Computational Soundness of Symbolic Zero-Knowledge Proofs[1]

Michael Backes[a,b] and Dominique Unruh[a]
[a] Saarland University
[b] Max Planck Institute for Software Systems (MPI-SWS)

September 10, 2009

Comments and corrections are welcome. Please send them to Dominique Unruh <unruh@mmci.uni-saarland.de>.

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 (non-interactive) 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.

Keywords: Computational soundness, zero-knowledge, symbolic cryptography.

Contents

[1] To appear in the Journal of Computer Security. A preliminary version appeared at CSF 2008 [BU08].

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, nAG97, 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 [AR02, Lau01, BPW03a, BP04, Lau04, MW04, CW05, CH06, SBB+06].

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 arguably the most prominent and most amazing such primitive (though not the only one[2]). 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 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 (non-interactive) 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 (non-interactive) 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.

Related work. Cryptographic underpinnings of a Dolev-Yao model were first addressed by Abadi and Rogaway in [AR02] for passive adversaries and symmetric encryption. The protocol

[2] Examples for other primitives studied in the Dolev-Yao model are blind-signatures (e.g., in [KR05]), Diffie-Hellman-style exponentiation (e.g., in [AF01]), or private contract signatures (e.g., in [KRW04]).

language and security properties handled were extended in [AJ01, Lau01, HLM03, ABW06], but still apply only for passive adversaries. This excludes most of the typical ways of attacking protocols, e.g., man-in-the-middle attacks and attacks by reusing a message part in a concurrent protocol run.

A cryptographic justification of a Dolev-Yao model for arbitrary active attacks and within arbitrary surrounding interactive protocols (within the Reactive Simulatability (RSIM) Framework [BPW07]) was first given by Backes, Pfitzmann, and Waidner in [BPW03a] with extensions in [BPW03b, BP04]. Tool support for this Dolev-Yao model was subsequently added in [SBB+06]. Laud [Lau04] has subsequently presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work enjoys a direct connection with a formal proof tool, but it is specific to certain confidentiality properties and restricts the surrounding protocols to straight-line programs. Micciancio and Warinschi [MW04] and Janvier, Lakhnech, and Mazaré [JLM05] have presented cryptographic underpinnings for a Dolev-Yao model of public-key encryption. Their results are narrower than those in [BPW03a] since they are specific for public-key encryption and restricted classes of protocols, but they consider simpler real implementations. Baudet, Cortier, and Kremer [BCK05] have established the soundness of specific classes of equational theories in a Dolev-Yao model under passive attacks. Canetti and Herzog [CH06] have shown that a Dolev-Yao-style symbolic analysis can be conducted using the framework of universal composability [Can01] for a restricted class of protocols.

Subsequent work concentrated on linking symbolic and cryptographic secrecy properties. Cortier and Warinschi [CW05] have shown that symbolically secret nonces are also computationally secret, i.e., indistinguishable from a fresh random value given the view of the adversary. Backes and Pfitzmann [BP05] and Canetti and Herzog [CH06] have established new symbolic criteria for showing that a key is cryptographically secret. Laud [Lau05] has designed a type system for proving payload secrecy of security protocols based on the BPW model [BPW03a]. His work is extended to key secrecy in [BL06]. Kremer and Mazaré [KM07] have established computational soundness results for static equivalence. Adão and Fournet [AF06] have shown computational soundness in the sense of observational equivalence of cryptographic implementations of processes. Cortier and Comon-Lundh [CLC08] have established computational soundness results as the preservation of observational equivalence within a fragment of the applied π-calculus (this fragment is restricted to protocols that do not branch).

Further work aimed at establishing computational soundness results for additional cryptographic primitives. Cortier, Kremer, Küsters, and Warinschi [CKKW06a] and Backes, Pfitzmann, and Waidner [BPW06] have shown computational soundness of hash functions in the random oracle model. Janvier, Lakhnech, and Mazaré [JLM07] have shown computational soundness of hash functions under a non-standard cryptographic assumption in the standard model, i.e., without random oracles. Garcia and van Rossum [GvR08] have showed computational soundness of hash functions under passive adversaries when implemented using perfect one-way hash functions [CMR98]. Bresson, Lakhnech, Mazaré, and Warinschi [BLMW07] have provided a computationally sound theory for reasoning about protocols based on the decisional Diffie-Hellman assumption (DDH) for passive adversaries. Limitations of computational soundness in the sense of Reactive Simultability were shown by Backes and Pfitzmann for hash functions [BPW06] and the XOR operation [BPW05].

Recently, efforts have also been started to formulate syntactic calculi with a probabilistic, polynomial-time semantics to directly reason about cryptographic primitives/protocol, including approaches based on process algebra [MMS98, LMMS98], security logics [IK03, DDM+05] and cryptographic games [Bla06, BP06, Cd06, Now07, BGJZ07, BBU08]. In general, this line of work is orthogonal to the work of justifying Dolev-Yao models, which offer a higher level of abstractions and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

Outline of the Paper. In Section 2, we briefly review the modeling of symbolic 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. Sections 3 and 4 contain the

symbolic and computational execution model. Section 5 contains our computational soundness result for symbolic zero-knowledge. Section 7 concludes and outlines future work.

Notation. By $[n]$ we denote $\{1,\dots,n\}$. We abbreviate $x_1,\dots,x_n$ 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 write $x \in (\mathsf{C},\mathsf{A})$, or we say "$x$ has the form (C,A)". We also write "x has the form (c,A)" for a given $c \in \mathsf{C}$. Given a term t and a substitution θ, we write $t\theta$ for the result of applying the substitution θ to t. We call two partial functions θ and θ' compatible if $\theta(x) = \theta'(x)$ whenever both are defined. Given two compatible partial functions θ and θ', we write $\theta \cup \theta'$ for the function satisfying $\theta \cup \theta'(x) = \theta(x)$ if $\theta(x)$ and $\theta \cup \theta'(x) = \theta'(x)$ otherwise. We write $\theta[x := y]$ for the function θ' defined by $\theta'(z) = \theta(z)$ for $z \neq x$ and $\theta'(x) = y$.

We call a function f negligible if for all positive polynomials p and sufficiently large n, $f(n) \geq 1/p(n)$. We call a function non-negligible if it is not negligible. We call a function f overwhelming if $f \geq 1 - \mu$ for some negligible function μ.

2 Zero-Knowledge Proofs

In this section, we first introduce our modeling of 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 computational (non-interactive) 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 symbolic zero-knowledge proofs.

2.1 Symbolic 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 symbolic 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 symbolic randomness ρ_1,ρ_2 and some values α_1,α_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:

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

Immediately including the values of β_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 a symbolic constructor $\mathsf{ZK}^R_F(\underline{r};\underline{a};\underline{b})$, called a *zero-knowledge proof*. (Recall that we abbreviate tuples $r_1,\dots,r_n$ by $\underline{r}$, similarly for $\underline{a}$ and $\underline{b}$). Its arguments are a symbolic 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 $z := \mathsf{ZK}^R_F(R_1,R_2;n,m;m_1,m_2,m_3,S,D,c,m)$. 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 a symbolic 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 a symbolic setting, comprising a larger set of base constructors such as blind signatures, we refer the interested reader to [BMU08].

2.2 Computational Zero-Knowledge Proofs

We now move to computational 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 symbolic model considers a proof as a single message that can be 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) (such a CRS is comparable to the public key of an 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 NP $\subseteq$ 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 symbolic 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 $z := \mathsf{ZK}^{R'}_F(R; m; c, a)$ 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 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 symbolic 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 $E(C, \mathit{proof}, \mathit{extd})$ 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 $E(C, \mathit{proof}, \mathit{extd})$ 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 $(\mathit{proof}_1, \mathit{proof}_2)$ 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 subcircuit individually does not contradict the properties we have discussed so far. In the symbolic model, however, given $\mathsf{ZK}^R_F(r; a; b)$ and $\mathsf{ZK}^{R'}_{F'}(r'; a'; b')$, it is not possible to construct a proof $\mathsf{ZK}^{\neg}_{F \wedge F'}(\_; \_; b, b')$ 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 $E(C,\cdot,\mathit{extd})$, i.e., when invoking the oracle $E(C,\cdot,\mathit{extd})$ with input *proof*, it returns $E(C,\mathit{proof},\mathit{extd})$ 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? (We only give an intuitive argument here as formally we will directly use the property of extraction zero-knowledge in our proofs.) 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 $\mathit{proof}_2 \neq \mathit{proof}_1$, 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. If the extraction oracle could produce a witness given proof_2, this would mean that the extraction oracle (together with the M and the simulator) has produced a witness for the satisfiability of C_2 (which we can assume to be hard). Since the extraction oracle, M, and the simulator are all polynomial-time, this happens with negligible probability. Hence in case (b) the extraction oracle fails to produce a witness for proof_2. Thus the adversary can distinguish the two cases, contradicting the extraction zero-knowledge property.

Unpredictability. We need an additional property for soundly implementing symbolic zero-knowledge proofs. If a proof using the same witness and the same public part is produced by two different agents in the symbolic 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 necessarily 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, $\mathit{proof}\neq\mathit{proof}'$.

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

Length-regularity. To get computational soundness, we additionally need that the length of a zero-knowledge proof is independent of its public part (although it may depend on the *length* of its public part). Consider a protocol which chooses a nonce N and then sends $\{\mathsf{ZK}^R_{\mathit{true}}(;;N)\}^R_{\mathsf{ek}(a)}$. That is, the protocol produces a trivial proof and includes the nonce N in the public part. Then it encrypts that proof. Abstractly, this should preserve the secrecy of N. If, however, in the computational model the length of $\mathsf{ZK}^R_{\mathit{true}}(;;N)$ could depend on the first bit of N, then $\{\mathsf{ZK}^R_{\mathit{true}}(;;N)\}^R_{\mathsf{ek}(a)}$ might leak the length of $\mathsf{ZK}^R_{\mathit{true}}(;;N)$ and thus the first bit of N.[3] This example can easily be extended to leak all bits of N and thus break the secrecy of N. Note that the requirement of length-regularity is not particular to zero-knowledge; even the encoding of pairs needs to be length-regular.

[3]Remember that encryption schemes cannot hide the length of their plaintexts.

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 (all probabilities are taken over the coin tosses of all algorithms and adversaries):*

- 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 η, $C(w) = 0$ 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 $w \leftarrow E(C, proof, extd)$. 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 $(C, w, state) \leftarrow \mathcal{A}^{E(\cdot,\cdot,extd)}(1^\eta, crs, simtd)$. Then let $proof \leftarrow P(C, w, crs)$ if $b = 0$ and $proof \leftarrow S(C, crs, simtd)$ if $b = 1$. Let $guess = \mathcal{A}^{E(\cdot,\cdot,extd)}(1^\eta, crs, simtd, state, proof)$. Let $P_b(\eta)$ denote the following probability:*

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

  Then $|P_0(\eta) - P_1(\eta)|$ is negligible.
- Length-regularity. *Let two circuits C, C' and witnesses w, w' with $C(w) = C'(w') = 1$ be given such that $|C| = |C'|$ and $|w| = |w'|$. Let $(crs, simtd, extd) \leftarrow K(1^\eta)$. Then let $proof \leftarrow P(C, w, crs)$ and $proof' \leftarrow P(C', w', crs)$. Then $|proof| = |proof'|$ holds with probability 1.*[4]
- 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. 4] under the assumption that enhanced trapdoor permutations [Gol04, Def. C.1.1] exist.[5] The latter exist, e.g., under the assumption that factoring is hard. (The length-regularity property can be ensured by a suitable padding.)

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

[4] We could relax this property to require that this equality holds only with overwhelming probability, or even to only require the distributions of $|proof|$ and $|proof'|$ to be indistinguishable. We opt for this simpler formulation because we are not aware of examples that need the more lenient formulations.

[5] [GO07] has a slightly different definition of extraction zero-knowledge where the extraction algorithm E is not given the circuit C (only the proof *proof*). However, their scheme can be easily modified such that valid proofs *proof* have to contain the circuit C, and then our and their definition coincide.

3 The Symbolic Model

In the following we define the symbolic model in which the execution of a symbolic protocol involving zero-knowledge proofs takes place. The basic ideas follow the framework presented in [CKKW06b]. However, to incorporate zero-knowledge proofs, we have to make various nontrivial modifications to the symbolic 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 A we denote the set of agent identifiers, by Nonce the set of nonces. We use elements from $\mathsf{Garbage}$ to represent ill-formed messages (corresponding to unparseable bitstrings in the computational model). Finally, elements of Rand denote symbolic randomness used in the construction of ciphertexts and zero-knowledge proofs. We assume that Nonce is partitioned into infinite sets Nonce_{ag} and Nonce_{adv}, representing the nonces of honest agents and the nonces of the adversary. Similarly, Rand is partitioned into infinite sets Rand_{ag} and 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 the set ZKTerm of *ZK-terms* be defined by the following grammar:

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

where $i, j = 1, 2, \ldots$. 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. The symbol β_i is a reference to the i-th component of the public part, and the symbols α_i and ρ_i are references to the witness of the zero-knowledge proof.

Note that we do not include signatures in the syntax of ZK-terms. This is because we explicitly deal with signatures using the Verify-ZK-atoms below.

A ZK-atom is a term of the form $\mathsf{ZKTerm} = \mathsf{ZKTerm}$ or $\mathrm{Decrypt}(\mathsf{ZKTerm}, \alpha_i, \mathsf{ek}(\beta_j)) = \mathsf{ZKTerm}$ or $\mathrm{Verify}(\mathsf{ZKTerm}, \mathsf{ZKTerm}, \mathsf{vk}(\beta_j))$. On an intuitive level, $\mathrm{Decrypt}(c, k, \mathsf{ek}(a)) = p$ means that k is the decryption key $\mathsf{dk}(a)$ corresponding to $\mathsf{ek}(a)$ and that c decrypts to p using k. We include the encryption key $\mathsf{ek}(a)$ in such a ZK-atom to allow the verifier to check that the decryption is indeed performed using the right decryption key. A ZK-atom of the form $\mathrm{Verify}(s, m, \mathsf{vk}(a))$ means that s is a valid signature of m using the verification key $\mathsf{vk}(a)$.

A ZK-formula F is a Boolean formula over ZK-atoms satisfying the following conditions: If ρ_i occurs in F, then $\rho_1, \ldots, \rho_i$ occur all in F.[6] Each ρ_i appears at most once.[7] We denote the set of ZK-formulas with $\mathsf{Formula}$. The α-arity of a ZK-formula F is the largest index of an α_i occurring in F. The ρ-arity and the β-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. (By $F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ we denote the result of substituting r_i for ρ_i, a_i for α_i, and b_i for β_i.) 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 $\mathsf{ek}(\beta_i)$ and $\mathsf{vk}(\beta_i)$ to appear in a formula, but not $\mathsf{ek}(\alpha_i)$ or $\mathsf{vk}(\alpha_i)$. This is due to the fact that a proof with a formula containing $\mathsf{ek}(\alpha_i)$ or $\mathsf{vk}(\alpha_i)$ 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 $\mathsf{ek}(\alpha_i)$ or $\mathsf{vk}(\alpha_i)$, one would have to encode a list of *all* public keys in

[6] This is a purely technical condition that simplifies the proof at one place. We include it because it does not seem to be a strong restriction.

[7] This excludes proofs that two different ciphertexts are produced using the same randomness. However, honest agents never produce two different ciphertexts with the same randomness anyway.

this circuit. On the other hand, in the case of $\mathsf{ek}(\beta_i)$, the value b_i substituted for β_i is known while constructing the circuit, thus we can directly hard-code $\mathsf{ek}(b_i)$ into the circuit.

Note further that we have introduced two different ways to formulate proofs about the content of an encryption: To show that c is an encryption of m, we can either show that there is some randomness R such that $\{m\}^R_{\mathsf{ek}(a)} = c$ (so the corresponding zero-knowledge proof would be $\mathsf{ZK}^{R'}_F(R;;m,c,a)$ with $F = (\{\beta_1\}^{\rho_1}_{\mathsf{ek}(\beta_3)} = \beta_2)$) or we can show that decrypting c yields m (the corresponding zero-knowledge proof would be $\mathsf{ZK}^{R'}_F(;\mathsf{dk}(a);m,c,\mathsf{ek}(a))$ with $F = (\mathrm{Decrypt}(\beta_2,\alpha_1,\beta_3) = \beta_1)$). However, the first type of proof can only be constructed by the agent that produced c, and the second can only be constructed by the agent that knows the decryption key for c. Hence, to cover all relevant cases, we need to enable both constructions.

Given the syntax for ZK-formulas, we can define the set M of messages as

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

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 ρ-arity, α-arity, and β-arity of F, respectively. In a message $\mathsf{ek}(a)$, $\mathsf{dk}(a)$, $\mathsf{vk}(a)$, and $\mathsf{sk}(a)$ represent encryption, decryption, verification, and signing 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 the 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.

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

We say Z is valid if for every subterm $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ of Z, we have that $b_i \in \mathsf{A}$.

Then $F = B(\underline{t})$ where all t_i are ZK-atoms and B is a Boolean formula. Let $g_i := t_i\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$. (By $t\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ we denote the result of substituting r_i for ρ_i, a_i for α_i, and b_i for β_i.)

We say g_i is true if it is of the form $m_1 = m_1$ or $\mathrm{Decrypt}(\{m_1\}^{r_1}_{\mathsf{ek}(a_1)}, \mathsf{dk}(a_1), \mathsf{ek}(a_1)) = m_1$ or $\mathrm{Verify}([m_1]^{r_1}_{\mathsf{sk}(a_1)}, m_1, \mathsf{vk}(a_1))$ with $m_1 \in \mathsf{M}$, $a_1 \in \mathsf{A}$, and $r_1 \in \mathsf{Rand}$. Let $h_i := \mathsf{True}$ if g_i is true, and $h_i := \mathsf{False}$ otherwise.

We say Z is a true proof *if it is valid and $B(\underline{h}) = \mathsf{True}$.*

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

Note that if Z is valid, every g_i has the form $\mathsf{M} = \mathsf{M}$ or $\mathrm{Decrypt}(\mathsf{M},\mathsf{M},\mathsf{M}) = \mathsf{M}$ or $\mathrm{Verify}(\mathsf{M},\mathsf{M},\mathsf{M})$ (we produce no terms that are not messages).

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 $\varphi \vdash \mathsf{ZK}^r_F(\underline{r};\underline{a};\underline{b}) \Longrightarrow \varphi \vdash \underline{b}$ (ZKPUB) corresponds to 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 $\underline{r},\underline{a}$.[8]

[8] Formally, of course, the zero-knowledge property does not correspond to any given rule, but to the absence of rules allowing to deduce more than $\underline{b}$ from $\mathsf{ZK}^r_F(\underline{r};\underline{a};\underline{b})$.

$$\frac{m \in \varphi}{\varphi \vdash m}\ \textsc{Assm} \qquad \frac{g, g' \in \mathsf{Garbage} \qquad r \in \mathsf{Rand}_{adv} \qquad \varphi \vdash m}{\varphi \vdash g \qquad \varphi \vdash \mathsf{ek}(g) \qquad \varphi \vdash \{g'\}^r_{\mathsf{ek}(g)} \qquad \varphi \vdash [m]^r_{\mathsf{sk}(g)}}\ \textsc{Garbage}$$

$$\frac{b \in \mathsf{A}}{\varphi \vdash \mathsf{ek}(b) \qquad \varphi \vdash \mathsf{vk}(b) \qquad \varphi \vdash b}\ \textsc{PubKey} \qquad \frac{\varphi \vdash m_1 \qquad \varphi \vdash m_2}{\varphi \vdash \langle m_1, m_2 \rangle}\ \textsc{Pair}$$

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

$$\frac{\varphi \vdash \{m\}^r_{\mathsf{ek}(b)} \qquad \varphi \vdash \mathsf{dk}(b)}{\varphi \vdash m}\ \textsc{Dec} \qquad \frac{\varphi \vdash [m]^r_{\mathsf{sk}(a)}}{\varphi \vdash m}\ \textsc{SigPub}$$

$$\frac{\varphi \vdash \mathsf{sk}(a) \qquad \varphi \vdash m \qquad r \in \mathsf{Rand}_{adv}}{\varphi \vdash [m]^r_{\mathsf{sk}(a)}}\ \textsc{Sig} \qquad \frac{\varphi \vdash \mathsf{ZK}^r_F(\underline{r}; \underline{a}; \underline{b})}{\varphi \vdash \underline{b}}\ \textsc{ZKPub}$$

$$\frac{\varphi \vdash \underline{a} \qquad \varphi \vdash \underline{b} \qquad F \in \mathsf{Formula} \qquad r \in \mathsf{Rand}_{adv} \qquad F\left\{\frac{r,\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\right\} \text{ is a true proof} \qquad \varphi \vdash_r \underline{r}}{\varphi \vdash \mathsf{ZK}^r_F(\underline{r}; \underline{a}; \underline{b})}\ \textsc{ZKTrue}$$

$$\frac{r \in \mathsf{Rand}_{adv}}{\varphi \vdash_r r}\ \textsc{RandAdv} \qquad \frac{\varphi \vdash \{t\}^r_{\mathsf{ek}(a)} \qquad \varphi \vdash \mathsf{dk}(a)}{\varphi \vdash_r r}\ \textsc{RandDec}$$

Figure 1: Deduction rules for the adversary.

More interesting and involved is the rule ZKTrue. 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 $F\{\frac{r,\underline{a},\underline{b}}{\rho,\underline{\alpha},\underline{\beta}}\}$ 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 symbolic 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. We capture these two possibilities of extracting randomness in the definition of the relation $\not\vdash_r$ (Figure 1). As an example why allowing the adversary to extract randomness from ciphertexts (rule RandDec) 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 $r_i \in \mathsf{Rand}_{adv}$ in the witness, the protocol would be secure abstractly, even if the adversary knows the secret decryption key $\mathsf{dk}(b)$. Yet a computational adversary could possibly (depending on the encryption scheme) extract the randomness from c and produce such a proof.[9]

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.[10]

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

$$\begin{gathered}\mathsf{Pat} = \mathsf{X} \mid \mathsf{ek}(\mathsf{X}.a) \mid \mathsf{vk}(\mathsf{X}.a) \mid \langle \mathsf{Pat},\mathsf{Pat}\rangle \mid \{\mathsf{Pat}\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(\mathsf{X}.a)} \mid \{\mathsf{Pat}\}^-_{\mathsf{ek}(\mathsf{X}.a)} \mid \\ \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\mathsf{Rand}^*_{ag};(\mathsf{Pat}|\mathsf{dk}(\mathsf{X}.a))^*;\mathsf{Pat}^*) \mid \mathsf{ZK}^-_{\mathsf{Formula}}(\_^*;\_^*;\mathsf{Pat}^*) \mid \\ [\mathsf{Pat}]^-_{\mathsf{sk}(\mathsf{X}.a)} \mid [\mathsf{Pat}]^{\mathsf{Rand}_{ag}}_{\mathsf{sk}(\mathsf{X}.a)}\end{gathered}$$

with the following additional conditions: For each subterm $\mathsf{ZK}^{\mathsf{Rand}_{ag}}_F(\underline{r};\underline{a};\underline{b})$ or $\mathsf{ZK}^-_F(\underline{r};\underline{a};\underline{b})$, we have that $|\underline{r}|,|\underline{a}|,|\underline{b}|$ are the ρ-arity, α-arity, and β-arity of F, respectively. This condition is needed to ensure that a pattern (containing no $\_$) becomes a valid message when the variables are instantiated.

[9] 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 computational adversary could use extracted randomness also in this case. The rough reason why this is not necessary is that if the computational randomness r is used to encrypt another message or under another key, we can consider it to be another symbolic randomness. Only if the *same* m is encrypted under the *same* $\mathsf{ek}(b)$, we will have to consider the computational 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.

Furthermore, the reader may wonder why the adversary can only extract randomness from ciphertexts, but not from signatures or zero-knowledge proofs. The intuitive reason is that even if the adversary could extract the randomness used in the construction of a signature or a zero-knowledge proof, this randomness would be useless in the witness of a zero-knowledge proof, as that randomness is only used for proving statements about encryptions.

[10] 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 $\mathsf{Garbage}$).

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 θ 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[11] 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 containing $\mathsf{dk}(\cdot)$ (except in the witnesses of zero-knowledge proofs) or $\mathsf{sk}(\cdot)$ since we do not allow protocols to explicitly use their private keys (except for decrypting or signing). 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. At this point, we give an informal description and postpone the exact definitions to Section 3.1. A *k-party protocol* Π 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 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 precisely, an edge is labeled with a pair (l, r) of patterns. Here l represents the pattern for matching incoming messages (parse-pattern), and r the pattern for constructing the answers to these messages (construct-pattern). The complete 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 σ, and an edge (l, r), the following steps take place:[12]

- 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 θ 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 previously assigned in σ, or assigned in the previous step (θ). 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 $X^j_N \in \mathsf{X}.n$ (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 [CKKW06b] 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

[11] 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.

[12] 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.

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. Or 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 matching an incoming message with the pattern l does not imply decrypting with someone else's secret key. Most conditions are of this kind and just guarantee that the symbolic protocol can indeed be implemented as a computational protocol. Complete details are given in Definition 4 in Section 3.1 below. 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 (except for decryption keys in the witness of a zero-knowledge proof). 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 then can send them.) This condition does not newly arise for zero-knowledge proofs; it is also present, e.g., in [CW05, CKKW06b].

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., [CW05, CKKW06b]) 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 (ρ-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-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 (True and effective subpatterns)**.** *For a given $R \in \mathsf{Rand}$, we call P an R-pattern if it is of the form $\{\mathsf{Pat}\}^R_{\mathsf{Pat}}$ or $[\mathsf{Pat}]^R_{\mathsf{Pat}}$ or $\mathsf{ZK}^R_{\mathsf{Formula}}(\mathsf{Rand}^*; \mathsf{Pat}^*; \mathsf{Pat}^*)$.*

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 $\mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b})$ of P, 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 *of P if it is an effective subpattern of P and an R-pattern.*

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 a symbolic protocol Π closely resembles the one in [CKKW06b]. A symbolic trace for a k-party protocol Π 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 φ is the set of messages the adversary learned so far (the *adversary knowledge*, initially set to Nonce_{adv}), the set Sid contains the ids of all sessions currently running, and the function f maps every session id *sid* in 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 σ that maps variables to (fragments of) messages received by the agent a in that session. See the discussion of the role tree on page 13.

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 φ in the next global state will be extended by $\{\mathsf{dk}(a_1), \ldots, \mathsf{dk}(a_l), \mathsf{sk}(a_1), \ldots, \mathsf{sk}(a_l)\}$, 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 Sid. The local state of *sid* is initialized as $(i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where ε is the root of the role tree $\Pi(i)$ and σ maps the variables A_j to a_j and the variables X_N^j to fresh nonces. In other words, a new session is initialized in which agent a_i runs role $\Pi(i)$ and supposedly interacts with agents $a_1, \ldots, a_k$. 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 φ. 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 13. If no such edge is found, both the local state as well as the knowledge of the adversary are unmodified. A $\mathbf{send}(sid, m)$ transition can only be taken if the adversary knows m, i.e., if $\varphi \vdash m$. Note that the only change with respect to the modeling in [CKKW06b] 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 traces* or *Dolev-Yao traces*. The set of Dolev-Yao traces for Π is denoted $\mathsf{Exec}^s(\Pi)$. The full definition is given in the next section.

3.1 Postponed definitions

In the following, let $A_i \in \mathsf{X}.a$ be pairwise distinct agent variables (for $i \in \mathbb{N}$), and let $X_N^j \in \mathsf{X}.n$ be pairwise distinct nonce variables (for $j \in \mathbb{N}$). Assume that $\mathsf{X}.a \setminus \{A_i : i \in \mathbb{N}\}$ and $\mathsf{X}.n \setminus \{X_N^j : j \in \mathbb{N}\}$ 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 $\{A_j : j \in [k]\} \cup \{X_N^j : j \in \mathbb{N}\}$ (where k is the number of parties).

Definition 4 (Role)**.** *A* role tree *RT is an ordered edge-labeled finite tree where each edge is labeled by an agent rule (l, r) where $l, r \in \mathsf{Pat}$. The patterns l are called* parse-patterns *and the patterns r are called* construct-patterns*.*

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

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 $\{t\}^{\mathsf{-}}_{t'}$ it holds that $t' = \mathsf{ek}(A_i)$.*
3. *For any subterm of l of the form $\{t\}^{\mathsf{Rand}}_{t'}$ it holds that t and t' do not contain free variables nor $\_$.*
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 nor $\_$.*
5. *r does not contain $\_$ nor free variables that are not free in l.*
6. *For any subterm of l or r of the form $\mathsf{ZK}^R_F(\ldots;\ldots;\underline{b})$ we have that if $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ is a subterm of F, then $b_i \in \mathsf{X}.a$.*
7. *For any subterm of l of the form $[t]^{\mathsf{Rand}}_{t'}$ it holds that $t' = \mathsf{sk}(A_i)$ and that t does not contain free variables nor $\_$.*
8. *For any subterm of r of the form $[t]^{\mathsf{Rand}}_{t'}$ it holds that $t' = \mathsf{sk}(A_i)$.*
9. *For any subterm of l or r of the form $\mathsf{dk}(t)$ it holds that $t = A_i$.*

Note that the syntax of Pat (page 12) imposes additional constraints that are not explicit in Definition 4.

Definition 5 (Symbolic protocol execution)**.** *Let a k-party protocol Π be given.*

A global state *is a triple $(\mathsf{Sid}, f, \varphi)$ where φ is a set of messages (the* adversary knowledge*), Sid is a finite set of session ids, and the function f maps every session id sid in 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 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.*

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

- 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 ε is the root of the role tree $\Pi(i)$ and the substitution σ is defined by $\sigma(A_j) := a_j$ for all $j \in [k]$ and $\sigma(X^j_N) := n^{a_i, j, sid}$ for every X^j_N 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* φ' *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* θ *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 trace *or* Dolev-Yao trace *for* Π. *The set of Dolev-Yao traces for* Π *is denoted* $\mathsf{Exec}^s(\Pi)$.

4 The Computational Model

Cryptographic assumptions. At this point, we list all the properties that the encryption scheme, the signature scheme, and the zero-knowledge proof system used in the computational model need to fulfill.

We assume that $\mathcal{ZK}$ is a symbolically-sound zero-knowledge proof system (Definition 1).

We assume that the encryption scheme $\mathcal{AE}$ has the following properties:

- *Valid encryption/decryption key pairs.* It is well-defined which is the decryption key belonging to a given encryption key. Formally, there is a function f such that (a) given ek and sk, it can be verified in deterministic polynomial time whether $sk = f(ek)$ and (b) with probability 1, the key generation algorithm produces a key pair (ek, sk) with $sk = f(ek)$. We call (ek, sk) with $sk = f(ek)$ a *valid encryption/decryption key pair.*
- *Perfect correctness.* If (ek, sk) is a valid encryption/decryption key pair, then for any message m, we have that encrypting m under ek and then decrypting the ciphertext using sk succeeds yields m with probability 1.
- *Deterministic decryption.* The decryption algorithm is deterministic.
- *IND-CCA.* The encryption scheme satisfies the IND-CCA security notion. Roughly, IND-CCA security means that the adversary that has access to an encryption oracle and a decryption oracle cannot distinguish between the case that the encryption oracle encrypts the message m it gets from the adversary, or the case where the encryption oracle encrypts a random message m' with $|m| = |m'|$. (Assuming that the adversary never queries the decryption oracle with any ciphertext produced by the encryption oracle.)[13] See [Gol04] for details (there IND-CCA is called indistinguishability of encryptions under a posteriori chosen ciphertext attacks).

We assume that the signature scheme $\mathcal{SIG}$ has the following properties:

- *Deterministic verification.* The verification algorithm is deterministic.
- *Perfect correctness.* For any message m and a verification/signing key pair (vk, sk) returned by the key generation algorithm, signing m using sk and then verifying the resulting signature using vk succeeds with probability 1.

[13] This notion is similar in structure to the extraction zero-knowledge property presented on page 6.

- *Unpredictability.* The signature scheme $\mathcal{SIG}$ is unpredictable (in the sense that for a fixed signature key and a fixed message, two honestly generated messages are equal only with negligible probability). This implies that $\mathcal{SIG}$ has to be probabilistic. Alternatively, one could have required $\mathcal{SIG}$ to be deterministic and removed the symbolic randomness in the symbolic model. Then, however, we would have to exclude probabilistic signature schemes.

- *Length-regularity.* The signature scheme $\mathcal{SIG}$ is length regular in the following sense: Given an honestly generated signing key sk and two messages m_1, m_2 of the same length, the signatures for m_1 and m_2 will have the same length, too. See the discussion on length-regularity on page 7.

- *Strong existential unforgeability.* The signature is strongly existentially unforgeable. Roughly, strong existential unforgeability means that no adversary with access to a signing oracle can produce a valid message/signature pair (m, σ) unless that pair was output by the signing oracle. (I.e., the adversary is not even allowed to produce a signature σ' for m if he received a signature $\sigma \neq \sigma'$ for m from the signing oracle.)[14] See [Gol04] for details (there strongly existentially unforgeable signatures are called super-secure signatures).

Note that for the encryption scheme $\mathcal{AE}$, we also need the unpredictability and the length-regularity properties. In the case of encryption schemes, however, they follow directly from the IND-CCA property, so we do not need to require them explicitly.

We assume all security properties to hold against *nonuniform* polynomial-time adversaries.

Computational execution. We now proceed to define the computational execution of a protocol Π. We use the same formalization of protocols Π as in the symbolic 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.

We first give a short overview over the computational execution model, stressing the design issues particular to the include of zero-knowledge proofs. We give the detailed definitions in Section 4.1.

Fix a security parameter $\eta \in \mathbb{N}$. A computational trace is a sequence of *computational global states* of the form (Sid, g, C) where Sid is the set of session ids, g maps sessions ids to computational local states and C is the list of corrupted agents. A *computational local state* is of the form $(i, \tau, p, (a_1, \ldots, a_k))$. As with the symbolic 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 τ 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 computational local state $(i, \tau, p, (a_1, \ldots, a_k))$ is allocated where in the mapping τ the variables A_j and X_N^j 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 τ (see below). This results in a new substitution τ' where the variables that were free in l are now assigned bitstrings. Then the pattern r is used with the variable assignments τ' to construct a new bitstring m' (see below). If both parsing and constructing succeed, this edge is taken, τ' 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.

[14] We require strong existential unforgeability instead of existential unforgeability. The latter notion does not exclude that the adversary may modify the randomness of existing honestly generated signatures. As the symbolic adversary cannot do this (because no corresponding rule exists in Figure 1), we need to use strong existential unforgeability. (Alternatively, we could also have given the adversary additional power in Figure 1 as was done in [CW05].)

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.[15] Then we define a function construct(r,τ) taking a pattern r and a partial mapping $\tau : \mathsf{X} \to \{0,1\}^*$. If, e.g., r is an encryption $r = \{r'\}^R_{\mathsf{ek}(b)}$, the function construct$(r,\tau)$ recursively invokes $m' :=$ construct(r',τ) and then encrypts m' using the public key of agent b and using randomness $tape^{a,sid}(R)$ for the encryption algorithm (here a denotes the agent that invokes construct). Pairs and zero-knowledge proofs are handled similarly. If $r \in \mathsf{X}$, then construct(r,τ) just returns the stored value $\tau(r)$. We give the details of the function construct in Definition 7 in Section 4.1 below. At this point, we would only like to comment on the operation of 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 $r_1,\dots,r_s;a_1,\dots,a_n$ and public part $b_1,\dots,b_m$ (where we assume that $\underline{r},\underline{a},\underline{b}$ have already been assigned bitstrings using recursive calls to construct), we first have to construct a circuit C whose satisfiability we will prove in zero-knowledge (such a circuit we call a ZK-circuit). 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 $a'_1,\dots,a'_n$ of lengths $l_1,\dots,l_n$ and arguments $r'_1,\dots,r'_s$ all of length η, and then performs the operations described by the ZK-formula F where ρ_i is instantiated with the input r'_i, α_i with input a'_i, and occurrences of β_i are replaced with the *hardcoded* value b_i. Details are given in Definition 6 in Section 4.1 below. Then the prover of the zero-knowledge scheme is invoked for the circuit C and for the witness $\underline{r},\underline{a}$ (as bitstrings) using randomness $tape^{a,sid}(R)$. Call the resulting proof z. Then the bitstring returned by 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.[16] If the zero-knowledge proof fails (because $\underline{r};\underline{a}$ is not a witness for C) the function 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 parse(m,l,τ) 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 parse(m',l',τ) is invoked. Pairs, signatures, and zero-knowledge proofs are handled analogously. When l is a free variable (i.e., unassigned in τ), it is checked whether m is of the right type and then assigned to $\tau(l)$ (resulting in an extended mapping τ). If l is a bound variable (assigned in τ), it is checked whether $m = \tau(l)$. If l is of the form $\{\cdot\}^R_\cdot$, $[\cdot]^R_\cdot$, or $\mathsf{ZK}^R_\cdot(\dots)$ (i.e., contains explicit randomness), the message m is not parsed further but compared to construct(l,τ) (this enables matching against encryptions or ciphertexts an agent has produced itself). Finally, if all checks succeeded, the (now possibly extended) mapping τ is returned. We give a detailed definition of the function parse in Definition 8 in Section 4.1.

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, signatures, 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 symbolic case where only true proofs can be assigned to $\mathsf{X}.z$. For analogous reasons, only verified signatures are of type signature.

Thus for any adversary $\mathcal{A}$ and any security parameter η, we get a distribution on computational traces which we denote by $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$. A detailed description of the computational

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

[16] 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.

execution is given in the next section.

4.1 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 identify the set A of agent identifiers with the set $\{0,1\}^*$ of all bitstrings. This implies, that in the computational setting, any bitstring is a valid agent identifier.

Tags and types. To be able to recognize whether a given bitstring is a ciphertext, signature, agent-id, zero-knowledge proof, etc., we use tagging. For this, we assume functions tag_x which intuitively take a tuple of bitstrings and returns a bitstring tagged with x. Formally, we assume that tag_x is efficiently computable, injective, and efficiently invertible, and that the ranges of tag_x and tag_y for $x \neq y$ are disjoint. We also assume that tag_x is length-regular in the sense that the length of $tag_x(a,b,c,\dots)$ depends only on the lengths of the bitstrings $a,b,c,\dots$. (See also the discussion on length-regularity on page 7.)

We will use the tags $sig, enc, zk, pair, nonce, agent, dk, ek, vk, sk$ for signatures, ciphertexts, zero-knowledge proofs, pairs, nonces, agent ids, decryption keys, encryption keys, verification keys, and signing keys, respectively.

Based on the tags, we can define types of bitstrings:

- If $m = tag_{sig}(s, m', vk)$ and s is a valid signature for m' with respect to the verification key vk, we say m has type signature.
- If $m = tag_{zk}(z, F, s, n, \underline{l}, \underline{b})$ and the 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, then we say m has type zero-knowledge.[17]
- If $m = tag_{enc}(m', ek)$, we say m has type ciphertext.
- If $m = tag_{pair}(m_1, m_2)$, we say m has type pair.
- If $m = tag_{nonce}(n)$, we say m has type nonce.
- If $m = tag_{agent}(a)$, we say m has type agent.
- If $m = tag_{dk}(k)$, $m = tag_{ek}(k)$, $m = tag_{vk}(k)$, or $m = tag_{sk}(k)$ we say m has type decryption key, encryption key, verification key, or signing key, respectively.

Note that the types signature and zero-knowledge check more than the tagging and the arity, they additionally ensure that the signature or the zero-knowledge proof are successfully verified.

Constructing bitstrings. We now present the definitions that are used in the process of constructing bitstrings from terms.

Definition 6 (ZK-circuits)**.** *Fix a security parameter η. Let a ZK-formula F of ρ-arity s, α-arity n and β-arity m be given, as well as bitstrings $b_1,\dots,b_m \in \{0,1\}^*$. Let $l_1,\dots,l_n \in \mathbb{N}$.*

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

- *It expects inputs $r_1,\dots,r_s$ of length η, and inputs $a_1,\dots,a_n$ of lengths $l_1,\dots,l_n$, respectively. (We assume that these inputs are concatenated to form a single bitstring w of length $s\eta + \sum l_i$.)*
- *If ($T = \mathsf{ek}(\beta_i)$ or $T = \mathsf{vk}(\beta_i)$ or $T = \{\cdot\}^{\cdot}_{\mathsf{ek}(\beta_i)}$) and b_i does not have type agent, then C is undefined.*

[17] We assume a single, globally available CRS that is used by the verifier of $\mathcal{ZK}$, cf. Definition 9.

- *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).*[18]
- *If $T = \mathsf{ek}(\beta_i)$ and b_i has type agent, then let $tag_{agent}(b_i') := b_i$ and C returns $tag_{ek}(ek)$ where ek is the encryption key of agent b_i' (i.e., C computes a constant function).*
- *If $T = \mathsf{vk}(\beta_i)$ and b_i has type agent, then let $tag_{agent}(b_i') := b_i$ and C returns $tag_{vk}(vk)$ where vk is the verification key of agent b_i' (i.e., C computes a constant function).*
- *If $T = \langle T_1, T_2\rangle$, then $C(w)$ computes $m_1 := C^{s,n,\underline{l}}_{T_1,\underline{b}}(w)$ and $m_2 := C^{s,n,\underline{l}}_{T_2,\underline{b}}(w)$, and then returns the bitstring $tag_{pair}(m_1, m_2)$.*
- *If $T = \{T'\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$ and b_j has type agent, then let $tag_{agent}(b_j') := b_j$ and $C(w)$ computes $m' := C^{s,n,\underline{l}}_{T',\underline{b}}(w)$ and then computes the encryption c of m' under the encryption key ek of b_j' using randomness r_i. It returns the bitstring $tag_{enc}(c, ek)$.*

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

- *It expects inputs $r_1, \ldots, r_s$ of length η, and inputs $a_1, \ldots, a_n$ of lengths $l_1, \ldots, l_n$, respectively.*
- *If T is of the form $T_1 = T_2$ where T_1 and T_2 are ZK-terms, then C computes $m_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}(\underline{r}, \underline{a})$ for $i = 1, 2$ and returns the bit 0 if $m_1 \neq m_2$ and the bit 1 if $m_1 = m_2$.*
- *If T is of the form $\mathrm{Decrypt}(T_1, T_2, T_3) = T_4$ then C computes $m_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}(\underline{r}, \underline{a})$ for $i = 1, \ldots, 4$. Then C checks whether m_1 is of type ciphertext, m_2 of type decryption key, and m_3 of type encryption key. C assigns $tag_{enc}(c, ek') := m_1$, $tag_{dk}(dk) := m_2$, $tag_{ek}(ek) := m_3$. Then C checks whether $ek' = ek$ and (ek, dk) is a valid encryption/decryption key pair and whether decrypting c using the decryption key dk succeeds and yields the plaintext m_4. If all checks succeed, C returns 1. Otherwise, C returns 0.*
- *If T is of the form $\mathrm{Verify}(T_1, T_2, T_3)$ then C computes $m_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}(\underline{r}, \underline{a})$ for $i = 1, \ldots, 3$. Then C checks whether m_1 is of type signature and m_3 of type verification key. C assigns $tag_{sig}(s, m', vk) := m_1$ and $tag_{vk}(vk') := m_3$. Then C checks whether $m' = m_2$ and whether $vk = vk'$. If all checks succeed, C returns 1. Otherwise, C returns 0.*[19]

ZK-formulas:

For a ZK-formula $F = B(T_1, \ldots, T_q)$ where B is a Boolean formula and $\underline{T}$ are ZK-atoms, the circuit $C^{s,n,\underline{l}}_{F,\underline{b}}$ is defined as follows:

- *It expects inputs $r_1, \ldots, r_s$ of length η, and inputs $a_1, \ldots, a_n$ of lengths $l_1, \ldots, l_n$, respectively.*
- *If s is not the ρ-arity of F, or n is not the α-arity of F, or $|\underline{l}| \neq n$, or $|\underline{b}|$ is not the β-arity of F, then C is undefined.*
- *C computes $t_i := C^{s,n,\underline{l}}_{T_i,\underline{b}}$ for $i \in [q]$ and returns $B(\underline{t})$. (If one of the circuits constructed recursively in the process is undefined, then C is undefined, too.)*

Note that by construction, $C^{s,n,\underline{l}}_{F,\underline{b}}$ is length-regular in the bitstrings $\underline{b}$. I.e., if $\underline{b}$ and $\underline{b}'$ satisfy $|b_i| = |b_i'|$ for all i, then $C^{s,n,\underline{l}}_{F,\underline{b}} = C^{s,n,\underline{l}}_{F,\underline{b}'}$. This property is important for the computational soundness result, see the discussion on length-regularity on page 7. It is important that the length-regularity of the construction is also present in an implementation. In particular, when

[18] Here it is important that for different values b_i of the same length, the corresponding circuits have the same size. See the discussion on length-regularity on page 7.

[19] Note that the type check of m_1 already implies an invocation of the verification algorithm.

performing some optimization of the circuit $C^{s,n,\underline{l}}_{F,\underline{b}}$ at runtime (e.g., identifying and merging subcircuits that appear more than once, or removing dead "code"), it is important that the size of the overall circuit does not depend on the actual values of the $\underline{b}$.

Without loss of generality, we also assume an encoding of $C^{s,n,\underline{l}}_{F,\underline{b}}$ that uniquely determines $F, s, n, \underline{l}, \underline{b}$.

Note also the close relation between the validity of a formula (as in Definition 2) and the definedness of $C^{s,n,\underline{l}}_{F,\underline{b}}$: For any ZK-formula F and any $\underline{r} \in \mathsf{Rand}$, $\underline{a}, \underline{b} \in \mathsf{M}$, the term $F\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ is valid if and only if for each subterm $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ of F, we have that $b_i \in \mathsf{A}$. And for any ZK-formula F and $\underline{b} \in \{0,1\}^*$, the circuit $C^{s,n,\underline{l}}_{F,\underline{b}}$ is defined if and only if for each subterm $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ of F the bitstring b_i is of type agent and $s, n, |\underline{b}|$ is the ρ-, α-, and β-arity of F, respectively, and $|l| = n$.

Definition 7 (Constructing bitstrings)**.** *Let a session id sid, an agent $a \in \mathsf{A}$, a pattern r, and a mapping τ 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 $\tau(x)$ is not defined, abort. Let $tag_{agent}(a') := \tau(x)$. Retrieve the encryption key ek belonging to agent id a'. (Note that since $\mathsf{A} = \{0,1\}^*$, a' is always a valid agent id.) Return $tag_{ek}(ek)$.

Case $r = \mathsf{vk}(x)$: If $\tau(x)$ is not defined, abort. Let $tag_{agent}(a') := \tau(x)$. Retrieve the verification key vk belonging to agent id a'. (Note that since $\mathsf{A} = \{0,1\}^*$, a' is always a valid agent id.) Return $tag_{vk}(vk)$.

Case $r = \mathsf{dk}(x)$: If $\tau(x)$ is not defined, abort. Let $tag_{agent}(a') := \tau(x)$. If $a \neq a'$, abort. Retrieve the decryption key dk belonging to agent id a'. If no such decryption key exists, abort. Otherwise return $tag_{dk}(dk)$.[20]

Case $r = \langle r_1, r_2\rangle$: Set $m_i := \mathrm{construct}^{a,sid}(r_i,\tau)$ for $i = 1,2$. Return $tag_{pair}(m_1,m_2)$.

Case $r = \{r'\}^R_{\mathsf{ek}(x)}$: Let $\tilde{R} := tape^{a,sid}(R)$, If $\tau(x)$ is not defined, abort. Let $tag_{agent}(a') := \tau(x)$ and let ek be the encryption key of agent a'. If a' is undefined, abort. Otherwise invoke $m' := \mathrm{construct}^{a,sid}(r',\tau)$. Compute the encryption m of m' under encryption key ek using the randomness $\tilde{R}$. Return $tag_{enc}(m, ek)$.

Case $r = [r']^R_{\mathsf{sk}(x)}$: Let $\tilde{R} := tape^{a,sid}(R)$. If $\tau(x)$ is not defined, abort. Let $tag_{agent}(a') := \tau(x)$ and let sk and vk be the signing and the verification key of agent a'. If $a \neq a'$, abort. Otherwise invoke $m' := \mathrm{construct}^{a,sid}(r',\tau)$. Compute the signature m of m' using the signing key sk and randomness $\tilde{R}$. Return $tag_{sig}(m, m', vk)$.

Case $r = \mathsf{ZK}^R_F(R_1,\dots,R_s; a_1,\dots,a_n; b_1,\dots,b_m)$: For each subterm T of each ZK-term of F, compute $\mathrm{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$ *and discard the result.*[21] *Let $\tilde{R} := tape^{a,sid}(R)$ and $\tilde{R}_i := tape^{a,sid}(R_i)$ for $i \in [s]$. For all $i \in [n]$, compute $\tilde{a}_i := \mathrm{construct}^{a,sid}(a_i,\tau)$, and for all $i \in [m]$, compute $\tilde{b}_i := \mathrm{construct}^{a,sid}(b_i,\tau)$. Set $l_i := |\tilde{a}_i|$ for all $i \in [n]$, and let $C := C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$. Set $w := \tilde{R}_1\|\dots\|\tilde{R}_s\|\tilde{a}_1\|\dots\|\tilde{a}_n$. If C is undefined or $C(w) \neq 1$, abort. Use the prover P of $\mathcal{ZK}$ to produce a proof z for circuit C and witness w where the prover uses the randomness $\tilde{R}$.*[22] *If z is not a valid proof for z (as determined by the verifier of $\mathcal{ZK}$), we say a ZK-failure occurred. Otherwise return $tag_{zk}(z, F, s, n, \underline{l}, \underline{\tilde{b}})$.*

[20] We do not need a corresponding rule for signing keys because the syntax of patterns forbids the use of signing keys except as part of a signature (and in the case of signatures, $\mathrm{construct}^{a,sid}([\cdot]^{\cdot}_{\mathsf{sk}(a)},\tau)$ does not perform a recursive invocation of $\mathrm{construct}^{a,sid}(\mathsf{sk}(a),\tau)$). In contrast, decryption keys can be used in the witness of a zero-knowledge proof.

[21] This computation has no effect (except for that $\mathrm{construct}^{a,sid}(r,\tau)$ might fail because an invocation $\mathrm{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$ fails), but it ensures that any bitstring occurring in the computation of $C(w)$ below will already have been computed by $\mathrm{construct}^{a,sid}$. This will allow for the construction of a function $\bar{c}$ later in the proof of the main theorem. When parsing bitstrings, this function performs a lookup in the list of values computed by $\mathrm{construct}^{a,sid}$, so we need that list to contain all relevant bitstrings.

[22] We assume a single, globally available CRS that is used by the prover, cf. Definition 9.

In any other case, abort (and if any of the recursive invocations of $\text{construct}^{a,sid}$ *aborts, abort, too).*

Parsing bitstrings. The next definitions specifies how incoming messages are parsed according to a pattern.

Definition 8 (Parsing bitstrings)**.** *Let a bitstring m, a pattern l, and a mapping τ from variables to bitstrings be given. We define* $\text{parse}^{a,sid}(m,l,\tau)$ *recursively as follows:*

Case $l = x \in X$: If the type of m does not match the sort of x,[23] *abort. If $\tau(x) = m$, then return τ. 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 encryption key, abort. Otherwise, let $tag_{ek}(ek) := m$. If there is an agent id b such that ek is the encryption key of b and $\tau(x)$ is not defined or equals $tag_{agent}(b)$, return $\tau[x := tag_{agent}(b)]$. Otherwise abort.

Case $l = \mathsf{vk}(x) \in \mathsf{vk}(X.a)$: If m is not of type encryption key, abort. Otherwise, let $tag_{vk}(vk) := m$. If there is an agent id b such that vk is the verification key of b and $\tau(x)$ is not defined or equals $tag_{agent}(b)$, return $\tau[x := tag_{agent}(b)]$. Otherwise abort.

Case $l = \langle l_1, l_2 \rangle$: If m is not of type pair, abort. Otherwise let $tag_{pair}(m_1, m_2) := m$ and let $\tau' := \text{parse}^{a,sid}(m_1, l_1, \tau)$ and $\tau'' := \text{parse}^{a,sid}(m_2, l_2, \tau)$. If τ' and τ'' are not compatible, abort. Otherwise return $\tau' \cup \tau''$.

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

Case $l = \{l'\}^{\_}_{\mathsf{ek}(A_j)}$ with $j \in [k]$: If m is not of type ciphertext, abort. Otherwise, let $tag_{enc}(c, ek) := m$. If ek is not the encryption key belonging to the agent id $\tau(A_j)$, abort. Otherwise decrypt c with the secret key belonging to agent a and let m' be the corresponding ciphertext.[24] *If this fails, abort. Otherwise let $\tau' := \text{parse}^{a,sid}(m', l', \tau)$ and return τ'.*

Case $l = [l']^{\_}_{\mathsf{sk}(x)}$ with $x \in X.a$: If m is not of type signature, abort. Otherwise, let $tag_{sig}(s, m', vk) := m$. Let $\tau' := \text{parse}^{a,sid}(tag_{vk}(vk), \mathsf{vk}(x), \tau)$ and $\tau'' := \text{parse}^{a,sid}(m', l', \tau)$. If τ' and τ'' are not compatible, abort. Otherwise, return $\tau' \cup \tau''$.

Case $l = \mathsf{ZK}^{\_}_F(\_^*; \_^*; t_1, \ldots, t_n)$: If m is not of type zero-knowledge proof, abort.[25] *Otherwise, let $tag_{zk}(z, F', s, n', \underline{l}, \underline{b}) := m$. If $F \neq F'$ or $n \neq n'$, abort. Otherwise for $i = 1, \ldots, n$ run the following: $\tau_i := \text{parse}^{a,sid}(b_i, t_i, \tau)$. If the τ_i are not pairwise compatible, abort. Otherwise return $\bigcup_i \tau_i$.*

In any other case, abort (and if any of the recursive invocations of $\text{parse}^{a,sid}$ *or* $\text{construct}^{a,sid}$ *aborts, abort, too).*

Definition 9 (Computational execution model)**.** *A* computational global state *is a triple (Sid, g, C) where Sid is a finite set of session ids, and g is a function mapping every $sid \in \mathsf{Sid}$ to a computational local state, and C is the set of corrupted parties.*

A computational 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 $\tau : \mathsf{X} \to \{0,1\}^*$ 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 computational trace $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$ *for security parameter η is a (distribution over) sequences of global states given by the following algorithm.*

- *Initially, pick $(crs, simtd, extd)$ using the key generation algorithm K from $\mathcal{ZK}$. Discard $simtd$ and $extd$. The CRS crs is made available to the adversary $\mathcal{A}$. Proofs and ZK-verifications performed by* $\text{construct}^{a,sid}$ *and* $\text{parse}^{a,sid}$ *use this CRS.*
- *Whenever an encryption, decryption, signing, or verification key of some agent $a \in \mathsf{A}$ is used for the first time, the corresponding key pair is generated using the key generation*

[23] If $x \in \mathsf{X}.z$ or $x \in \mathsf{X}.s$, this implies invoking the verification algorithm of $\mathcal{ZK}$ or $\mathcal{SIG}$.
[24] Note that due to Definition 4, condition 2, we always have $\tau(A_j) = tag_{agent}(a)$ here.
[25] By our definition of the type zero-knowledge proof, this implies invoking the verification algorithm of $\mathcal{ZK}$.

algorithm of $\mathcal{AE}$ or $\mathcal{SIG}$, respectively. We further give $\mathcal{A}$ access to the encryption and verification keys of all agents. We also give $\mathcal{A}$ and all agents the possibility to perform a reverse lookup: Given an encryption or verification key, it is possible to lookup the corresponding agent id.

- *The initial global state is $(\varnothing, \varnothing, \varnothing)$. In the first step, $\mathcal{A}$ is invoked with input 1^η.*
- *When $\mathcal{A}$ outputs* **corrupt**$(a_1, \ldots, a_l)$ *with $a_1, \ldots, a_l \in \mathsf{A}$ in its first activation, the adversary is given the decryption and signing keys of $a_1, \ldots, a_l$ as input. The next global state is $(\varnothing, \varnothing, \{a_1, \ldots, a_l\})$.*
- *When $\mathcal{A}$ outputs* **new**$(i, a_1, \ldots, a_k)$ *in global state (Sid, g, C) where $i \in [k]$ and $a_1, \ldots, a_k \in \mathsf{A}$, the next global state is (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 ε is the root of the role tree $\Pi(i)$ and the substitution τ is defined by $\tau(A_j) := tag_{agent}(a_j)$ for all $j \in [k]$ and $\tau(X_N^j) := tag_{nonce}(n_j)$ for every X_N^j occurring in $\Pi(i)$ where the n_j are independently uniformly chosen $n_j \in \{0,1\}^\eta$. The adversary is given empty input.
- *When $\mathcal{A}$ outputs* **send**(sid, m) *in global state (Sid, g, C) where $sid \in \mathsf{Sid}$, the next global state is (Sid, g', C).*

  Here $g'(sid') := g(sid')$ for all $sid \neq sid'$, $g'(sid) := (i, \tau', p', (a_1, \ldots, a_k))$ is computed from $(i, \tau, p, (a_1, \ldots, a_k)) := g(sid)$ 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 $\tau'' := \mathrm{parse}^{a_i, sid}(m, l, \tau)$. Then invoke $m' := \mathrm{construct}^{a_i, sid}(r, \tau'')$. If $\mathrm{parse}^{a_i, sid}$ or $\mathrm{construct}^{a_i, sid}$ aborts, 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 τ', 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 computational trace ends at this point.*

5 Computational Soundness

In the preceding two sections, we have described the symbolic and the computational execution model involving zero-knowledge proofs and encryptions. To be able to state our main computational soundness result, we have to formalize the statement that a given computational trace t^c corresponds to a given symbolic trace t^s. Here we follow [MW04, CW05, CKKW06b] 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 [CKKW06b], except that we add the requirement that the adversary corrupts the same agents in the symbolic and the computational trace.

Definition 10 (Computational instantiations)**.** *Let $t^s = (\mathsf{Sid}_1^s, f_1, \varphi_1), \ldots, (\mathsf{Sid}_m^s, f_m, \varphi_m)$ be a symbolic trace and $t^c = (\mathsf{Sid}_1^c, g_1, C_1), \ldots, (\mathsf{Sid}_n^c, g_n, C_n)$ a computational trace.*

We say that the trace t^c is a computational 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 $\mathsf{Sid}_\ell^s = \mathsf{Sid}_\ell^c$, and $C_\ell = \{a : \mathsf{dk}(a) \in \varphi_\ell\}$, and for every $sid \in \mathsf{Sid}_\ell^s$ the following holds:*

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

We say that t^c is a computational 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 computational trace is a computational instantiation of some symbolic Dolev-Yao trace.

Theorem 1 (Computational soundness of zero-knowledge proofs)**.** *Assume that the encryption scheme $\mathcal{AE}$, the signature scheme $\mathcal{SIG}$, and the zero-knowledge proof system $\mathcal{ZK}$ satisfy the requirements listed on page 17.*

Let Π be a k-party protocol. Let $\mathcal{A}$ be a nonuniform polynomial-time adversary. Then the following probability is overwhelming in η:

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

We first give a proof sketch of the theorem that covers the main ideas. The full proof is then given in Section 6.

Proof sketch. To establish the theorem, it is sufficient to find an injective mapping $\bar{c}$ that maps bitstrings to terms such that a computational 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.[26] 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 m 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 because the adversary sends 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 ℓ, a message m is sent such that

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

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 computational execution in several steps into an adversary that guesses some bitstrings of high entropy or that fakes a signature.

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 $\mathit{Proof}(x, w)$; when the implementation of $\bar{c}$ extracts the witness of some zero-knowledge proof z, it invokes $\mathit{Extract}(z)$. Note that for this, it must be guaranteed that each zero-knowledge proof produced by honest agents uses a different randomness R,[27] 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 $\mathsf{ZK}^R_F(\dots)$, then R

[26] E.g., the Cramer-Shoup cryptosystem, cf. footnote 11.

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

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 (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:

- 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 computational 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.[28] Whether a given symbolic trace is a Dolev-Yao trace can be checked in polynomial-time. Thus from the computational indistinguishability of the symbolic traces in both cases, it follows that the probability that the symbolic 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 encrypts random messages (and thus does not use 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 a underivable subterm. In order to derive a contradiction from (1), we have to identify a subterm of $\bar{c}(m)$ whose "fault" it is that $\bar{c}(m)$ cannot be derived. We will then use this term to get a contradiction. 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 ℓ, and let $S' := S \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv}$ (the knowledge of the adversary after that step). Assume that $S' \vdash M$ (this follows from (1)).*

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

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

Furthermore, we have that $S' \nvdash T$ and that T satisfies one of the following conditions:

(a) $T \in \mathsf{Nonce}_{ag}$, or

[28] Note that here it is important that $\mathcal{ZK}$ has the extraction zero-knowledge property and not only the zero-knowledge property and the extractability individually, as we replace *Proof* by *Simulate* in the presence of calls to an extraction oracle.

(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, $S' \nvdash_r r_i$.

(e) $T = [\cdot]^{\mathsf{Rand}_{adv}}_{\mathsf{sk}(a)}$ *where* $a \notin C \cup \mathsf{Garbage}$.

(f) $T = [\cdot]^{\mathsf{Rand}_{ag}}_{\mathsf{sk}(a)}$.

(g) $T = \mathsf{sk}(a)$ *or* $T = \mathsf{dk}(a)$ *for some* $a \in \mathsf{A} \setminus C$.

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

***T* is a nonce.** In case (a) we have $T \in \mathsf{Nonce}_{ag}$. Since $S' \nvdash T$, 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 $\langle\cdot,\cdot\rangle$, $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$, $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$, or $[\cdot]^{\cdot}_{\mathsf{sk}(\cdot)}$. The pair and the signature can directly be parsed (since we assume that signatures are always tagged with the signed message), 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 Rand_{adv} and not 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), (c), (f), (e), and (g) of Lemma 1 are taken care of similarly: In the first three cases, the adversary guesses some randomized message m that is never actually used; this contradicts the unpredictability of $\mathcal{AE}$, $\mathcal{ZK}$, or $\mathcal{SIG}$, respectively. In case (e) the adversary produces a signature m that was never produced by the signing oracle; this contradicts the strong existential unforgeability of $\mathcal{SIG}$. In case (g) the adversary guesses some secret key m.

***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 $S' \nvdash_r r_i$. 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 computational 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 encryption key $\mathsf{ek}(a)$ performed by an honest agent (otherwise the function $\bar{c}$ would have assigned a randomness $r_i \in \mathsf{Rand}_{adv}$ implying $S' \vdash_r r_i$). 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 $S' \nvdash c$. 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. □

6 Proof of the Main Theorem

Proof of Theorem 1. In the following, let a k-party protocol Π 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 η.

For this, we will give a construction of an injective mapping $\bar{c} : \{0,1\}^* \to \mathsf{M}$ that may depend on η as well as of some of the values occurring in $\mathsf{Exec}^c_{\Pi,\mathcal{A}}(\eta)$ such as the CRS, its extraction trapdoor, the private keys of the parties, or their random tapes. We extend this mapping to computational traces as follows: A computational trace $t^c = (\mathsf{Sid}_1, g_1, C_1), \ldots, (\mathsf{Sid}_n, g_n, C_n)$ is mapped to a symbolic trace $\bar{c}(t^c) := (\mathsf{Sid}_1, f_1, \varphi_1), \ldots, (\mathsf{Sid}_n, f_n, \varphi_n)$ as follows. We let $f_\ell(sid) = (i, \bar{c} \circ \tau, p, (a_1, \ldots, a_k))$ where $(i, \tau, p, (a_1, \ldots, a_k)) := g_\ell(sid)$, and we define the adversary's knowledge φ_ℓ as follows: We set $\varphi_1 := \mathsf{Nonce}_{adv}$. For $\varphi_{\ell+1}$ we distinguish three cases. If the ℓ-th transition was a **new**-transition, we set $\varphi_{\ell+1} := \varphi_\ell$. If the ℓ-th transition was a **corrupt**$(a_1, \ldots, a_l)$-transition, we set $\varphi_{\ell+1} := \varphi_\ell \cup \{\mathsf{dk}(a_j), \mathsf{sk}(a_j) : j \in [l]\}$. If the ℓ-th transition was a **send**(sid, m)-transition, let $\tilde{m}$ be the answer given to the adversary after taking that transition, and set $\varphi_{\ell+1} := \varphi_\ell \cup \{\bar{c}(\tilde{m})\}$ (or $\varphi_{\ell+1} := \varphi_\ell$ if the adversary gets an empty answer).

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 $c := \bar{c}^{-1}$ is an injective partial function, and if $\bar{c}(t^c)$ is a Dolev-Yao trace, by construction of $\bar{c}(t^c)$ we have that $\bar{c}(t^c) \preceq^c t^c$. (Here, we use that $\bar{c}(t^c)$ is a Dolev-Yao trace to ensure that no $\mathsf{dk}(a_j)$ or $\mathsf{sk}(a_j)$ occurs in φ_ℓ unless a_j was corrupted in the first step.) Thus, assuming that $\bar{c}(t^s)$ is a Dolev-Yao trace with probability $1 - \mu$, we have

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

Note that the construction of $\bar{c}$ actually depends on the particular execution of Π. For example, the construction of $\bar{c}$ may depend on the secret keys, or other random choices made during the execution. Hence also c will depend on these random choices. However, since Definition 10 only requires the existence of c, not that it can efficiently be computed from the information contained in t^c, such a randomness-dependent construction of c is sufficient.

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. This 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 parse function from Definition 8. However, in contrast to 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 to decrypt all ciphertexts. Moreover, $\bar{c}$ will have to assign symbolic randomness from 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.[29] 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 to 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} (in the case that m was generated by 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

[29] E.g., the Cramer-Shoup cryptosystem, cf. footnote 11.

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 $G : \{0,1\}^* \cup \{\bot\} \to \mathsf{Garbage}$ and write their arguments as superscripts. Intuitively, 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 is of type agent"*. Let $tag_{agent}(a) := m$ and return a.

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

Case *"m is of type verification key"*. Analogous to the case of encryption keys.

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

Case *"m is of type signing key"*. Analogous to the case of decryption keys.

Case *"m is of type pair"*. Let $tag_{pair}(m_1, m_2) := m$. 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_N by some honest agent a in some session sid. More exactly, check whether a global state (Sid, g, C) occurs in the trace where $g(sid) = (i, \tau, \cdot, (a_1, \ldots, a_n))$ for some i, τ, and $\underline{a}$, and $\tau(X^j_N) = m$ for some j. If so, return $n^{a_i,j,sid}$. (Remember that $n^{a_i,j,sid}$ is the nonce that is assigned in the symbolic model to the nonce variable X^j_N in session sid run by agent a_i.) Otherwise, return B^m_{adv}.

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 τ from variables to bitstrings. Let $t' := t\gamma^{a,sid}(\bar{c} \circ \tau)$.[30] Then return $\{t'\}^{r^{a,R,sid}}_{\mathsf{ek}(\bar{c}(\tau(x)))}$. (Remember that $r^{a,R,sid}$ is the randomness that is actually used in the symbolic model when agent a instantiates a pattern with randomness R in session sid.)

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

Case *"m is of type signature and m has been generated by an honest agent"*. I.e., m was the result of an invocation of $\mathrm{construct}^{a,sid}([t]^R_{\mathsf{sk}(x)}, \tau)$ by agent a in session sid for some $R \in \mathsf{Rand}_{ag}$, $x \in X.a$, and some mapping τ. Let $tag_{sig}(s, m', vk) := m$. Let $\mathsf{vk}(a) := \bar{c}(tag_{vk}(vk))$. Return $[\bar{c}(m')]^{r^{a,R,sid}}_{\mathsf{sk}(a)}$.

Case *"m is of type signature and m has not been generated by an honest agent"*. Let $tag_{sig}(s, m', vk) := m$. Let $\mathsf{vk}(a) := \bar{c}(vk)$. Return $[\bar{c}(m')]^{R^m_{adv}}_{\mathsf{sk}(a)}$.

[30] $t\gamma^{a,sid}(\bar{c} \circ \tau)$ denotes the application of the substitution $\bar{c} \circ \tau$ to the term $t\gamma^{a,sid}$, i.e., the variable x in $t\gamma^{a,sid}$ is replaced by $\bar{c}(\tau(x))$.

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 $\mathrm{construct}^{a,sid}(\mathsf{ZK}_F^R(R_1,\dots,R_l;a_1,\dots,a_n;b_1,\dots,b_s),\tau)$ 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 τ from variables to bitstrings. Let $a_i' := a_i\gamma^{a,sid}(\bar{c}\circ\tau)$ and $b_i' := b_i\gamma^{a,sid}(\bar{c}\circ\tau)$. Then return $\mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\dots,r^{a,R_l,sid};\underline{a}';\underline{b}')$. (Remember that $r^{a,R_i,sid}$ is the randomness that is actually used in the symbolic model when agent a instantiates a pattern with randomness R_i in session sid.)

Case *"m is of type zero-knowledge proof and m has not been generated by an honest agent"*.
Let $tag_{zk}(z,F,s,n,\underline{l},\underline{b}) := m$. Compute the circuit $C := C_{F,\underline{b}}^{s,n,\underline{l}}$. Apply the ZK extraction algorithm to z and the extraction trapdoor of $\mathcal{ZK}$ (cf. Definition 1) to extract a bitstring w with $C(w)=1$. If the extraction algorithm fails or $C(w)\neq 1$, return G^m. In this case, we say a *ZK-break occurred*. (We will later see that this happens only with negligible probability.) Parse w as $r_1\|\dots\|r_l\|a_1\|\dots\|a_n$ with $|r_i|=\eta$ and $|a_i|=l_i$. Let $b_i' := \bar{c}(b_i)$ and $a_i' := \bar{c}(a_i)$ for all i. For each $i\in[s]$, compute the symbolic randomness r_i' as follows: Let e be the unique subterm of F of the form $e=\{t'\}_t^{\rho_i}$. Let $e' := C_{e,\underline{b}}^{s,n,\underline{l}}(w)$ (this gives the bitstring that corresponds to e in an evaluation of C). Then e' will be a bitstring of type ciphertext. Let $e'' := \bar{c}(e')$. Then e'' will have the form $\{\dots\}_\dots^{R'}$. Set $r_i' := R'$. Finally, return $\mathsf{ZK}_F^{R_{adv}^m}(\underline{r}',\underline{a}',\underline{b}')$.

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

Later in the proof, we will also need the following function c^* that will turn out to be the inverse of $\bar{c}$:

- $c^*(a) := tag_{agent}(a)$ for $a\in\mathsf{A}$.
- $c^*(\mathsf{ek}(a)) := tag_{ek}(ek_a)$ for $a\in\mathsf{A}$ where ek_a is the encryption key of agent a and analogously for dk, vk, and sk.
- $c^*(\langle t_1,t_2\rangle) := tag_{pair}(c^*(t_1),c^*(t_2))$.
- $c^*(n^{a_i,j,sid}) := \tau(X_N^j)$ if there is a global state (Sid,g,C) with $g(sid)=(i,\tau,\cdot,(a_1,\dots,a_k))$.
- $c^*(t) := m$ for $t\in\{G^m,\mathsf{ek}(G^m),\mathsf{vk}(G^m),B_{adv}^m,\{\cdot\}_\cdot^{R_{adv}^m},[\cdot]_\cdot^{R_{adv}^m},\mathsf{ZK}_\cdot^{R_{adv}^m}(\dots)\}$.
- $t=\{t'\}_{\mathsf{ek}(a)}^{r^{a,R,sid}}$ for $R\in\mathsf{Rand}_{ag}$ and $a\in\mathsf{A}$. Then the bitstring $c^*(t)$ is produced like in a computation of $\mathrm{construct}^{a,sid}(\{\tilde{r}\}_{\mathsf{ek}(a)}^R,\tau)$ for some $\tilde{r}$ and τ with $\tau(x)=tag_{agent}(a)$, except that instead of computing the recursive $m' := \mathrm{construct}^{a,sid}(\tilde{r},\tau)$ (which is then encrypted to give m), we use $m' := c^*(t'\gamma^{a,sid}(\bar{c}\circ\tau))$ instead. (Note that this does not depend on the pattern $\tilde{r}$ and the mapping τ, since $\mathrm{construct}^{a,sid}(\{\tilde{r}\}_{\mathsf{ek}(a)}^R,\tau)$ uses $\tilde{r}$ and τ only as the argument to $\mathrm{construct}^{a,sid}(\tilde{r},\tau)$.)
- $t=[t']_{\mathsf{sk}(a)}^{r^{a,R,sid}}$ for $R\in\mathsf{Rand}_{ag}$ and $a\in\mathsf{A}$. Then the bitstring $c^*(t)$ is produced like in a computation of $\mathrm{construct}^{a,sid}([\tilde{r}]_{\mathsf{sk}(x)}^R,\tau)$ for some $\tilde{r}$ and τ with $\tau(x)=tag_{agent}(a)$, except that instead of computing the recursive $m' := \mathrm{construct}^{a,sid}(\tilde{r},\tau)$ (which is then signed), we use $m' := c^*(t'\gamma^{a,sid}(\bar{c}\circ\tau))$ instead.
- $t=\mathsf{ZK}_F^{r^{a,R,sid}}(\underline{r};\underline{a};\underline{b})$ for $R\in\mathsf{Rand}_{ag}$ and $a\in\mathsf{A}$. Then the bitstring $c^*(t)$ is produced like in a computation of $\mathsf{ZK}_F^{r^{a,R,sid}}(\underline{r};\underline{\hat{a}};\underline{\hat{b}})$ for some $\hat{a}_i,\hat{b}_i$, except that instead of computing the recursive $\tilde{a}_i := \mathrm{construct}^{a,sid}(\hat{a}_i,\tau)$ and $\tilde{b}_i := \mathrm{construct}^{a,sid}(\hat{b}_i,\tau)$ (which are then used for constructing witness and circuit), we use $\tilde{a}_i := c^*(a_i\gamma^{a,sid}(\bar{c}\circ\tau))$ and $\tilde{b}_i := c^*(b_i\gamma^{a,sid}(\bar{c}\circ\tau))$ instead.

Properties of $\bar{c}$. Note that $\bar{c}$ is injective: Fix some term $t=\bar{c}(m)$. We need to see that t has only one preimage. If t is a pair or an agent id this is obvious. If t is of the form $B_{adv}^{m'}$, $G^{m'}$, $R_{adv}^{m'}$, $\mathsf{ek}(G^{m'})$, $\mathsf{vk}(G^{m'})$, $\mathsf{sk}(G^{m'})$, $\mathsf{dk}(G^{m'})$, $\{\cdot\}_\dots^{R_{adv}^{m'}}$, $[\cdot]_\dots^{R_{adv}^{m'}}$, $\mathsf{ZK}_\dots^{R_{adv}^{m'}}(\dots)$, the preimage is determined by m'.

If $t=\mathsf{ek}(a)$ for $a\in\mathsf{A}$, then the preimage of t must be $tag_{agent}(a)$. Analogously for $\mathsf{dk}(a)$, $\mathsf{vk}(a)$, and $\mathsf{sk}(a)$.

For $t = \{\cdot\}^{r^{a,R,sid}}_{\ldots}$ we have that the preimage m was the result of a call $\text{construct}^{a,sid}(\{\ldots\}^{\tilde{R}}_{\ldots}, \tau)$. In the session *sid*, any two calls $\text{construct}^{a,sid}(\{\cdot\}^{R}_{\ldots}, \tau)$ and $\text{construct}^{a,sid}(\{\cdot\}^{R}_{\ldots}, \tau')$ with the same R will satisfy that $\tau \subseteq \tau'$ or $\tau' \subseteq \tau$ (i.e., one of the two substitutions is an extension of the other) because in any path of a role tree $\Pi(i)$ associated with the session *sid*, there can be at most one pattern of the form $\{\cdot\}^{R}_{\ldots}$ for a given R (uniqueness of effective R-subpatterns, Definition 4). Since $\text{construct}^{a,sid}(\{\cdot\}^{R}_{\ldots}, \tau)$ and $\text{construct}^{a,sid}(\{\cdot\}^{R}_{\ldots}, \tau')$ will return the same bitstring if one of τ and τ' is an extension of the other (as long as both invocations succeed), it follows that m is uniquely defined.

Analogous reasoning applies for the cases $t = [\cdot]^{r^{a,R,sid}}_{\ldots}$ and $t = \mathsf{ZK}^{r^{a,R,sid}}_{\ldots}(\ldots)$. Hence $\bar{c}$ is injective.

Furthermore, 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 z such that w is not a witness for z, although the proof has been successfully verified. Since all machines involved in the execution, as well as the construction of $\bar{c}$ are polynomial-time, this is a contradiction to the extractability property of $\mathcal{ZK}$.

The following lemmas guarantee that the various definitions given so far are compatible. For example, when computing $m := \text{construct}^{a,sid}(r, \tau)$ and then parsing m using $\bar{c}$, we get back r (up to the substitution of the variables occuring in r).

Lemma 2. *Whenever a circuit $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is constructed (through a call to $\text{construct}^{a,sid}(r,\tau)$), we have that $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is defined.*

Lemma 3. *Whenever a circuit $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is constructed and $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w)$ is evaluated with some $w = \tilde{R}_1 \| \ldots \| \tilde{R}_l \| \tilde{a}_1 \| \ldots \| \tilde{a}_n$ (indirectly through a call to $\text{construct}^{a,sid}(r,\tau)$ with $r = \mathsf{ZK}^{R}_{F}(\underline{R}; \underline{a}; \underline{b})$), we have for every ZK-term T which is a subterm of F that $C^{s,n,\underline{l}}_{T,\underline{\tilde{b}}}(w) = \text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}, \tau)$.*

Lemma 4. *Assume that in a computational execution, no two honestly generated nonces, encryption keys, verification keys, ciphertexts, signatures, or zero-knowledge proofs are equal. Assume that no ZK-failure occurs.*

Then for any invocation of $\text{construct}^{a,sid}(r,\tau)$ that is performed during the execution, we have that $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\gamma^{a,sid}(\bar{c} \circ \tau)$ contains true proofs and that in this case $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\gamma^{a,sid}(\bar{c} \circ \tau)$.

Lemma 5. *Assume that in a computational execution, no two honestly generated nonces, encryption keys, verification keys, ciphertexts, signatures, or zero-knowledge proofs are equal. Assume that no ZK-failure occurs.*

Whenever $\bar{c}(m')$ is computed for some m', $\bar{c}(m')$ contains true proofs.

Lemma 6. *Assume that in a computational execution, no two honestly generated nonces, encryption keys, verification keys, ciphertexts, signatures, or zero-knowledge proofs are equal. Assume that no ZK-failure or ZK-break occurs.*

Then for any invocation of $\text{parse}^{a,sid}(m,l,\tau)$ that is performed during the execution, we have that $\text{parse}^{a,sid}$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ (let θ be the matcher) and that in this case $\bar{c} \circ \text{parse}^{a,sid}(m,l,\tau) = (\bar{c} \circ \tau) \cup \theta$.

Lemma 7. *Assume that in a computational execution, no two honestly generated nonces, encryption keys, verification keys, ciphertexts, signatures, or zero-knowledge proofs are equal. Assume that no ZK-failure occurs.*

For all $m \in \{0,1\}^*$, $c^*(\bar{c}(m)) = m$.

The proofs of these lemmas are very lengthy inductions and consist mainly of case distinctions in which it is verified that two differently computed terms are indeed equal. Since these proofs essentially only check that we did not make mistakes in writing down the definitions of $\text{construct}^{a,sid}$, $\text{parse}^{a,sid}$, $\bar{c}$, and c^*, but do not involve actual cryptographic arguments, we defer the proofs to Appendix A.

Note that the assumptions of these lemmas are fulfilled with overwhelming probability: Nonces are pairwise different with overwhelming probability because of their high entropy. If encryption keys and verification keys were not pairwise different with overwhelming probability, by honestly executing the key generation algorithm, there would be a non-negligible probability of producing the secret key for a public key that is already being used, thus breaking the IND-CCA or the strong existential unforgeability property, respectively. Encryptions of the same message are pairwise different with overwhelming probability because otherwise one could distinguish between the encryption of a given message m and of a random string by reencrypting m, thus breaking the IND-CCA property. Signatures and zero-knowledge proofs are pairwise different because of the unpredictability property. The probability than a ZK-failure occurs is negligible because of the completeness of $\mathcal{ZK}$. As discussed before, ZK-breaks also occur only with negligible probability.

The trace $\bar{c}(t^c)$ is a pre-DY trace. In the following, by a *pre-DY trace* we denote a symbolic trace that satisfies Definition 5 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 symbolic 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 ℓ, the ℓ-th transition in $\bar{c}(t^c)$ is a valid transition. If $\ell = 1$ and the ℓ-th transition of t^c is of the form

$$(\varnothing, \varnothing, \varnothing) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\varnothing, \varnothing, \{a_1, \ldots, a_l\}),$$

then $\bar{c}$ maps that transition to $(\varnothing, \varnothing, \mathsf{Nonce}_{adv}) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\varnothing, \varnothing, \mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a_j), \mathsf{sk}(a_j) : j \in [l]\})$ which is a valid symbolic transition.

If the ℓ-th transition of t^c is of the form

$$(\mathsf{Sid}, g, C) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{Sid}', g', C)$$

then $\bar{c}$ maps that transition to $(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{Sid}', f', \varphi)$ for some φ and with $f(sid) = (i, \bar{c}\circ\tau, p, \underline{a})$ where $(i, \tau, p, \underline{a}) := g(sid)$ and f' analogously. As g and g' only differ in $sid' := |\mathsf{Sid}| + 1$, so do f and f'. Furthermore, $g'(sid') = (i, \tau, \varepsilon, (a_1, \ldots, a_k))$ where ε is the root of $\Pi(i)$ and $\tau(X_N^j) = tag_{nonce}(n_j)$ and $\tau(A_j) = tag_{agent}(a_j)$. Then, by definition of $\bar{c}$, $\bar{c} \circ \tau(X_N^j) = n^{a_i,j,sid'}$ and $\bar{c} \circ \tau(A_j) = a_j$. Hence $(\mathsf{Sid}, f, \varphi) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{Sid}', f', \varphi)$ is a valid symbolic transition.

If the ℓ-th transition of t^c is of the form

$$(\mathsf{Sid}, g, C) \xrightarrow{\mathbf{send}(sid,m)} (\mathsf{Sid}, g', C),$$

(where the adversary gets answer $\tilde{m}$) then $\bar{c}$ maps that transition to

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

for some φ where $\varphi' = \varphi \cup \{\bar{c}(\tilde{m})\}$ and $f(sid) = (i, \bar{c} \circ \tau, p, \underline{a})$ where $(i, \tau, p, \underline{a}) := g(sid)$ and f' analogously. Furthermore, from the definition of the computational trace, we have that $g'(sid) = (i, \tau', p', (a_1, \ldots, a_k))$ where p' is the node reached through the first edge (l, r) leaving p such that $\tau'' := \mathrm{parse}^{a_i,sid}(m, l, \tau)$ and $m' := \mathrm{construct}^{a_i,sid}(r, \tau'')$ succeed, and in that case $\tau' := \tau''$ and $\tilde{m} := m'$. (If no such edge exists, $\tau' = \tau$ and $p' = p$.)

When the assumptions of Lemmas 6 and 4 are fulfilled (and this happens with overwhelming probability as discussed above), by Lemmas 6 and 4, p' is also the node reached through the first edge leaving p such that $\bar{c}(m)$ matches $l\gamma^{a_i,sid}(\bar{c} \circ \tau)$ (with matcher θ) and $r\gamma^{a_i,sid}((\bar{c} \circ \tau) \cup \theta) = r\gamma^{a_i,sid}(\bar{c} \circ \tau)\theta$ contains true proofs (or no such edge exists). In this case (still by the same lemmas), we have that $\bar{c} \circ \tau' = (\bar{c} \circ \tau) \cup \theta$ and $\bar{c}(\tilde{m}) = r\gamma^{a_i,sid}((\bar{c} \circ \tau) \cup \theta) = r\gamma^{a_i,sid}(\bar{c} \circ \tau)\theta$. Hence

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

is a valid symbolic transition in a pre-DY trace (cf. Definition 5; remember that for pre-DY traces, we do not need $\varphi \vdash \bar{c}(m)$).

The initial state $(\varnothing, \varnothing, \varnothing)$ of t^c is mapped to the valid initial state $(\varnothing, \varnothing, \mathsf{Nonce}_{adv})$ of $\bar{c}(t^c)$.

Thus with overwhelming probability, $\bar{c}(t^c)$ is a pre-DY trace. Furthermore, by construction of $\bar{c}$ (as extended to traces) we have that $\varphi_\ell = \mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \{\bar{c}(m) : m \in S_\ell\}$ where φ_ℓ is the adversary knowledge in the ℓ-th step of $\bar{c}(t^c)$, C denotes the corrupted agents and S_ℓ are the messages given to the adversary in the send transitions in the computational model up to the transition leading to φ_ℓ.

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 φ_ℓ in the ℓ-th step of $\bar{c}(t^c)$ is $\varphi_\ell = \mathsf{Nonce}_{adv} \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\}$ (where C denotes the corrupted agents and S_ℓ the messages given to the adversary), it will be enough to show that in every $\mathbf{send}(sid, m)$ transition in the computational model, we have that

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

(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 ℓ, a message m is sent such that

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

From this we will derive a contradiction by transforming the computational execution in several steps using the security properties of $\mathcal{ZK}$, $\mathcal{SIG}$, and $\mathcal{AE}$.

Emulating $\mathrm{construct}^{a,sid}$ **and** $\mathrm{parse}^{a,sid}$**.** We will now redefine $\mathrm{construct}^{a,sid}$ and $\mathrm{parse}^{a,sid}$ in such a way that they still compute the same functions but are more amenable to certain modifications later on.

By Lemma 4, $\bar{c}(\mathrm{construct}^{a,sid}(r,\tau)) = r\gamma^{a,sid}(\bar{c} \circ \tau)$ where $\mathrm{construct}^{a,sid}(r,\tau)$ succeeds iff $r\gamma^{a,sid}(\bar{c} \circ \tau)$ contains true proofs. Thus we can redefine $\mathrm{construct}^{a,sid}(r,\tau) := c^*(r\gamma^{a,sid}(\bar{c} \circ \tau))$ (or $\mathrm{construct}^{a,sid}(r,\tau) := \bot$ if $r\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain true proofs).

Note also that whenever $m := \mathrm{construct}^{a,sid}(r,\tau)$ is computed and sent, we have that $\bar{c}(m) = r\gamma^{a,sid}(\bar{c} \circ \tau)$ and $m = \mathrm{construct}^{a,sid}(r,\tau) = c^*(r\gamma^{a,sid}(\bar{c} \circ \tau)) = c^*(\bar{c}(m))$. Hence, whenever a message m is sent by the protocol, we have that m was the result of an invocation $c^*(t)$ where $t = \bar{c}(m)$ (although t was not necessarily computed as $\bar{c}(m)$, it just happens to coincide with $\bar{c}(m)$).

By Lemma 6, $\bar{c} \circ \mathrm{parse}^{a,sid}(m,l,\tau) = (\bar{c} \circ \tau) \cup \theta$ where θ is the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ (or $\bar{c} \circ \mathrm{parse}^{a,sid}(m,l,\tau) = \bot$ if there is no match).

Note that the redefined $\mathrm{construct}^{a,sid}$ only uses $\tilde{\tau} := \bar{c} \circ \tau$, but never τ directly. And we can also compute $\bar{c} \circ \mathrm{parse}^{a,sid}(m,l,\tau)$ directly from $\bar{c} \circ \tau$. Hence instead of computing and storing τ in the computational trace, we can instead directly store $\tilde{\tau} := \bar{c} \circ \tau$ in the trace and use the following modified functions $\mathrm{construct}^{a,sid}(r,\tilde{\tau}) := c^*(r\gamma^{a,sid}\tilde{\tau})$ (or $\mathrm{construct}^{a,sid}(r,\tilde{\tau}) := \bot$ if $r\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain true proofs) and $\mathrm{parse}^{a,sid}(m,l,\tilde{\tau}) := \tilde{\tau} \cup \theta$ where θ is the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ (or $\mathrm{parse}^{a,sid}(m,l,\tilde{\tau}) := \bot$ if there is no match).

Note that the changed $\mathrm{construct}^{a,sid}$ and $\mathrm{parse}^{a,sid}$ do not directly access any secrets any more, all accesses and cryptographic operations are performed through $\bar{c}$ and c^*.

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*. A query to the *Proof* takes a circuit and a witness as argument and outputs a zero-knowledge proof (honestly generated using the publicly available CRS). A query to the *Extract* oracle takes

a zero-knowledge proof and applies the extraction algorithm to it (using the extraction trapdoor that was generated together with the CRS).

We now modify c^* and $\bar{c}$ as follows: Whenever $c^*(t)$ with $t = \mathsf{ZK}_\cdot^{\mathsf{Rand}_{ag}}(\dots)$ would produce a zero-knowledge proof for the circuit C and the witness w, it queries the *Proof*-oracle instead (with C and w as arguments). Furthermore, we change $c^*(t)$ not to query the *Proof*-oracle twice for the same term t. Instead, the result of the last query is cached. And when $\bar{c}$ would use the extraction algorithm for extracting a witness w from a zero-knowledge proof z (in the case of not honestly generated zero-knowledge proofs), then $\bar{c}$ queries the *Extract*-oracle instead.

Obviously, this modification changes the computational trace only if c^* uses some randomness for some ZK-proof, and the same randomness is used for a later ZK-proof or for a different operation (i.e., encryption or signing). However, Definition 4, condition 1 guarantees that the original construct^{a,sid}, if it uses some randomness $tape^{a,sid}(R)$ to construct some zero-knowledge proof, it uses that randomness only for producing the same proof again. Hence also c^* does not reuse randomness used for producing ZK-proofs. Hence replacing the proof and the extraction algorithm does not change the probability of (2).

We introduce yet another oracle *Simulate*: A query to *Simulate* consists of a circuit C and a witness w, and *Simulate* runs the simulation algorithm on the circuit C using the simulation trapdoor that was produced together with the CRS (and the witness w is ignored).

Lemma 8. *No polynomial-time machine M can distinguish between $\mathcal{O} :=$ Proof and $\mathcal{O} :=$ Simulate (with more than negligible probability), even when given access to the Extract-oracle, as long as M obeys the following rules: M never queries the Extract-oracle with C, z where z was the output of a query to $\mathcal{O}$ with input C, w, and M never invokes $\mathcal{O}$ with C, w such that $C(w) \neq 1$.*

Proof. Assume a polynomial-time machine M that distinguishes with non-negligible probability between *Proof* and *Simulate* given access to the *Extract*-oracle (and that obeys the rules stated in the lemma). We define an oracle $\mathcal{H}_i$ that behaves like *Proof* for the first i-proofs and like *Simulate* afterwards. Let q be a polynomial upper bound on the number of queries performed by M. Since M distinguishes between *Proof* and *Simulate*, we have that M also distinguishes with non-negligible probability between $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ for some i (that may depend on the security parameter). We now construct an adversary $\mathcal{A}$ that has access to an oracle $\mathcal{O}$ and to *Extract*, and that expects as input the CRS and the corresponding simulation trapdoor. This adversary then simulates M. All queries of M sent to the *Extract*-oracle are forwarded to that oracle. The first i queries that M sends to $\mathcal{O}$, $\mathcal{A}$ answers by invoking the prover P of $\mathcal{ZK}$. The $(i+1)$-st query $\mathcal{A}$ forwards to the oracle $\mathcal{O}$ that was given to $\mathcal{A}$. All further queries $\mathcal{A}$ answers by invoking the simulator S of $\mathcal{ZK}$ (note that $\mathcal{A}$ knows the simulation trapdoor, thus he can execute the simulator). When M terminates with some output, then $\mathcal{A}$ terminates with that output. By construction, $\mathcal{A}$ with access to *Simulate* simulates an execution of M with access to $\mathcal{H}_i$. And $\mathcal{A}$ with access to *Proof* simulates an execution of M with access to $\mathcal{H}_{i+1}$. Since M distinguishes $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ with non-negligible probability, $\mathcal{A}$ distinguishes between $\mathcal{O} :=$ *Proof* and $\mathcal{O} :=$ *Simulate* with non-negligible probability, and $\mathcal{A}$ performs only a single query to that oracle. Furthermore, since M obeys the rules given in the lemma, $\mathcal{A}$ never queries (C, proof) from *Extract* where C is the circuit send to $\mathcal{O}$ and *proof* the answer of $\mathcal{O}$. And $\mathcal{A}$ only sends (C, w) to $\mathcal{O}$ with $C(w) = 1$. Thus $\mathcal{A}$ is an adversary that contradicts the extraction zero-knowledge property (Definition 1). □

We show that $\bar{c}$ queries the *Extract*-oracle with arguments (C, z) only if z was not the output of a query to *Proof* with arguments (C, w): The only case where $\bar{c}$ invokes the *Extract* is when given $m = tag_{zk}(z, F, s, n, \underline{l}, \underline{b})$ where m was not produced by construct^{a,sid}. Hence, if z was the output of the *Proof*-oracle, then it was produced by construct^{a,sid} as part of a message $m' = tag_{zk}(z, F', s', n', \underline{l}', \underline{b}')$ with $(F', s', n', \underline{l}', \underline{b}') \neq (F, s, n, \underline{l}, \underline{b})$. Since $C_{F,\tilde{\underline{b}}}^{s,n,\underline{l}}$ determines $F, s, n, \underline{l}, \underline{b}$, we have that z was produced by a query to the *Proof*-oracle with a circuit $C_{F',\tilde{\underline{b}'}}^{s',n',\underline{l}'} \neq C_{F,\tilde{\underline{b}}}^{s,n,\underline{l}} =: C$. Hence z was not the output of a query to *Proof* with arguments (C, w) for any w.

Furthermore, we have that c^* never queries the *Proof*-oracle on (C, w) with $C(w) \neq 1$ since c^* checks whether $C(w) \neq 1$ and aborts otherwise (this check is inherited from the original construction of construct^{a,sid}).

Thus, by virtue of Lemma 8, we can replace all queries to the *Proof*-oracle by queries to the *Simulate*-oracle, and the probability of (2) changes by at most a negligible amount. Thus, in a computational execution using the *Simulate*-oracle, (2) holds with non-negligible probability.

Reducing direct access to the random tape and private keys. The functions $\bar{c}$ and c^* access the random tape $tape^{a,sid}$ and the decryption and signing keys of uncorrupted agents in various places:

- $\bar{c}$ uses the decryption keys for decrypting.
- c^* uses the signing keys for signing.
- $\bar{c}$ uses the decryption when parsing a bitstring $m = tag_{dk}(dk)$ that is tagged as a decryption key. In this case, however, it is sufficient that $\bar{c}$ can *compare* dk to all decryption/signing keys of honest agents. Analogously for the signing keys.
- c^* uses the random tape to get the randomness for encrypting and signing. (The randomness for zero-knowledge proofs is not taken from the random tape but produced by the *Simulate*-oracle directly.)
- c^* uses the random tape and the decryption keys to produce witnesses w for the zero-knowledge proofs. More precisely, c^* uses w to check whether $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$, and to compute the length of w that is given to the *Simulate*-oracle.

In order to relate (2) to the security of $\mathcal{ZK}$, $\mathcal{AE}$, and $\mathcal{SIG}$, we have to minimize the direct accesses that c^* and $\bar{c}$ perform to secret data, and instead ensure that c^* and $\bar{c}$ only access this data through well-defined oracles.

First, we get rid of the computation of the witnesses through $c^*(t)$ with $t = \mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b})$. Observe that the length of w can be computed given only t because all our primitives are length regular (and the lengths of nonces are fixed). Furthermore, $c^*(t)$ accesses the witness w to check whether $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$ holds for the circuit $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$. However, since $c^*(t)$ is only executed by the modified construct$^{a,sid}(r, \tau)$ if t contains true proofs, in which case a call to the original construct$^{a,sid}(r, \tau)$ with $t = r\tau$ would succeed (Lemma 4), we have that $c^*(t) = \text{construct}^{a,sid}(r, \tau)$ succeeds. Since $c^*(t) = \text{construct}^{a,sid}(r, \tau)$ would fail if $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) \neq 1$, we know that when $c^*(t)$ is invoked by the modified construct^{a,sid}, we always have $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$. Since $c^*(t)$ is only invoked by construct^{a,sid}, we can hence remove the check $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$ from c^*. Thus c^* does not compute the witness any more. This removes all access to the random tape and the decryption keys during to the computation of the witness by construct^{a,sid}.

In order to get rid of the remaining access to the random tape, we introduce two more oracles, an encryption, a decryption, and a signing oracle.

The encryption oracle *Encrypt* supports the following queries: *Generate key pair:* This produces a new encryption/decryption key pair and returns the encryption key. *Encrypt:* This takes a message m and a previously generated encryption key ek and returns the encryption of m under ek. *Check key:* This takes a decryption key dk and returns if this decryption key was produced in a call to *generate key pair*, and if so, what the corresponding encryption key was. *Decrypt:* This takes a previously generated encryption key ek and a ciphertext c. Then it looks up the decryption key dk corresponding to ek and decrypts c using dk and returns the plaintext. *Get decryption key:* This takes a previously generated encryption key and returns the corresponding decryption key.

The signing oracle *Sign* supports the following queries: *Generate key pair:* This produces a new verification/signing key pair and returns the verification key. *Check key:* This takes a

signing key *sk* and returns whether this signing key was produced in a call to *generate key pair*, and if so, what the corresponding verification key was. *Sign:* This takes a previously generated verification key *vk* and a message m. Then it looks up the signing key *sk* corresponding to *vk* and signs m using *sk* and returns the resulting signature. *Get signing key:* This takes a previously generated verification key and returns the corresponding signing key.

We now change $\bar{c}$ and c^* to use these oracles instead of decrypting and signing directly. Furthermore, we change c^* not to query the encryption or signing oracle twice for the same term t. Instead, the result of the last query is cached. Furthermore, when $\bar{c}$ parses a bitstring tagged as a decryption or signing key, it uses the *check key* query of *Encrypt* and *Sign*.

Hence, with the modified definitions of $\bar{c}$ and c^*, (2) still holds, and the decryption and signing keys of uncorrupted agents are only accessed through the encryption and signing oracles, and the random tapes are never accessed.

Using fake encryptions. We now introduce a new oracle *FakeEncrypt*. This oracle behaves like *Encrypt* with the following difference: Upon *encrypt* query with encryption key *ek* and message m, if no *get decryption key* query has earlier been performed for *ek*, the oracle encrypts the plaintext $0^{|m|}$ using *ek* (instead of encrypting m). If a *get decryption key* query has earlier been performed, the message m is encrypted (as in the oracle *Encrypt*).

Lemma 9. *No polynomial-time machine M can distinguish between Encrypt and FakeEncrypt (with more than negligible probability), as long as M obeys the following rules: No* get decryption key *query for some encryption key ek may be preceded by an* encrypt *call with respect to the same query. No* decrypt *query may be given a ciphertext c that was the result of an* encrypt *query with respect to the same key.*

Proof. Assume a polynomial-time machine M that distinguishes with non-negligible probability between *Encrypt* and *FakeEncrypt* (and that obeys the rules stated in the lemma). We define an oracle $\mathcal{H}_i$ that behaves like *Encrypt* with respect to the first i encryption keys and like *FakeEncrypt* afterwards. Let q be a polynomial upper bound on the number of queries performed by M. Since M distinguishes between *Encrypt* and *FakeEncrypt*, we have that M also distinguishes with non-negligible probability between $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ for some i (that may depend on the security parameter). We now construct an adversary $\mathcal{B}$ that has access to an oracle $\mathcal{O}$. This adversary then simulates M. All oracle queries by M with respect to the first i encryption keys are answered by $\mathcal{B}$. For this, $\mathcal{B}$ picks his own encryption/decryption key pair and then simulates the oracle *Encrypt*. The queries with respect to the $(i+1)$-st encryption key are forwarded to the oracle $\mathcal{O}$ that is given to $\mathcal{B}$. Queries with respect to all following encryption keys are answered by $\mathcal{B}$ by simulating the oracle *FakeEncrypt* (again picking his own key pair). *Check key* queries are answered by $\mathcal{B}$ as follows: Let *dk* be the argument of the *check key* query. For each encryption key *ek* (including those chosen by $\mathcal{B}$ and the one chosen by the oracle $\mathcal{O}$ given to $\mathcal{B}$), $\mathcal{B}$ picks a random message m, encrypts m with *ek*, and decrypts the resulting ciphertext with *dk*. If this returns m again, then $\mathcal{B}$ returns *ek* as answer to the query. If this fails for all *ek*, $\mathcal{B}$ returns "no" as answer to the query.

Then $\mathcal{B}$ with access to $\mathcal{O} :=$ *Encrypt* is a faithful simulation (up to a negligible error) of M with access to $\mathcal{H}_{i+1}$. (For this, note that encrypting a message with *ek* and decrypting with *dk* will succeed (except for a negligible probability) only given the right decryption key.) And $\mathcal{B}$ with access to $\mathcal{O} :=$ *FakeEncrypt* is a faithful simulation (up to a negligible error) of M with access to $\mathcal{H}_i$. Since M distinguishes between $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ with non-negligible probability, $\mathcal{B}$ distinguishes between $\mathcal{O} :=$ *Encrypt* and $\mathcal{O} :=$ *FakeEncrypt* with non-negligible probability.

Furthermore, since M obeys the rules stated in the lemma, $\mathcal{B}$ queries $\mathcal{O}$ only with respect to a single encryption key, does not perform *check key* queries, never performs a *decrypt* query on a ciphertext that was returned by an *encrypt* query, and it only performs *encrypt* queries if no *get decryption key* query was performed before.

From $\mathcal{B}$ we construct another adversary $\mathcal{A}'$ that also has access to an oracle $\mathcal{O}$. $\mathcal{A}'$ chooses a random bit b, and then simulates $\mathcal{B}$ and forwards $\mathcal{B}$'s oracle queries to $\mathcal{O}$. If $b = 0$ and $\mathcal{B}$ performs a *get decryption key* query, $\mathcal{A}'$ aborts with no output. If $b = 1$ and $\mathcal{B}$ does not perform

a *get decryption key* query, $\mathcal{A}'$ aborts with no output. Since b is independent of the actions of $\mathcal{B}$, $\mathcal{A}'$ aborts with probability $\frac{1}{2}$, and the distribution of the output of $\mathcal{A}'$ conditioned on a non-abort is equal to the distribution of $\mathcal{B}$. Hence $\mathcal{A}'$ also distinguishes between $\mathcal{O} := \mathit{Encrypt}$ and $\mathcal{O} := \mathit{FakeEncrypt}$ with non-negligible probability.

Now we construct an adversary $\mathcal{A}$ from $\mathcal{A}'$. If $b = 1$, $\mathcal{A}$ simulates an instance of *Encrypt* for $\mathcal{A}'$ (and ignores the oracle $\mathcal{O}$ it gets as input). If $b = 0$, $\mathcal{A}$ forwards all queries of $\mathcal{A}'$ to the oracle $\mathcal{O}$ that is given to $\mathcal{A}$. Given access to $\mathcal{O} := \mathit{Encrypt}$, the distributions of the outputs of $\mathcal{A}$ and $\mathcal{A}'$ are equal by construction. Given $\mathcal{O} := \mathit{FakeEncrypt}$, $\mathcal{A}$ and $\mathcal{A}'$ behave slightly differently: If $b = 1$, $\mathcal{A}$ simulates *Encrypt*, while $\mathcal{A}'$ has access to *FakeEncrypt*. However, in the case $b = 1$, no *encrypt* queries are performed by $\mathcal{A}'$ (since it performs a *get decryption key* query). Thus, since *Encrypt* and *FakeEncrypt* differ only in their handling of *encrypt* queries, also given access to $\mathcal{O} := \mathit{Encrypt}$, the distributions of the outputs of $\mathcal{A}$ and $\mathcal{A}'$ are equal. Hence $\mathcal{A}$ distinguishes between $\mathcal{O} := \mathit{Encrypt}$ and $\mathcal{O} := \mathit{FakeEncrypt}$ with non-negligible probability.

The adversary $\mathcal{A}$ only performs one *generate key pair* query, no *check key* or *get decryption* queries, and never performs a *decrypt* query on a ciphertext it received from an *encrypt* query. Thus the adversary $\mathcal{A}$ is a valid adversary against the IND-CCA property that succeeds with non-negligible probability.[31] Thus we have a contradiction against the IND-CCA property of $\mathcal{AE}$ which shows that no machine M with the properties stated in the lemma exists. □

Since in the computational execution, *get decryption key* queries are only performed at the very beginning of the execution, no *encrypt* query precedes a *get decryption key* query. Furthermore, both c^* and $\bar{c}$ only perform decryption queries for messages that were not generated honestly, hence no *decrypt* query is given a ciphertext c that was the result of an *encrypt* query with respect to the same key.

Thus we can replace the *Encrypt*-oracle by the *FakeEncrypt*-oracle, and by Lemma 9, the probability that (2) holds changes only by a negligible amount. Hence (2) holds with non-negligible probability in the modified execution that uses the *FakeEncrypt*-oracle.

Reducing direct access to the random tape and private keys (even more). Since the *FakeEncrypt*-oracle does not use the plaintext it is given (only its length) in an *encrypt* query with respect to an encryption key of some agent $a \notin C$, we can modify c^* as follows: Instead of performing an *encrypt* query with arguments ek and $m' = c^*(t')$ for some t', it uses the arguments ek and $\bar{m} := 0^{|m'|}$. Furthermore, the length of m' can be computed given only t' because all our primitives are length regular (and the lengths of nonces are fixed). Hence we can modify c^* such that $\bar{m}$ can be computed without invoking $c^*(t')$. Summarizing, using the modified construction, $c^*(\{t'\}^R_{\mathsf{ek}(a)})$ with $R \in \mathsf{Rand}_{adv}$ will not invoke $c^*(t')$ any more for $a \in \mathsf{A} \setminus C$.

Identifying an underivable subterm. In order to derive a contradiction from (2), we have to identify a 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 property of $\mathcal{AE}$ or the strong existential unforgeability of $\mathcal{SIG}$. For this, we need the following characterization of underivable messages:

Lemma 10. *Fix $C \subseteq \mathsf{A}$, $S \subseteq \mathsf{M}$ and $M \in \mathsf{M}$ and set $S' := S \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv}$. Assume $S' \nvdash M$. Assume further that M contains true proofs.*

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

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

Furthermore, we have that $S' \nvdash T$ and that T satisfies one of the following conditions:

(a) $T \in \mathsf{Nonce}_{ag}$.

[31] We assume the definition of IND-CCA where the adversary is allowed to perform an arbitrary number of encryption queries. One also commonly defines IND-CCA with respect to adversaries that perform only a single encryption query; however, the two variants of IND-CCA are well-known to be equivalent.

(b) $T = \{\cdot\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(a)}$.

(c) $T = \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\dots)$.

(d) $T = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r};\underline{a};\underline{b})$ *and for some* i, $S' \nvdash_r r_i$.

(e) $T = [\cdot]^{\mathsf{Rand}_{adv}}_{\mathsf{sk}(a)}$ *where* $a \notin C \cup \mathsf{Garbage}$.

(f) $T = [\cdot]^{\mathsf{Rand}_{ag}}_{\mathsf{sk}(a)}$.

(g) $T = \mathsf{sk}(a)$ *or* $T = \mathsf{dk}(a)$ *for some* $a \in \mathsf{A} \setminus C$.

Proof. We prove the lemma by structural induction on M. We distinguish the following cases:

Case 1: *"M is an agent identifier, an encryption or a verification key, or garbage".*
In this case, we have $S' \vdash M$, so the premises of the lemma do not hold and there is nothing to show.

Case 2: *"M is a nonce".*
If $M \in \mathsf{Nonce}_{adv}$, $S' \vdash M$ holds and there is nothing to show. If $M \in \mathsf{Nonce}_{ag}$, the conclusion of the lemma is fulfilled with $T := M$.

Case 3: *"$M = \mathsf{sk}(a)$ or $M = \mathsf{dk}(a)$".*
If $a \in C$, $S' \vdash M$ holds and there is nothing to show. If $a \notin C$, the conclusion of the lemma is fulfilled with $T := M$.

Case 4: *"$M = \langle M_1, M_2 \rangle$".*
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 instead of M, and this T is also a subterm of M satisfying the conclusion for M.

Case 5: *"$M = \{\cdot\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(a)}$ or $M = \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\dots)$ or $M = [\cdot]^{\mathsf{Rand}_{ag}}_{\mathsf{sk}(a)}$ or $M = [\cdot]^{\mathsf{Rand}_{adv}}_{\mathsf{sk}(b)}$ with $b \notin C \cup \mathsf{Garbage}$".*
In these cases $T := M$ fulfills the conclusion of the lemma.

Case 6: *"$M = \{M'\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(a)}$".*
In this case, since $S \nvdash M$, by the rules from Figure 1 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 7: *"$M = [M']^{\mathsf{Rand}_{adv}}_{\mathsf{sk}(a)}$ with $a \in C \cup \mathsf{Garbage}$".*
In this case, since $S \nvdash M$, by the rules from Figure 1 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 8: *"$M = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r};\underline{a};\underline{b})$ and $S' \vdash_r \underline{r}$".*
Since $S' \vdash_r \underline{r}$, 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 9: *"$M = \mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\underline{r};\underline{a};\underline{b})$ and for some i, $S' \nvdash_r r_i$".*
Then $T := M$ fulfills the conclusion of the lemma.

□

In a trace satisfying (2), we can apply this lemma with C being the set of corrupted agents and $S := \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\}$ being the messages received by the adversary up to the ℓ-th step, and $M := \bar{c}(m)$ being the message in the ℓ-th step. By Lemma 5, M contains true proofs. Thus,

since (2) holds with non-negligible probability, we have that with non-negligible probability, a subterm T of M with the properties specified in Lemma 10 exists.

So the term T satisfies one of the properties (a–g) with non-negligible probability. In the following, we will examine each of these conditions separately and in each case derive a contradiction.

***T* is a nonce (Case** (a)**).** In Case (a) we have $T \in \mathsf{Nonce}_{ag}$, so T is of the form $T = n^{a,j,sid}$. As discussed on page 33, each message m' sent by the protocol has been produced as $c^*(t)$ where $t = \bar{c}(m')$. Hence all messages $m' \in S_\ell$ sent by the protocol were computed as $m' = c^*(t)$ with $t \in S = \{\bar{c}(\tilde{m}) : \tilde{m} \in S_\ell\}$. Furthermore, since c^* is only invoked to compute messages that are to be sent, it follows that for every invocation of $c^*(t)$ we have that $t \in S$ and hence $\{t\} \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv} \not\vdash T$. Hence T occurs in t only as a subterm of an encryption $\{\cdot\}^R_{\mathsf{ek}(a)}$ with $a \notin C$ or as a subterm of the witness of ZK-proof $\mathsf{ZK}^R_F(\dots)$ (because otherwise the deduction rules would allow to derive T from $\{t\} \cup \{\mathsf{dk}(a), \mathsf{sk}(a) : a \in C\} \cup \mathsf{Nonce}_{adv}$). Hence, by definition of c^*, in the computation of $c^*(t)$, the recursive call $c^*(T) = c^*(n^{a,j,sid})$ will never be performed (remember that we changed c^* not to compute the witnesses of ZK-proofs nor the plaintexts of encryptions). Hence c^* will never read the bitstring $\tau(X^j_N)$ (where τ is the substitution the from session sid). Furthermore, by construction, an invocation $\bar{c}$ might read the bitstring $\tau(X^j_N)$, but only for comparing it to another bitstring m''. Note that the adversary does not have access to $\tau(X^j_N)$. Hence the only operation performed on $\tau(X^j_N)$ before receiving the message m from the adversary is a comparison to already computed strings (and only a polynomial number of these comparisons are performed). Hence it is information-theoretically impossible to guess $\tau(X^j_N)$ (except for a negligible probability).

On the other hand, by Lemma 10, $\bar{c}(m) = M = D[T]$ where all terms on the path from $D[T]$ to T are of the form $\langle\cdot,\cdot\rangle$ or $\{\cdot\}^{\mathsf{Rand}_{adv}}_{\mathsf{ek}(\cdot)}$ or $\mathsf{ZK}^{\mathsf{Rand}_{adv}}_{\mathsf{Formula}}(\dots)$ or $[\cdot]_{\mathsf{sk}(\cdot)}$. By definition of $\bar{c}$, $\bar{c}(m) = D[T]$ then implies that there was a recursive invocation $\bar{c}(\hat{m})$ (indirectly through $\bar{c}(m)$) such that $\bar{c}(\hat{m}) = T = n^{a,j,sid}$. By definition of $\bar{c}$, this implies that $\hat{m} = \tau(X^j_N)$. Thus $\tau(X^j_N)$ occurs in the computation of $\bar{c}$. This is a contradiction to the fact that it is information-theoretically impossible to guess $\tau(X^j_N)$.

***T* is an honestly-generated encryption (Case** (b)**)** Fully analogous as in the case of T being a nonce (Case (a)), we have that the $c^*(T)$ will never be computed during the execution, and that $\bar{c}(\hat{m})$ is executed with $\bar{c}(\hat{m}) = T$. However, by definition of $\bar{c}$, $\bar{c}(\hat{m}) = T = \{T'\}^R_{\mathsf{ek}(a)}$ with $R \in \mathsf{Rand}_{ag}$ implies that $\hat{m}$ is the result of a prior invocation $\mathrm{construct}^{a,sid}(r, \tilde{\tau})$ with $r = \{r'\}^{R'}_{\mathsf{ek}(x)}$ and $r'\gamma^{a,sid}\tilde{\tau} = T'$ and $R = r^{a,R',sid}$ and $\tilde{\tau}(x) = a$. Hence $r\gamma^{a,sid}\tilde{\tau} = T$. Since we have defined $\mathrm{construct}^{a,sid}(r, \tilde{\tau})$ to compute $c^*(r\gamma^{a,sid}\tilde{\tau}) = c^*(T)$, this is a contradiction to the fact that $c^*(T)$ is never executed.

***T* is an honestly-generated signature or zero-knowledge proof (Cases** (f) **and** (c)**).** These cases are analogous to the case of T being an honestly-generated encryption.

***T* is an adversary-generated zero-knowledge proof (Case** (d)**).** Then $T = \mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b})$ and $R \in \mathsf{Rand}_{adv}$ and $S' \not\vdash_r r_i$. Fully analogous to the case of T being a nonce (Case (a)), we have that $\bar{c}(\hat{m})$ is executed with $\bar{c}(\hat{m}) = T$ for some bitstring $\hat{m}$. By definition of $\bar{c}$, we have that $\bar{c}(\hat{m}) = T = \mathsf{ZK}^R_F(\underline{r}; \underline{a}; \underline{b})$ with $R \in \mathsf{Rand}_{adv}$ implies that

- The *Extract*-oracle returns a witness w on input $(C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}, z)$ where $tag_{zk}(z, F, s, n, \underline{l}, \underline{\tilde{b}}) := \hat{m}$.
- $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$.
- Let $\tilde{R}_1 \| \dots \| \tilde{R}_l \| \tilde{a}_1 \| \dots \| \tilde{a}_n := w$. Then there is a subterm $e = \{e'\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$ of F such that $\bar{c}(C_e(w)) = \{t\}^{r_i}_{\mathsf{ek}(a)} =: g$ for some $a \in \mathsf{A}$ and t. (We abbreviate $C_e := C^{s,n,\underline{l}}_{e,\underline{\tilde{b}}}$.)

Furthermore, since $S' \not\vdash_r r_i$, we have that $r_i \in \mathsf{Rand}_{ag}$. Since $\bar{c}(C_e(w)) = g = \{t\}^{r_i}_{\mathsf{ek}(a)}$ with $r_i \in \mathsf{Rand}_{ag}$, by definition of $\bar{c}$, we have that $C_e(w)$ is the result of a prior invocation $\mathrm{construct}^{a,sid}(r, \tilde{\tau})$

with $r = \{r'\}^{R'}_{\mathsf{ek}(x)}$ and $r'\gamma^{a,sid}\tilde{\tau} = t$ and $r_i = r^{a,R',sid}$ and $\tilde{\tau}(x) = a$. Hence $r\gamma^{a,sid}\tilde{\tau} = g$. Since we have defined construct$^{a,sid}(r, \tilde{\tau})$ to compute $c^*(r\gamma^{a,sid}\tilde{\tau}) = c^*(g)$, it follows that $c^*(g)$ has been computed at some point in time and that $C_e(w) = c^*(g)$.

We distinguish the following cases:

Case 1: *"$a \in C$".*
In this case, since $S' \not\vdash_r r_i$ and $S' \vdash \mathsf{dk}(a)$, we have $S' \not\vdash \{t\}^{r_i}_{\mathsf{ek}(a)}$. Hence (like in the case of T being a nonce) we have that $c^*(g)$ is never computed in contradiction to the fact that $c^*(g)$ was computed at some point.

Case 2: *"$a \notin C$".*
In this case, by definition of ZK-circuits, $C_e(w) = c^*(g)$ implies that encrypting $m' := C_{e'}(w)$ using the encryption key of agent $\bar{c}(C_{\beta_j}(w)) = a$ and randomness $\tilde{r} := C_{\rho_i}(w)$ results in the ciphertext $C_e(w) = c^*(g)$. However, since $a \notin C$, $c^*(g)$ was produced by an *encrypt* query to the *FakeEncrypt*-oracle with respect to a key for which no *get decryption key* query has been performed. Hence we can guess the randomness used in an encryption performed by the *FakeEncrypt*-oracle which can be easily seen to contradict the IND-CCA property of $\mathcal{AE}$.

T is an adversary-generated signature (Case (e)). Fully analogous to the case of T being a nonce (Case (a)), we have that the $c^*(T)$ will never be computed during the execution, and that $\bar{c}(\hat{m})$ is executed with $\bar{c}(\hat{m}) = T$. Since $\bar{c}(\hat{m}) = T = [T']^R_{\mathsf{sk}(a)}$ for some T', a, R with $R \in \mathsf{Rand}_{adv}$ and $a \in \mathsf{A} \setminus C$, we have that $\hat{m} = tag_{sig}(m, s', vk)$ where vk is the verification key of agent a and s' is a valid signature on m with respect to vk. Furthermore, $\hat{m}$ was not computed by c^* since otherwise $\bar{c}(\hat{m})$ would have been $[\cdot]^{\mathsf{Rand}_{ag}}$. Since only c^* performs signing queries, this implies that the signature s' has not been produced by a call to the signing oracle with arguments m, vk. Furthermore, since $a \notin C$, no *get signing key* query has been made with argument vk. Thus a pair (s', m) has been produced where s' is a valid signature on m with respect to vk although s' has never been produced by the signing oracle with respect to that same key vk, and the secret key has never been revealed. This is a direct contradiction to the strong existential unforgeability of $\mathcal{SIG}$.

T is a decryption or signing key (Case (g)). Fully analogous to the case of T being a nonce (Case (a)), that $\bar{c}(\hat{m})$ is executed with $\bar{c}(\hat{m}) = T$. Assume first that $T = \mathsf{dk}(a)$ with $a \in \mathsf{A} \setminus C$. By definition of $\bar{c}$, $\bar{c}(\hat{m}) = \mathsf{dk}(a)$ means that a *check key* query with argument $\hat{m}$ succeeded and returned the encryption key vk of agent a (i.e., $\hat{m}$ is the decryption key of agent a). Furthermore, since $a \notin C$, no *get decryption key* query has been performed with argument vk. In other words, a decryption key has been guessed given only encryption and decryption queries; this is a contradiction to the IND-CCA property of $\mathcal{AE}$. In the case $T = \mathsf{sk}(a)$, we analogously get a signing key is guessed using only signing queries; this contradicts the strong existential unforgeability of $\mathcal{SIG}$. □

7 Conclusions

We have presented the first computational soundness theorem for (non-interactive) symbolic zero-knowledge proofs. This allows to analyze protocols in a simple symbolic model supporting encryptions, signatures, and zero-knowledge proofs; the computational soundness theorem then guarantees that the trace properties shown in the symbolic model carry over to the computational implementation.

Open questions and directions for further research include:

- Show analogous computational soundness results for zero-knowledge proofs in other frameworks, e.g., the BPW model [BPW03a] or the CoSP-framework [BHU09]. We believe that our techniques can easily be integrated into these frameworks under essentially the same security assumptions. In particular a proof of the computational soundness of zero-knowledge

proofs in the CoSP-framework would be worthwhile because computational soundness in that framework automatically entails computational soundness for established symbolic calculi like the applied π-calculus that can be analyzed with standard tools (e.g., ProVerif [Bla01]).

- Give computational soundness results for zero-knowledge proofs under relaxed assumptions. We require a property called extraction zero-knowledge which is a very strong requirement. More efficient zero-knowledge schemes might be possible under weaker requirements such as extractability alone. Of course, since this gives additional power to the computational adversary, the deduction relation for the symbolic adversary needs to be adapted as well to make the symbolic adversary stronger, too. First results in this direction have been achieved in [Moh09].
- Extend our framework to handle interactive zero-knowledge proofs. These need a completely different symbolic modeling as it does not make sense to, say, sign or store an interactive proof. Interactive zero-knowledge proofs exist under weaker assumptions and are often more efficient, so a result for interactive zero-knowledge proofs can lead to more efficient computational protocols.
- Give general criteria for the computational soundness of zero-knowledge proofs when interacting with other cryptographic primitives. In this paper, we showed that zero-knowledge proofs together with encryptions and signatures are computationally sound. If another primitive is added, the proof needs to be redone. General criteria might allow to add more primitives to the model without having to redo all proofs each time.
- Find a simpler proof. Currently, the largest part of our proof is devoted to checking that various seemingly obvious conditions hold. Using a simpler (or at least more uniform) symbolic and computational model might allow to strongly reduce the complexity of the proof. Alternatively, an automated or semi-automated verification of these conditions might be desirable.

Acknowledgements. We thank Esfandiar Mohammadi for valuable discussions and help in writing the final version of this manuscript. We also thank the anonymous referees for valuable suggestions on how to improve this manuscript. This work has been partially funded by the German Research Foundation (DFG) through the Cluster of Excellence "Multimodal Computing and Interaction" and the DFG grant 3194/1-1.

A Postponed proofs

Proof of Lemma 2. By Definition 6, $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is defined if the following two conditions are satisfied:

(a) For any subterm of the form $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ of F, we have that $\tilde{b}_i$ is of type agent.

(b) s is the ρ-arity of F, n is the α-arity of F, $|\underline{l}| = n$, and $|\underline{\tilde{b}}|$ is the β-arity of F.

By definition of $\mathrm{construct}^{a,sid}(r,\tau)$, when $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is constructed, we have that $r = \mathsf{ZK}^R_F(R_1,\ldots,R_s;a_1,\ldots,a_n;b_1,\ldots,b_m)$, $\tilde{b}_i = \mathrm{construct}^{a,sid}(b_i,\tau)$, $|\underline{\tilde{b}}| = m$, and $|\underline{l}| = n$. By the definition of the syntax of patterns (page 12), we have that the ρ-, α-, and β-arity of F are s, n, and m, respectively. Hence (b) is fulfilled.

By Definition 4, condition 6, for any subterm of the form $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ of F we have that $b_i \in \mathsf{X}.a$. Hence $\tau(b_i)$ has type agent. By Definition 7, we have that $\tilde{b}_i = \mathrm{construct}^{a,sid}(b_i,\tau) = \tau(b_i)$. Thus $\tilde{b}_i$ has type agent and (a) is fulfilled. □

Proof of Lemma 3. $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w)$ is constructed and invoked by $\mathrm{construct}^{a,sid}(r,\tau)$ with $w = \tilde{R}_1\|\ldots\|\tilde{R}_s\|\tilde{a}_1\|\ldots\|\tilde{a}_n$. By definition of $\mathrm{construct}^{a,sid}$ (Definition 7), this only happens if $r =$

$\mathsf{ZK}_F^R(R_1,\ldots,R_s;a_1,\ldots,a_n;b_1,\ldots,b_m)$ with $|\underline{l}| = n$ and $l_i = |\tilde{a}_i|$ and $\tilde{a}_i = \text{construct}^{a,sid}(a_i,\tau)$ and $\tilde{b}_i = \text{construct}^{a,sid}(b_i,\tau)$ and $\tilde{R}_i = tape^{a,sid}(R_i)$. We perform an induction on the structure of T. (And abbreviate $C_T := C^{s,n,\underline{l}}_{T,\underline{\tilde{b}}}$.)

Case 1: *"T is a ZK-term of the form $\mathsf{ek}(\beta_i)$".*
By Definition 4, condition 6, $b_i \in \mathsf{X}.a$ and thus $\tilde{b}_i = \text{construct}^{a,sid}(b_i,\tau) = \tau(b_i) = tag_{agent}(b'_i)$ for some b'_i. Then by definition of $\text{construct}^{a,sid}$ (Definition 7) and of ZK-circuits (Definition 6), we have $\text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau) = \text{construct}^{a,sid}(\mathsf{ek}(b_i),\tau) = tag_{ek}(ek_{b'_i}) = C_T(w)$ where $\mathsf{ek}_{b'_i}$ is the encryption key of agent b'_i.

Case 2: *"T is a ZK-term of the form $\mathsf{vk}(\beta_i)$".*
Fully analogous to the previous case.

Case 3: *"T is a ZK-term of the form α_i".*
We have that $C_T(w) = \tilde{a}_i = \text{construct}^{a,sid}(a_i,\tau) = \text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$.

Case 4: *"T is a ZK-term of the form β_i".*
We have that $C_T(w) = \tilde{b}_i = \text{construct}^{a,sid}(b_i,\tau) = \text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$.

Case 5: *"T is a ZK-term of the form $\langle T_1,T_2\rangle$".*
By definition of $\text{construct}^{a,sid}$ (Definition 7) and of ZK-circuits (Definition 6), and using the induction hypothesis, we have that $C_T(w) = tag_{pair}(C_{T_1}(w),C_{T_2}(w)) = tag_{pair}(\text{construct}^{a,sid}(T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau),\text{construct}^{a,sid}(T_2\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)) = \text{construct}^{a,sid}(\langle T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},T_2\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\rangle,\tau) = \text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$.

Case 6: *"T is a ZK-term of the form $\{T'\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$".*
By Definition 4, condition 6, $b_j \in \mathsf{X}.a$ and thus $\tilde{b}_j = \text{construct}^{a,sid}(b_j,\tau) = \tau(b_j) = tag_{agent}(b'_j)$ for some b'_j. Then by definition of $\text{construct}^{a,sid}$, we have $\text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau) = \text{construct}^{a,sid}(\{T'\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\}^{R_i}_{\mathsf{ek}(b_j)},\tau) = tag_{enc}(m,\mathsf{ek}_{b'_j})$ where $\mathsf{ek}_{b'_j}$ is the encryption key of agent b'_j, and m is the result of encrypting $m' := \text{construct}(T'\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$ using the key $\mathsf{ek}_{b'_j}$ and the randomness $tape^{a,sid}(R_i) = \tilde{R}_i$. Furthermore, by definition of ZK-circuits, $C_T(w) = tag_{enc}(\bar{m},\mathsf{ek}_{b'_j})$ where $\bar{m}$ is the result of encrypting $\bar{m}' := C_{T'}(w)$ using the key $\mathsf{ek}_{b'_j}$ and the randomness $\tilde{R}_i$. By induction hypothesis, $\bar{m}' = m'$, and hence $\bar{m} = m$ and thus $C_T(w) = \text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$.

□

Proof (of Lemmas 4 and 5). In the following, we abbreviate $\gamma^{a,sid}(\bar{c}\circ\tau)$ as $\triangle$ and $\gamma^{a',sid'}(\bar{c}\circ\tau')$ as $\triangle'$.

We show the following three statements:

(a) For any invocation of $\text{construct}^{a,sid}(r,\tau)$ that is performed during the execution, we have that $\text{construct}^{a,sid}$ succeeds iff $r\triangle$ contains true proofs and that in this case $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$.

(b) Whenever a circuit $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is constructed and $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w)$ is evaluated with some $w = \tilde{R}_1\|\ldots\|\tilde{R}_l\|\tilde{a}_1\|\ldots\|\tilde{a}_n$ (indirectly through a call to $\text{construct}^{a,sid}(r,\tau)$ with $r = \mathsf{ZK}_F^R(\underline{R};\underline{a};\underline{b})$), we have that $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w) = 1$ iff $\mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\ldots,r^{a,R_l,sid};\bar{c}(\tilde{a}_1),\ldots,\bar{c}(\tilde{a}_n);\bar{c}(\tilde{b}_1),\ldots,\bar{c}(\tilde{b}_s))$ is a true proof.

(c) Whenever $\bar{c}(m')$ is computed for some m', $\bar{c}(m')$ contains true proofs.

Lemmas 4 and 5 then follow from statements (a) and (c).

We prove the statements (a), (b), and (c) by induction on the point in time where the computation is performed (more exactly, the point in time at which the computation finishes). That is, when showing that (a) holds, we can assume that (a), (b), and (c) hold for all earlier points in time; and similarly for (b) and (c).

Showing (a)**.** We distinguish the following cases depending on r. (Note that r is a subterm of a pattern; cf. page 12 for the syntax of patterns):

Case a-1: *"$r \in \mathsf{X}$".*
In this case r is a variable in a construct-pattern occurring in a role. By Definition 4, condition 5, it is guaranteed that r is a free variable of a parse-pattern occurring earlier on the same path in the role. Hence $\tau(r)$ will be defined in the invocation of $\text{construct}^{a,sid}(r,\tau)$. Then $\text{construct}^{a,sid}(r,\tau)$ succeeds and returns $\tau(r)$. Thus $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \bar{c}(\tau(r))$. Furthermore $r\triangle = r(\bar{c}\circ\tau) = \bar{c}(\tau(r))$. Hence $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$. Furthermore, by induction hypothesis, (c) holds for $m := \tau(r)$, hence $r\triangle = \bar{c}(m)$ contains true proofs. Thus $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs (namely always).

Case a-2: *"$r = \mathsf{ek}(x)$ for some $x \in \mathsf{X}.a$".*
As in the case $r \in \mathsf{X}$, we have that $\tau(x)$ is defined. Furthermore, since $x \in \mathsf{X}.a$ we have that $\tau(x)$ is of type agent. Thus $\tau(x) = tag_{agent}(a')$ for some $a' \in \mathsf{A}$. Hence $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \bar{c}(tag_{ek}(ek_{a'}))$ where $ek_{a'}$ is the encryption key of agent a'. Since $ek_{a'}$ is the encryption key of an existing agent a', $\bar{c}(tag_{ek}(ek_{a'})) = \mathsf{ek}(a')$. Moreover $r\triangle = \mathsf{ek}(x)(\bar{c}\circ\tau) = \mathsf{ek}(\bar{c}(\tau(x))) = \mathsf{ek}(\bar{c}(tag_{agent}(a'))) = \mathsf{ek}(a')$. Thus $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$. Furthermore, as $\mathsf{ek}(a')$ does not contain any ZK-subterms, it always contains true proofs.

Case a-3: *"$r = \mathsf{vk}(x)$ for some $x \in \mathsf{X}.a$".*
Fully analogous to the previous case.

Case a-4: *"$r = \mathsf{dk}(x)$ for some $x \in \mathsf{X}.a$".*
Let i be such that the session sid executes the i-th role. By Definition 4, condition 9, it follows that $x = A_i$, and hence $\tau(x) = tag_{agent}(a)$. Hence $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \bar{c}(tag_{dk}(dk_a))$ where dk_a is the decryption key of agent a. Since dk_a is the decryption key of an existing agent a, $\bar{c}(tag_{dk}(dk_a)) = \mathsf{dk}(a)$. Moreover $r\triangle = \mathsf{dk}(x)(\bar{c}\circ\tau) = \mathsf{dk}(\bar{c}(\tau(x))) = \mathsf{dk}(\bar{c}(tag_{agent}(a))) = \mathsf{dk}(a)$. Thus $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$. Furthermore, as $\mathsf{dk}(a)$ does not contain any ZK-subterms, it always contains true proofs.

Case a-5: *"$r = \mathsf{sk}(x)$".*
This case does not occur because a pattern can contain the subterm $\mathsf{sk}(x)$ only within $[\dots]^{\dots}_{\mathsf{sk}(x)}$. However, a call $\text{construct}^{a,sid}([\dots]^{\dots}_{\mathsf{sk}(x)},\dots)$ does not invoke $\text{construct}^{a,sid}(\mathsf{sk}(x),\dots)$ but instead accesses the needed signing key directly. Hence no call of the form $\text{construct}^{a,sid}(\mathsf{sk}(x),\dots)$ will ever be performed.

Case a-6: *"$r = \langle r_1, r_2\rangle$".*
By definition, $\text{construct}^{a,sid}(r,\tau) = tag_{pair}(\text{construct}^{a,sid}(r_1,\tau), \text{construct}^{a,sid}(r_2,\tau))$. Hence $\text{construct}^{a,sid}(r_1,\tau)$ is invoked before the completion of the computation of $\text{construct}^{a,sid}(r,\tau)$. And if $\text{construct}^{a,sid}(r_1,\tau)$ succeeds, $\text{construct}^{a,sid}(r_2,\tau)$ is invoked before the completion of the computation of $\text{construct}^{a,sid}(r,\tau)$.

Assume that $\text{construct}^{a,sid}(r_1,\tau)$ or $\text{construct}^{a,sid}(r_2,\tau)$ does not succeed. Then $\text{construct}^{a,sid}(r,\tau)$ does not succeed. And by induction hypothesis, $r_1\triangle$ or $r_2\triangle$ does not contain true proofs. Hence $r\triangle = \langle r_1\triangle, r_2\triangle\rangle$ does not contain true proofs either. Thus in this case, $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs (namely never).

Assume that both $\text{construct}^{a,sid}(r_1,\tau)$ and $\text{construct}^{a,sid}(r_1,\tau)$ succeed. Then $\text{construct}^{a,sid}(r,\tau)$ succeeds. And by the induction hypothesis, $r_1\triangle$ and $r_2\triangle$ contain true proofs. Thus $r\triangle = \langle r_1\triangle, r_2\triangle\rangle$ contains true proofs.

Thus in this case, $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs (namely always). Furthermore, again using the induction hypothesis, $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \bar{c}(tag_{pair}(\text{construct}^{a,sid}(r_1,\tau), \text{construct}^{a,sid}(r_2,\tau))) = \langle \bar{c}(\text{construct}^{a,sid}(r_1,\tau)), \bar{c}(\text{construct}^{a,sid}(r_2,\tau))\rangle = \langle r_1\triangle, r_2\triangle\rangle = r\triangle$.

Case a-7: *"$r = \{r'\}^R_{\mathsf{ek}(x)}$ with $R \in \mathsf{Rand}_{ag}$ and $x \in \mathsf{X}.a$".*
As in the case $r = \mathsf{ek}(x)$, we have that $\tau(x)$ is defined and $\tau(x) = tag_{agent}(a')$ for some $a' \in \mathsf{A}$. By definition, $\text{construct}^{a,sid}(r,\tau)$ invokes $\text{construct}^{a,sid}(r',\tau)$ and succeeds iff $\text{construct}^{a,sid}(r',\tau)$ succeeds. Hence $\text{construct}^{a,sid}(r',\tau)$ is invoked before the completion of the computation of $\text{construct}^{a,sid}(r,\tau)$. From the induction hypothesis, it then follows that $\text{construct}^{a,sid}(r',\tau)$ succeeds iff $r'\triangle$ contains true proofs. Since $r'\triangle$ contains true proofs iff $r\triangle = \{r'\triangle\}^{r^{a,R,sid}}_{\mathsf{ek}(a')}$ contains true proofs, it follows that $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs.

In the case that $\text{construct}^{a,sid}(r,\tau)$ succeeds, $\text{construct}^{a,sid}(r,\tau) = \text{construct}^{a,sid}(\{r'\}^R_{\mathsf{ek}(x)})$ is of type ciphertext, and hence $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \{r'\triangle\}^{r^{a,R,sid}}_{\mathsf{ek}(\bar{c}(\tau(x)))}$ (we are in the case of the definition of $\bar{c}$ where the argument of $\bar{c}$ is of type ciphertext and has been generated by a call to $\text{construct}^{a,sid}$). Since $r\triangle = \{r'\triangle\}^{r^{\triangle}}_{\mathsf{ek}(\bar{c}(\tau(x\gamma^{a,sid})))} = \{r'\triangle\}^{r^{a,R,sid}}_{\mathsf{ek}(\bar{c}(\tau(x)))}$, we have that $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$.

Case a-8: *"$r = [r']^R_{\mathsf{sk}(x)}$ where $R \in \mathsf{Rand}_{ag}$ and $x \in \mathsf{X}.a$".*
Let i be such that the session sid executes the i-th role. By Definition 4, condition 8, we have that $x = A_i$ and hence we have that $\tau(x)$ is defined and $\tau(x) = tag_{agent}(a)$. By definition, $\text{construct}^{a,sid}(r,\tau)$ invokes $\text{construct}^{a,sid}(r',\tau)$ and succeeds iff $\text{construct}^{a,sid}(r',\tau)$ succeeds. Hence $\text{construct}^{a,sid}(r',\tau)$ is invoked before the completion of the computation of $\text{construct}^{a,sid}(r,\tau)$. From the induction hypothesis, it then follows that $\text{construct}^{a,sid}(r',\tau)$ succeeds iff $r'\triangle$ contains true proofs. Since $r'\triangle$ contains true proofs iff $r\triangle = [r'\triangle]^{r^{a,R,sid}}_{\mathsf{sk}(a)}$ contains true proofs, it follows that $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs.

Let vk_a and sk_a be the verification and signing key of agent a. In the case that $\text{construct}^{a,sid}(r,\tau)$ succeeds, $\text{construct}^{a,sid}(r,\tau) = \text{construct}^{a,sid}([r']^R_{\mathsf{sk}(x)}) = tag_{sig}(s, m', vk_a)$ where $m' = \text{construct}^{a,sid}(r',\tau)$ and s is a signature of m' using sk_a. Then $\text{construct}^{a,sid}(r,\tau)$ is of type signature and $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = [\bar{c}(m')]^{r^{a,R,sid}}_{\mathsf{sk}(a)}$ (we are in the case of the definition of $\bar{c}$ where the argument of $\bar{c}$ has been generated by a call to $\text{construct}^{a,sid}$). From the induction hypothesis, we have that $\bar{c}(m') = r'\triangle$ and hence $r\triangle = [r'\triangle]^{r^{a,R,sid}}_{\mathsf{sk}(\bar{c}(\tau(x)))} = [\bar{c}(m')]^{r^{a,R,sid}}_{\mathsf{sk}(a)}$. Thus $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = r\triangle$.

Case a-9: *"$r = \{r'\}^{\_}_{\mathsf{ek}(x)}$ or $r = [r']^{\_}_{\mathsf{sk}(x)}$ or $r = \mathsf{ZK}^{\_}_F(\_^*;\_^*;\underline{r})$".*
This case cannot occur, since r contains $\_$ which is excluded by Definition 4, condition 5.

Case a-10: *"$r = \mathsf{ZK}^R_F(R_1,\ldots,R_s;a_1,\ldots,a_n;b_1,\ldots,b_m)$ where $R, \underline{R} \in \mathsf{Rand}_{ag}$".*
We begin by showing that $\text{construct}^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs. To ease formulations, we introduce abbreviations (i)–(viii) for the following statements:

(i) $\text{construct}^{a,sid}(r,\tau)$ succeeds.
(ii) $\text{construct}^{a,sid}(T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)$ succeeds for all subterms T of F.
(iii) For all $t \in \{\underline{a},\underline{b}\}$, $\text{construct}^{a,sid}(t,\tau)$ succeeds (we denote the return values as $\tilde{\underline{a}},\tilde{\underline{b}}$).
(iv) $C := C^{s,n,\underline{l}}_{F,\tilde{\underline{b}}}$ is defined and $C(w) = 1$ where $w := \tilde{R}_1\|\ldots\|\tilde{R}_s\|\tilde{a}_1\|\ldots\|\tilde{a}_n$ and $l_i := |\tilde{a}_i|$ and $\tilde{R}_i := tape^{a,sid}(R_i)$.
(v) $T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ contains true proofs for all subterms T of F.
(vi) For all $t \in \{\underline{a},\underline{b}\}$, $t\triangle$ contains true proofs.
(vii) $r\triangle$ is a true proof.
(viii) $r\triangle$ contains true proofs.

By definition of construct^{a,sid}, we have (i) $\Leftrightarrow$ (ii)$\wedge$(iii)$\wedge$(iv). (Note that construct$^{a,sid}(t,\tau)$ may also abort if a ZK-failure occurs, however, this is explicitly excluded by the assumptions of Lemma 4.) Since the calls to construct^{a,sid} in statements (ii) and (iii) are performed before the completion of the invocation construct$^{a,sid}(r,\tau)$, we can apply the induction hypothesis and get (ii) $\wedge$ (iii) $\Leftrightarrow$ (v) $\wedge$ (vi). Hence (i) $\Leftrightarrow$ (ii) $\wedge$ (iii) $\wedge$ (iv) $\Leftrightarrow$ (v) $\wedge$ (vi) $\wedge$ (iv). Moreover, by induction hypothesis we have $\bar{c}(\tilde{a}_i) = \bar{c}(\text{construct}^{a,sid}(a_i,\tau)) = a_i\triangle$ and analogously $\bar{c}(\tilde{b}_i) = b_i\triangle$. By Lemma 2 we have that $C = C^{s,n,\underline{l}}_{F,\tilde{b}}$ is defined, hence (iv) holds iff $C(w) = 1$. By the induction hypothesis, (b) holds for $C(w)$, thus $C(w) = 1$ iff $z := \mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\dots;\bar{c}(\tilde{a}_1),\dots;\bar{c}(\tilde{b}_1),\dots)$ is a true proof. Since $\bar{c}(\tilde{a}_i) = a_i\triangle$ and $\bar{c}(\tilde{b}_i) = b_i\triangle$, we have that $z = r\triangle$. Thus $C(w) = 1$ iff $r\triangle$ is a true proof, hence (iv) $\Leftrightarrow$ (vii). Hence (i) $\Leftrightarrow$ (v) $\wedge$ (vi) $\wedge$ (iv) $\Leftrightarrow$ (v) $\wedge$ (vi) $\wedge$ (vii). Since $r\triangle = \mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\dots;a_1\triangle,\dots;b_1\triangle,\dots) =: z$ contains true proofs iff z is a true proof and all proper subterms of z contain true proofs, we have that (viii) $\Leftrightarrow$ (vii) $\wedge$ (vi). Hence (i) $\Leftrightarrow$ (v) $\wedge$ (vi) $\wedge$ (vii) $\Leftrightarrow$ (v) $\wedge$ (viii). Assume that (v) does not hold, i.e., for some subterm T of F, $T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ does not contain true proofs. Since F and hence T does not contain a ZK-constructor itself, this implies that $\chi\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ does not contain true proofs for some $\chi \in \{\underline{\alpha},\underline{\beta}\}$. Since $\chi\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle = t\triangle$ for some $t \in \{\underline{a},\underline{b}\}$, and $t\triangle$ is a subterm of $r\triangle$, this implies that $r\triangle$ does not contain true proofs, i.e., (viii) does not hold. Thus by contraposition, (viii) $\Rightarrow$ (v) and hence (i) $\Leftrightarrow$ (v) $\wedge$ (viii) $\Leftrightarrow$ (viii). Hence we have that construct$^{a,sid}(r,\tau)$ succeeds iff $r\triangle$ contains true proofs.

In the case that construct$^{a,sid}(r,\tau)$ succeeds, construct$^{a,sid}(r,\tau)$ returns a bitstring of type zero-knowledge proof (unless a ZK-failure occurs which is excluded by the assumptions of Lemma 4). Hence by definition of $\bar{c}$ (in the case of honestly generated zero-knowledge proofs), we have that $\bar{c}(\text{construct}^{a,sid}(r,\tau)) = \mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\dots;a_1\triangle;b_1\triangle) = r\triangle$.

This covers all cases for r and thus shows (a) (assuming the induction hypothesis).

Showing (b)**.** In case (b), $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}(w)$ is constructed and invoked by construct$^{a,sid}(r,\tau)$ with $w = \tilde{R}_1\|\dots\|\tilde{R}_s\|\tilde{a}_1\|\dots\|\tilde{a}_n$ and $r = \mathsf{ZK}^R_F(R_1,\dots,R_s;a_1,\dots,a_n;b_1,\dots,b_m)$. By definition of construct^{a,sid} this implies $|\tilde{l}| = n$ and $l_i = |\tilde{a}_i|$ and $\tilde{a}_i = \text{construct}^{a,sid}(a_i,\tau)$ and $\tilde{b}_i = \text{construct}^{a,sid}(b_i,\tau)$ and $\tilde{R}_i = \mathit{tape}^{a,sid}(R_i)$. In the following, we abbreviate $C_T := C^{s,n,\underline{l}}_{T,\underline{\tilde{b}}}$ for any subterm T of F. Note that from the induction hypothesis, for any subterm T of F we have

$$T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{\text{(a)}}{=} \bar{c}(\text{construct}^{a,sid}(T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\},\tau)) \overset{\text{L.3}}{=} \bar{c}(C_T(w)). \tag{3}$$

By definition of ZK-formulas, F can be written as $B(T_1,\dots,T_q)$ where B is a Boolean formula and $T_1,\dots,T_q$ are ZK-atoms. We first show that for every $i \in [q]$, we have that

$$T_i\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \text{ true} \quad\Longleftrightarrow\quad C_{T_i}(w) = 1. \tag{4}$$

Here "true" is used in the sense of Definition 2.

Note that whenever some $\mathsf{ek}(\beta_j)$ or $\mathsf{vk}(\beta_j)$ occurs in T, then by Definition 4, condition 6, it follows that $b_j \in \mathsf{X}.a$, hence $\tau(b_j) = \mathit{tag}_{\mathit{agent}}(b')$ for some agent b', and thus $b_j\triangle = \bar{c}(\tau(b_j)) = b' \in \mathsf{A}$. Hence, in the following we will implicitly use the facts that if $\mathsf{ek}(\beta_j)$ or $\mathsf{vk}(\beta_j)$ occurs in T, then $b_j\triangle \in \mathsf{A}$ and $b_j\triangle = \bar{c}(\tau(b_j))$ and $\tau(b_j) = \mathit{tag}_{\mathit{agent}}(b_j\triangle)$.

We distinguish three cases for $T := T_i$:

Case b-1: *"$T = (t_1 = t_2)$ for ZK-terms t_1, t_2".*

By Definition 2, we have that $T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $t_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle = t_2\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$. By (3), this holds iff $\bar{c}(C_{t_1}(w)) = \bar{c}(C_{t_2}(w))$. By the injectivity of $\bar{c}$, this holds iff $C_{t_1}(w) = C_{t_2}(w)$. And by the definition of ZK-circuits, this holds iff $C_T(w) = 1$. Hence $T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $C_T(w) = 1$.

Case b-2: *"$T = (\text{Decrypt}(t_1, \alpha_i, \mathsf{ek}(\beta_j)) = t_2)$ for ZK-terms t_1, t_2".*
To facilitate notation, we introduce the abbreviations (i)–(x) for the following statements:

(i) $T\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$ is true.

(ii) $t_1\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle = \{t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle\}^{R'}_{\mathsf{ek}(b_j\triangle)}$ for some $R' \in \mathsf{Rand}$.

(iii) $a_i\triangle = \mathsf{dk}(b_j\triangle)$.

(iv) $C_{t_1}(w)$ is of type ciphertext. In this case, let $tag_{enc}(c, ek') := C_{t_1}(w)$.

(v) $C_{\alpha_i}(w)$ is of type decryption key. In this case, let $tag_{dk}(dk) := C_{\alpha_i}(w)$.

(vi) $C_{\mathsf{ek}(\beta_j)}(w)$ is of type encryption key. In this case, let $tag_{ek}(ek) := C_{\mathsf{ek}(\beta_j)}$.

(vii) $ek = ek'$.

(viii) (ek, dk) is a valid encryption/decryption key pair.

(ix) Decrypting c using the decryption key dk yields the plaintext $C_{t_2}(w)$.

(x) $C_T(w) = 1$.

$T\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$ is equal to $(\text{Decrypt}(t_1\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle, a_i\triangle, \mathsf{ek}(b_j\triangle)) = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle)$. By Definition 2, $T\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$ is true iff it has the form $\text{Decrypt}(\{\bar{m}_1\}^{\bar{r}_1}_{\mathsf{ek}(\bar{a}_1)}, \mathsf{dk}(\bar{a}_1), \mathsf{ek}(\bar{a}_1)) = \bar{m}_1$ for some terms $\bar{r}_1, \bar{m}_1, \bar{a}_1$. Combining these two facts, we have that $T\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$ is true iff (ii) and (iii) hold. Thus (i) $\Leftrightarrow$ (ii) $\wedge$ (iii).

Furthermore, (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix) $\Leftrightarrow$ (x) by definition of ZK-circuits (Definition 6).

To show (ii) $\wedge$ (iii) $\Leftrightarrow$ (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix) and thus conclude the proof of (i) $\Leftrightarrow$ (x), we need to distinguish two subcases:

Case b-2.1: *"$C_{t_1}(w)$ equals the result of some earlier call $\text{construct}^{a',sid'}(\{t'\}^{R''}_{\mathsf{ek}(x')}, \tau')$ for some $a', sid', R'', x', t', \tau'$".*
Assume that (ii) and (iii) hold. Then $x' \in \mathsf{X}.a$ and $b' := \bar{c}(\tau'(x)) \in \mathsf{A}$. Then, by definition of $\text{construct}^{a',sid'}$, it follows that $C_{t_1}(w) = tag_{enc}(c, ek')$ where ek' is the encryption key of agent b', and c is the result of encrypting $m := \text{construct}^{a',sid'}(t', \tau')$ with encryption key ek'. Hence (iv) holds. Moreover, we have

$$\{t'\triangle'\}^{r^{a',R'',sid'}}_{\mathsf{ek}(b')} \overset{(a)}{=} \bar{c}(\text{construct}^{a',sid'}(\{t'\}^{R''}_{\mathsf{ek}(x')}, \tau')) = \bar{c}(C_{t_1}(w)) \overset{(3)}{=} t_1\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(ii)}{=} \{t_2\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle\}^{R'}_{\mathsf{ek}(b_j\triangle)}. \quad (5)$$

From (5), we have that $b' = b_j\triangle$. Then, by definition of ZK-circuits, $C_{\mathsf{ek}(\beta_j)}(w) = tag_{ek}(ek')$ since ek' is the encryption key of agent b'. Thus (vi) and (vii) follow.
Let dk denote the decryption key of agent b'. Then

$$\bar{c}(tag_{dk}(dk)) = \mathsf{dk}(b') = \mathsf{dk}(b_j\triangle) \overset{(iii)}{=} a_i\triangle = \alpha_i\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(3)}{=} \bar{c}(C_{\alpha_i}(w)).$$

Since $\bar{c}$ is injective, we have $C_{\alpha_i}(w) = tag_{dk}(dk)$. This shows (v) and (viii) (the latter because we already know that $C_{\mathsf{ek}(\beta_j)}(w) = tag_{ek}(ek)$ where ek is the encryption key of b').
Then

$$\bar{c}(m) = \bar{c}(\text{construct}^{a',sid'}(t', \tau')) \overset{(a)}{=} t'\triangle' \overset{(5)}{=} t_2\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_2}(w)).$$

Since $\bar{c}$ is injective, we have that $C_{t_2}(w) = m$. Furthermore, since c is the result of encrypting m with encryption key ek', and since dk is the decryption key corresponding

to $ek' = ek$, we have that decrypting c using the decryption key dk yields the plaintext $m = C_{t_2}(w)$. Thus (ix) holds.
Summarizing, we have that (ii) $\wedge$ (iii) $\Rightarrow$ (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix).

Assume now that (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix) holds.
Let $b' := b_j\triangle$. Then $C_{\mathsf{ek}(\beta_j)}(w) = tag_{ek}(ek)$ where ek is the encryption key of b'. Furthermore, since (ek, dk) is a valid encryption/decryption key pair (by (viii)), and since for every encryption key, there is at most one corresponding decryption key, we have that dk is the decryption key of b'. Hence $a_i\triangle = \alpha_i\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(3)}{=} \bar{c}(C_{\alpha_i}(w)) \overset{(v)}{=} \bar{c}(tag_{dk}(dk)) = \mathsf{dk}(b') = \mathsf{dk}(b_j\triangle)$. Hence (iii) holds.
Since $C_{t_1}(w) = \text{construct}^{a',sid'}(\{t'\}^{R''}_{\mathsf{ek}(x')}, \tau')$, by definition of $\text{construct}^{a',sid'}$, we have that $C_{t_1} = tag_{enc}(\hat{c}, \hat{ek})$ where $\hat{ek}$ is the encryption key of agent $\bar{c}(\tau'(x'))$ and $\hat{c}$ is the result of encrypting $m := \text{construct}^{a',sid'}(t', \tau')$ under $\hat{ek}$. Since $C_{t_1}(w) \overset{(iv)}{=} tag_{enc}(c, ek')$, it follows that $\hat{c} = c$ and $\hat{ek} = ek' \overset{(vii)}{=} ek$. Since dk and ek are the decryption and encryption key of b', decrypting c with dk then yields m. With (ix) this implies $m = C_{t_2}(w)$.
Then using the induction hypothesis, $\bar{c}(C_{t_2}(w)) = \bar{c}(m) = \bar{c}(\text{construct}^{a',sid'}(t', \tau')) \overset{(a)}{=} t'\triangle'$. And by (3), $\bar{c}(C_{t_2}(w)) = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. Thus $t'\triangle' = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. And since $\hat{ek} = ek$ is the encryption key of $b' = b_j\triangle$ and of $\bar{c}(\tau'(x'))$, we have that $\bar{c}(\tau'(x')) = b_j\triangle$. Then with $R' := r^{a',R'',sid'}$ we get

$$t_1\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_1}(w)) = \bar{c}(\text{construct}^{a',sid'}(\{t'\}^{R''}_{\mathsf{ek}(x')}, \tau')) \overset{(a)}{=} \{t'\triangle'\}^{R'}_{\mathsf{ek}(\bar{c}(\tau'(x')))} = \{t_2\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle\}^{R'}_{\mathsf{ek}(b_j\triangle)}$$

This proves (ii).
Thus we have that (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix) $\Rightarrow$ (ii) $\wedge$ (iii).
So altogether, we have (iv) $\wedge$ (v) $\wedge$ (vi) $\wedge$ (vii) $\wedge$ (viii) $\wedge$ (ix) $\Leftrightarrow$ (ii) $\wedge$ (iii) in Case b-2.1.

Case b-2.2: *"$C_{t_1}(w)$ is not the result of some earlier call* $\text{construct}^{a',sid'}(\{t'\}^{R''}_{\mathsf{ek}(x')}, \tau')$*".*
Assume that (ii) and (iii) hold. Let $b' := b_j\triangle$. By (3), we have $\bar{c}(C_{t_1}(w)) = t_1\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. Using (ii), this implies $\bar{c}(C_{t_1}(w)) = \{t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle\}^{R'}_{\mathsf{ek}(b')}$. Furthermore, since $G^\perp$ occurs in the range of $\bar{c}$ only under $\{\cdot\}^{\cdot}_{\cdot}$ and $G^\perp$ does not occur in $t_2, \underline{a}, \underline{b}$, we have $t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle \neq G^\perp$. By definition of $\bar{c}$ (the case of ciphertexts not generated by an honest agent), $\bar{c}(C_{t_1}(w)) = \{t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle\}^{R'}_{\mathsf{ek}(b')}$ with $t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle \neq G^\perp$ can only happen if $C_{t_1}(w) = tag_{enc}(c, ek')$ for some c and ek' where ek' is the encryption key of agent b' and decrypting c with the decryption key dk of agent b' results in a plaintext m' such that $\bar{c}(m') = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. This shows (iv).
Then, by definition of ZK-circuits, $C_{\mathsf{ek}(\beta_j)} = tag_{ek}(ek')$ since ek' is the encryption key of agent b'. Thus (vi) and (vii) follow.
Let dk denote the decryption key of agent b'. Then

$$\bar{c}(tag_{dk}(dk)) = \mathsf{dk}(b') = \mathsf{dk}(b_j\triangle) \overset{(iii)}{=} a_i\triangle = \alpha_i\{\tfrac{r,a,b}{\rho,\alpha,\beta}\}\triangle \overset{(3)}{=} \bar{c}(C_{\alpha_i}(w))$$

Since $\bar{c}$ is injective, we have $C_{\alpha_i}(w) = tag_{dk}(dk)$. This shows (v) and (viii) (the latter because we already know that $C_{\mathsf{ek}(\beta_j)}(w) = tag_{ek}(ek')$ where ek' is the encryption key of b').
Finally, as noted above, decrypting c with the decryption key dk results in a plaintext m' with $\bar{c}(m') = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. By (3), $\bar{c}(C_{t_2}(w)) = t_2\{\frac{r,a,b}{\rho,\alpha,\beta}\}\triangle$. Thus $\bar{c}(m') = \bar{c}(C_{t_2}(w))$, and using the injectivity of $\bar{c}$ we have $m' = C_{t_2}(w)$. This shows (ix).

Thus we have that (ii) ∧ (iii) ⇒ (iv) ∧ (v) ∧ (vi) ∧ (vii) ∧ (viii) ∧ (ix).

Assume now that (iv) ∧ (v) ∧ (vi) ∧ (vii) ∧ (viii) ∧ (ix) holds. As in Case b-2.1, (iii) follows and $ek' = ek$ is the encryption key of agent $b' := b_j\triangle$ and dk is the decryption key of agent b'. By (ix) and the definition of $\bar{c}$ (the case of not honestly generated ciphertexts) we have that $\bar{c}(C_{t_1}(w)) = \{\bar{c}(C_{t_2}(w))\}^{R'}_{\mathsf{ek}(b')}$ with $R' := R_{adv}^{C^{t_1}(w)}$. Hence

$$t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_1}(w)) = \{\bar{c}(C_{t_2}(w))\}^{R'}_{\mathsf{ek}(b')} \overset{(3)}{=} \{t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle\}^{R'}_{\mathsf{ek}(b_j\triangle)}$$

Hence (ii) follows.
Thus we have shown that (iv) ∧ (v) ∧ (vi) ∧ (vii) ∧ (viii) ∧ (ix) ⇒ (ii) ∧ (iii).
Thus altogether we have shown (iv) ∧ (v) ∧ (vi) ∧ (vii) ∧ (viii) ∧ (ix) ⇔ (ii) ∧ (iii).

Thus we have shown

$$\begin{aligned}\text{(i)} &\Leftrightarrow \text{(ii)} \wedge \text{(iii)}\\ &\Leftrightarrow \text{(iv)} \wedge \text{(v)} \wedge \text{(vi)} \wedge \text{(vii)} \wedge \text{(viii)} \wedge \text{(ix)}\\ &\Leftrightarrow \text{(x)},\end{aligned}$$

so $T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $C_T(w) = 1$.

This concludes the proof of Case b-2.

Case b-3: *"$T = \mathrm{Verify}(t_1, t_2, \mathsf{vk}(\beta_j))$ for ZK-terms t_1, t_2".*
To facilitate notation, we introduce the abbreviations (i)–(vii) for the following statements:

(i) $T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true.

(ii) $t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle = [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b_j\triangle)}$ for some $R' \in \mathsf{Rand}$.

(iii) $C_{t_1}(w)$ is of type signature. In this case, let $tag_{sig}(s, m', vk) := C_{t_1}(w)$.

(iv) $C_{\mathsf{vk}(\beta_j)}(w)$ is of type verification key. In this case, let $tag_{vk}(vk') := C_{\mathsf{vk}(\beta_j)}(w)$.

(v) $m' = C_{t_2}(w)$.

(vi) $vk = vk'$.

(vii) $C_T(w) = 1$.

Let $b' := b_j\triangle = \beta_j\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$. By Definition 2, $T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is of the form $[t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(\beta_j\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle)}$. Since $b' = \beta_j\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$, this implies (i) ⇔ (ii).

By definition of ZK-circuits (Definition 6), we have that (vii) ⇔ (iii) ∧ (iv) ∧ (v) ∧ (vi).

To show (ii) ⇔ (iii) ∧ (iv) ∧ (v) ∧ (vi) and conclude the proof of (i) ⇔ (vii), we need to distinguish two subcases:

Case b-3.1: *"$C_{t_1}(w)$ equals the result of some earlier call* $\mathrm{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')}, \tau')$ *for some $a', sid', R'', x', t', \tau'$".*
Assume that (ii) holds. By Definition 4, condition 6, we have that $x' \in \mathsf{X}.a$ and hence $\tau'(x') = tag_{agent}(b')$ for some $b' \in \mathsf{A}$. Let $b'' := b_j\triangle$. Since $C_{t_1}(w) = \mathrm{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')}, \tau')$, by definition of $\mathrm{construct}^{a',sid'}$, $C_{t_1}(w) = tag_{sig}(s, m', vk)$ where $m' = \mathrm{construct}^{a',sid'}(t', \tau')$ and vk and sk are the verification and signing key of agent b' and s is the signature resulting from signing m' using sk. Hence $C_{t_1}(w)$ is of type signature and (iii) holds. Furthermore $C_{\mathsf{vk}(\beta_j)}(w) = tag_{vk}(vk')$ where vk' is the verification key of agent b''. Hence (iv) holds.

We have

$$[t'\triangle']^{r^{a',R'',sid'}}_{\mathsf{sk}(b')} \overset{(a)}{=} \bar{c}(\text{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')},\tau')) = \bar{c}(C_{t_1}(w))$$
$$\overset{(3)}{=} t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(ii)}{=} [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b_j\triangle)} = [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b'')} \quad (6)$$

Hence $b' = b''$. Since vk and vk' are the verification keys of agent b' and b'', respectively, we have $vk = vk'$ and thus (vi) holds.

Moreover, we have

$$\bar{c}(m') \;=\; \bar{c}(\text{construct}^{a',sid'}(t',\tau')) \;\overset{(a)}{=}\; t'\triangle' \;\overset{(6)}{=}\; t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \;\overset{(3)}{=}\; \bar{c}(C_{t_2}(w))$$

Since $\bar{c}$ is injective, this implies $m' = C_{t_2}(w)$ and thus (v).

Thus (ii) $\Rightarrow$ (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi) holds in Case b-3.1.

Assume now that we have (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi). Since $C_{t_1}(w) = \text{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')},\tau')$, by definition of $\text{construct}^{a',sid'}$, $C_{t_1}(w) = tag_{sig}(\hat{s},\hat{m'},\hat{vk})$ where $\hat{m'} = \text{construct}^{a',sid'}(t',\tau')$ and $\hat{vk}$ and $\hat{sk}$ are the verification and signing key of the agent $\hat{b'}$ with $\tau'(x') = tag_{agent}(\hat{b'})$ and $\hat{s}$ is the signature resulting from signing $\hat{m'}$ using $\hat{sk}$. By (iii) we have $s = \hat{s}$, $C_{t_2}(w) \overset{(v)}{=} m' = \hat{m'}$ and $vk' \overset{(vi)}{=} vk = \hat{vk}$. Furthermore, by (iv) and definition of ZK-circuits, we have that vk' is the verification key of agent $b' := b_j\triangle$. Since vk' is also the verification key of agent $\hat{b'}$, we have $b' = \hat{b'}$. We then have

$$t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \;\overset{(3)}{=}\; \bar{c}(C_{t_2}(w)) \;=\; \bar{c}(\hat{m'}) \;=\; \bar{c}(\text{construct}^{a',sid'}(t',\tau')) \;\overset{(a)}{=}\; t'\;\triangle' \;. \quad (7)$$

Then we have with $R' := r^{a',R'',sid'}$ that

$$t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_1}(w))$$
$$= \bar{c}(\text{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')},\tau')) \overset{(a)}{=} [t'\triangle']^{R'}_{\mathsf{sk}(\hat{b'})}$$
$$\overset{(7)}{=} [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(\hat{b'})} = [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b_j\triangle)}$$

Hence (ii) holds. This shows (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi) $\Rightarrow$ (ii) in Case b-3.1.

Hence we have that (ii) $\Leftrightarrow$ (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi) in Case b-3.1.

Case b-3.2: *"$C_{t_1}(w)$ does not equal the result of some earlier call* $\text{construct}^{a',sid'}([t']^{R''}_{\mathsf{sk}(x')},\tau')$*".*

Assume that (ii) holds. Then

$$\bar{c}(C_{t_1}(w)) \overset{(3)}{=} t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(ii)}{=} [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b_j\triangle)}.$$

By definition of $\bar{c}$ this only happens if $C_{t_1}(w)$ is of type signature and $C_{t_1}(w) = tag_{sig}(s,m',vk)$ for some s, m', and vk where $\bar{c}(m') = t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ and vk is the verification key of agent $b' := b_j\triangle$. Hence (iii) holds. Furthermore, by definition of ZK-circuits, $C_{\mathsf{vk}(\beta_j)}(w) = tag_{vk}(vk')$ where vk' is the verification key of $b' = \beta_j\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$. Hence (iv) holds. Furthermore, since vk' and vk are verification keys of the same agent, $vk = vk'$ and thus (vi) holds. Moreover we have $\bar{c}(m') = t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_2}(w))$. Since $\bar{c}$ is injective, this implies $m' = C_{t_2}(w)$, so (v) holds. This shows (ii) $\Rightarrow$ (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi) in Case b-3.2.

Assume now that we have (iii) $\wedge$ (iv) $\wedge$ (v) $\wedge$ (vi). By definition of ZK-circuits and from (iv) we have that $vk' \overset{(vi)}{=} vk$ is the verification key of agent $b_j\triangle$. By (iii) and the

definition of $\bar{c}$ (the case of not honestly generated signatures) we have that $\bar{c}(C_{t_1}(w)) = [\bar{c}(m')]^{R'}_{\mathsf{sk}(b_j\triangle)}$ for $R' := R_{adv}^{C_{t_1}(w)}$. Hence

$$t_1\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle \overset{(3)}{=} \bar{c}(C_{t_1}(w)) = [\bar{c}(m')]^{R'}_{\mathsf{sk}(b_j\triangle)} \overset{(v)}{=} [\bar{c}(C_{t_2}(w))]^{R'}_{\mathsf{sk}(b_j\triangle)} \overset{(3)}{=} [t_2\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle]^{R'}_{\mathsf{sk}(b_j\triangle)}.$$

Thus (ii) holds. This shows (iii) ∧ (iv) ∧ (v) ∧ (vi) ⇒ (ii) in Case b-3.2.
Hence we have that (ii) ⇔ (iii) ∧ (iv) ∧ (v) ∧ (vi) in Case b-3.2.

Thus we have shown

$$\begin{aligned}&\text{(i)} \Leftrightarrow \text{(ii)}\\ &\quad\Leftrightarrow \text{(iii)} \wedge \text{(iv)} \wedge \text{(v)} \wedge \text{(vi)}\\ &\quad\Leftrightarrow \text{(vii)}\end{aligned}$$

so $T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $C_T(w) = 1$.

This concludes the proof of Case b-3.

This covers all cases for T, hence for every $i \in [q]$, we have shown that $T_i\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true iff $C_{T_i}(w) = 1$.

Let $g_i := \mathsf{True}$ if $T_i\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\triangle$ is true and $g_i := \mathsf{False}$ otherwise. Then $g_i = C_{T_i}(w)$. (We associate True with 1 and False with 0.)

We proceed to show that $\mathsf{ZK}_F^{r^{a,R,sid}}(r^{a,R_1,sid},\ldots;a_1\triangle,\ldots;b_1\triangle,\ldots) =: Z$ is true iff $C_F(w) = 1$. By Definition 4, condition 6, if a subterm $\mathsf{ek}(\beta_i)$ or $\mathsf{vk}(\beta_i)$ occurs in Z, then $b_i \in \mathsf{X}.a$. In this case, $b_i\triangle = \bar{c}(\tau(x)) \in \mathsf{A}$, hence Z is valid in the sense of Definition 2. Thus Z is true iff $B(\underline{g}) = \mathsf{True}$. Thus Z is true iff $C_F(w) = B(C_{T_1}(w),\ldots,C_{T_q}(w)) = B(\underline{g}) = 1$. This shows (b) (assuming the induction hypothesis).

Showing (c)**.** If $\bar{c}(m')$ outputs a term that does not contain true proofs, then there is a recursive invocation of $\bar{c}(m)$ for some m such that $\bar{c}(m)$ is of the form $\mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$ and $\bar{c}(m)$ is not a true proof. Hence we can assume that $\bar{c}(m) = \mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$, and to show (c), it is sufficient to show that $\mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$ is a true proof.

$\bar{c}(m)$ outputs a term of the form $t_z := \mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$ only if m is of type zero-knowledge proof. If m was output by a call to $\mathrm{construct}^{a',sid'}(r',\tau)$ for some a', sid', r', τ', by the induction hypothesis (a), we have that $r'\triangle'$ contains true proofs and that $\bar{c}(m) = \bar{c}(\mathrm{construct}^{a',sid'}(r',\tau)) = r'\triangle'$. Thus $\bar{c}(m)$ contains true proofs and hence $\bar{c}(m) = t_z$ is a true proof. Hence we assume in the following that m is not the output of a call to construct. In this case, by definition of $\bar{c}$, if $\bar{c}(m) = \mathsf{ZK}_F^R(\underline{r};\underline{a};\underline{b})$ the follow holds:

We have that $m = tag_{zk}(z,F,s,n,\underline{l},\underline{\tilde{b}})$ such that $C_F := C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is defined (this is implied by the type zero-knowledge). There is a bitstring w with $C_F(w) = 1$ and $w = \tilde{r}_1\|\ldots\|\tilde{r}_l\|\tilde{a}_1\|\ldots\|\tilde{a}_n$ and $a_i = \bar{c}(\tilde{a}_i)$ and $b_i = \bar{c}(\tilde{b}_i)$ for all i. If T is a subterm of F of the form $\{\cdot\}^{\rho_i}_{\cdot}$, then $\bar{c}(C_T(w)) = \{\cdot\}^{r_i}_{\cdot}$ where $C_T := C^{s,n,\underline{l}}_{T,\underline{\tilde{b}}}$.

Since C_F is defined, it follows by definition of ZK-circuits that if there is a subterm $\mathsf{ek}(t)$ or $\mathsf{vk}(t)$ in F, then $t = \beta_i$ for some i, and $\tilde{b}_i = tag_{agent}(b')$ for some $b' \in \mathsf{A}$. Then $b_i = \bar{c}(\tilde{b}_i) = b' \in \mathsf{A}$.

We first show that for all ZK-terms T that are subterms of F, we have that

$$\bar{c}(C_T(w)) = T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}. \tag{8}$$

We prove this by induction on the structure of T.

Case c-1: *"$T = \alpha_i$".*
Then $\bar{c}(C_T(w)) = \bar{c}(\tilde{a}_i) = a_i = T\{\tfrac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-2: *"$T = \beta_i$".*
Then $\bar{c}(C_T(w)) = \bar{c}(\tilde{b}_i) = b_i = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-3: *"$T = \langle T_1, T_2\rangle$".*
Then $C_T(w) = tag_{pair}(C_{T_1}(w), C_{T_2}(w))$ and hence using the induction hypothesis (8), $\bar{c}(C_T(w)) = \langle \bar{c}(C_{T_1}(w)), \bar{c}(C_{T_2}(w))\rangle \overset{(8)}{=} \langle T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}, T_2\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\rangle = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-4: *"$T = \mathsf{ek}(\beta_i)$".*
In this case $C_T(w) = ek$ where ek is the encryption key of agent $b' = b_i$. Then $\bar{c}(ek) = \mathsf{ek}(b_i) = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-5: *"$T = \mathsf{vk}(\beta_i)$".*
In this case $C_T(w) = vk$ where vk is the encryption key of agent $b' = b_i$. Then $\bar{c}(vk) = \mathsf{vk}(b_i) = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-6: *"$T = \{T_1\}^{\rho_i}_{\mathsf{ek}(\beta_j)}$".*
In this case, $C_T(w)$ is of type ciphertext. We distinguish two cases.

Case c-6.1: *"$C_T(w)$ is the result of an earlier call* $\mathrm{construct}^{a',sid'}(r',\tau')$ *with* $r' = \{t'\}^{R'}_{\mathsf{ek}(x')}$*".*
Note that then $x' \in \mathsf{X}.a$ and hence $b^* := \bar{c}(\tau'(x')) \in \mathsf{A}$. By the induction hypothesis (a), $\bar{c}(C_T(w)) = \bar{c}(\mathrm{construct}^{a',sid'}(r',\tau')) \overset{(a)}{=} r'\triangle' = \{t'\triangle'\}^{R'\triangle'}_{\mathsf{ek}(b^*)}$. Hence $r_i = R'\triangle'$ (see the properties of r_i mentioned before (8)). By definition of $\mathrm{construct}^{a',sid'}$, $\mathrm{construct}^{a',sid'}(r',\tau') = tag_{enc}(\mathsf{ek}_{b^*}, m')$ where m' is the result of encrypting $\mathrm{construct}^{a',sid'}(t',\tau')$ using the encryption key ek_{b^*} of agent b^*. And by definition of ZK-circuits, $C_T(w) = tag_{enc}(\mathsf{ek}_{b_j}, m'')$ where m'' is the result of encrypting $C_{T_1}(w)$ using the encryption key ek_{b_j} of agent b_j. Since $C_T(w) = \mathrm{construct}^{a',sid'}(r',\tau')$ this implies that $C_{T_1}(w) = \mathrm{construct}^{a',sid'}(t',\tau')$ and $\mathsf{ek}_{b^*} = \mathsf{ek}_{b_j}$, and thus $b^* = b_j$. By induction hypothesis (a), we also have $\bar{c}(C_{T_1}(w)) = \bar{c}(\mathrm{construct}^{a',sid'}(t',\tau')) \overset{(a)}{=} t'\triangle'$. And by induction hypothesis (8), $\bar{c}(C_{T_1}(w)) \overset{(8)}{=} T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ and thus $t'\triangle' = T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$. Thus $\bar{c}(C_T(w)) = \{t'\triangle'\}^{R'\triangle'}_{\mathsf{ek}(b^*)} = \{T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\}^{r_i}_{\mathsf{ek}(b_j)} = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

Case c-6.2: *"$C_T(w)$ is not the result of an earlier call* $\mathrm{construct}^{a',sid'}(r',\tau')$ *with* $r' = \{t'\}^{R'}_{\mathsf{ek}(x')}$*".*
By definition of ZK-circuits, $C_T(w)$ is the result of encrypting $C_{T_1}(w)$ using using the encryption key ek_{b_j} of agent b_j. Hence $\bar{c}(C_T(w)) = \{\bar{c}(C_{T_1}(w))\}^{R'}_{\mathsf{ek}(b_j)}$ for some R'. Hence $r_i = R'$ (see the properties of r_i mentioned before (8)). By induction hypothesis (8), $\bar{c}(C_{T_1}(w)) = T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$. Thus $\bar{c}(C_T(w)) = \{T_1\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}\}^{r_i}_{\mathsf{ek}(b_j)} = T\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$.

This shows (8).

By definition of ZK-formulas, F can be written as $B(T_1,\ldots,T_q)$ where B is a Boolean formula and $T_1,\ldots,T_q$ are ZK-atoms. We claim that for every $i \in [q]$, we have that $T_i\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ is true (in the sense of Definition 2) iff $C_{T_i}(w) = 1$. The proof of this fact is identical to that of (4) (i.e., to Cases b-1–b-3), except that all occurrences of $\triangle$ are removed (or equivalently, $\triangle$ is defined as the empty substitution), and (8) is used instead of (3). (Note that $\triangle'$ is used unchanged, i.e., $\triangle' = \gamma^{a',sid'}(\bar{c}\circ\tau')$.)

Let $g_i :=$ True iff $T_i\{\frac{\underline{r},\underline{a},\underline{b}}{\underline{\rho},\underline{\alpha},\underline{\beta}}\}$ is true and $g_i :=$ False otherwise. Then $g_i = C_{T_i}(w)$. (We associate True with 1 and False with 0.) By assumption (see the discussion at the beginning of the proof of (c)), $C_F(w) = 1$. By definition of ZK-circuits, $C_F(w) = B(C_{T_1}(w),\ldots,C_{T_q}(w)) = B(\underline{g})$, so we have that $B(\underline{g}) =$ True. Since we assumed that $b_i \in \mathsf{A}$ for all occurrences of $\mathsf{ek}(\beta_i)$ and $\mathsf{vk}(\beta_i)$ in F, we have that $t_z = \mathsf{ZK}^R_F(\underline{r};\underline{a};\underline{b})$ is valid in the sense of Definition 2. Hence by Definition 2, t_z is a true proof. This concludes the proof of (c).

Summarizing, we have shown (a), (b) and (c). Since (a) implies Lemma 4 and (c) implies Lemma 5, this concludes the proof. □

Proof of Lemma 6. We prove the lemma by induction over the structure of the pattern l. We distinguish the following cases for the pattern l:

Case 1: *"$l = x \in \mathsf{X}$ and $\tau(x) = \bot$".*
By definition of $\bar{c}$, we have that: $\bar{c}(m) \in \mathsf{A}$ iff m is of type agent. $\bar{c}(m) \in \mathsf{Nonce}$ iff m is of type nonce. $\bar{c}(m) = \langle \cdot, \cdot \rangle$ iff m is of type pair. $\bar{c}(m) = \{\cdot\}^{\cdot}_{\cdot}$ iff m is of type encryption. $\bar{c}(m) = [\cdot]^{\cdot}_{\cdot}$ iff m is of type signature. $\bar{c}(m) = \mathsf{ZK}^{\cdot}_{\cdot}(\dots)$ iff m is of type zero-knowledge (since we assume that no ZK-break occurs).

We have that $\bar{c}(m)$ matches x iff $\bar{c}(m)$ is of the type corresponding to the sort of x. (E.g., if $x \in \mathsf{X}.c$, $\bar{c}(m)$ matches x iff $\bar{c}(m) = \{\cdot\}^{\cdot}_{\cdot}$.) Since there are only variables of sorts agent, nonce, pair, ciphertext, signature, and ZK-proof, it follows that $\bar{c}(m)$ matches x iff m is of the type corresponding to the sort of x. Furthermore, by definition of $\mathrm{parse}^{a,sid}$, $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff the type of m matches the sort of $l = x$. Thus $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = x$.

In that case, the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau) = x$ is $\theta = (x \mapsto \bar{c}(m))$. And the return value of $\mathrm{parse}^{a,sid}(m, l, \tau)$ is $(x \mapsto m) \cup \tau$. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = (x \mapsto \bar{c}(m)) \cup (\bar{c} \circ \tau) = \theta \cup (\bar{c} \circ \tau)$.

Case 2: *"$l = x \in \mathsf{X}$ and $\tau(x) \neq \bot$".*
$\mathrm{parse}^{a,sid}(m, l, \tau)$ is defined iff $\tau(x) = m$ (note that in this case, the type of $\tau(x)$ already matches the sort of x). And $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \bar{c}(\tau(x))$ iff $\bar{c}(m) = \bar{c}(\tau(x))$. Since $\bar{c}$ is injective, this happens iff $m = \tau(x)$. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\mathrm{parse}^{a,sid}(m, l, \tau)$ is defined.

In this case, the matcher is $\theta = \varnothing$, and the return value of $\mathrm{parse}^{a,sid}(m, l, \tau)$ is τ. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = \bar{c} \circ \tau = \theta \cup (\bar{c} \circ \tau)$.

Case 3: *"$l = \mathsf{ek}(x)$ with $x \in \mathsf{X}.a$ and $\tau(x) = \bot$".*
$\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff m is the encryption key of some agent b. $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \mathsf{ek}(x)$ iff $\bar{c}(m) = \mathsf{ek}(b')$ for some $b' \in \mathsf{A}$. This happens iff m is the encryption key of some agent b'. Hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$.

In this case, the matcher is $\theta = (x \mapsto b')$, and the return value of $\mathrm{parse}^{a,sid}(m, l, \tau)$ is $\tau[x := tag_{agent}(b')] = (x \mapsto tag_{agent}(b')) \cup \tau$. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = (x \mapsto b') \cup \bar{c} \circ \tau = \theta \cup (\bar{c} \circ \tau)$.

Case 4: *"$l = \mathsf{ek}(x)$ with $x \in \mathsf{X}.a$ and $\tau(x) \neq \bot$".*
$\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff m is the encryption key of some agent b' and $\tau(x) = tag_{agent}(b')$. $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \mathsf{ek}(\bar{c}(\tau(x)))$ iff $\bar{c}(m) = \mathsf{ek}(b')$ for some $b' \in \mathsf{A}$ with $\bar{c}(\tau(x)) = b'$. This holds iff m is the encryption key of some agent b' and $\tau(x) = tag_{agent}(b')$. Hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$.

In this case, the matcher is $\theta = \varnothing$, and the return value of $\mathrm{parse}^{a,sid}(m, l, \tau)$ is $\tau[x := tag_{agent}(b')] = \tau$. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = \bar{c} \circ \tau = \theta \cup (\bar{c} \circ \tau)$.

Case 5: *"$l = \mathsf{vk}(x)$ with $x \in \mathsf{X}.a$ and $\tau(x) = \bot$".*
Analogous to the $l = \mathsf{ek}(x)$ case.

Case 6: *"$l = \mathsf{vk}(x)$ with $x \in \mathsf{X}.a$ and $\tau(x) \neq \bot$".*
Analogous to the $l = \mathsf{ek}(x)$ case.

Case 7: *"$l = \langle l_1, l_2 \rangle$".*
$\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $m = tag_{pair}(m_1, m_2)$ for some m_1, m_2 and $\tau_i := \mathrm{parse}^{a,sid}(m_i, l_i, \tau)$ succeeds for $i = 1, 2$ and τ_1 and τ_2 are compatible. By induction hypothesis, $\tau_i := \mathrm{parse}^{a,sid}(m_i, l_i, \tau)$ succeeds iff $\bar{c}(m_i)$ matches $l_i\gamma^{a,sid}(\bar{c} \circ \tau)$ (let θ_i denote the matcher). Furthermore, in this case $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$. Since $l_i\gamma^{a,sid}(\bar{c} \circ \tau)$ does

not contain variables in the domain of $\bar{c} \circ \tau$, $\bar{c} \circ \tau$ and θ_i have disjoint domains. Hence θ_1 and θ_2 are compatible iff $(\bar{c} \circ \tau) \cup \theta_1$ and $(\bar{c} \circ \tau) \cup \theta_2$ are compatible. Since $\bar{c}$ is injective and $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$, this holds iff τ_1 and τ_2 are compatible. Thus $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $m = tag_{pair}(m_1, m_2)$ for some m_1, m_2 and $\bar{c}(m_i)$ matches $l_i\gamma^{a,sid}(\bar{c} \circ \tau)$ and the resulting matchers θ_1 and θ_2 are compatible. By definition of $\bar{c}$, this happens iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \langle l_1\gamma^{a,sid}(\bar{c} \circ \tau), l_2\gamma^{a,sid}(\bar{c} \circ \tau)\rangle$.

In this case, $\mathrm{parse}^{a,sid}(m, l, \tau) = \tau_1 \cup \tau_2$ and the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ is $\theta := \theta_1 \cup \theta_2$. By induction hypothesis, $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$ for $i = 1, 2$, hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = \bar{c} \circ \tau_1 \cup \bar{c} \circ \tau_2 = (\bar{c} \circ \tau) \cup \theta_1 \cup (\bar{c} \circ \tau) \cup \theta_2 = (\bar{c} \circ \tau) \cup \theta$.

Case 8: *"$l \in \{\mathsf{Pat}\}^{\mathsf{Rand}_{ag}}_{\mathsf{ek}(\mathsf{X}.a)}$ or $l \in [\mathsf{Pat}]^{\mathsf{Rand}_{ag}}_{\mathsf{sk}(X.a)}$ or $l \in \mathsf{ZK}^{\mathsf{Rand}_{ag}}_{\mathsf{Formula}}(\mathsf{Rand}^*_{ag}; (\mathsf{Pat}|\mathsf{dk}(\mathsf{X}.a))^*; \mathsf{Pat}^*)$".*

In this case, $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bot \neq \mathrm{construct}^{a,sid}(l, \tau) = m$. Note also that l does not contain free variables (due to Definition 4, conditions 3, 4, and 7), hence l does not contain variables outside the domain of τ and thus $l\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain variables.

First consider the case that $l\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain true proofs. Then by Lemma 4, $\mathrm{construct}^{a,sid}(l, \tau) = \bot$ and hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ fails. Furthermore, by Lemma 5, $\bar{c}(m)$ contains true proofs, hence $\bar{c}(m) \neq l\gamma^{a,sid}(\bar{c} \circ \tau)$. Since $l\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain variables, this implies that $\bar{c}(m)$ does not match $l\gamma^{a,sid}(\bar{c} \circ \tau)$. Thus in the case that $l\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain true proofs, we have that $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ (namely never).

Now consider the case that $l\gamma^{a,sid}(\bar{c} \circ \tau)$ contains true proofs. In this case, by Lemma 4, $\mathrm{construct}^{a,sid}(l, \tau)$ succeeds and $\bar{c}(\mathrm{construct}^{a,sid}(l, \tau)) = l\gamma^{a,sid}(\bar{c} \circ \tau)$. We have that $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\mathrm{construct}^{a,sid}(l, \tau) = m$. Since $\bar{c}$ is injective, this holds iff $\bar{c}(\mathrm{construct}^{a,sid}(l, \tau)) = \bar{c}(m)$. Since $l\gamma^{a,sid}(\bar{c} \circ \tau)$ does not contain variables and $\bar{c}(\mathrm{construct}^{a,sid}(l, \tau)) = l\gamma^{a,sid}(\bar{c} \circ \tau)$, this holds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$.

Hence in both cases, $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$.

In this case, $\mathrm{parse}^{a,sid}(m, l, \tau) = \tau$ and the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ is $\theta = \varnothing$. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = \bar{c} \circ \tau = (\bar{c} \circ \tau) \cup \theta$.

Case 9: *"$l = \{l'\}^{-}_{\mathsf{ek}(x)}$".*

Let $j \in [k]$ such that the session sid executes the j-th role. Then by Definition 4, condition 2, $x = A_j$. Hence $\tau(x) = \tau(A_j) = tag_{agent}(a)$ and $\bar{c}(\tau(x)) = a$. Hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff $m = tag_{enc}(c, ek_a)$ where ek_a is the encryption key of agent a and decrypting c with decryption key dk_a succeeds and results in a plaintext m' such that $\mathrm{parse}^{a,sid}(m', l', \tau)$ succeeds. In this case, $\mathrm{parse}^{a,sid}(m, l, \tau) = \mathrm{parse}^{a,sid}(m', l', \tau)$. We distinguish two cases for m:

Case 9.1: *"m was the result of some call* $\mathrm{construct}^{a',sid'}(\{t\}^{R}_{\mathsf{ek}(y)}, \tau')$".

In this case, by Lemma 4, $\bar{c}(m) = \{t\gamma^{a',sid'}(\bar{c} \circ \tau')\}^{R'}_{\mathsf{ek}(b')}$ with $R' := r^{a',R,sid'}$ and $b' := \bar{c}(\tau(y))$. So $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \{l'\gamma^{a,sid}(\bar{c} \circ \tau)\}^{-}_{\mathsf{ek}(a)}$ iff $t\gamma^{a',sid'}(\bar{c} \circ \tau')$ matches $l'\gamma^{a,sid}(\bar{c} \circ \tau)$ and $b' = a$.

By definition of $\mathrm{construct}^{a',sid'}$, we have that $b' = a$ iff $m = tag_{enc}(c, ek_a)$ for some c where ek_a is the encryption key of agent a. And in that case, using the decryption key sk_a of agent a, c decrypts to $m' := \mathrm{construct}^{a',sid'}(t, \tau')$. By Lemma 4, $\bar{c}(m') = t\gamma^{a',sid'}(\bar{c} \circ \tau')$. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $m = tag_{enc}(c, ek_a)$ for some c such that decrypting c using the decryption key sk_a results in a plaintext m' such that $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c} \circ \tau)$. By induction hypothesis, $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\mathrm{parse}^{a,sid}(m', l', \tau)$ succeeds. Together with the observations at the beginning of Case 9, it follows that $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds.

In this case, $\mathrm{parse}^{a,sid}(m, l, \tau) = \mathrm{parse}^{a,sid}(m', l', \tau)$. And the matcher θ of $\bar{c}(m) = \{t\gamma^{a',sid'}(\bar{c} \circ \tau')\}^{R'}_{\mathsf{ek}(a')}$ and $l\gamma^{a,sid}(\bar{c} \circ \tau) = \{l'\gamma^{a,sid}(\bar{c} \circ \tau)\}^{-}_{\mathsf{ek}(a)}$ and the matcher θ' of $\bar{c}(m') = t\gamma^{a',sid'}(\bar{c} \circ \tau')$ and $l'\gamma^{a,sid}(\bar{c} \circ \tau)$ are equal. By induction hypothesis, $\bar{c} \circ \mathrm{parse}^{a,sid}(m', l', \tau) = (\bar{c} \circ \tau) \cup \theta'$. Hence $\bar{c} \circ \mathrm{parse}^{a,sid}(m, l, \tau) = (\bar{c} \circ \tau) \cup \theta$.

Case 9.2: *"m was not the result of some call* $\mathrm{construct}^{a',sid'}(\{t\}^R_{\mathsf{ek}(y)}, \tau')$*".*

Then $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau) = \{l'\gamma^{a,sid}(\bar{c}\circ\tau)\}^{\text{-}}_{\mathsf{ek}(a)}$ iff $m = tag_{enc}(c, \mathsf{ek}_a)$ where ek_a is the encryption key of agent a and decrypting c with decryption key dk_a succeeds and results in a plaintext m' such that $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$. By induction hypothesis, $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\mathrm{parse}^{a,sid}(m', l', \tau)$ succeeds. Together with the observations at the beginning of Case 9, we have that $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds.

In this case, the unifier θ of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c}\circ\tau)$ is the same as the unifier θ' of $\bar{c}(m')$ and $l'\gamma^{a,sid}(\bar{c}\circ\tau)$. Furthermore, $\mathrm{parse}^{a,sid}(m, l, \tau) = \mathrm{parse}^{a,sid}(m', l', \tau)$ and by induction hypothesis, $\bar{c}\circ\mathrm{parse}^{a,sid}(m', l', \tau) = (\bar{c}\circ\tau)\cup\theta'$. Hence $\bar{c}\circ\mathrm{parse}^{a,sid}(m, l, \tau) = (\bar{c}\circ\tau)\cup\theta$.

Case 10: *"$l = [l']^{\text{-}}_{\mathsf{sk}(x)}$ with $x \in \mathsf{X}.a$".*

By definition of $\mathrm{parse}^{a,sid}$, we have that $\mathrm{parse}^{a,sid}(m, l, \tau)$ succeeds iff s is of type signature and $m = tag_{sig}(s, m', vk)$, and $\tau_1 := \mathrm{parse}^{a,sid}(tag_{vk}(vk), \mathsf{vk}(x), \tau)$ and $\tau_2 := \mathrm{parse}^{a,sid}(m', l', \tau)$ succeed and τ_1 and τ_2 are compatible. We distinguish several cases:

Case 10.1: *"m is not of type signature".*

Then $\mathrm{parse}^{a,sid}(m, l, \tau)$ fails. And $\bar{c}(m)$ is not of the form $[\cdot]^{\cdot}$, hence $\bar{c}(m)$ does not match $l\gamma^{a,sid}(\bar{c}\circ\tau) = [l_1\gamma^{a,sid}(\bar{c}\circ\tau)]^{\text{-}}_{\mathsf{sk}(x\gamma^{a,sid}(\bar{c}\circ\tau))}$. Thus $\mathrm{parse}^{a,sid}(m, l, \tau)$ never succeeds and $\bar{c}(m)$ never matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$, so the lemma is vacuously true in this case.

We can hence assume in the next cases that m is of type signature and that $m = tag_{sig}(s, m', vk)$ for some bitstrings s, m', vk.

Case 10.2: *"vk is not the verification key of some agent".*

Then $\mathrm{parse}^{a,sid}(tag_{vk}(vk), \mathsf{vk}(x), \tau)$ fails and hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ fails. Moreover, in this case m was not generated by some call $\mathrm{construct}^{a',sid'}([t]^R_{\mathsf{sk}(x)}, \tau)$, and m is not of the form $tag_{sig}(s, m', vk)$ with vk being the verification key of some agent. Hence, by definition of $\bar{c}$, $\bar{c}(m)$ will not be of the form $[\cdot]^{\cdot}_{\mathsf{vk}(b')}$ for some $b' \in \mathsf{A}$. Since $x\gamma^{a,sid}(\bar{c}\circ\tau) \in \mathsf{X}.a \cup \mathsf{A}$, this implies that $\bar{c}(m)$ does not match $l\gamma^{a,sid}(\bar{c}\circ\tau) = [l_1\gamma^{a,sid}(\bar{c}\circ\tau)]^{\text{-}}_{\mathsf{sk}(x\gamma^{a,sid}(\bar{c}\circ\tau))}$. Thus $\mathrm{parse}^{a,sid}(m, l, \tau)$ never succeeds and $\bar{c}(m)$ never matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$, so the lemma is vacuously true in this case.

We can hence assume in the next cases that vk is the verification key of some agent b.

Case 10.3: *"$\tau(x) \neq \bot$ and $\tau(x) \neq tag_{agent}(b)$".*

Hence $\tau(x) = tag_{agent}(b')$ for some $b' \neq b$. Then $\mathrm{parse}^{a,sid}(tag_{vk}(vk), \mathsf{vk}(x), \tau)$ fails and hence $\mathrm{parse}^{a,sid}(m, l, \tau)$ fails (since vk is the verification key of agent b). And $\bar{c}(m)$ is of the form $[\cdot]^{\cdot}_{\mathsf{sk}(b)}$. Thus $\bar{c}(m)$ does not match $l\gamma^{a,sid}(\bar{c}\circ\tau) = [l_1\gamma^{a,sid}(\bar{c}\circ\tau)]^{\text{-}}_{\mathsf{sk}(b')}$. Thus the lemma is vacuously true in this case.

We can hence assume in the next cases that $\tau(x) = \bot$ or $\tau(x) = tag_{agent}(b)$ and hence $\bar{x} := x\gamma^{a,sid}(\bar{c}\circ\tau) \in \{x, b\}$. Moreover, since vk is the verification key of agent b, both for $\tau(x) = \bot$ and $\tau(x) = tag_{agent}(b)$, $\tau_1 := \mathrm{parse}^{a,sid}(tag_{vk}(vk), \mathsf{vk}(x), \tau)$ succeeds and $\tau_1 = \tau[x := tag_{agent}(b)]$.

Case 10.4: *"m was the result of some call* $\mathrm{construct}^{a',sid'}([t]^R_{\mathsf{sk}(y)}, \tau')$*".*

By Lemma 4, $\bar{c}(m) = \bar{c}(\mathrm{construct}^{a',sid'}([t]^R_{\mathsf{sk}(y)}, \tau')) = [t\gamma^{a',sid'}(\bar{c}\circ\tau')]^{R'}_{\mathsf{sk}(b')}$ with $R' := r^{a',R,sid'}$ and $b' := \bar{c}(\tau'(y))$. Hence $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau) = [l'\gamma^{a,sid}(\bar{c}\circ\tau)]^{\text{-}}_{\mathsf{sk}(\bar{x})}$ iff $t\gamma^{a',sid'}(\bar{c}\circ\tau')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ (with some matcher θ') and θ' is compatible with θ'' where $\theta'' := (x \mapsto b)$ if $\tau(x) = \bot$ and $\theta'' := \varnothing$ otherwise. Moreover, θ' and θ'' are compatible iff $\theta' \cup (\bar{c}\circ\tau)$ and $(x \mapsto b)$ are compatible.

By definition of $\mathrm{construct}^{a',sid'}$, and since $\mathrm{construct}^{a',sid'}([t]^R_{\mathsf{sk}(y)}, \tau') = m = tag_{sig}(s, m', vk)$, we have that $m' = \mathrm{construct}^{a',sid'}(t, \tau')$. Hence, by Lemma 4, $\bar{c}(m') = t\gamma^{a',sid'}(\bar{c}\circ\tau')$.

Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ (with some matcher θ') and $\theta'\cup(\bar{c}\circ\tau)$ and $(x\mapsto b)$ are compatible.

By induction hypothesis, $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ (with some matcher θ') iff $\mathrm{parse}^{a,sid}(m',l',\tau)$ succeeds and in this case, $\bar{c}\circ(\mathrm{parse}^{a,sid}(m',l',\tau))=(\bar{c}\circ\tau)\cup\theta'$.

Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\tau_2:=\mathrm{parse}^{a,sid}(m',l',\tau)$ succeeds and $\bar{c}\circ\tau_2$ and $(x\mapsto b)$ are compatible.

Since we assume that $m=tag_{sig}(s,m',vk)$ is of type signature, that $\tau_1:=\mathrm{parse}^{a,sid}(tag_{vk}(vk),\mathsf{vk}(x),\tau)$ succeeds, and that $\tau_1=\tau[x:=tag_{agent}(b)]$, we have that $\mathrm{parse}^{a,sid}(m,l,\tau)$ succeeds iff $\tau_2:=\mathrm{parse}^{a,sid}(m',l',\tau)$ succeeds and τ_2 is compatible with the partial function $(x\mapsto tag_{agent}(b))$. (Note that $\tau_2=\tau$ wherever τ is defined.) Since $\bar{c}$ is injective, τ_2 and $(x\mapsto tag_{agent}(b))$ are compatible iff $\bar{c}\circ\tau_2$ and $(x\mapsto b)=\bar{c}\circ(x\mapsto tag_{agent}(b))$ are. Thus $\mathrm{parse}^{a,sid}(m,l,\tau)$ succeeds iff $\tau_2:=\mathrm{parse}^{a,sid}(m',l',\tau)$ succeeds and $\bar{c}\circ\tau_2$ and $(x\mapsto b)$ are compatible, which in turn holds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$.

In this case, $\mathrm{parse}^{a,sid}(m,l,\tau)=\tau_1\cup\tau_2=\tau_2\cup\tau[x:=tag_{agent}(b)]=\tau_2\cup(x\mapsto tag_{agent}(b))$. Moreover, the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c}\circ\tau)=[l'\gamma^{a,sid}(\bar{c}\circ\tau)]^{\_}_{\mathsf{sk}(\bar{x})}$ is $\theta:=\theta'\cup\theta''$. By induction hypothesis we have that $\bar{c}\circ\tau_2=(\bar{c}\circ\tau)\cup\theta'$. Hence $\bar{c}\circ\mathrm{parse}^{a,sid}(m,l,r)=(\bar{c}\circ\tau_2)\cup(x\mapsto b)=(\bar{c}\circ\tau)\cup\theta'\cup\theta''=(\bar{c}\circ\tau)\cup\theta$.

Case 10.5: *"m was not the result of some call* $\mathrm{construct}^{a',sid'}([t]^R_{\mathsf{sk}(y)},\tau)$*".*

Since we assume that $m=tag_{sig}(s,m',vk)$ is of type signature and that vk is the verification key of agent b, we have that $\bar{c}(m)=[\bar{c}(m')]^{R'}_{\mathsf{sk}(b)}$ for some R'. Hence $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)=[l'\gamma^{a,sid}(\bar{c}\circ\tau)]^{\_}_{\mathsf{sk}(\bar{x})}$ iff $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ (with some matcher θ') and θ' and θ'' are compatible where $\theta'':=(x\mapsto b)$ if $\tau(x)=\bot$ and $\theta'':=\varnothing$ otherwise. Moreover, θ' and θ'' are compatible iff $\theta'\cup(\bar{c}\circ\tau)$ and $(x\mapsto b)$ are compatible.

By induction hypothesis, $\bar{c}(m')$ matches $l'\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\tau_2:=\mathrm{parse}^{a,sid}(l',m',\tau)$ succeeds, and in this case we have $\bar{c}\circ\tau_2=(\bar{c}\circ\tau)\cup\theta'$. From the injectivity of $\bar{c}$ we have in this case that $\theta'\cup(\bar{c}\circ\tau)=\bar{c}\circ\tau_2$ and $(x\mapsto b)=\bar{c}\circ(x\mapsto tag_{agent}(b))$ are compatible iff τ_2 and $(x\mapsto tag_{agent}(b))$ are. So $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$ iff $\tau_2:=\mathrm{parse}^{a,sid}(l',m',\tau)$ succeeds and τ_2 and $(x\mapsto tag_{agent}(b))$ are compatible.

Since we assume that $m=tag_{sig}(s,m',vk)$ is of type signature, that $\tau_1:=\mathrm{parse}^{a,sid}(tag_{vk}(vk),\mathsf{vk}(x),\tau)$ succeeds, and that $\tau_1=\tau[x:=tag_{agent}(b)]$, we have that $\mathrm{parse}^{a,sid}(m,l,\tau)$ succeeds iff $\tau_2:=\mathrm{parse}^{a,sid}(m',l',\tau)$ succeeds and τ_2 and $(x\mapsto tag_{agent}(b))$ are compatible. Thus $\mathrm{parse}^{a,sid}(m,l,\tau)$ succeeds iff $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)$.

In this case, $\mathrm{parse}^{a,sid}(m,l,\tau)=\tau_1\cup\tau_2=\tau_2\cup\tau[x:=tag_{agent}(b)]=\tau_2\cup(x\mapsto tag_{agent}(b))$. Moreover, the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c}\circ\tau)=[l'\gamma^{a,sid}(\bar{c}\circ\tau)]^{\_}_{\mathsf{sk}(\bar{x})}$ is $\theta:=\theta'\cup\theta''$. By induction hypothesis we have that $\bar{c}\circ\tau_2=(\bar{c}\circ\tau)\cup\theta'$. Hence $\bar{c}\circ\mathrm{parse}^{a,sid}(m,l,r)=(\bar{c}\circ\tau_2)\cup(x\mapsto b)=(\bar{c}\circ\tau)\cup\theta'\cup\theta''=(\bar{c}\circ\tau)\cup\theta$.

Case 11: *"$l=\mathsf{ZK}^{\_}_F(\_^*;\_^*;t_1,\ldots,t_n)$".*

Then $\mathrm{parse}^{a,sid}(m,l,\tau)$ succeeds iff m is of type zero-knowledge, $m=tag_{zk}(\ldots,F',\ldots,\underline{\tilde{b}})$ with $F'=F$, $\tau_i:=\mathrm{parse}^{a,sid}(\tilde{b}_i,t_i,\tau)$ succeeds for all $i\in[n]$, and all τ_i are compatible. (The condition $n=n'$ from the definition of $\mathrm{parse}^{a,sid}$ is automatically fulfilled if m is of type zero-knowledge because the latter implies that the circuit $C^{s,n,\underline{l}}_{F,\underline{\tilde{b}}}$ is defined.) In this case, $\mathrm{parse}^{a,sid}(m,l,\tau)=\bigcup_i\tau_i$. We distinguish three cases for m:

Case 11.1: *"m was the result of some call* $\mathrm{construct}^{a',sid'}(\mathsf{ZK}^R_{F'}(\underline{R};\underline{a};\underline{b}),\tau')$*".*

Then by Lemma 4, $\bar{c}(m)=\mathsf{ZK}^{R'}_{F'}(\underline{R}';\underline{a}';\underline{b}')$ for some $R',\underline{R}',\underline{a}'$ and $b'_i:=b_i\gamma^{a',sid'}(\bar{c}\circ\tau')$. Hence $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c}\circ\tau)=\mathsf{ZK}^{\_}_F(\_^*;\_^*;\underline{t}\gamma^{a,sid}(\bar{c}\circ\tau))$ iff $F=F'$ and for all $i\in[n]$, b'_i matches $t_i\gamma^{a,sid}(\bar{c}\circ\tau)$ (with some matcher θ_i) and all matchers θ_i are compatible.

Moreover, by definition of construct, we have that $m = tag_{zk}(\ldots, \underline{\tilde{b}})$ with $\tilde{b}_i = \text{construct}^{a',sid'}(b_i, \tau')$. Thus by Lemma 4, $\bar{c}(\tilde{b}_i) = b'_i$. Then by induction hypothesis, b'_i matches $t_i\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\tau_i := \text{parse}^{a,sid}(\tilde{b}_i, t_i, \tau)$ succeeds and in this case, $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$. Moreover, the matchers θ_i are compatible iff all $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$ are compatible (because the domains of τ and θ_i are disjoint). Because of the injectivity of $\bar{c}$, this holds iff all τ_i are compatible. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $F = F'$, $\tau_i := \text{parse}^{a,sid}(\tilde{b}_i, t_i, \tau)$ succeeds for all i, and all τ_i are compatible. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\text{parse}^{a,sid}(m, l, \tau)$ succeeds.

In this case, the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ is $\theta := \bigcup_i \theta_i$. And $\tau := \text{parse}^{a,sid}(m, l, \tau) = \bigcup_i \tau_i$. By induction hypothesis, $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$. Hence $\bar{c} \circ \tau = \bigcup_i \big((\bar{c} \circ \tau) \cup \theta_i\big) = (\bar{c} \circ \tau) \cup \bigcup_i \theta_i = (\bar{c} \circ \tau) \cup \theta$.

Case 11.2: *"m is not of type zero-knowledge".*
Then $\text{parse}^{a,sid}(m, l, \tau)$ fails and $\bar{c}(m)$ is not of the form $\mathsf{ZK}_\cdot(\ldots)$ and hence does not match $l\gamma^{a,sid}(\bar{c} \circ \tau)$. Thus the lemma is vacuously true.

Case 11.3: *"m is of type zero-knowledge but m was not the result of some call* $\text{construct}^{a',sid'}(\mathsf{ZK}^R_{F'}(\underline{R}; \underline{a}; \underline{b}), \tau')$*".*
Then $m = tag_{zk}(\ldots, F', \ldots, \underline{\tilde{b}})$ for some F' and $\underline{\tilde{b}}$. And $\bar{c}(m) = \mathsf{ZK}^{R'}_{F'}(\underline{R}'; \underline{a}'; \underline{b}')$ for some $R', \underline{R}', \underline{a}'$ and $b'_i := \bar{c}(\tilde{b}_i)$. (The case $\bar{c}(m) = G^m$ does not occur because we assume that no ZK-break occurs.)

Hence $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau) = \mathsf{ZK}^-_F(\_^*; \_^*; \underline{t}\gamma^{a,sid}(\bar{c} \circ \tau))$ iff $F = F'$ and for all $i \in [n]$, b'_i matches $t_i\gamma^{a,sid}(\bar{c} \circ \tau)$ (with some matcher θ_i) and all matchers θ_i are compatible.

By induction hypothesis, b'_i matches $t_i\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\tau_i := \text{parse}^{a,sid}(\tilde{b}_i, t_i, \tau)$ succeeds and in this case, $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$. Moreover, the matchers θ_i are compatible iff all $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$ are compatible (because the domains of τ and θ_i are disjoint). Because of the injectivity of $\bar{c}$, this holds iff all τ_i are compatible. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $F = F'$, $\tau_i := \text{parse}^{a,sid}(\tilde{b}_i, t_i, \tau)$ succeeds for all i and all τ_i are compatible. Thus $\bar{c}(m)$ matches $l\gamma^{a,sid}(\bar{c} \circ \tau)$ iff $\text{parse}^{a,sid}(m, l, \tau)$ succeeds.

In this case, the matcher of $\bar{c}(m)$ and $l\gamma^{a,sid}(\bar{c} \circ \tau)$ is $\theta := \bigcup_i \theta_i$. And $\tau := \text{parse}^{a,sid}(m, l, \tau) = \bigcup_i \tau_i$. By induction hypothesis, $\bar{c} \circ \tau_i = (\bar{c} \circ \tau) \cup \theta_i$. Hence $\bar{c} \circ \tau = \bigcup_i \big((\bar{c} \circ \tau) \cup \theta_i\big) = (\bar{c} \circ \tau) \cup \bigcup_i \theta_i = (\bar{c} \circ \tau) \cup \theta$.

Case 12: *"All other values of l".*
According to the syntax of patterns (page 12), the only missing case is $l = \{l'\}^-_{\mathsf{ek}(x)}$ with $x \notin \{A_1, \ldots, A_k\}$. But by Definition 4, condition 2, such a pattern l cannot occur in a role. Hence $\text{parse}^{a,sid}(m, l, \tau)$ will never be invoked.

□

Proof of Lemma 7. We prove the lemma by induction on the length of m. All cases except $\bar{c}(m) = \{t'\}^{r^{a,R,sid}}_{\mathsf{ek}(a)}$, $\bar{c}(m) = [t']^{r^{a,R,sid}}_{\mathsf{sk}(a)}$, $\bar{c}(m) = \mathsf{ZK}^{r^{a,R,sid}}_F(\underline{r}; \underline{a}; \underline{b})$ follow directly from the definition of $\bar{c}$.

In the case $\bar{c}(m) = \{t'\}^{r^{a,R,sid}}_{\mathsf{ek}(a)}$, by definition of $\bar{c}$, we have that $m = \text{construct}^{a,sid}(\{r'\}^R_{\mathsf{ek}(x)}, \tau)$ where $r'\gamma^{a,sid}(\bar{c} \circ \tau) = t'$ and $\tau(x) = tag_{agent}(a)$. By Lemma 4 we have $\bar{c}(\text{construct}^{a,sid}(r', \tau)) = r'\gamma^{a,sid}(\bar{c} \circ \tau)$ and thus by induction hypothesis, $\text{construct}^{a,sid}(r', \tau) = c^*(r'\gamma^{a,sid}(\bar{c} \circ \tau))$. Hence by construction of c^*, we have that $c^*(\{t'\}^{r^{a,R,sid}}_{\mathsf{ek}(a)}) = \text{construct}^{a,sid}(\{r'\}^R_{\mathsf{ek}(x)}, \tau) = m$. Hence $c^*(\bar{c}(m)) = m$.

The cases $\bar{c}(m) = [t']^{r^{a,R,sid}}_{\mathsf{sk}(a)}$, $\bar{c}(m) = \mathsf{ZK}^{r^{a,R,sid}}_F(\underline{r}; \underline{a}; \underline{b})$ are handled analogously. □

References

[ABW06] Martin Abadi, Mathieu Baudet, and Bogdan Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Proc. 9th International Confer-*

ence on Foundations of Software Science and Computation Structures (FOSSACS), volume 3921 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2006.

[AF01] Martí Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 104–115, New York, NY, USA, 2001. ACM Press.

[AF06] Pedro Adão and Cédric Fournet. Cryptographically sound implementations for communicating processes. In *Proc. 32nd International Confererence on Automata, Languages and Programming (ICALP)*, pages 83–94, 2006.

[AJ01] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.

[AR02] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[BBU08] Michael Backes, Matthias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. In *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330 of *Lecture Notes in Computer Science*, pages 353–376, 2008.

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

[BCJ+06] Frederik Butler, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of kerberos 5. *Theoretical Computer Science*, 367(1):57–87, 2006.

[BCK05] Mathieu Baudet, Véronique Cortier, and Steve Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

[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`.

[BGJZ07] Gilles Barthe, Benjamin Gregoire, Romain Janvier, and Santiago Zanella Beguelin. Formal certification of code-based cryptographic proofs. IACR ePrint Archive, August 2007. Online available at `http://eprint.iacr.org/2007/314`.

[BHU09] Michael Backes, Dennis Hofheinz, and Dominique Unruh. A general framework for computational soundness proofs - or - the computational soundness of the applied pi-calculus. IACR ePrint Archive 2009/080, 2009.

[BL06] Michael Backes and Peeter Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proc. 13th ACM Conference on Computer and Communications Security*, 2006.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society Press, 2001.

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 140–154, 2006.

[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.

[BLMW07] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *LNCS*, pages 482–499. Springer, 2007.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy 2008*, pages 158–169, May 2008.

[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.

[BP05] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.

[BP06] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology: CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[BPW03a] 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/2003/015`.

[BPW03b] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003.

[BPW05] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Reactively secure signature schemes. *International Journal of Information Security*, 4(4):242–252, 2005.

[BPW06] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science. Springer, 2006. Preliminary version in IACR Cryptology ePrint Archive 2006/068, Feb. 2006, `http://eprint.iacr.org/`.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[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.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[Cd06] Ricardo Corin and Jerry den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[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.

[CKKW06a] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proc. 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 176–187, 2006.

[CKKW06b] Véronique Cortier, Steve Kremer, Ralf Kï¿½sters, and Bogdan 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.

[CLC08] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proc. ACM Conference on Computer and Communications Security*, pages 109–118, 2008.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 131–140, 1998.

[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.

[DDM+05] Anupam Datta, Ante Derek, John Mitchell, Vitalij Shmatikov, and Matthieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 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.cs.ucla.edu/~rafail/PUBLIC/85.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`.

[GvR08] Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theor. Comput. Sci.*, 394(1-2):112–133, 2008.

[HLM03] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

[IK03] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[JLM05] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 172–185, 2005.

[JLM07] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. *Electronic Notes in Theoretical Computer Science*, 186:121 – 139, 2007. Proceedings of the First Workshop in Information and Computer Security (ICS 2006).

[KKW05] Detlef Kähler, Ralf Küsters, and Thomas Wilke. Deciding properties of contract-signing protocols. In *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 158–169. Springer-Verlag, 2005.

[KM07] Steve Kremer and Laurent Mazaré. Adaptive soundness of static equivalence. In *Proc. 12th European Symposium On Research In Computer Security (ESORICS)*, pages 610–625, 2007.

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

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming (ESOP)*, LNCS, pages 186–200. Springer-Verlag, 2005.

[KRW04] Detlef Kähler, Küsters Ralf, and Thomas Wilke. Deciding properties of contract-signing protocols. Technical Report IFI 0409, CAU Kiel, 2004. Online available at `http://www.ti.informatik.uni-kiel.de/~kuesters/publications_html/KaehlerKuestersWilke-IFI-TR-0409-2004.pdf`, short version in [KKW05].

[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.

[Lau05] Peeter Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 26–35, 2005.

[LMMS98] Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[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.

[MMS98] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

[Moh09] Esfandiar Mohammadi. Computational soundness for symbolic zero-knowledge proofs against active attackes under relaxed assumptions. Master's thesis, Saarland University, Department of Computer Science, 2009.

[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.

[nAG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[Now07] David Nowak. A framework for game-based security proofs. IACR Cryptology ePrint Archive 2007/199, 2007. `http://eprint.iacr.org/`.

[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.

[SBB+06] Christoph Sprenger, Michael Backes, David Basin, Birgit Pfitzmann, and Michael Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.

[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.

Index