of a protocol $\Pi$. We use the same protocols $\Pi$ as in the abstract model in the preceding section, but the messages exchanged over the network now are bitstrings, and the patterns $(l, r)$ on the edges of the role tree specify how to parse or construct these bitstrings, respectively.

Since the concrete execution model is quite straightforward and furthermore very similar to the model from [CKKW06], we only sketch it here and concentrate on the design issues particular to the inclusion of zero-knowledge proofs. The details are postponed to Appendix B.

Fix a security parameter $\eta \in \mathbb{N}$. A concrete trace is a sequence of *concrete global states* of the form $(\mathsf{Sid}, g, C)$ where $\mathsf{Sid}$ is the set of session ids, $g$ maps sessions ids to concrete local states and $C$ is the list of corrupted agents. A *concrete local state* is of the form $(i, \tau, p, (a_1, \ldots, a_k))$. As for the abstract state, $i$ is the role executed by $a_i$, the node $p$ indicates which point of the role tree the agent $a_i$ has reached so far, and $(a_1, \ldots, a_k)$ list the communication partners. The mapping $\tau$ maps variables to bitstrings (instead of terms) that result from parsing incoming messages. The transitions between the global states are invoked by a probabilistic polynomial-time adversary $\mathcal{A}$. The adversary may invoke a $\mathbf{corrupt}(a_1, \ldots, a_l)$ transition (only in the first step) and will then learn the secret keys of the agents $a_1, \ldots, a_l$. Further the ids $a_1, \ldots, a_l$ are stored in the set $C$ in the global state. The adversary may invoke a $\mathbf{new}(i, a_1, \ldots, a_k)$ transition. In this case a new session id $sid$ with concrete local state $(i, \tau, p, (a_1, \ldots, a_k))$ is al-

located where in the mapping $\tau$ the variables $A_j$ and $X^j_{X_i}$ are preinitialized to $a_j$ and fresh nonces, respectively. Finally, the adversary may invoke a $\mathbf{send}(sid, m)$ transition where $m$ is a bitstring. In this case, for each edge leaving the current node $p$, the following is tried: Let $(l, r)$ be the label of that edge. Then the bitstring $m$ is parsed according to the pattern $l$ using the variable substitution $\tau$ (see below). This results in a new substitution $\tau'$ where the variables that were free in $l$ are now assigned bitstrings. Then the pattern $r$ is used with the variable assignments $\tau'$ to construct a new bitstring $m'$ (see below). If both parsing and constructing succeed, this edge is taken, $\tau'$ becomes part of the new local state of the session $sid$, and the adversary gets $m$ as input. If no edge matches, no action is taken.

It is left to explain how a pattern is used to parse or construct a bitstring. For *constructing* bitstrings, we first randomly choose a family of random values $tape^{a,sid} : \mathsf{Rand}_{ag} \to \{0,1\}^\eta$ parametrized over the agent $a$, the session $sid$.[7] Then we define a function $\mathrm{construct}(r, \tau)$ taking a pattern $r$ and a partial mapping $\tau : \mathsf{X} \to \{0,1\}^*$. If, e.g., $r$ is an encryption $r = \{r'\}^R_{\mathsf{ek}(a)}$, the function $\mathrm{construct}(r, \tau)$ recursively invokes $m' := \mathrm{construct}(r', \tau)$ and then encrypts $m'$ using the public key of agent $a$ and using randomness $tape^{a,sid}(R)$ for the encryption algorithm. Pairs and zero-knowledge proofs are handled similarly. If $r = \mathsf{X}$, then $\mathrm{construct}(r, \tau)$ just returns the stored value $\tau(r)$. We give the details of $\mathrm{construct}$ in Definition 8 in Appendix B. At this point, we would only like to comment on the operation of $\mathrm{construct}(\mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b}), \tau)$, i.e., on the construction of zero-knowledge proofs, since it contains several relevant points.

To produce a zero-knowledge proof for witness $r_1, \ldots, r_s; a_1, \ldots, a_n$ and public part $b_1, \ldots, b_m$ (where we assume that $\underline{r}, \underline{a}, \underline{b}$ have already been assigned bitstrings using recursive calls to $\mathrm{construct}$), we first have to construct a circuit $C$ whose satisfiability we will prove in zero-knowledge. For this, let $l_i := |a_i|$. Then by $C := C^{s,n,\underline{l}}_{F,\underline{b}}$ we denote the circuit that expects arguments $a'_1, \ldots, a'_n$ of lengths $l_1, \ldots, l_n$ and arguments $r'_1, \ldots, r'_n$ all of length $\eta$, and then performs the operations described by the ZK-formula $F$ where $\rho_i$ is instantiated with the input $r'_i$, $\alpha_i$ with input $a'_i$, and occurrences of $\beta_i$ are replaced with the *hardcoded* value $b_i$. Details are given in Definition 7 in Appendix A. Then the prover of the zero-knowledge scheme is invoked for the circuit $C$ and for witness $\underline{r}, \underline{a}$ (as bitstrings) using randomness $tape^{a,sid}(R)$. Call the resulting proof $z$. Then the bitstring returned by $\mathrm{construct}(\mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b}), \tau)$ is the tuple $(z, F, s, n, \underline{l}, \underline{b})$ with appropriate tagging to mark it as a zero-knowledge proof. Note that this construction does not completely hide all information on the witness since it leaks the length of the individual components. This is comparable to the situation with encryption schemes which also cannot completely hide the length of the plaintext.[8] If the zero-knowledge proof fails (because $\underline{r}; \underline{a}$ is not a witness for $C$) the function $\mathrm{construct}$ aborts (and the next edge in the role tree is tried). Note that the circuit $C = C^{s,n,\underline{l}}_{F,\underline{b}}$ can be constructed given only $F, s, n, \underline{l}, \underline{b}$; this is important since for verifying a proof, we need to construct $C$ first.

For *parsing* bitstrings, we define a function $\mathrm{parse}(m, l, \tau)$ taking a bitstring $m$, a pattern $l$, and a partial mapping $\tau : \mathsf{X} \to \{0,1\}^*$. Then if, e.g., $l$ is an encryption pattern of the form $\{l'\}^-_{\mathsf{ek}(A_i)}$ where $i$ is the role executed by agent $a$ in the current session, the bitstring $m$ is decrypted with the secret key of agent $a$ resulting in the plaintext $m'$, and then function $\mathrm{parse}(m', l', \tau)$ is invoked. Pairs and zero-knowledge proofs are handled analogously. When $l$ is a free variable (i.e., unassigned in $\tau$), it is checked whether $m$ is of the right type and then assigned to $\tau(l)$ (resulting in an extended mapping $\tau$). If $l$ is a bound variable (assigned in $\tau$), it is checked whether $m = \tau(l)$. If $l$ is of the form $\{\cdot\}^R_\cdot$ or $\mathsf{ZK}^R_\cdot(\ldots)$ (i.e., contains explicit randomness), the message $m$ is not parsed further but compared to $\mathrm{construct}(l, \tau)$ (this enables matching against encryptions or ciphertexts an agent produced itself). Finally, if all checks succeeded, the (now possibly extended) mapping $\tau$ is returned. Details are postponed to Definition 9 in Appendix B.

We assume explicit type information on each bitstring. We achieve this by requiring that every bitstring carries a type tag distinguishing between agents, nonces, pairs, ciphertexts and zero-knowledge proofs. Furthermore, we require that a bitstring tagged as a zero-knowledge proof is only considered to be of type zero-knowledge proof if it additionally passes the verification. This is necessary since otherwise a bitstring could be assigned to a variable $\mathsf{X}.z$ that later will not pass verification, in contrast to the abstract case where only true proofs can be assigned to $\mathsf{X}.z$.

Thus for any adversary $\mathcal{A}$ and any security parameter $\eta$, we get a distribution on computational traces which we denote by $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$. A detailed description of the concrete execution is given in Definition 10 in Appendix B.

## 5 Computational Soundness

In the preceding two sections, we have described the abstract and the concrete execution model involving zero-knowledge proofs and encryptions. To be able to state

[7] For notational simplicity, we assume that any operation, be it encrypting or performing zero-knowledge proofs, needs at most $\eta$ bits. This can be easily achieved by using a pseudorandom generator if the operation needs more randomness.

[8] Note however that in the case of zero-knowledge proofs, this is not a principal impossibility. For example, [BG02] present so-called universal arguments that can be transformed into length-hiding zero-knowledge proofs. These schemes however are very complex and far from being practically usable.

our main computational soundness result, we have to formalize the statement that a given concrete trace $t^c$ corresponds to a given abstract trace $t^s$. Here we follow [CW05, MW04, CKKW06] and require that there exists a mapping $c$ that maps every message from $t^s$ to a bitstring of $t^c$ in a consistent fashion. The exact definition is almost identical to the one of [CKKW06], except that we add the requirement that the adversary corrupts the same agents in the abstract and the concrete trace.

**Definition 4** (Concrete instantiations)**.** *Let* $t^s = (\mathsf{Sid}^s_1, f_1, \varphi_1), \ldots, (\mathsf{Sid}^s_m, f_m, \varphi_m)$ *be a symbolic execution trace and* $t^c = (\mathsf{Sid}^c_1, g_1, C_1), \ldots, (\mathsf{Sid}^c_n, g_n, C_n)$ *a concrete execution trace.*

*We say that the trace* $t^c$ *is a* concrete instantiation *of* $t^s$ *with partial mapping* $c : \mathsf{M} \to \{0,1\}^*$ *(written* $t^s \preceq^c t^c$*) if* $m = n$ *and for every* $\ell \in [n]$ *it holds that* $\mathsf{Sid}^s_\ell = \mathsf{Sid}^c_\ell$*, and* $C_\ell = \{a : \mathsf{dk}(a) \in \varphi_\ell\}$*, and for every* $sid \in \mathsf{Sid}^s_\ell$ *the following holds:*

*For* $(\sigma, i, p, (a_1, \ldots, a_k)) := f_\ell(sid)$ *and* $(\tau, j, q, (b_1, \ldots, b_n)) := g_\ell(sid)$ *we have that* $\tau = c \circ \sigma$*, and* $i = j$*, and* $p = q$*, and* $(a_1, \ldots, a_n) = (b_1, \ldots, b_n)$*.*

*We say that* $t^c$ *is a* concrete instantiation *of* $t^s$ *(written* $t^s \preceq t^c$*) if there exists a partial injective function* $c : \mathsf{M} \to \{0,1\}^*$ *such that* $t^s \preceq^c t^c$*.*

Equipped with this definition, we can formulate our soundness result. Namely, with overwhelming probability, a concrete trace is a concrete instantiation of some abstract Dolev-Yao trace.

**Theorem 1** (Computational soundness of zero-knowledge proofs)**.** *Let* $\Pi$ *be a* $k$*-party protocol. Assume that* $\mathcal{AE}$ *is an* IND-CCA *secure encryption scheme and that* $\mathcal{ZK}$ *is a symbolically-sound zero-knowledge proof system. Let* $\mathcal{A}$ *be a nonuniform polynomial-time adversary. Then the following probability is overwhelming in* $\eta$*:*

$$\Pr\big[\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta) \in \{t^c : \exists t^s \in \mathsf{Exec}^s(\Pi) \text{ such that } t^s \preceq t^c\}\big].$$

*Proof sketch.* (The full proof is postponed to Appendix C.) To establish the theorem, it is sufficient to find an injective mapping $\bar{c}$ that maps bitstrings to terms such that a concrete trace $t^c$ (chosen according to $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$) will be mapped to a Dolev-Yao trace $\bar{c}(t^c)$. Then the inverse $\bar{c}^{-1}$ satisfies $\bar{c}(t^c) \preceq^{\bar{c}^{-1}} t^c$, which proves the theorem. The mapping $\bar{c}$ is defined in the canonical way, namely by parsing every bitstring $m$ to a term. To this aim, we use the decryption keys to parse encryptions, and the extraction trapdoor $E$ of $\mathcal{ZK}$ to recover the witnesses of zero-knowledge proofs. Unparseable bitstrings are mapped to distinct terms in $\mathsf{Garbage}$. A small difficulty occurs when trying to extract the randomness used for encryptions or zero-knowledge proofs. In general, an encryption scheme may not allow to extract the randomness used when decrypting, even given knowledge of the secret key.[9] Moreover, some of the randomness might even be information-theoretically lost, so it is impossible to recover the randomness that is actually used. Thus for adversary-generated bitstrings $m$, we do not aim to extract the randomness but instead consider the full bitstring as its own randomness.

We have to show that $\bar{c}(t^c)$ constitutes a Dolev-Yao trace with overwhelming probability. Assume that $\bar{c}(t^c)$ is not a Dolev-Yao trace. This can be because $\bar{c}(t^c)$ does not fulfill the syntactic conditions of a trace (e.g., the knowledge of the adversary changes in an unexpected way, or the local state of some machine does not correspond to the messages received), or the adversary might send a message that cannot be deduced from the messages that were output by the honest agents. In this proof sketch, we will only consider the latter case. We will therefore assume that with non-negligible probability, in step $\ell$, a message $m$ is sent such that

$$\mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a) : a \in C\} \cup \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\} \nvdash \bar{c}(m) \tag{1}$$

holds, where $C$ denotes the set of corrupted agents. From this we will derive a contradiction to the cryptographic assumptions used in the theorem by transforming the concrete execution in several steps into an adversary against the IND-CCA assumption.

**Simulating the zero-knowledge proofs.** As a first step towards a contradiction, we will replace all zero-knowledge proofs by fake proofs produced by the simulator. For this, we first introduce two oracles into our execution: A proof oracle $Proof$ and an extraction oracle $Extract$. Whenever an honest agent wants to produce a zero-knowledge proof of some statement $x$ with witness $w$, it invokes $Proof(x, w)$; when the implementation of $\bar{c}$ extracts the witness of some zero-knowledge proof $z$, it invokes $Extract(z)$. Note that for this, it must be guaranteed that each zero-knowledge proof produced by honest agents uses a different randomness $R$,[10] and that this randomness is only used for the zero-knowledge proof. By the definition of valid roles, we have that for any randomness $R$, there is at most one effective $R$-subpattern in any path of the role tree of any agent. If this effective $R$-subpattern is a term of the form $\mathsf{ZK}^R_F(\ldots)$, then $R$ does not appear in the witness of any zero-knowledge proof since terms of the form $\mathsf{ZK}^R_F(\ldots)$ may not appear in ZK-formulas. Thus any randomness $R$ that is used for some ZK-proof is used only for that proof (if the proof is performed several times with the same witness, statement and randomness, the $Proof$ oracle will not be invoked again but the old result will be reused). Note the following facts:

[9] E.g., the Cramer-Shoup cryptosystem, cf. footnote 5.

[10] Here and in the following, when we reason about a randomness $R \in \mathsf{Rand}_{ag}$ in the concrete model, we mean the symbolic value $R$ that is used to select the corresponding bitstring from the random tape using the function $tape$.

- The oracle $Extract$ is never invoked with a proof $z$ that has previously been output by $Proof$. This holds since $\bar{c}$ by definition only extracts proofs that have not been generated by an honest agent, and only honest agents use $Proof$.
- The oracle $Proof$ is never invoked with $(x, w)$ such that $w$ is not a witness of $x$. This holds since honest agents check whether $w$ is a witness before constructing a proof.

Hence, since both the execution of the concrete trace, as well as the application of the mapping $\bar{c}$ run in polynomial-time, we can exploit that $\mathcal{ZK}$ has the extraction zero-knowledge property, and hence replace the $Proof$ oracle by a simulation oracle $Simulate$ using the simulation trapdoor of the CRS such that $\bar{c}(t^c)$ (which is the output of an *efficient* function $\bar{c}$) is computationally indistinguishable in both cases.[11] Whether a given abstract trace is a Dolev-Yao trace can checked in polynomial-time. Thus from the computational indistinguishability of the abstract traces in both cases, it follows that the probability that the abstract trace is a Dolev-Yao trace changes only by a negligible amount when replacing $Proof$ by $Simulate$. Thus (1) still holds with non-negligible probability. Moreover, in contrast to $Proof$, the oracle $Simulate$ only expects the statement $x$ as input, but no witness.

**Using fake encryptions.** The next step towards deriving a contradiction is to replace the encryptions created by honest agents by fake encryptions. Since this step is very similar to the introduction of the oracles $Simulate$ and $Extract$, we only give a rough idea. All encryptions and decryptions performed by honest agents (with respect to public keys of uncorrupted agents) are replaced by calls to an encryption or decryption oracle. By performing a lookup in the list of all encryptions produced so far, we can ensure that the decryption oracle is only invoked for ciphertexts not produced by the encryption oracle. Then the IND-CCA property guarantees that we can replace the encryption oracle by an oracle $FakeEncrypt$ that encryptions random messages (and thus is independent of its input). Some care has to be taken concerning the randomness: We do not guarantee that the randomness used by the encryption oracle is used exactly once, but instead may also use it in the witnesses of zero-knowledge proofs. However, exploiting that $Simulate$ does not need a witness, one can show that the replacement of the encryption by $FakeEncrypt$ leads to an indistinguishable trace. We refer to the full proof for details.

**Identifying the underivable subterm.** In order to derive a contradiction from (1), we have to identify the subterm of $\bar{c}(m)$ whose "fault" it is that $\bar{c}(m)$ cannot be derived. We will then use this term to construct an attack against the IND-CCA. For this, we need the following characterization of underivable messages:

**Lemma 1.** *Let $C$ be the set of corrupted agents, let $M := \bar{c}(m)$, let $S$ be the set of messages output by honest agents up to step $\ell$, and let $S' := S \cup \{\mathsf{dk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv}$ (the knowledge of the adversary after that step).*

*Then there exists a term $T \in \mathsf{M}$ and a context $D$ such that $M = D[T]$ and all terms on the path from $M = D[T]$ to $T$ (not including $T$) are of the form*

$$\langle \cdot, \cdot \rangle \quad or \quad \{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)} \quad or \quad \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$$

*Furthermore, we have that $S' \not\vdash T$ and that $T$ satisfies one of the following conditions: (a) $T \in \mathsf{Nonce}_{ag}$, or (b) $T = \{\cdot\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(a)}$, or (c) $T = \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\dots)$, or (d) $T = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ and for some $i$, $r_i$ is not known.*

Thus by (1) such a subterm $T$ of $M = \bar{c}(m)$ exists. We have to show that each of the four cases leads to a contradiction.

***T* is a nonce.** In case (a) we have $T \in \mathsf{Nonce}_{ag}$. Since $S' \not\vdash T$, for any message $m$ sent to the adversary, the nonce $T$ occurs in $\bar{c}(m)$ only inside an encryption (with a public key $\mathsf{ek}(a)$ with $a \notin C$) or inside the witness of a zero-knowledge proof. Since honest agents construct such encryptions and zero-knowledge proofs using the oracles $Simulate$ and $FakeEncrypt$, the message $m$ is computed without using the bitstring corresponding to $T$; thus it is not possible to extract that bitstring from $m$. On the other hand, from the message $m$ sent by the adversary, we can retrieve the nonce as follows. In $M = \bar{c}(m)$, the nonce $T$ is protected only by terms of the form $\langle \cdot, \cdot \rangle$, $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$ or $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$. The pair can directly be parsed, in the case of $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$ or $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$, we can call the oracles $Decrypt$ and $Extract$, respectively. Since these oracles are also used by the function $\bar{c}$ (at least for terms where $\bar{c}$ assigns randomness $\mathsf{Rand}_{adv}$ and not $\mathsf{Rand}_{ag}$), these oracles will answer consistently with the parsing $M = \bar{c}(m)$ of $m$. Thus we can guess the nonce $T$, leading to a contradiction. Cases (b) and (c) of Lemma 1 are taken care of similarly.

***T* is an adversary-generated zero-knowledge proof.** In case (d), we have that $T = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ and that $r_i$ is not known (in the sense of Lemma 1). In this case the argumentation used for case (a) cannot be used because $T$ does not correspond to a bitstring generated by an honest agent. However, as in the preceding argument, the adversary can extract the bitstring corresponding to $T$, and using the oracle $Extract$ it can extract the concrete randomness corresponding to $r_i$. By definition of the function $\bar{c}$, this randomness will be the randomness used in an encryption with respect to some $\mathsf{ek}(a)$ performed by an honest agent (otherwise the function $\bar{c}$ would have assigned a randomness $r_i \in \mathsf{Rand}_{adv}$). We distinguish two cases: $a \notin C$ and

[11] Here we really need extraction zero-knowledge and not only the extraction oracle $Extract$ is used.

$a \in C$. If $a \notin C$, then the encryption has been generated using the encryption oracle $Encrypt$. Being able to retrieve the randomness used in that encryption contradicts the IND-CCA property of $\mathcal{AE}$. If $a \in C$, then the randomness has been used to generate the bitstring corresponding to a term $c = \{t\}_{r_i}^{\mathsf{Rand}_{ag}}$ with $a \in C$. Since $r_i$ is not known, we have that $S' \nvdash c$. With an analogous argument as above, we can see that all bitstrings sent by honest agents can be computed without actually computing the bitstring corresponding to $c$. But in this case, that fact that the adversary is able to guess the randomness used to produce $c$ is a contradiction. □

# A The Abstract Model – Postponed Definitions

In the following, let $A_i \in \mathsf{X}.a$ be pairwise distinct agent variables (for $i \in \mathbb{N}$), and let $X^j_{A_i} \in \mathsf{X}.n$ be pairwise distinct nonce variables (for $i, j \in \mathbb{N}$). Assume that $\mathsf{X}.a \setminus \{A_i : i \in \mathbb{N}\}$ and $\mathsf{X}.n \setminus \{X^j_{A_i}\}$ are infinite.

By $n^{a,j,s} \in \mathsf{Nonce}_{ag}$ with $a \in \mathsf{A}$ and $j, s \in \mathbb{N}$ we denote distinct nonces. By $r^{a,j,s} \in \mathsf{Rand}_{ag}$ with $a \in A, j \in \mathsf{Rand}_{ag}, s \in \mathbb{N}$ we denote distinct symbolic randomnesses. By $\gamma_{a,s}$ we denote a mapping that maps every $r \in \mathsf{Rand}_{ag}$ to $r^{a,r,s}$.

For the following definition, we use the following notation: For an edge $p \xrightarrow{l,r} q$ in a labeled tree, the free variables of $(l, r)$ are the variables that occur in $l$ or $r$ but are not in the label of any edge on the path from the root to $p$ nor are in $\{A_j : j \in [k]\} \cup \{X^j_{A_i} : j \in \mathbb{N}\}$ (where the number $k$ of parties and the role-index $i$ will be clear from the context below).

**Definition 5** (Role)**.** *A* role tree $R$ *is an ordered edge-labeled finite tree where each edge is labeled by an agent rule* $(l, r)$ *where* $l, r \in \mathsf{Pat}$.

*A* role for agent $i$ in a $k$-party protocol *is a role tree* $R$ *satisfying the following conditions for each node* $p$ *of* $R$*:*

1. *For every* $r \in \mathsf{Rand}_{ag}$*, there is at most one effective* $r$*-subpattern in the labels of the path to* $p$ *(but that effective* $r$*-subpattern may occur several times).*
2. *For any subterm of* $l$ *of the form* $\{t\}_{\bar{t}'}$ *it holds that* $t' = \mathsf{ek}(A_i)$.
3. *For any subterm of* $l$ *of the form* $\{t\}_{t'}^{\mathsf{Rand}}$ *it holds that* $t$ *and* $t'$ *do not contain free variables.*
4. *For any subterm of* $l$ *of the form* $\mathsf{ZK}^{\mathsf{Rand}}_{\mathsf{Formula}}(\mathsf{Rand}^*; \underline{a}; \underline{b})$ *it holds that* $\underline{a}$ *and* $\underline{b}$ *do not contain free variables.*
5. $r$ *does not contain* _ *nor free variables that are not free in* $l$.
6. $l$ *and* $r$ *do not contain subterms that are in* $\mathsf{Rand} \setminus \mathsf{Rand}_{ag}$.

**Definition 6** (Formal execution)**.** *Let* $k$*-party protocol* $\Pi$ *be given.*

*A* global state *is a triple* $(\mathsf{Sid}, f, \varphi)$ *where* $\varphi$ *is a set of messages (the* adversary knowledge*),* $\mathsf{Sid}$ *is a finite set of session ids, and the function* $f$ *maps every session id* $sid$ *in* $\mathsf{Sid}$ *to the current state of the session* $sid$*. This state is called the* local state *and is of the form* $(i, \sigma, p, (a_1, \ldots, a_k))$ *where* $i \in [k]$ *is the index of the role executed in this session, the partial function* $\sigma : \mathsf{X} \to \mathsf{M}$ *is a substitution mapping variables of the agent to messages,* $p$ *is a node in the role tree* $\Pi(i)$*, and* $a_j \in \mathsf{A}$ *is the agent identifier assigned to role* $j$ *in this session (thus* $a_i$ *is the agent carrying out this session).*

*The initial state is* $q_I = (\varnothing, \varnothing, \mathsf{Nonce}_{adv})$.

*We allow three kinds of transitions between global states.*

- Corruption. *The adversary corrupts a subset of the parties involved in the protocol and learns their private keys. This transition can only be applied in the beginning.*

$$q_I \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\varnothing, \varnothing, \mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a_j) : j \in [l]\}).$$

- Session initialization. *The adversary can initialize new sessions.*

$$(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{Sid}', f', \varphi).$$

  *Here* $sid := |\mathsf{Sid}|+1$ *is the identifier of the new session and* $\mathsf{Sid}' := \mathsf{Sid} \cup \{sid\}$*. The function* $f'$ *is defined as* $f'(sid') = f(sid')$ *for* $sid' \in \mathsf{Sid}$ *and* $f'(sid) = (i, \sigma, \varepsilon, (a_1, \ldots, a_k))$*. Here* $\varepsilon$ *is the root of the role tree* $\Pi(i)$ *and the substitution* $\sigma$ *is defined by* $\sigma(A_j) := a_j$ *for all* $j \in [k]$ *and* $\sigma(X^j_{A_i}) := n^{a_i,j,s}$ *for every* $X^j_{A_i}$ *occurring in* $\Pi(i)$.

- Sending of messages. *The adversary can send messages to agents.*

$$(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{send}(sid,m)} (\mathsf{Sid}, f', \varphi').$$

  *Here we require* $sid \in \mathsf{Sid}$, $m \in \mathsf{M}$, $\varphi \vdash m$*, and* $\varphi'$ *and* $f'$ *are defined as follows. We define* $f'(sid') = f(sid')$ *for every* $sid' \neq sid$*. Let* $(i, \sigma, p, (a_1, \ldots, a_k)) := f(sid)$*. Then let* $(l, r)$ *be the label of the first outgoing edge from* $p$ *such that the following holds:*
  - *The message* $m$ *matches the pattern* $l\gamma_{a_i,sid}\sigma$*. Let* $\theta$ *denote the matcher.*
  - *Let* $\tilde{m} := r\gamma_{a_i,sid}\sigma\theta$*. Then* $\tilde{m}$ *contains true proofs.*

  *If no such edge exists, let* $f'(sid) = f(sid)$ *and* $\varphi' = \varphi$*. Otherwise, let* $f'(sid) = (i, \sigma \cup \theta, p', (a_1, \ldots, a_k))$ *where* $p'$ *is the successor of* $p$ *along that edge, and let* $\varphi' := \varphi \cup \{\tilde{m}\}$.

*A finite sequence of global states starting with* $q_I$ *with the above transitions is called a* symbolic execution trace *or* Dolev-Yao trace *for* $\Pi$*. The set of Dolev-Yao traces for* $\Pi$ *is denoted* $\mathsf{Exec}^s(\Pi)$.

## B The Concrete Model – Postponed Definitions

For the following definitions, we assume that $tape^{a,sid}(R) \in \{0,1\}^\eta$ are uniformly and independently chosen for each $a \in \mathsf{A}$, $sid \in \mathbb{N}$, and $R \in \mathsf{Rand}_{ag}$. In an implementation, these values would, of course, be sampled upon first use. Similarly, we assume that $crs$ is chosen according to the CRS-generation algorithm $K$ of $\mathcal{ZK}$.

For convenience, we will identify elements of $\mathsf{A}$ with the bitstrings encoding them.

**Definition 7** (Circuits for ZK-formulas)**.** *Fix a security parameter $\eta$. Let a ZK-formula $F$ of $\rho$-arity $s$, $\alpha$-arity $n$ and $\beta$-arity $m$ be given, as well as bitstrings $b_1, \ldots, b_m \in \{0,1\}^*$. Let $l_1, \ldots, l_n \in \mathbb{N}$.*

*For a ZK-term $T$, the circuit $C = C^{s,n,\underline{l}}_{T,\underline{b}}$ is recursively defined as follows:*

- *It expects inputs $r_1, \ldots, r_s$ of length $\eta$, and inputs $a_1, \ldots, a_n$ of lengths $l_1, \ldots, l_n$, respectively.*
- *If $s$ is not the $\rho$-arity of $T$, or $n$ is not the $\alpha$-arity of $T$, or $|\underline{l}| \neq n$, or $|\underline{b}|$ is not the $\beta$-arity of $T$, then $C$ is undefined.*
- *If $T = \mathsf{ek}(\beta_i)$ or $T = \{\cdot\}^{\cdot}_{\mathsf{ek}(\beta_i)}$ with $b_i \notin \mathsf{A}$, then $C$ is undefined.*
- *If $T = \alpha_i$, then $C$ computes $a_i$ (i.e., $C$ is a projection).*
- *If $T = \beta_i$, then $C$ returns $b_i$ (i.e., $C$ computes a constant function).*
- *If $T = \mathsf{ek}(\beta_i)$ and $b_i \in \mathsf{A}$, then $C$ returns the public key of agent $b_i$ (i.e., $C$ computes a constant function).*
- *If $T = \langle T_1, T_2 \rangle$, then $C$ computes $m_1 := C^{s,n,\underline{l}}_{T_1,\underline{b}}$ and $m_2 := C^{s,n,\underline{l}}_{T_2,\underline{b}}$, and then returns the bitstring $\langle m_1, m_2 \rangle$.*
- *If $T = \{T'\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$ and $b_i \in \mathsf{A}$, then $C$ computes $m' := C^{s,n,\underline{l}}_{T',\underline{b}}$ and then returns the encryption of $m'$ under the public key of $b_i$ using randomness $r_i$.*

*The circuit $C^{s,n,\underline{l}}_{T_1=T_2,\underline{b}}$ expects inputs $r_1, \ldots, r_s$ of length $\eta$, and inputs $a_1, \ldots, a_n$ of lengths $l_1, \ldots, l_n$, respectively. It computes $m_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}$ for $i = 1, 2$ and returns $1$ if $m_1 = m_2$ and $0$ otherwise.*

*For a ZK-formula $F = B(T_1, \ldots, T_q)$ where $B$ is a Boolean predicate and $\underline{T}$ are of the form $\mathsf{ZKTerm} = \mathsf{ZKTerm}$, the circuit $C^{s,n,\underline{l}}_{F,\underline{b}}$ expects inputs $r_1, \ldots, r_s$ of length $\eta$, and inputs $a_1, \ldots, a_n$ of lengths $l_1, \ldots, l_n$, respectively. It computes $t_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}$ for $i \in [q]$ and returns $B(\underline{t})$.*

**Types of bitstrings** We assume that pairs, agent ids, ciphertexts, zero-knowledge proofs, and nonces in the concrete model all come for different efficiently recognizable sets, so that it is meaningful to speak about bitstrings of type pair, agent id, ciphertext, zero-knowledge proof, or nonce. One possibility to achieve this is to add type tags to the bitstrings. We assume that encryptions are additionally tagged with their public key. In the case of the type zero-knowledge proofs, we impose an additional condition. A bitstring of type zero-knowledge proof is a tuple $(z, F, s, n, \underline{l}, \underline{b})$ satisfying that circuit $C := C^{s,n,\underline{l}}_{F,\underline{b}}$ is defined and the verification algorithm of $\mathcal{ZK}$ accepts the proof $z$ for the circuit $C$. This restriction on the type zero-knowledge proof has the effect that only valid proofs can be assigned to variables $\mathsf{X}.z$.

**Definition 8** (Constructing bitstrings)**.** *Let a session id $sid$, an agent $a \in \mathsf{A}$, a pattern $r$, and a mapping $\tau$ from variables to bitstrings be given. We define $\mathrm{construct}^{a,sid}(r, \tau)$ recursively as follows:*

*Case $r = x \in \mathsf{X}$: If $\tau(x)$ is defined, return $\tau(x)$. Otherwise, abort.*

*Case $r = \mathsf{ek}(x)$: If $a := \tau(x)$ is not defined, abort. Otherwise, retrieve the public key $pk$ belonging to agent id $a$. If no such public key exists, abort. otherwise return $pk$.*

*Case $r = \langle r_1, r_2 \rangle$: Set $m_i := \mathrm{construct}^{a,sid}(r_i, \tau)$ for $i = 1, 2$. If one of the invocations aborts, abort. Otherwise return the pair $\langle m_1, m_2 \rangle$.*

*Case $r = \{r'\}^R_{\mathsf{ek}(x)}$: Let $\tilde{R} := tape^{a,sid}(R)$, let $a := \tau(x)$, and let $pk$ be the public key of agent $a$. If $a$ or $pk$ is undefined, abort. Otherwise invoke $m' := \mathrm{construct}^{a,sid}(r', \tau)$. If it aborts, abort. Otherwise compute the encryption $m$ of $m'$ under public key $pk$ using randomness $\tilde{R}$. Return $m$.*

*Case $r = \mathsf{ZK}^R_F(R_1, \ldots, R_l; a_1, \ldots, a_n; b_1, \ldots, b_s)$: Let $\tilde{R} := tape^{a,sid}(R)$ and $\tilde{R}_i := tape^{a,sid}(R_i)$ for $i = [l]$. If one of these is undefined, abort. For all $i \in [n]$, compute $\tilde{a}_i := \mathrm{construct}^{a,sid}(a_i, \tau)$, and for all $i \in [s]$, compute $\tilde{b}_i := \mathrm{construct}^{a,sid}(b_i, \tau)$. If one of these invocations aborts, abort. Otherwise set $l_i := |\tilde{a}_i|$ and let $C := C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$. Set $w := (\underline{\tilde{R}}, \underline{\tilde{a}})$. Use the prover $P$ of $\mathcal{ZK}$ to produce a proof $z$ for circuit $C$ and witness $w$ where the prover uses randomness $\tilde{R}$. If this fails (e.g., because the witness does not fulfill the statement), abort. Otherwise return $z$.*

*In any other case, abort.*

**Definition 9** (Parsing bitstrings)**.** *Let a bitstring $m$, a pattern $l$, and a mapping $\tau$ from variables to bitstrings be given. We define $\mathrm{parse}^{a,sid}(m, l, \tau)$ recursively as follows:*

*Case $l = x \in X$: If the type of $m$ does not match the type of $x$,[12] abort. If $\tau(x) = m$, then return $\tau$. If $\tau(x)$ is defined, but $\tau(x) \neq m$, abort. Otherwise return $\tau[x := m]$.*

*Case $l = \mathsf{ek}(x) \in \mathsf{ek}(X.a)$: If $m$ is not of type ciphertext, abort. Otherwise, if there is an agent id $a$ such that $m$ is the public key of $a$ and $\tau(x)$ is not defined or equals $a$, return $\tau(x)[x := a]$. Otherwise abort.*

*Case $l = \langle l_1, l_2 \rangle$: If $m$ is not a pair, abort. Otherwise parse $m$ as $\langle m_1, m_2 \rangle$ and let $\tau' := \mathrm{parse}^{a,sid}(m_1, l_1, \tau)$*

[12] If $\mathsf{X}.z$ this implies invoking the verification algorithm of $\mathcal{ZK}$ on $m$.

*and $\tau'' := (m_2, l_2, \tau')$. If one of the invocations aborts, abort. Otherwise return $\tau''$.*

*Case $l \in \{\mathsf{Pat}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{X}.a)}$ or $l \in \mathsf{ZK}^{\mathsf{Rand}}_{\mathsf{Formula}}(\mathsf{Rand}^*; \mathsf{Pat}^*; \mathsf{Pat}^*)$: Invoke $m' := \mathrm{construct}^{a,sid}(l, \tau)$. If $m \neq m'$, abort. Otherwise, return $\tau$.*

*Case $l = \{l'\}^{-}_{\mathsf{ek}(A_j)}$ with $j \in [k]$: If $m$ is not an encryption, abort. Otherwise, extract the public key $pk$ from $m$. If $pk$ is not the public key belonging to the agent id $\tau(A_j)$, abort. Otherwise decrypt $m$ with the secret key belonging to $a := \tau(A_j)$ and let $m'$ be the corresponding ciphertext.*[13] *If this fails, abort. Otherwise let $\tau' := \mathrm{parse}^{a,sid}(m', l', \tau)$ and return $\tau'$ (or abort if the invocation aborts).*

*Case $l \in \mathsf{ZK}^{-}_{F}(\_^*; \_^*; m_1, \ldots, m_n)$: If $m$ is not of type zero-knowledge proof, abort.*[14] *Otherwise, $m =: (z, F', s, n, \underline{l}, \underline{b})$. If $F \neq F'$, abort. Otherwise for $i = 1, \ldots, n$ run the following: $\tau_i := \mathrm{parse}^{a,sid}(b_i, l_i, \tau_{i-1})$ with $\tau_0 := \tau$. If one of the invocations aborts, abort. Otherwise return $\tau_n$.*

*In any other case, abort.*

**Definition 10** (Concrete execution model)**.** *A* concrete global state *is a triple $(\mathsf{Sid}, g, C)$ where $\mathsf{Sid}$ is a finite set of session-ids, and $g$ is a function mapping every $sid \in \mathsf{Sid}$ to a concrete local state, and $C$ is the set of corrupted parties.*

*A* concrete local state *is of the form $(i, \tau, p, (a_1, \ldots, a_k))$ where $i \in [k]$ is the index of the role executed in this session, the partial function $\sigma : \mathsf{X} \to \{0,1\}^*$ is a substitution mapping variables to bitstrings, $p$ is a node in $\Pi(i)$, and $a_j \in \mathsf{A}$ is the agent identifier assigned to role $j$ in this session.*

*Let a probabilistic interactive Turing machine $\mathcal{A}$ be given. The* concrete trace $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$ *for security parameter $\eta$ is a (distribution over) sequences of global states given by the following algorithm.*

- *Whenever a public or secret key of some agent $a \in \mathsf{A}$ is used for the first time, the key pair is generated using the key generation algorithm of $\mathcal{AE}$. We further give $\mathcal{A}$ access to the public keys of all agents.*
- *The initial global state is $(\varnothing, \varnothing, \varnothing)$. In the first step, $\mathcal{A}$ is invoked with input $1^\eta$.*
- *When $\mathcal{A}$ outputs $\mathrm{corrupt}(a_1, \ldots, a_l)$ with $a_1, \ldots, a_n \in \mathsf{A}$ in its first activation, the adversary is given the secrets keys of $a_1, \ldots, a_n$ as input. The next global state is $(\varnothing, \varnothing, \{a_1, \ldots, a_l\})$.*
- *When $\mathcal{A}$ outputs $\mathrm{new}(i, a_1, \ldots, a_k)$ in global state $(\mathsf{Sid}, g, C)$ where $i \in [k]$ and $a_1, \ldots, a_k \in \mathsf{A}$, the next global state is $(\mathsf{Sid}', g', C)$.*
  *Here $sid := |\mathsf{Sid}| + 1$ is the identifier of the new session and $\mathsf{Sid}' := \mathsf{Sid} \cup \{sid\}$. The function $g'$ is defined as $g'(sid') = g(sid')$ for $sid' \in \mathsf{Sid}$ and $g'(sid) = (i, \tau, \varepsilon, (a_1, \ldots, a_k))$. Here $\varepsilon$ is the root of the role tree $\Pi(i)$ and the substitution $\tau$ is defined by $\sigma(A_j) := a_j$ for all $j \in [k]$ and $\sigma(X^j_{A_i})$ is initialized with a random $\eta$-bit nonce for every $X^j_{A_i}$ occurring in $\Pi(i)$. The adversary is given empty input.*
- *When $\mathcal{A}$ outputs $\mathrm{send}(sid, m)$ in global state $(\mathsf{Sid}, g, C)$ where $sid \in \mathsf{Sid}$, the next global state is $(\mathsf{Sid}, g', C)$.*
  *Here $g'(sid') := g(sid')$ for all $sid \neq sid'$, $g'(sid) := (\tau', i, p', (a_1, \ldots, a_k))$ is computed from $(\tau, i, p, (a_1, \ldots, a_k)) := g(sid)$ as follows:*
  *For each edge $p \xrightarrow{l,r} p''$ starting in $p$ (in their natural order, remember that the role tree $\Pi(i)$ has ordered edges), first invoke $\tau'' := \mathrm{parse}^{a_i,sid}(m, l, \tau)$. If this fails, continue with the next edge. Then invoke $m' := \mathrm{construct}^{a_i,sid}(r, \tau'')$. If this fails, continue with the next edge. Otherwise set $\tau' := \tau''$ and $p' := p''$, let the next input of $\mathcal{A}$ be $m'$, and do not proceed with the next edges.*
  *If no edge lead to a definition of $\tau', p'$ and an output for $\mathcal{A}$, set $\tau' := \tau$ and $p' := p$ and let the next input of $\mathcal{A}$ be the empty string.*
- *When the adversary outputs anything else, the execution terminates and the concrete trace ends at this point.*

[13] Note that due to the definition of roles, this will only need to be done by agent $a$ itself.

[14] By our definition of the type zero-knowledge proof, this implies invoking the verification algorithm of $\mathcal{ZK}$.

# C Proof of Theorem 1

*Proof.* In the following, let a $k$-party protocol $\Pi$ and a polynomial-time adversary $\mathcal{A}$ be fixed. We have to show that

$$\Pr\big[\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta) \in \{t^c : \exists t^s \in \mathsf{Exec}^s(\Pi) \text{ such that } t^s \preceq t^c\}\big].$$

is overwhelming in $\eta$.

For this, we will give a construction of an injective mapping $\bar{c} : \{0,1\}^* \to \mathsf{M}$ that may depend on $\eta$ as well as of some of the values occurring in $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$ such as the CRS, its extraction trapdoor, or the private keys of the parties. We extend this mapping to concrete traces as follows: A concrete execution trace $t^c = (\mathsf{Sid}_1, g_1), \ldots, (\mathsf{Sid}_n, g_n)$ is mapped to an abstract trace $\bar{c}(t^c) := (\mathsf{Sid}_1, f_1, \varphi_1), \ldots, (\mathsf{Sid}_n, f_n, \varphi_n)$ as follows. We let $f_\ell(sid) = (c \circ \tau, i, p, (a_1, \ldots, a_k))$ where $(\tau, i, p, (a_1, \ldots, a_k)) := g_\ell(sid)$, and we define the adversary's knowledge $\varphi_\ell$ as follows: We set $\varphi_1 := \mathsf{Nonce}_{adv}$. For $\varphi_{\ell+1}$ we distinguish three cases. If the $\ell$-th transition was a $\mathbf{new}$-transition, we set $\varphi_{\ell+1} := \varphi_\ell$. If the $\ell$-th transition was a $\mathbf{corrupt}(a_1, \ldots, a_l)$-transition, we set $\varphi_{\ell+1} := \varphi_\ell \cup \{\mathsf{dk}(a_j) : j \in [l]\}$. If the $\ell$-th transition was a $\mathbf{send}(sid, m)$-transition, let $\tilde{m}$ be defined as in the corresponding part of Definition 6 and set $\varphi_{\ell+1} := \varphi_\ell \cup \{\tilde{m}\}$ if $\tilde{m}$ is defined and $\varphi_{\ell+1} := \varphi_\ell$ otherwise.

We will then show that if $t^c$ is chosen according to the distribution $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$, with some overwhelming probability $1-\mu(\eta)$ we have that $\bar{c}(t^s)$ is a Dolev-Yao trace.

Since $\bar{c}$ is injective, we have that $\bar{c}^{-1}$ is an injective partial function, and by construction of $\bar{c}(t^c)$ we have that $\bar{c}(t^c) \preceq^{\bar{c}^{-1}} t^c$. Thus, assuming that $\bar{c}(t^s)$ is a Dolev-Yao trace with probability $1-\mu$, we have

$$\Pr\big[\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta) \in \{t^c : \exists t^s \in \mathsf{Exec}^s(\Pi) \text{ such that } t^s \preceq t^c\}\big] \geq 1-\mu(\eta).$$

Thus in the remainder of this proof, let $t^c$ be distributed according to $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$. We will construct $\bar{c}$ and show that $\bar{c}(t^c)$ is a Dolev-Yao trace with overwhelming probability which then establishes the theorem.

**The mapping $\bar{c}$.** The mapping $\bar{c}$ works by parsing any message $m \in \{0,1\}^*$ in a manner similar to the $\mathrm{parse}$ function from Definition 9. However, in contrast to $\mathrm{parse}$, the mapping $\bar{c}$ has to parse any bitstring and not only terms matching some pattern. In particular, $\bar{c}$ has to extract the witnesses of the zero-knowledge proofs and decrypt all ciphertexts. Moreover, $\bar{c}$ will have to assign symbolic randomness from $\mathsf{Rand}$ to any ciphertext or zero-knowledge proof.

Decrypting the ciphertexts is easy, since we allow $\bar{c}$ to access the secret keys of all agents. This allows to decrypt all ciphertexts that use a public key corresponding to an existent agent. All other ciphertexts may be safely considered as invalid, since no honest party will ever be able to decrypt them.

To extract the witnesses from the zero-knowledge proofs, we use the extractability property of $\mathcal{ZK}$. Using the extraction trapdoor for the CRS, $\bar{c}$ can recover the witness for the proof and use it for further parsing.

However, extracting the randomness is non-trivial. In general, an encryption scheme may not allow to extract the randomness used when decrypting, even given knowledge of the secret key.[15] Moreover, some of the randomness might even be information-theoretically lost, so even an inefficient mapping would not be able to recover that randomness. Similar reasoning applies for the zero-knowledge proofs. Fortunately, it turns out not to be necessary that $\bar{c}$ identifies the actual randomness, but only some value such that different encryptions or proofs of the data will result in different terms. Thus, instead of trying to extract the randomness from a message $m$ generated by the adversary, we interpret the whole message $m$ as its randomness and map $m$ to a symbolic randomness $R^m_{adv}$ (depending on whether $m$ was generated by an honest party or the adversary).

Furthermore, we will define $\bar{c}$ in a way so that it can be efficiently evaluated without decrypting the ciphertexts generated by honest agents or extracting from zero-knowledge proofs generated by honest agents. This can be done because if an honest agent explicitly computed the bitstring, we simply store the inputs of that operation.

For the actual definition of $\bar{c}$, we fix arbitrary (but efficient) injective mappings, $R_{adv} : \{0,1\}^* \to \mathsf{Rand}_{adv}$, $B_{adv} : \{0,1\}^* \to \mathsf{Nonce}_{adv}$, and $G : \{0,1\}^* \to \mathsf{Garbage}$ and write their arguments as superscripts. Here $R^m_{adv}$ denotes the randomness used by the adversary to construct the message $m$. Similarly, $B^m_{adv}$ denotes the nonce $m$ when $m$ is generated by the adversary. And finally, $G^m$ will be used to abstractly represent unparsable bitstrings.

The term $\bar{c}(m) \in \mathsf{M}$ is then recursively defined as follows:

Case "$m \in \mathsf{A}$". Return $m$.

Case "*$m$ is of type public key*". Find $a \in \mathsf{A}$ such that $m$ is the public key for $a$. If no such $a$ exists, return $\mathsf{ek}(G^m)$. Otherwise return $\mathsf{ek}(a)$.

Case "*$m$ is of type pair*". Parse $m$ as $\langle m_1, m_2\rangle$ and return $\langle \bar{c}(m_1), \bar{c}(m_2)\rangle$.

Case "*$m$ is of type nonce*". First check whether $m$ was generated as the value of some variable $X^j_{A_i}$ by some honest agent $a$ in some session $sid$. More exactly, check whether a global state $(\mathsf{Sid}, f, \varphi)$ occurs in the trace where $f(sid) = (i, \sigma, \dots, \dots)$ for some $i$ and $\sigma$, and $\sigma(X^j_{A_i}) = m$ for some $j$. If so, return $n^{a,j,sid}$. (Remember that $n^{a,j,sid}$ is the nonce that is assigned in the abstract model to the nonce variable $X^j_{A_i}$ in session $sid$ run by agent $a$.) Otherwise, return $B^m_{adv}$.

Case "*$m$ is of type ciphertext and $m$ has not been generated by an honest agent*". Extract the public key $pk$ contained in $m$. Let $\mathsf{ek}(a) := \bar{c}(pk)$. If $a \in \mathsf{Garbage}$, return $\{G^m\}^{R^m_{adv}}_{\mathsf{ek}(a)}$. Otherwise, let $sk$ be the secret key of agent $a$ and decrypt $m$ using $sk$ and call the result $m'$. If this fails, return $\{G^m\}^{R^m_{adv}}_{\mathsf{ek}(a)}$. Otherwise, return $\{m'\}^{R^m_{adv}}_{\mathsf{ek}(a)}$.

Case "*$m$ is of type ciphertext and $m$ has been generated by an honest agent*". I.e., $m$ was the result of a call $\mathrm{construct}^{a,sid}(\{t\}^R_{\mathsf{ek}(x)}, \tau)$ by agent $a$ in session $sid$ for some $R \in \mathsf{Rand}_{ag}$, $x \in X.a$, and some mapping $\tau$ from variables to bitstrings. Let $t'$ be the bitstring that was computed by $\mathrm{construct}^{a,sid}(t,\tau)$. Let then $t'' := \bar{c}(t')$ (this gives the same result as applying $\bar{c}$ to all variables in $\tau$ and then computing $t'' := t\tau'$ where $\tau'$ is the mapping resulting from that substitution). Then return $\{t''\}^{r^{a,R,sid}}_{\mathsf{ek}(\tau(x))}$. (Remember that $r^{a,R,sid}$ is the randomness that is actually used in the abstract model when agent $a$ instantiates a pattern with randomness $R$ in session $sid$.)

Case "*$m$ is of type zero-knowledge proof and $m$ has not been generated by an honest agent*". Extract the formula $F$ and the statement $x$ from $m$. Use the

[15] E.g., the Cramer-Shoup cryptosystem, cf. footnote 5.

ZK extraction algorithm and the extraction trapdoor of $\mathcal{ZK}$ (cf. Definition 1) to extract the witness $w$ of $z$. Parse $x$ as a tuple $(b_1, \ldots, b_s)$, and parse $w$ as a tuple $(r_1, \ldots, r_l; a_1, \ldots, a_n)$. Let $b'_i := \bar{c}(b_i)$ and $a'_i := \bar{c}(a_i)$ for all $i$. To compute the abstract randomness $r'_i$ for some $i$, proceed as follows: Let $e$ be an arbitrary subterm of $F$ of the form $e = \{t'\}_t^{r_i}$. Construct the bitstring $e'$ corresponding to $e$. Then $e'$ will be a bitstring of type ciphertext. Let $e'' := \bar{c}(e')$. Then $e''$ will have the form $\{\ldots\}_{\ldots}^{r'}$. Set $r'_i := r'$. If any of these operations fails (which can only happen if the ZK extraction algorithm fails or $w$ is not a witness of $x$ under the circuit $C_F$), we return $G^0$ and say that a ZK-break occurred (the value $G^0$ is arbitrary, we will later see that ZK-breaks occur only with negligible probability anyway). Otherwise, return $\mathsf{ZK}_F^{R^m_{adv}}(\underline{r}', \underline{a}', \underline{b}')$.

Case "*$m$ is of type zero-knowledge proof and $m$ has been generated by an honest agent*". I.e., $m$ was the result of a call $\mathrm{construct}^{a,sid}(\mathsf{ZK}_F^R(r_1, \ldots, r_l; a_1, \ldots, a_n; b_1, \ldots, b_s), \tau)$ by agent $a$ in session $sid$ for some $R, r_i \in \mathsf{Rand}_{ag}$, $a_i, b_i \in \mathsf{M}$, and some mapping $\tau$ from variables to bitstrings. Let $a'_i$ be the bitstring that was computed by $\mathrm{construct}^{a,sid}(a_i, \tau)$ and $b_i$ analogous. Let then $a''_i := \bar{c}(a'_i)$ (this gives the same result as applying $\bar{c}$ to all variables in $\tau$ and then computing $a''_i := a_i\tau'$ where $\tau'$ is the mapping resulting from that substitution). Define $b''_i$ analogously. Then return $\mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,r_1,sid}, \ldots, r^{a,r_l,sid}; \underline{a}''; \underline{b}'')$. (Remember that $r^{a,R,sid}$ is the randomness that is actually used in the abstract model when agent $a$ instantiates a pattern with randomness $R$ in session $sid$.)

Case "*$m$ does not match any of the above cases*". Return $G^m$.

Note that $\bar{c}$ is injective unless a ZK-break occurs: For pairs and agent ids this is obvious. In the case of nonces, garbage, encryptions and zero-knowledge proofs $m$ generated by the adversary, this follows since we include the message $m$ explicitly in the superscript of $B^m_{adv}$, $G^m$, and $R^m_{adv}$, respectively. For nonces, encryptions and zero-knowledge proofs $m$ generated by honest parties, this also follows, since we assign $m$ the abstract message $t$ whose evaluation resulted in $m$, and since repeated evaluation of $t$ does not yield different values in a concrete execution (the randomness to be used is explicitly referenced in $t$), each term $t$ can only be assigned to a single value $m$.

Finally, we see that ZK-breaks occur only with negligible probability: A ZK-break implies that a witness $w$ is extracted from a zero-knowledge proof that is not a witness for the statement of that proof, although the proof has been successfully verified. This is a contradiction to the proof of knowledge property of $\mathcal{ZK}$.

**The trace $\bar{c}(t^c)$ is a pre-DY trace.** In the following, by a *pre-DY trace* we denote an abstract trace that satisfies Definition 6 with the (possible) exception of the condition that in the transition

$$(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{send}(sid,m)} (\mathsf{Sid}, f', \varphi'),$$

we have $\varphi \vdash m$. That is, in a pre-DY trace we allow the adversary to send messages it cannot derive.

To see that $\bar{c}(t^c)$ is a pre-DY trace with overwhelming probability, we have to check that for any $\ell$, the $\ell$-th transition in $\bar{c}(t^c)$ is a valid transition. In the case of **corrupt** and **new** transitions, this follows directly from the definition of the corresponding transitions in the concrete execution and from the construction of $\bar{c}(t^c)$ (and in particular of the knowledge $\varphi_{\ell+1}$ in that trace). We therefore consider the case that the $\ell$-th transition is a transition of the form

$$(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{send}(sid,\bar{c}(m))} (\mathsf{Sid}, f', \varphi').$$

First note that all (honestly generated) nonces and all randomnesses of honest agents in $t^c$ are assigned different values with overwhelming probability. Similarly, any two zero-knowledge proofs or encryptions (unless generated with the same randomness) produced by honest agents are different because of the unpredictability property.[16]

Thus, assume that all nonces, randomnesses, encryptions, and zero-knowledge proofs of honest agents in $t^c$ are assigned different values.

Then a detailed case analysis over the construction of $\mathrm{parse}$ and the definition of the **send** transition in the concrete model shows that for a concrete transition $\mathbf{send}(sid, m)$ with local state $(i, \tau, p, (a_1, \ldots, a_k))$, we have that the invocation $\tau'' := \mathrm{parse}^{a_i,sid}(m, l, \tau)$ succeeds if and only if $\bar{c}(m)$ matches $l\gamma_{a_i,sid}\sigma$ where $\sigma := \bar{c}\circ\tau$. Furthermore, if it matches with matcher $\theta$, we have that $\sigma \cup \theta = \bar{c} \circ \tau''$.

Similarly, a detailed case analysis over the construction of $\mathrm{construct}$ and the definition of the **send** transition in the concrete model shows that for a concrete transition $\mathbf{send}(sid, m)$ with local state $(i, \tau, p, (a_1, \ldots, a_k))$, we have that the invocation $m' := \mathrm{construct}^{a_i,sid}(r, \tau'')$ succeeds if and only if $r\gamma_{a_i,sid}(\bar{c} \circ \tau'')$ contains true proofs. (Note that the condition of having true proofs corresponds to the fact that the construction of a zero-knowledge proof in $\mathrm{construct}$ will fail if the witness constructed is not actually a witness.) And in this case, we have $\bar{c}(m') = r\gamma_{a_i,sid}(\bar{c} \circ \tau'') = r\gamma_{a_i,sid}\sigma\theta$.

Thus the first edge satisfying the conditions in the definition of the **send** transition in the abstract model is the same as the first edge satisfying the corresponding conditions in

[16] Which holds for $\mathcal{ZK}$ by assumption and which can be easily seen to also hold for any IND-CCA secure encryption scheme.

the concrete model. From this we can conclude that

$$(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{send}(sid, \bar{c}(m))} (\mathsf{Sid}, f', \varphi').$$

is a valid transition. Note that the fact $\bar{c}(m') = r\gamma_{a_i,sid}\sigma\theta$ also implies that $\varphi' = \varphi \cup \{\bar{c}(m')\}$ where $m'$ is the message passed to the adversary in the concrete execution $t^c$ (if such a message was given to the adversary).

Thus we have shown that with overwhelming probability, $\bar{c}(t^c)$ is a pre-DY trace. Furthermore, from the analysis of the **send** transition we know that $\varphi_\ell = \{\mathsf{dk}(a) : a \in C\} \cup \{\bar{c}(m) : m \in S_\ell\}$ where $C$ denotes the corrupted agents and $S_\ell$ the messages given to the adversary in the send transitions in the concrete model up to the transition leading to $\varphi_\ell$.

**The trace $\bar{c}(t^c)$ is a Dolev-Yao trace.** We will now proceed to show that $\bar{c}(t^c)$ is a Dolev-Yao trace with overwhelming probability. Since we already know that $\bar{c}(t^c)$ is a pre-DY trace, and that the adversary's knowledge $\varphi_\ell$ in the $\ell$-th step of $\bar{c}(t^c)$ is $\varphi_\ell = \mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a) : a \in C\} \cup \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\}$ (where $C$ denotes the corrupted agents and $S_\ell$ the messages given to the adversary), it will be enough to show that in every $\mathbf{send}(sid, m)$ transition in the concrete model, we have that

$$\mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a) : a \in C\} \cup \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\} \vdash \bar{c}(m)$$

(with overwhelming probability).

In order to prove this, we will assume that this is not the case, i.e., that with non-negligible probability, in step $\ell$, a message $m$ is sent such that

$$\mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a) : a \in C\} \cup \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\} \nvdash \bar{c}(m) \tag{2}$$

From this we will derive a contradiction to the cryptographic assumptions used in the theorem by transforming the concrete execution in several steps into an adversary against the IND-CCA assumption.

**Simulating the zero-knowledge proofs.** As a first step towards a contradiction, we will replace all zero-knowledge proofs by fake proofs produced by the simulator. For this, we first introduce two oracles into our execution: A proof oracle $Proof$ and an extraction oracle $Extract$. Whenever an honest agent wants to produce a zero-knowledge proofs of some statement $x$ with witness $w$, it invokes $Proof(x, w)$, and when the implementation of $\bar{c}$ extract the witness of some zero-knowledge proof $z$, it invokes $Extract(z)$. Note that for this, it must be guaranteed that each zero-knowledge proof produced by honest agents uses a different randomness $R$,[17] and that this randomness is only used for the zero-knowledge proof. By Definition 5, we know that for any randomness $R$, there is at most one effective $R$-subpattern in any path of the role tree of any agent. If this effective $R$-subpattern is a term of the form $\mathsf{ZK}_F^R(\dots)$, then $R$ does not appear in the witness of any zero-knowledge proof since terms of the form $\mathsf{ZK}_F^R(\dots)$ may not appear in ZK-formulas. Thus any randomness $R$ that is used for some ZK-proof is used only for that proof (the proof may be performed several times with the same witness, statement and randomness, but in this case the $Proof$ oracle will not be invoked again but the old result will be reused).

Note the following facts:

- The oracle $Extract$ is never invoked with a proof $z$ that has previously been output by $Proof$. This holds since $\bar{c}$ by definition only extracts proofs that have not been generated by an honest agent, and only honest agents use $Proof$.
- The oracle $Proof$ is never invoked with $(x, w)$ such that $w$ is not a witness of $x$. This holds since by Definition 8 honest agents check whether $w$ is a witness before constructing a proof.

Hence, since both the execution of the concrete trace, as well as the application of the mapping $\bar{c}$ run in polynomial-time, we can use the fact that $\mathcal{ZK}$ has the extraction zero-knowledge property and replace the $Proof$ oracle by a simulation oracle $Simulate$ using the simulation trapdoor of the CRS such that $\bar{c}(t^c)$ (which is the output of an *efficient* function $\bar{c}$) is computationally indistinguishable in both cases.[18] It can easily be seen that it can be checked in polynomial-time whether a given abstract trace is a Dolev-Yao trace. Thus from the computational indistinguishability of the abstract traces in both cases, it follows that the probability that the abstract trace is a Dolev-Yao trace changes only by a negligible amount when replacing $Proof$ by $Simulate$. Thus (2) still holds with non-negligible probability. Note further that in contrast to $Proof$, the oracle $Simulate$ only expects the statement $x$ as input, but no witness.

**Using fake encryptions.** The next step towards deriving a contradiction is to replace the encryptions created by honest agents by fake encryptions. Let $C$ denote the set of corrupted agents. Since the **corrupt** transition must be the first transition, $C$ is known whenever an encryption has to be produced. We can now introduce an encryption oracle $Encrypt$ and a decryption oracle $Decrypt$ that handle encryptions and decryptions performed by honest agents with respect to any key $\mathsf{ek}(a)$ or $\mathsf{dk}(a)$ with $a \in \mathsf{A} \setminus C$.

[17] Here and in the following, when we reason about a randomness $R \in \mathsf{Rand}_{ag}$ in the concrete model, we mean the symbolic value $R$ that is used to select the corresponding bitstring from the random tape using the function $tape$.

[18] Here we really need extraction zero-knowledge and not only the extraction oracle $Extract$ is used.

Again, we have to verify that the randomness used for an encryption is never reused. Fix a randomness $R$ that is used in some encryption $c := \{t\}^R_{\mathsf{ek}(a)}$. Again, we exploit that for any randomness $R$, there is at most one effective $R$-subpattern in any path of the role tree of any agent. In this case, it will be $c$. Then $c$ may occur directly as a subterm of the message to be sent, or it may result from substituting the $\underline{\rho}, \underline{\alpha}, \underline{\beta}$ in a ZK-formula $F$ with some other terms. Thus the randomness $R$ may be used in three places: In the computation of the bitstring corresponding to $c$, in the verification whether the witness $w$ used for some ZK-proof with some formula $F$ is valid, and as part of the witness $w$. The first case is a normal encryption and thus can be replaced by an invocation of $Encrypt$. The third case is captured by the fact that we do not need to construct the witness $w$ since the oracle $Simulate$ introduced above does not need a witness. It remains to see that we can check whether the witness $w$ is valid without explicitly constructing it or accessing the randomness $R$. However, any ZK-term contained in the ZK-formula $F$ will, after substituting the witness, contain $R$ only in subterms that are equal to $c$ (otherwise $c$ there would be more than one effective $R$-subterm). For these subterms, we reuse the result of the invocation of $Encrypt$ (or, if $c$ has not been evaluated yet, invoke $Encrypt$ now and reuse the result later when it is needed for constructing a bitstring). Thus we do not need to access the randomness $R$ used by the encryption oracle $Encrypt$ directly.

We can further change the invocations of the decryption oracle as follows. If a ciphertext $c$ is to be decrypted with respect to $\mathsf{dk}(a)$ that was previously returned by the oracle $Encrypt$ for some plaintext $m$ with respect to $\mathsf{ek}(a)$, then do not call $Decrypt$ but use the plaintext $m$ directly. We end up being in a situation where $Decrypt$ is only invoked for encryptions that have not been returned by $Encrypt$. Hence, because $\mathcal{AE}$ is IND-CCA secure, we can replace $Encrypt$ by an oracle $FakeEncrypt$ that instead of a plaintext $m$ expects only its length $l$, and that returns the encryption of a random string of length $l$. This will lead to a computationally indistinguishable trace $\bar{c}(t^c)$, and thus (2) still holds with non-negligible probability.

**Identifying the underivable subterm.** In order to derive a contradiction from (2), we have to identify the subterm of $\bar{c}(m)$ whose "fault" it is that $\bar{c}(m)$ cannot be derived. We will then use this term to construct an attack against the IND-CCA. For this, we need the following characterization of underivable messages:

**Lemma 2.** *Fix $C \subseteq \mathsf{A}$, $S \subseteq \mathsf{M}$ and $M \in \mathsf{M}$ and set $S' := S \cup \{\mathsf{dk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv}$. Assume $S' \nvdash M$ and that $M, S$ do not contain a subterm of the form $\mathsf{dk}(\mathsf{A})$. Assume further that $M$ contains true proofs. We say $r \in \mathsf{Rand}$ is* known *if $r \in \mathsf{Rand}_{adv}$ or there are $a \in C$ and $t \in \mathsf{M}$ such that $S' \vdash \{t\}^r_{\mathsf{ek}(a)}$.*

*Then there exists a term $T \in \mathsf{M}$ and a context $D$ such that $M = D[T]$ and all terms on the path from $M = D[T]$ to $T$ (not including $T$) are of the form*

$$\langle \cdot, \cdot \rangle \quad or \quad \{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)} \quad or \quad \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$$

*Furthermore, we have that $S' \nvdash T$ and that $T$ satisfies one of the following conditions:*
*(a) $T \in \mathsf{Nonce}_{ag}$, or*
*(b) $T = \{\cdot\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(a)}$, or*
*(c) $T = \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\dots)$, or*
*(d) $T = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ and for some $i$, $r_i$ is not known.*

*Proof.* We prove the lemma by structural induction on $M$. In the base case, $M$ is a nonce, an agent identifier, a public key, or garbage. In the last three cases, we have $S' \vdash M$, so the lemma is not applicable. If $M$ is a nonce, we distinguish $M \in \mathsf{Nonce}_{adv}$ and $M \in \mathsf{Nonce}_{ag}$. In the first case, $S' \vdash M$ holds, thus $M$ is not applicable. In the second case, the conclusion of the lemma is fulfilled with $T := M$.

In the induction step we distinguish the following cases:

Case "$M = \langle M_1, M_2 \rangle$". Since $S' \nvdash M$, we have $S' \nvdash M_i$ for some $i \in \{1, 2\}$. Hence there exists a subterm $T$ of $M_i$ satisfying the conclusion of the lemma for $M_i$, and this $T$ is also a subterm of $M$ satisfying the conclusion for $M$.

Case "$M = \{M'\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(a)}$ *or* $M = \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\dots)$". In these cases $T := M$ fulfills the conclusion of the lemma.

Case "$M = \{M'\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(a)}$". In this case, since $S \nvdash M$ and $S \vdash \mathsf{ek}(a)$, we have that $S \nvdash M'$. Hence there exists a subterm $T$ of $M'$ satisfying the conclusion of the lemma for $M'$, and this $T$ is also a subterm of $M$ satisfying the conclusion for $M$.

Case "$M = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ *and all $\underline{r}$ are known*". Since $\underline{r}$ are known, if we had $S' \vdash \underline{a}, \underline{b}$ we would also have $S' \vdash M$. Thus for some $M' \in \{\underline{a}, \underline{b}\}$ we have $S' \nvdash M'$. Hence there exists a subterm $T$ of $M'$ satisfying the conclusion of the lemma for $M'$, and this $T$ is also a subterm of $M$ satisfying the conclusion for $M$.

Case "$M = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ *and one of the $\underline{r}$ is not known*". Then $T := M$ fulfills the conclusion of the lemma. □

In a trace satisfying (2), we can apply this lemma with $C$ being the corrupted agents and $S := \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\}$ being the messages up to the $\ell$-th step, and $M := \bar{c}(m)$ being the message in the $\ell$-th step. The condition that $S, M$ do not contain subterms of the form $\mathsf{dk}(\mathsf{A})$ is fulfilled since the function $\bar{c}$ by definition never outputs such terms. Similarly, $\bar{c}$ will not output an $M$ that does not contain true proofs

since $\bar{c}$ checks the witnesses it extracts. Since (2) holds with non-negligible probability, we have that with non-negligible probability, a subterm $T$ of $M$ as in Lemma 2 exists.

The term $T$ satisfies one of the four properties (a–d). In the following, we will examine each of these conditions separately and in each case derive a contradiction.

***T* is a nonce.** In case (a) we have $T \in \mathsf{Nonce}_{ag}$. Since $S' \not\vdash T$, for any message $m$ sent to the adversary, the nonce $T$ occurs in $\bar{c}(m)$ only inside an encryption (with a public key $\mathsf{ek}(a)$ with $a \notin C$) or inside the witness of a zero-knowledge proof. Since honest agents construct such encryptions and zero-knowledge proofs using the oracles $Simulate$ and $FakeEncrypt$, the message $m$ can be computed without knowing the bitstring corresponding to $T$.[19] Then the value of $T$ is only used in comparisons, namely when checking whether a given witness $(\underline{r}; \underline{a})$ is valid or to perform pattern matching. Thus it might be that these comparisons leak information about the nonce $T$ (up to one bit per comparison). However, it is easy to see that only by comparing (polynomially many) values to $T$, it is not possible to guess $T$ with more than negligible probability. On the other hand, from the message $m$ sent by the adversary, we can retrieve the nonce as follows. In $M = \bar{c}(m)$, the nonce $T$ is protected only by terms of the form $\langle \cdot, \cdot \rangle$, $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$ or $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$. The pair can directly be parsed, in the case of $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$ or $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$, we can call the oracles $Decrypt$ and $Extract$, respectively. Since these oracles are also used by the function $\bar{c}$ (at least for terms where $\bar{c}$ assigns randomness $\mathsf{Rand}_{adv}$ and not $\mathsf{Rand}_{ag}$), these oracles will give answers consistent with the parsing $M = \bar{c}(m)$ of $m$. Thus we can guess the nonce $T$ which leads to a contradiction.

***T* is an honestly-generated encryption or zero-knowledge proof.** In the cases (b) and (c) of Lemma 2, completely analogous reasoning to case (a) applies. In these cases, the term $T$ will correspond to an encryption or zero-knowledge proof that was generated by honest agents. Since zero-knowledge proofs and encryptions have the unpredictability property, we get a contradiction by showing that the adversary can guess $T$ without $T$ being used.

***T* is an adversary-generated zero-knowledge proof.** In case (d), we have that $T = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r}; \underline{a}; \underline{b})$ and that $r_i$ is not known (in the sense of Lemma 2). In this case the argumentation used for the cases (a–c) cannot be used because $T$ does not correspond to a bitstring generated by an honest agent. However, as in the preceding paragraphs, the adversary can extract the bitstring corresponding to $T$, and using the oracle $Extract$ it can extract the concrete randomness corresponding to $r_i$. By definition of the function $\bar{c}$, this randomness will be the randomness used in an encryption with respect to some $\mathsf{ek}(a)$ performed by an honest agent (otherwise the function $\bar{c}$ would have assigned a randomness $r_i \in \mathsf{Rand}_{adv}$). We distinguish two cases: $a \notin C$ and $a \in C$. If $a \notin C$, then the encryption has been generated using the encryption oracle $Encrypt$. Being able to retrieve the randomness used in that encryption contradicts the IND-CCA property of $\mathcal{AE}$. If $a \in C$, then the randomness has been used to generate the bitstring corresponding to a term $c = \{t\}^{\mathsf{Rand}_{ag}}_{r_i}$ with $a \in C$. Since $r_i$ is not known, we have that $S' \not\vdash c$. With an analogous argument as above, we can see that all bitstrings sent by honest agents can be computed without actually computing the bitstring corresponding to $c$. But in this case, that fact that the adversary is able to guess the randomness used to produce $c$ is a contradiction. □

[19] These oracles expect only the *length* of the witness or plaintext, respectively. This length can be computed since the length of the nonce is fixed and thus known.

## References


[AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[BCC04] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[BCJ+06] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of kerberos 5. *Theoretical Computer Science*, 367(1):57–87, 2006.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *17th Annual IEEE Conference on Computational Complexity, Proceedings of CCC'02*, pages 194–203. IEEE Computer Society, 2002. Online available at `http://www.cs.princeton.edu/~boaz/Papers/uargs.ps`.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1998.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. IEEE Symposium on Security and Privacy 2008, May 2008. To appear. Full version available at `http://eprint.iacr.org/2007/289`.

[BMV04] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[BP04] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption—how to encrypt with RSA. In Alfredo de Santis, editor, *Advances in Cryptology, Proceedings of EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995. Extended version online available at `http://www.cs.ucsd.edu/users/mihir/papers/oae.ps`.

[BU08] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *21st IEEE Computer Security Foundations Symposium, CSF 2008*, 2008. To appear.

[CH06] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[CKKW06] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology, Proceedings of CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998. Online available at `http://eprint.iacr.org/1998/006`.

[CW05] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[DS81] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[EG83] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[Fis03] Dennis Fisher. Millions of .Net Passport accounts put at risk. *eWeek*, May 2003. (Flaw detected by Muhammad Faisal Rauf Danka).

[FOPS04] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 2004. Online available at `http://www.di.ens.fr/~pointche/Documents/Papers/2004_joc.pdf`.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991. Online available at `http://www.wisdom.weizmann.ac.il/~oded/X/gmw1j.pdf`.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007. Full version available at `http://www.brics.dk/~jg/MultiStringModelFull.pdf`. The definition of extraction zero-knowledge is only contained in the full version.

[Gol01] Oded Goldreich. *Foundations of Cryptography – Volume 1 (Basic Tools)*. Cambridge University Press, August 2001. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[Gol04] Oded Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications)*. Cambridge University Press, May 2004. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[KMM94] Richard Kemmerer, Catherine Meadows, and Jon Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[Lau01] Peeter Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[Lau04] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[Mer83] Michael Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[MW04] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[NS78] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.

[Pau98] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[Sch96] Steve Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

[WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.