Termination-Insensitive Computational Indistinguishability
(and applications to computational soundness)

Dominique Unruh
Saarland University

Abstract

We defined a new notion of computational indistinguishability: termination-insensitive computational indistinguishability (tic-indistinguishability). Tic-indistinguishability models indistinguishability with respect to distinguishers that cannot distinguish between termination and non-termination.

We sketch how the new notion allows to get computational soundness results of symbolic models for equivalence-based security properties (such as anonymity) for processes that contain loops, solving an open problem.

Contents

1 Introduction

We present the notion of termination-insensitive computational indistinguishability (tic-indistinguishability for short). In a nutshell, two interactive machines A and B are tic-indistinguishable (written: $A \approx_{tic} B$) if they cannot be distinguished by a distinguisher that is computationally limited, and that cannot distinguish between non-termination and termination of the machines it interacts with. That is, if for example upon some query A returns 1, and B never returns, A and B will be considered tic-indistinguishable. Furthermore, needing superpolynomial time for answering a request is treated as non-termination. That is, if B returns 0 after an exponential number of steps, we still have $A \approx_{tic} B$.

Before we explain how tic-indistinguishability can be formally defined, we present a motivating example that explains why such a notion can be necessary in the first place. Assume that we have designed a file server A that supports the following queries: Store content c under filename f. Read (part of) the content of file f. Concatenate the contents of two files f_1, f_2 and store the result in file f_3. Encrypt the content of file f_1 and store the result in file f_2. (Encryption takes place with respect to some IND-CCA secure encryption scheme and some publicly known public key; the secret key is not revealed or even used.)

We wish to compare this file server with the file server B which, instead of encrypting the content of c of file f_1, encrypts $0^{|c|}$. Intuitively, we would expect that A and B are computationally indistinguishable. Typically, computational indistinguishability of interactive machines A and B would be defined by requiring that for any polynomial-time machine Z, Z cannot guess whether it interacts with A or with B. Consider the following Z: Z creates a file f_0 with content 1. Then, through a sequence of concatenation operations, Z creates polynomially-many files f_i such that the content of f_i is 1^{2^i}. Finally, Z produces a file f containing the encryption of the content 1^{2^η} of f_η (η being the security parameter), and requests and outputs the first bit of the content of f. Since IND-CCA security only deals with polynomial-time adversaries, it does not contradict IND-CCA security if the first bit of the encryption of a message m of length 2^η equals the first bit of m. Assume that we are using such a scheme. Then Z, interacting with A, will output 1. And interacting with B, Z will output 0. Thus A and B are not computationally indistinguishable.

What options do we have to resolve this issue?

- We could reject all machines that do not run in polynomial time and never even ask the question whether they are computationally indistinguishable. This would, however, exclude even such innocuous looking machines as our file server. And we believe that many practical systems do in fact have specifications that allow to perform operations that are not a priori bounded in their running time.
- We could try to strengthen the definition of IND-CCA security such that it allows to make A and B computationally indistinguishable. It is not clear whether such an approach is feasible in general. Anyway, it seems strange to introduce additional computational assumptions in order to avoid attacks that occur after a superpolynomial amount of time (from the point of view of cryptography, they never occur).
- One could try to define a notion of computational indistinguishability that requires

A to answer in polynomial-time whenever B answers in polynomial-time, and that in this case the answers are indistinguishable. This is, however, not trivial to define, because there is no strict border between polynomial-time and superpolynomial-time (that is, one cannot specify a number t such that t steps would be polynomial-time, and $t+1$ steps would be superpolynomial-time).
- Or, one could use our notion of tic-indistinguishability. We have $A \approx_{tic} B$ because the encryption scheme can only leak information at a point where A and B are already considered non-terminating.

One might complain that tic-indistinguishability is too weak a notion because it considers non-termination and termination to be indistinguishable although in practice, these are clearly distinguishable: We fix a suitable timeout, and once that timeout has elapsed, we can assume that the system we are communicating with does not terminate. One should observe, however, that the same applies when distinguishing machines with polynomial running times of, say, η^2 and η^{100}. These can easily be distinguished by using a timeout, but commonly we treat them as indistinguishable because we choose to abstract away from timing issues. The alternative would be to explicitly model the timing; in this case tic-indistinguishability would not make sense. If we do, however, choose to abstract away from the running time, it seems natural not to make an exception for exponential running time. That is, if we cannot distinguish between η^2 and η^{100}, we should also not distinguish between, say, η^2 and 2^η. On the other hand, after 2^η steps, the actual answer does not matter any more, so that answer should not be used to distinguish. This is exactly what tic-indistinguishability does. Thus, from a more philosophical point of view, natural notions would be some kind of timing-sensitive indistinguishability (if we consider timing), and tic-indistinguishability (if we abstract away from timing), while normal computational indistinguishability seems to model a mixture of two approaches (considering and abstracting away from timing).

On the more practical side, we also explain below (Section 1.4) that tic-indistinguishability allows to state certain computational soundness results[1] that would, with respect to computational indistinguishability, be subject to strong side-conditions.

Our contribution. We present the notion of tic-indistinguishability. We show that the definition of tic-indistinguishability is machine-model independent. We relate tic-indistinguishability to computational and statistical indistinguishability; this allows us to perform subproofs with respect to more familiar indistinguishability notions. We show how tic-indistinguishability can be used to get computational soundness results for equivalence-based security properties (such as anonymity) for processes that may contain loops, solving an open problem.

1.1 How to define tic-indistinguishability

In order to understand the definition of tic-indistinguishability, let us first consider a simpler case: We do not consider computational limitations of the distinguisher, we do not allow

[1]A computational soundness result is a theorem that states that, if a protocol is shown secure in a symbolic model (a.k.a. Dolev-Yao model) using formal methods, then this protocol is also secure in a computational/cryptographic sense.

for a negligible probability of failure, and assume that the machines A and B that we distinguish are non-interactive and either output a bit b or run forever. Let p_b^A and $p_\perp^A$ denote the probability that A outputs b or does not terminate, respectively. Define p_B analogously. We will consider A and B as indistinguishable if we can redistribute the probability of non-termination onto the probabilities for the outputs 0 and 1 in such a way, that the probability distributions of the outputs of A and B become the same. (I.e., we treat the non-termination probability as a kind of "wildcard" that can be reinterpreted as any output.) More precisely, A and B are indistinguishable if there are values $\hat{p}_b^X \geq 0$ for $b \in \{0,1\}$ and $X \in \{A, B\}$ such that $\hat{p}_0^A + \hat{p}_1^A = p_\perp^A$ and $\hat{p}_0^B + \hat{p}_1^B = p_\perp^B$ (this models that the $\hat{p}_b^X$ are a redistribution of the non-termination probability) and such that $p_0^A + \hat{p}_0^A = p_0^B + \hat{p}_0^B$ and $p_1^A + \hat{p}_1^A = p_1^B + \hat{p}_1^B$ (this models that after redistributing probabilities, the output distributions of A and B are equal). One easily sees that such $\hat{p}_b^X$ exist iff $p_0^A + p_1^B \leq 1$ and $p_1^A + p_0^B \leq 1$. Thus we can define A and B as indistinguishable iff for all $a \neq b$ we have $p_0^A + p_1^B \leq 1$ and $p_1^A + p_0^B \leq 1$.

Now, let us consider the case that A and B are arbitrary interactive machines. To cover this case, we introduce a distinguisher Z whose goal it is two interacts with A and B and whose output represents a guess whether it is interacting with A or B. We also add the possibility of a negligible error. This leads to the following definition of indistinguishability: For any machine Z there is a negligible μ such that for all $a \neq b$,

$$\Pr[Z + A \Downarrow a] + \Pr[Z + B \Downarrow b] \leq 1 + \mu.$$

Here $Z + A \Downarrow a$ denotes the event that an interaction between Z and A terminates and Z outputs a.

So far, we have not considered the running time of A, B, or Z, leading to a statistical notion of indistinguishability. To get a computational notion, we need to ensure two things: Z runs in polynomial time, and anything that A and B do after polynomial-time is not counted. In other words, we have to consider a polynomial-time prefix of the interaction between Z and A or B. This leads to the following definition: For any machine Z and all polynomials p, q,

$$\Pr[Z + A \Downarrow_p a] + \Pr[Z + B \Downarrow_q b] \leq 1 + \mu.$$

Here $Z + A \Downarrow_p a$ denotes the event that an interaction between Z and A terminates after at most p steps (counting the running time of both Z and A) and Z outputs a. Notice that if this condition is violated for some p, q, it is also violated for the polynomials $p' := p + q$ and $q' := p' = p + q$. Thus, without loss of generality, we can assume $p = q$. Thus our definition is ready:

Definition 1 (tic-indistinguishability – informal) *We call A and B tic-indistinguishable ($A \approx_{tic} B$) if for all machines Z and all polynomials p, there is a negligible μ such that for all $a \neq b$ we have*

$$\Pr[Z + A \Downarrow_p a] + \Pr[Z + B \Downarrow_p b] \leq 1 + \mu.$$

1.2 Machine model independence

When considering Definition 1, one may fear that the definition is very sensitive to the underlying machine model. Instead of quantifying over all polynomial-time machines (which is a class that is equal for all common machine models), we explicitly consider the running time of the machines A and B and of the distinguisher Z.

Fortunately, it turns out that we have machine-model independence in the following strong sense: Given two machines A and A', we call A and A' *time-equivalent* (written $A \approx_{time} A'$) if A and A' compute the same function, and any computations performed by A are take at most polynomially longer than the same computation performed by A' and vice versa. (We give a precise definition in Section 3.) We call two machine models $\mathfrak{M}, \mathfrak{M}'$ time-equivalent if for any machine in model $\mathfrak{M}$, there is a time-equivalent machine in model $\mathfrak{M}'$ and vice versa. In other words, whatever can be done in machine model $\mathfrak{M}$ can also be done in machine model $\mathfrak{M}'$ with only polynomial overhead. Most natural machine models are time-equivalent. We show the following lemma:

Lemma 2 (Informal) *If $A \approx_{tic} B$ with respect to some machine model $\mathfrak{M}$, then also $A \approx_{tic} B$ with respect to any time-equivalent machine model $\mathfrak{M}$. Also, if $A \approx_{time} A'$ and $A \approx_{tic} B$, then $A' \approx_{tic} B$ (and the same for B).*

This lemma allow us to fix some machine model without loss of generality (say, Turing machines), while still getting results that apply to most natural machine models. Also, when describing machine A and B that we wish to compare, it is not necessary to exactly know the algorithm they use, it is only necessary to describe them up to time-equality. Thus the situation is comparable to the machine model independence that we have with computational indistinguishability.

1.3 Using tic-indistinguishability

In order to derive tic-indistinguishability of two machines, it is often not necessary to invoke the definition of tic-indistinguishability directly. Instead, there are a few useful properties that allow to derive tic-indistinguishability from common indistinguishability notions.

First, we have that if A and B are statistically indistinguishable (no unbounded Z can guess whether it is talking to A or B), then $A \approx_{tic} B$. The converse does not hold: For example, if A terminates but B does not, A and B can be tic-indistinguishable, but not statistically indistinguishable.

Second, if A and B are reactive polynomial-time machines,[2] then $A \approx_{tic} B$ iff A and B are computationally indistinguishable. This gives an indication that tic-indistinguishability indeed exactly captures computational indistinguishability augmented with the concept of not being able to recognize non-termination.

[2]A machine is reactively polynomial-time [8] if it runs in polynomial-time whenever it interacts with a polynomial-time machine. Intuitively, this means that a machine reacts to any incoming message with a polynomial-time computation.

We also have composability: If $A \approx_{tic} B$, then $A + C \approx_{tic} B + C$ for all C. ($A + C$ denotes the machine consisting of A and C interacting with each other.)

For example, if we want to show that $A \approx_{tic} B$ were A and B are the file servers from our initial motivating example, we can proceed as follows: Let C be an implementation of the file server that invokes an external machine E for performing the encryptions. I.e., E behaves like an encryption oracle, and whenever C wishes to encrypt a message m, C sends m to E and E returns the ciphertext. Since outsourcing a computation incurs only a polynomial-time overhead, we have that $A \approx_{time} C + E$. Furthermore, let F be defined like E, except that F encrypts $0^{|m|}$ instead of m. Then $B \approx_{time} C + F$. It is easy to see that E and F are reactive polynomial-time (their complexity is polynomial in the length of the messages they have to encrypt). And if the encryption scheme is IND-CCA (or IND-CPA), E and F are computationally indistinguishable. Thus $E \approx_{tic} F$, and, using composability, $C + E \approx_{tic} C + F$. By Lemma 2, $A \approx_{tic} B$.

One should not, however, that some kinds of reasoning are excluded: $\approx_{tic}$ is not transitive. This can be easily seen: If M_0 is a machine that never terminates, $A \approx_{tic} M_0 \approx_{tic} B$ for all A, B, but not necessarily $A \approx_{tic} B$. Thus intransitivity is a necessary consequence of making non-termination indistinguishable from any other behavior. To allow for step-wise rewriting as often used in cryptographic proofs (game-based proofs), we give the following lemma:

Lemma 3 (Informal) *If A and B are reactive polynomial-time and computationally indistinguishable, and if $C + A \approx_{tic} D + A$ ($C + A$ denotes the composition of C and A), then $C + B \approx_{tic} D + B$.*

This lemma allows to derive a tic-indistinguishability in a simpler setting (where B is replaced by some simpler but computationally indistinguishable A). This lemma is also used centrally in our computational soundness proof (see next section).

1.4 Computational soundness

There are two main approaches on how to model and verify the security of protocols. In the computational approach, protocol machines are modeled as computational entities (e.g., Turing machines) that send and receive bitstrings. Cryptographic operations on these bitstrings are modeled as algorithms on bitstrings; the adversary is allowed to perform any polynomial-time computation. In the symbolic approach, messages are not modeled as bitstrings, but as terms over a suitable algebra. The constructors in this algebra model the various available cryptographic operations (such as encryption). The adversary is limited to perform certain well-defined symbolic operations on the terms in his knowledge (e.g., an adversary can derive a plaintext m if and only if he knows the key k and the ciphertext $enc(k, m)$). Although the cryptographic approach comes much closer to reality, the symbolic approach is very popular because it allows for very powerful machine-assisted verification of the security of protocols. In order to get the best of both worlds, Abadi and Rogaway [1] suggested so-called computational soundness results. Such a result states, that, if a protocol is secure in the symbolic setting, that protocol is secure in the computational setting. Most computational soundness results that consider security against active attackers only consider so-called trace properties, e.g., [3, 7, 6, 4, 2]. That is, they show that if, in the symbolic setting, a certain event does not occur, it does not occur in the computational setting, either. (Examples for such events would be that the adversary guesses a key or fakes an authentication.) Many security properties cannot be modeled using trace properties. Instead, we need equivalence-based properties. For example, anonymity would be modeled by stating that the protocol using identity a is indistinguishable from the protocol using identity b. A computational soundness result for such properties has been given by Cortier and Comon-Lundh [5].

They show that if two processes are indistinguishable in the symbolic setting, then they are computationally indistinguishable in the computational setting. Their result is, however, restricted to a small class of processes. In particular, they disallow processes that contain loops. The reason for this is that, if a process contains a loop, it might run a superpolynomial number of steps. When comparing two processes P and Q, one of them might react immediately to any query, while the other might first enter a loop that takes a superpolynomial number of steps. These processes would be considered equivalent in the symbolic setting (in the symbolic setting there is no notion of polynomial- or superpolynomial-time). Thus, in order to get a computational soundness result that allows to deal with such processes, we need a computational notion of indistinguishability that treats P and Q as indistinguishable. Normal computational soundness cannot be used: When trying to prove the computational indistinguishability of P and Q, we have to invoke computational assumption (such as IND-CCA security) which do not necessarily hold when P or Q runs in superpolynomial-time (same problem as with the file servers from the motivating example). It was an open problem how to define the indistinguishability on the computational side in order to make computational soundness for equivalence-based security properties possible.[3] Tic-indistinguishability solves this problem. We show (for a toy-calculus), that symbolic indistinguishability implies computational indistinguishability. The parts of our proof related to computational soundness are quite separate from those related to tic-indistinguishability; thus our approach can be easily adapted to more complex settings than our toy-calculus.

2 Termination-insensitive computational indistinguishability

In order to formally define the notion of tic-indistinguishability, we first need to fix a machine model. As we will see in Section 3, the precise choice of machine model is irrelevant. However, to derive and formally state this irrelevance, we first need some formalism that

[3] One way out is, of course, to just disallow processes which run, in the computational setting, more than a polynomial number of steps. But this approach has several problems: First, we disallow innocuous seeming processes such as our file servers. Second, checking that a protocol runs in polynomial-time can be difficult. In particular, one cannot determine this by only considering the symbolic description of the process. For example, a process that performs η nested encryptions will, if the encryption scheme doubles the size of the plaintext, need time 2^η. If the encryption scheme only adds an additive offset, the nested encryptions can be computed in polynomial time.

allows to express machines in different machine models. We will do this by first defining the concept of a machine, is a fashion independent of a particular machine model. Such a machine M is specified by a set of interfaces I_M (over which it can interact with other machines), an initial state σ_M, and a probabilistic state transition function δ_M. In order to model non-termination, δ_M can be partial, that is, the probabilities of the different outcomes do not need to add up to 1. The "missing probability" is then interpreted as the non-termination probability. The function δ_M does not only output the new state and the message to be sent, but also outputs how much time passed during the current invocation. This allows to encode different machine models with different running times in our formalism.

Definition 4 (Machines) *A* machine M *consists of a non-empty set of interfaces* I_M*, an initial state* σ_M*, and a partial probabilistic function* $\delta_M : \mathbb{N} \times \{0,1\}^* \times I_M \times \{0,1\}^* \to \mathbb{N} \times \{0,1\}^* \times I_M \times \{0,1\}^*$*. (Intuitively, an invocation* $\delta_M(\eta, \sigma, \mathit{ifc}, m)$ *means that the security parameter is* η*, the current state* σ*, and the message* m *arrives on interface* ifc*. If* δ_M *returns* $(t, \sigma', \mathit{ifc}', m')$*, time* t *has passed, the new state is* σ'*, and the message* m' *is being sent on interface* ifc'*.) We call a machine* M terminating *if* δ_M *is total.*

Given this notion of a machine, we can easily express different machine models in our setting. We simply define the set $\mathfrak{M}$ of machines that exist with respect to the machine model under consideration.

Definition 5 (Machine model) *A* machine model *is a set of machines.*

For example, we can define the machine model $\mathfrak{M}_{\text{Turing}}$ consisting of probabilistic Turing machines. This model contains all machines M where δ_M is computable, and the running time that δ_M returns corresponds to the running time a Turing machine would need for that computation.

Another example for a machine model is the model $\mathfrak{M}_0$. $\mathfrak{M}_0$ is the set of all machines with zero running time. More precisely, $M \in \mathfrak{M}_0$ iff for all $\eta, \sigma, \mathit{ifc}, m$, we have $\Pr[t > 0 : (t, \sigma', \mathit{ifc}', m') \leftarrow \delta_M(\eta, \sigma, \mathit{ifc}, m)] = 0$. Notice that $\mathfrak{M}_0$ is not a realistic machine model (it imposes no computational restrictions at all).

Next, we need the concept of a network of machines. Since, at least in principle, a network of machines is nothing else than another machine simulating these machines, we model networks by just composing different machines. When composing machines A and B, we simply connect the interfaces that A and B share, and let all other interfaces be external interfaces (that can be connected with other machines when composing further).

Definition 6 (Composition of machines) *Given two machines* A *and* B*, we define the machine* $A + B$ *as follows:* $I_{A+B} := I_A \triangle I_B$ *(symmetric set difference) and* $\sigma_{A+B} := (\sigma_A, \sigma_B)$*. And* δ_{A+B} *is the probabilistic function described by the following algorithm:*

1. *Let* $(\eta, \sigma, \mathit{ifc}, m)$ *be the inputs to* δ_{A+B}*.*
2. *Let* $M := A$ *if* $\mathit{ifc} \in I_A$ *and* $M := B$ *if* $\mathit{ifc} \in I_B$*.*[4] *Let* $t_{sum} := 0$*. Parse* $\sigma =: (\sigma_A, \sigma_B)$*.*

[4]We always have either $\mathit{ifc} \in I_A$ or $\mathit{ifc} \in I_B$, but never both, since $\mathit{ifc} \in I_{A+B} = I_A \triangle I_B$.

3. *Invoke* $(t, \sigma_M, ifc, m) \leftarrow \delta_M(\eta, \sigma_M, ifc, m)$.
4. *Let* $t_{sum} := t_{sum} + t$.
5. *If* $ifc \in I_A \cap I_B$, *let* $M := \bar{M}$ *where* $\bar{A} := B$ *and* $\bar{B} := A$ *and goto step 3.*
6. *Otherwise, return* $(t_{max}, (\sigma_A, \sigma_B), ifc, m)$.

Once we have composed a network of different machines into a single machine M with a single interface *ifc* (which is to be used to transport the initial input to and the final output of the network), the execution of the whole network is modeled by sending a single message to M on *ifc* and waiting for the answer. Machines with a single interface we call closed; the following definition also introduces notation to speak about the outcome of the execution of a network.

Definition 7 (Closed machines) *A machine M is* closed *if $|I_M| = 1$. Given a closed machine M and values $z, x \in \{0,1\}^*$ and $t_{max}, \eta \in \mathbb{N}$, we write* $\Pr[M(\eta, z) \Downarrow_{t_{max}} x]$ *for* $\Pr[t \le t_{max} \wedge m = x : (t, \sigma', ifc', m) \leftarrow \delta_M(\eta, \sigma_M, ifc, z)]$ *where ifc is the element of I_M. Similarly, we write* $\Pr[M(\eta, z) \Downarrow_{t_{max}}]$ *for* $\Pr[t \le t_{max} : (t, \sigma', ifc', m) \leftarrow \delta_M(\eta, \sigma_M, ifc, z)]$.

(Intuitively, z is an auxiliary input given to the submachine of M that has the interface *ifc*.)

We can finally define tic-indistinguishability. We parametrize the notion over the machine model $\mathfrak{M}$ the distinguisher Z is chosen from. Once one has decided on a particular machine model, $\mathfrak{M}$ will be fixed.

Definition 8 (Tic-indistinguishability) *We write $A \sim B$ iff A and B are closed and $I_A = I_B$ and for all polynomials p, there is a negligible function μ such that for all $z, a, b \in \{0,1\}^*$ with $a \neq b$ and all $\eta \in \mathbb{N}$, we have that*

$$\Pr[A(\eta, z) \Downarrow_{p(\eta)} a] + \Pr[B(\eta, z) \Downarrow_{p(\eta)} b] \le 1 + \mu(\eta).$$

*We call Z I-*closing *if $I_Z = I \cup \{ifc\}$ for some $ifc \notin I$.*

We call two machines A and B termination-insensitively computationally indistinguishable *with respect to a machine model $\mathfrak{M}$ (short* tic-indistinguishable*, written $A \approx_{tic}^{\mathfrak{M}} B$) if $I_A = I_B$ and for all I_A-closing $Z \in \mathfrak{M}$, we have $Z + A \sim Z + B$.*

3 Machine-model independence

We proceed to show that tic-indistinguishability is independent of the machine model (assuming a reasonable machine model). In order to formalize this, we first need to define what it means for two machines to behave the same and to have polynomially-related running times (time-equivalence). In order to make the notion independent of the encoding of the internal state of the machines, we define time-equivalence by requiring that for any sequence of inputs (modeled by a machine Z that provides the inputs but does not, itself, use up running time), if one of the machines achieves to perform a certain task (modeled by Z outputting 1) in polynomial time p, then the other machine can achieve the same task, with at least the same probability, in another (possibly larger) polynomial time q.

Definition 9 (Time-equivalence) *We say A is* time-majorized *by B (written $A \lesssim_{time} B$) if $I_A = I_B$ and for any machine I_A-closing $Z \in \mathfrak{M}_0$ and any polynomial p, there exists a polynomial q and a negligible function μ such that for all $\eta \in \mathbb{N}$ and $z \in \{0,1\}^*$,* $\Pr[(A+Z)(\eta,z) \Downarrow_{q(\eta)} 1] \geq \Pr[(B+Z)(\eta,z) \Downarrow_{p(\eta)} 1] - \mu(\eta)$.

We say A and B are time-equivalent *(written $A \approx_{time} B$) if $A \lesssim_{time} B \lesssim_{time} A$.*

Lemma 10 *If $A \lesssim_{time} A'$ and $A \sim B$, then $A' \sim B$.*

Lemma 11 *If $A \lesssim_{time} B$ then $A + C \lesssim_{time} B + C$.*

Lemma 12 *If $A \lesssim_{time} A'$ and $B \lesssim_{time} B'$ and $A \approx_{tic}^{\mathfrak{M}} B$, then $A' \approx_{tic}^{\mathfrak{M}} B'$.*

Corollary 13 *If $A \approx_{time} A'$, and $B \approx_{time} B'$, then $A \approx_{tic}^{\mathfrak{M}} B$ iff $A' \approx_{tic}^{\mathfrak{M}} B'$.*

Definition 14 *Let machine models $\mathfrak{M}$ and $\mathfrak{M}'$ be given. We say that $\mathfrak{M}$ is* time-majorized *by $\mathfrak{M}'$ ($\mathfrak{M} \lesssim_{time} \mathfrak{M}'$) iff for any $B \in \mathfrak{M}'$ there is an $A \in \mathfrak{M}$ such that $A \lesssim_{time} B$.*

We call $\mathfrak{M}$ and $\mathfrak{M}'$ time-equivalent *(written $\mathfrak{M} \approx_{time} \mathfrak{M}'$) if $\mathfrak{M} \lesssim_{time} \mathfrak{M}' \lesssim_{time} \mathfrak{M}$.*

Lemma 15 *If $\mathfrak{M} \lesssim_{time} \mathfrak{M}'$ and $A \approx_{tic}^{\mathfrak{M}} B$, then $A \approx_{tic}^{\mathfrak{M}'} B$.*

Corollary 16 *If $\mathfrak{M} \approx_{time} \mathfrak{M}'$ then $\approx_{tic}^{\mathfrak{M}} = \approx_{tic}^{\mathfrak{M}'}$.*

Due to Corollary 16, the exact choice of machine model is irrelevant since most machine models are time-equivalent to the Turing machine model $\mathfrak{M}_{\text{Turing}}$. Thus we define $\approx_{tic} := \approx_{tic}^{\mathfrak{M}_{\text{Turing}}}$ and concentrate on $\approx_{tic}$ in the following.

4 Properties of tic-indistinguishability

Definition 17 (Closed machine models) *We call a machine model $\mathfrak{M}$* closed *if for any $A, B \in \mathfrak{M}$, there is a $C \in \mathfrak{M}$ such that $C \lesssim_{time} A + B$.*

Examples of closed machines models are $\mathfrak{M}_{\text{Turing}}$ and $\mathfrak{M}_0$, as well as any machine model time-equivalent to $\mathfrak{M}_{\text{Turing}}$.

Lemma 18 (Composition) *Assume that $\mathfrak{M}$ is a closed machine model. If $A \approx_{tic}^{\mathfrak{M}} B$ and $C \in \mathfrak{M}$, then $A + C \approx_{tic}^{\mathfrak{M}} B + C$.*

Definition 19 (Reactive polynomial-time, following [8]) *We call a machine M reactively polynomial-time if $M \in \mathfrak{M}_{\text{Turing}}$ and for all polynomial-time I_M-closing Z,[5] there is a negligible function μ such that for all $\eta \in \mathbb{N}$ and $z \in \{0,1\}^*$ we have* $\Pr[(M+Z)(\eta,z) \Downarrow_{p(\eta)}]$ $\geq 1 - \mu(\eta)$.

[5] Polynomial-time means that Z only runs $p(\eta)$ steps altogether, for a fixed polynomial p, no matter what M does. The formal definition is given on page 15.

Definition 20 (Computational indistinguishability) *We call two machines A and B with $I_A = I_B$* computationally indistinguishable *(short: $A \approx_{comp} B$) iff for any polynomial-time I_A-closing Z, there exists a negligible function μ such that for all $\eta \in \mathbb{N}$ and $z \in \{0,1\}^*$ we have that $\bigl|\Pr[(Z+A)(\eta,z) \Downarrow_\infty 1] - \Pr[(Z+B)(\eta,z) \Downarrow_\infty 1]\bigr| \le \mu(\eta)$.*

Lemma 21 *Fix machines A and B with $I_A = I_B$. If $A \approx_{comp} B$ then $A \approx_{tic} B$. If A and B are reactively polynomial-time, we have $A \approx_{comp} B$ iff $A \approx_{tic} B$.*

Lemma 22 *Fix machines $A, B, C, D \in \mathfrak{M}_{\text{Turing}}$ with $I_A = I_B$ and $I_C = I_D$. Assume that A is reactively polynomial-time, that $A \approx_{comp} B$, and that $C + A \approx_{tic} D + A$. Then $C + B \approx_{tic} D + B$.*

Definition 23 (Statistical indistinguishability) *We call two machines A and B with $I_A = I_B$* statistically indistinguishable *(short: $A \approx_{stat} B$) iff for any I_A-closing Z (not restricted to any particular machine model), there exists a negligible function μ such that for all $\eta \in \mathbb{N}$ and $z \in \{0,1\}^*$ we have that $\bigl|\Pr[(Z+A)(\eta,z) \Downarrow_\infty 1] - \Pr[(Z+B)(\eta,z) \Downarrow_\infty 1]\bigr| \le \mu(\eta)$.*

Lemma 24 *Fix machines A and B with $I_A = I_B$. If $A \approx_{stat} B$ then $A \approx_{tic}^{\mathfrak{M}} B$ for all machine models $\mathfrak{M}$.*

5 Application: Computational soundness

In the following, we sketch how to get a computational soundness result for equivalence-based security properties (such as anonimity) by using tic-indistinguishability. In our exposition, we will focus on the proof steps related to tic-indistinguishability while omitting those that relate to established notions such as computational indistinguishability and statistical indistinguishability and that are thus more or less standard.

To state our result, we introduce a toy-calculus. For our calculus, we assume countably infinite sets of variables x, y, nonces N, keys K, channels c, randomness symbols R, and abstract lengths $L \in \mathbf{L}$. We assume that the nonces are partitioned into two infinite sets, the protocol nonces and the adversary nonces. We assume a designated channel net, all other channels are called private channels. Terms $M \in \mathit{terms}$ and processes P, Q are of the following grammars:

$$\begin{aligned}
M &::= x \mid N \mid (M, M') \mid \mathsf{enc}(K, M, R) \mid \mathsf{garb}^L(N)\\
P, Q &::= 0 \mid (P|Q) \mid \bar{c}\langle M\rangle.P \mid c(x).P \mid !c(x).P \mid \nu x.P\\
&\quad \mid \mathsf{if}\ M = M'\ \mathsf{then}\ P\ \mathsf{else}\ Q\\
&\quad \mid \mathsf{let}\ (x,y) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q\\
&\quad \mid \mathsf{let}\ x = \mathsf{Dec}_K(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q\\
&\quad \mid \mathsf{let}\ x = \mathsf{Enc}_K(M)\ \mathsf{in}\ P\\
&\quad \mid \mathsf{let}\ x = (M, M')\ \mathsf{in}\ P
\end{aligned}$$

A term $\mathsf{enc}(K, M, R)$ denotes a public-key encryption of message M with the public key pk_K and randomness R. $\mathsf{garb}^L(N)$ represents an invalid message of abstract length L. $P|Q$ denotes processes P and Q running in parallel. $c(x).P$ denotes a process that wait for a message on channel c, assigns the message to the variable x and proceeds as P. $!c(x).P$ does the same, but forks a new copy of P for each message received. $\bar{c}\langle M\rangle.P$ sends M on c and proceeds as P. The combination of $!c(x).P$ and $\bar{c}\langle M\rangle.P$ allows to construct processes that loop. $\mathsf{let}\ x = \mathsf{Enc}_K(M)\ \mathsf{in}\ P$ assigns $\mathsf{enc}(K, M, R)$ to x (with fresh R) and proceeds as P. $\mathsf{let}\ x = \mathsf{Dec}_K(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ decrypts M (if $M = \mathsf{enc}(K, M', R)$) and proceeds as P (or as Q if decryption fails). $\nu x.P$ chooses a fresh protocol nonce N, assigns it to x, and proceeds as P.

We define a notion of equivalence between two processes, called *labeled bisimilarity* ($\cong_{bisi}$). Intuitively, we have $P \cong_{bisi} Q$ if in any (symbolic) execution of P or Q, one cannot distinguish between P and Q given only the terms sent by P and Q. We call a process P *deterministic*, if at no point in its symbolic execution, P has two possible successor states (which could happen if, e.g., $P = P_1|P_2$ where both P_1 and P_2 send a message). We call P *closed* if it has no free variables and *honest* if every term in P is a variable.[6] Finally, we define the computational execution of a process P. It is modeled as a machine M_P with $I_{M_P} = \mathsf{net}$ that simulates P, except that it uses bitstrings instead of terms, and, whenever it encounters a let, performs a corresponding computational operation (e.g., encrypting using an IND-CCA secure public-key encryption scheme). Similarly, when encountering $\nu x.P$, a random value is assigned to x. Messages over the public channel net are sent over the interface net.

Theorem 25 (Computational soundness) *Fix closed, honest, and deterministic processes P and Q. Then, if $P \cong_{bisi} Q$ then $M_P \approx_{tic} M_Q$.*

We sketch how to prove Theorem 25. We first define some auxiliary machines. Note that M_P can be split into two machines: A machine M_P^* with $I_{M_P^*} = \{\mathsf{cmd}\}$ that simulates P, and a machine M_{cmd} with $I_{M_{cmd}} = \{\mathsf{cmd}, \mathsf{net}\}$ that performs the actual computations on bitstrings. Over the channel cmd, M_P^* instructs M_{cmd} which operations to perform and which bitstrings to send over net. A salient point is that M_P^* never handles any bitstrings. All bitstrings are stored in M_{cmd} and referenced by M_P^* via handles. Since $M_P^* + M_{cmd}$ is the same as M_P except that some operations are outsourced to M_{cmd}, any computation performed by M_P^* takes at most polynomially longer in $M_P^* + M_{cmd}$ and vice versa. Thus we have the following lemma:

Lemma 26 $M_P \approx_{time} M_P^* + M_{cmd}$ *and* $M_Q \approx_{time} M_Q^* + M_{cmd}$.

Then we define a machine M_{sym} with $I_{M_{sym}} = \{\mathsf{cmd}, \mathsf{sim}\}$. From the point of view of M_P^*, M_{sym} behaves like M_{cmd}. Internally, however M_{sym} stores symbolic terms instead of bitstrings. And instead of sending/receiving bitstrings over net, M_{sym} provides access

[6]One can transform any process into an honest process by picking nonces using νx, and constructing terms using let.

to these terms over the channel sim. When M_{cmd} would send a bitstring over net, M_{sym} instead allocates a handle for that term and sends it over sim. Then it provides symbolic operations that allow to a machine connected over sim to perform exactly the operations and tests on the terms that are considered in the definition of labeled bisimilarity. (For example, there are queries for splitting a pair or performing an encryption, but not for decrypting with respect to an unknown key.) Then we construct a simulator Sim with $I_{\mathit{Sim}} = \{\mathsf{sim}, \mathsf{net}\}$. The task of this simulator is to simulate the bitstrings that M_{cmd} would send on net by only getting access to the messages sent by M_{sym} on sim. For example, when Sim is informed by M_{sym} that a message has been sent, Sim inquires whether it is an encryption, and if so, Sim produces a ciphertext containing an all-zero string of the right length (since Sim cannot find out what is contained in a symbolic encryption, he is forced to encrypt fake data). We then prove:

Lemma 27 *$M_{cmd} \approx_{comp} M_{sym} + \mathit{Sim}$.*

Moreover, if we only consider $M_{sym} + \mathit{Sim}$ (without M_P^*), we have removed the part of the system that potentially runs in superpolynomial-time (due to a looping process P). We have:

Lemma 28 *$M_{sym} + \mathit{Sim}$ is reactively polynomial-time.*

Finally, since $M_P^* + M_{sym}$ is a faithful simulation of the symbolic execution of P, and the analogously for $M_Q^* + M_{sym}$, and since the machine connected over sim can only perform operations and tests considered in the definition of $\cong_{bisi}$, we get:

Lemma 29 *$M_P^* + M_{sym} \approx_{stat} M_Q^* + M_{sym}$.*

We can plug these four lemmas together to get a proof of Theorem 25:

Proof of Theorem 25. By Lemma 29, we have $M_P^* + M_{sym} \approx_{stat} M_Q^* + M_{sym}$. By Lemma 24, we get $M_P^* + M_{sym} \approx_{tic} M_Q^* + M_{sym}$. By Lemma 18, we get $M_P^* + M_{sym} + \mathit{Sim} \approx_{tic} M_Q^* + M_{sym} + \mathit{Sim}$. By Lemma 28, $M_{sym} + \mathit{Sim}$ is reactively polynomial-time, and by Lemma 27, $M_{sym} + \mathit{Sim} \approx_{comp} M_{cmd}$. Thus by Lemma 22, $M_P^* + M_{cmd} \approx_{tic} M_Q^* + M_{cmd}$. By Lemma 26, $M_P \approx_{time} M_P^* + M_{cmd}$ and $M_Q \approx_{time} M_Q^* + M_{cmd}$. By Corollary 13, $M_P \approx_{tic} M_Q$ follows. □

We stress that the technically difficult parts are the proofs of Lemmas 27 and 29 which are unrelated to the notion of tic-indistinguishability. Thus we expect that computational soundness results for more complex calculi can be handled in exactly the same way, without any additional trouble due to the notion of tic-indistinguishability.

A Postponed proofs for Section 3 (machine-model independence)

Proof of Lemma 10. Since $A \sim B$ and $A \lesssim_{time} A'$, $I_{A'} = I_A = I_B = \{\mathit{ifc}\}$ for some ifc. For contradiction, assume $A' \not\sim B$. Then there is a polynomial p and sequences a_η, b_η, z_η

with $a_\eta \neq b_\eta$ such that $\nu - 1$ is not upper-bounded by a negligible function where

$$\nu(\eta) := \Pr[A'(\eta, z_\eta) \Downarrow_{p(\eta)} a_\eta] + \Pr[B(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta].$$

Let $Z \in \mathfrak{M}_0$ be a machine with interfaces $I_Z = \{\mathit{ifc}, \mathit{ifc}'\}$ for some $\mathit{ifc}' \neq \mathit{ifc}$ and with the following behavior: When receiving x on ifc', send x on ifc. When receiving a_η on ifc, send 1 on ifc', and when receiving $y \neq a_\eta$ on ifc, send 0 on ifc'.

Since $A \lesssim_{\mathit{time}} A'$, there is a polynomial q and a negligible function μ such that for all η, z, we have

$$\Pr[(A + Z)(\eta, z) \Downarrow_{q(\eta)} 1] \geq \Pr[(A' + Z)(\eta, z) \Downarrow_{p(\eta)} 1] - \mu(\eta). \tag{1}$$

We then have

$$\begin{aligned}
&\Pr[A(\eta, z_\eta) \Downarrow_{q(\eta)+p(\eta)} a_\eta] + \Pr[B(\eta, z_\eta) \Downarrow_{q(\eta)+p(\eta)} b_\eta] \\
&\geq \Pr[A(\eta, z_\eta) \Downarrow_{q(\eta)} a_\eta] + \Pr[B(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta] \\
&\overset{(*)}{=} \Pr[(A + Z)(\eta, z_\eta) \Downarrow_{q(\eta)} 1] + \Pr[B(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta] \\
&\overset{(1)}{\geq} \Pr[(A' + Z)(\eta, z_\eta) \Downarrow_{p(\eta)} 1] + \Pr[B(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta] \\
&\overset{(*)}{=} \Pr[A'(\eta, z_\eta) \Downarrow_{p(\eta)} a_\eta] + \Pr[B(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta] \\
&= \nu(\eta).
\end{aligned}$$

For $(*)$, note that Z, being in $\mathfrak{M}_0$, does not increase the running time.

Since $q + p$ is a polynomial, and ν is not upper-bounded by a negligible function, it follows that $A \not\sim B$. This contradicts $A \sim B$, so our assumption that $A' \not\sim B$ holds was false. $\square$

Proof of Lemma 11. Assume $A \lesssim_{\mathit{time}} B$. Fix an $I_A \triangle I_C$-closing machine $Z \in \mathfrak{M}_0$ and a polynomial p. To show $A + C \lesssim_{\mathit{time}} B + C$, we have to construct a polynomial q and a negligible function μ such that

$$\forall \eta, z.\ \Pr[(A + C + Z)(\eta, z) \Downarrow_{q(\eta)} 1] \geq \Pr[(B + C + Z)(\eta, z) \Downarrow_{p(\eta)} 1] - \mu(\eta). \tag{2}$$

Let ifc be the interface such that $I_Z = (I_A \triangle I_C) \mathbin{\dot\cup} \{\mathit{ifc}\}$. We construct an I_A-closing machine $Z^* \in \mathfrak{M}_0$ that does the following: It simulates $C + Z$. When Z outputs x on the interface ifc, Z^* outputs y on ifc, where y is defined as follows: If $x = 1$ and the total time spent by the simulated C is less-equal $p(\eta)$, $y := 1$. Otherwise $y := 0$. (Note: Being in $\mathfrak{M}_0$, Z^* spends zero time performing the simulation. It does, however, keep track of the running time of C for computing y.)

Since $A \lesssim_{\mathit{time}} B$, there are a polynomial q^* and a negligible function μ such that

$$\forall \eta, z.\ \Pr[(A + Z^*)(\eta, z) \Downarrow_{q^*(\eta)} 1] \geq \Pr[(B + Z^*)(\eta, z) \Downarrow_{p(\eta)} 1] - \mu(\eta). \tag{3}$$

Note that $A + Z^*$ outputs 1 in time $q^*(\eta)$ iff the simulated Z outputs 1, A runs at most $q^*(\eta)$ steps, and the simulated C runs at most $p(\eta)$ steps. Note further that $A + C + Z$

outputs 1 in time $p(\eta) + q^*(\eta)$ if (but not "iff") Z outputs 1, A runs at most $q^*(\eta)$ steps, and C runs at most $p(\eta)$ steps. Thus

$$\forall \eta, z.\ \Pr[(A + Z^*)(\eta, z) \Downarrow_{q^*(\eta)} 1] \leq \Pr[(A + C + Z)(\eta, z) \Downarrow_{p(\eta)+q^*(\eta)} 1]. \tag{4}$$

Note that if $B + C + Z$ outputs 1 in time $p(\eta)$, then Z outputs 1, B runs at most $p(\eta)$ steps, and C runs at most $p(\eta)$ steps. Note further that $B + Z^*$ outputs 1 in time $p(\eta)$ iff the simulated Z outputs 1, B runs at most $p(\eta)$ steps, and the simulated C runs at most $p(\eta)$ steps. Thus

$$\forall \eta, z.\ \Pr[(B + C + Z)(\eta, z) \Downarrow_{p(\eta)} 1] \leq \Pr[(B + Z^*)(\eta, z) \Downarrow_{p(\eta)} 1]. \tag{5}$$

Let $q := p + q^*$. Then (2) follows from (4), (3), and (5). □

Proof of Lemma 12. Fix an I_A-closing $Z \in \mathfrak{M}$. By Lemma 11, we have $Z + A \lesssim_{time} Z + A'$ and $Z + B \lesssim_{time} Z + B'$. Since $A \approx_{tic}^{\mathfrak{M}} B$, we have $Z + A \sim Z + B$. By Lemma 10, we have $Z + A' \sim Z + B'$. Since this holds for every $Z \in \mathfrak{M}$, we have $A' \approx_{tic}^{\mathfrak{M}} B'$. □

Proof of Corollary 13. Immediate from Lemma 12 □

Proof of Lemma 15. Fix an I_A-closing $Z' \in \mathfrak{M}'$. Since $\mathfrak{M} \lesssim_{time} \mathfrak{M}'$, there is a $Z \in \mathfrak{M}$ with $Z \lesssim_{time} Z'$. By Lemma 11, $Z + A \lesssim_{time} Z' + A$ and $Z + B \lesssim_{time} Z' + B$. Since $A \approx_{tic}^{\mathfrak{M}} B$ and $Z \in \mathfrak{M}$, we have $Z + A \sim Z + B$. By Lemma 10, it follows that $Z' + A \sim Z' + B$. Since this holds for every $Z' \in \mathfrak{M}'$, we have $A \approx_{tic}^{\mathfrak{M}'} B$. □

Proof of Corollary 16. Immediate from Lemma 15. □

B Postponed proofs and definitions for Section 4 (properties of tic-indistinguishability)

Proof of Lemma 18. Fix some $Z \in \mathfrak{M}$. To show $C + A \approx_{tic}^{\mathfrak{M}} C + B$, we have to show that $A + C + Z \sim B + C + Z$. Since $C, Z \in \mathfrak{M}$ and $\mathfrak{M}$ is closed, there is a machine $Z' \in \mathfrak{M}$ such that $Z' \lesssim_{time} C + Z$. Since $A \approx_{tic}^{\mathfrak{M}} B$, we have $A + Z' \sim B + Z'$. By Lemma 10, $A + C + Z \sim B + C + Z$. □

Definition 30 (Polynomial-time) *We call a machine M* polynomial-time with respect to *$ifc \in I_M$ if $M \in \mathfrak{M}_{\text{Turing}}$ and there is a polynomial p such that for all terminating $Z \in \mathfrak{M}_0$ with $I_Z = I_M \setminus \{ifc\}$, we have that for all $\eta \in \mathbb{N}$ and $z \in \{0,1\}^*$, $\Pr[(M+Z)(\eta, z) \Downarrow_{p(\eta)}] = 1$.*

We omit specifying *ifc* if it is clear from the context.

Proof of Lemma 21. We first show that if $A \approx_{comp} B$, then $A \approx_{tic} B$.

Fix an I_A-closing $Z \in \mathfrak{M}_{\mathrm{Turing}}$ and a polynomial p. To show $A \approx_{tic} B$, we need to show that there is a negligible function μ such that for all z, a, b with $a \neq b$ and all $\eta \in \mathbb{N}$, we have

$$\Pr[(Z+A)(\eta, z) \Downarrow_{p(\eta)} a] + \Pr[(Z+B)(\eta, z) \Downarrow_{p(\eta)} b] \leq 1 + \mu(\eta). \tag{6}$$

Let *ifc* be the element of $I_Z \setminus I_A$. Let Z_p be a machine that does the following: When activated with message (z, b) on interface *ifc*,[7] it activates a simulated Z with message z on interface *ifc*. Messages sent/received by Z on interfaces other than *ifc* are forwarded. When Z runs more than a total number of $p(\eta)$ steps, Z_p aborts and outputs 0 on *ifc*. When Z outputs m on *ifc*, Z checks whether $m = b$, and, if so, sends 1 on *ifc*, otherwise it sends 0 on *ifc*.

Notice that, since $Z \in \mathfrak{M}_{\mathrm{Turing}}$, Z_p can be implemented by a polynomial-time Turing machine. Thus Z_p can be chosen to be polynomial-time.

By construction of Z_p, for $a \neq b$ we have that

$$\Pr[(Z+A)(\eta, z) \Downarrow_{p(\eta)} a] \leq \Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 0] \tag{7}$$

and

$$\Pr[(Z+B)(\eta, z) \Downarrow_{p(\eta)} b] \leq \Pr[(Z_p+B)(\eta, (z,b)) \Downarrow_{\infty} 1]. \tag{8}$$

Since Z_p is polynomial-time and A and B are computationally indistinguishable, we have that there is a negligible μ such that for all $z, b \in \{0,1\}^*$ and $\eta \in \mathbb{N}$, we have

$$\bigl|\Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 1] - \Pr[(Z_p+B)(\eta, (z,b)) \Downarrow_{\infty} 1]\bigr| \leq \mu(\eta). \tag{9}$$

Then we can show (6) and conclude the proof of $A \approx_{tic} B$:

$$\begin{aligned}
&\Pr[(Z+A)(\eta, z) \Downarrow_{p(\eta)} a] + \Pr[(Z+B)(\eta, z) \Downarrow_{p(\eta)} b]\\
&\overset{(7,8)}{\leq} \Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 0] + \Pr[(Z_p+B)(\eta, (z,b)) \Downarrow_{\infty} 1]\\
&\leq 1 - \Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 1] + \Pr[(Z_p+B)(\eta, (z,b)) \Downarrow_{\infty} 1]\\
&\overset{(9)}{\leq} 1 - \Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 1] + \Pr[(Z_p+A)(\eta, (z,b)) \Downarrow_{\infty} 1] + \mu(\eta)\\
&= 1 + \mu(\eta).
\end{aligned}$$

We now show that if A and B are reactively polynomial-time and $A \approx_{tic} B$, then $A \approx_{comp} B$.

Fix a polynomial-time Z. To show that $A \approx_{comp} B$, we have to show that there exists a negligible μ such that for all z and η,

$$\bigl|\Pr[(Z+A)(\eta, z) \Downarrow_{\infty} 1] - \Pr[(Z+B)(\eta, z) \Downarrow_{\infty} 1]\bigr| \leq \mu(\eta). \tag{10}$$

Let Z' be a machine that internally simulates Z, except that when Z sends $m \neq 1$ on *ifc*, Z' sends 0 on *ifc* instead. Since Z is polynomial-time, we can choose Z' to be polynomial-time as well.

[7] We assume that (m_1, m_2) is encoded by suitably interleaving the bits of m_1 and m_2, so that even if m_i is very long, m_{3-i} can be read in time $O(|m_{3-i}|)$.

Since Z' is polynomial-time and A and B are reactively polynomial-time, there is a polynomial p and a negligible function μ' such that

$$\forall \eta, z.\ \Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)}] \geq 1-\mu'(\eta) \quad \text{and} \quad \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)}] \geq 1-\mu'(\eta). \tag{11}$$

Since Z' only outputs 0 or 1, this implies

$$\forall \eta, z.\ \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 0] + \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 1] \geq 1-\mu'(\eta) \tag{12}$$

and

$$\forall \eta, z.\ \Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)} 0] + \Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)} 1] \geq 1-\mu'(\eta). \tag{13}$$

Since $A \approx_{tic} B$, there is a negligible μ'' such that for all z, η we have

$$\Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)} 1] + \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 0] \leq 1+\mu''(\eta) \tag{14}$$

and

$$\Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)} 0] + \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 1] \leq 1+\mu''(\eta). \tag{15}$$

For all η, z, we then have

$$\begin{aligned}
&\Pr[(Z+A)(\eta,z) \Downarrow_\infty 1] \\
&\overset{(*)}{=} \Pr[(Z'+A)(\eta,z) \Downarrow_\infty 1] \\
&\overset{(11)}{\leq} \Pr[(Z'+A)(\eta,z) \Downarrow_{p(\eta)} 1] + \mu'(\eta) \\
&\overset{(14)}{\leq} 1 - \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 0] + \mu''(\eta) + \mu'(\eta) \\
&\overset{(12)}{\leq} \Pr[(Z'+B)(\eta,z) \Downarrow_{p(\eta)} 1] + \mu''(\eta) + 2\mu'(\eta) \\
&\leq \Pr[(Z'+B)(\eta,z) \Downarrow_\infty 1] + \mu''(\eta) + 2\mu'(\eta) \\
&\overset{(*)}{=} \Pr[(Z+B)(\eta,z) \Downarrow_\infty 1] + \mu''(\eta) + 2\mu'(\eta).
\end{aligned}$$

Here $(*)$ uses that Z' outputs 1 iff the simulated Z does.

Using symmetric reasoning (and (15), (13) instead of (14), (12)), we get

$$\Pr[(Z+B)(\eta,z) \Downarrow_\infty 1] \leq \Pr[(Z+A)(\eta,z) \Downarrow_\infty 1] + \mu''(\eta) + 2\mu'(\eta)$$

With $\mu := \mu'' + 2\mu'$, (10) follows. □

Proof of Lemma 22. Assume that A is reactively polynomial-time, that $A \approx_{comp} B$, and that $C + A \approx_{tic} D + A$. Let $I := I_A \triangle I_C$. For contradiction, assume that $C + B \approx_{tic} D + B$ does not hold. Then there is an I-closing machine $Z \in \mathfrak{M}_{\text{Turing}}$, a polynomial p, a non-negligible function ν, and sequences z_η, a_η, b_η with $a_\eta \neq b_\eta$ such that for all η,

$$\Pr[(Z+C+B)(\eta,z_\eta) \Downarrow_{p(\eta)} a_\eta] + \Pr[(Z+D+B)(\eta,z_\eta) \Downarrow_{p(\eta)} b_\eta] \geq 1+\nu(\eta).$$

We will abbreviate this as

$$\Pr[Z + C + B \Downarrow_p a] + \Pr[Z + D + B \Downarrow_p b] \geq 1 + \nu. \tag{16}$$

Without loss of generality, there is a bitstring `stop` such that $a_\eta, b_\eta \neq$ `stop` for all η. (E.g., we can choose `stop` to be 0, and let Z prefix every output on *ifc* with 1.)

We construct the following machine ZC_p: It has interfaces $I_{ZC_p} = I_Z \triangle I_C$. Internally, it simulates $Z + C$, with the following modification: As soon as $Z + C$ runs more than a total number of $p(\eta)$ steps (all invocations together), ZC_p outputs `stop` on *ifc*. Intuitively, ZC_p is the machine that simulates the first $p(\eta)$ steps of $Z + C$. Analogously, we define ZD_p, simulating $p(\eta)$ steps of $Z + D$.

By construction, every output made by $Z + C + B$ within at most $p(\eta)$ steps will also be made by $ZC_p + B$. (Note that we do not claim that $ZC_p + B$ makes that output within $p(\eta)$ steps. This would not be the case as $ZC_p + B$ incurs some computational overhead for counting the steps of $Z + C$.) Hence $\Pr[ZC_p + B \Downarrow_\infty a] \geq \Pr[Z + C + B \Downarrow_p a]$. By construction, ZC_p is polynomial-time. Since $A \approx_{comp} B$, this implies that $\Pr[ZC_p + A \Downarrow_\infty a] \geq \Pr[ZC_p + B \Downarrow_\infty a] - \mu_C$ for some negligible function μ_C. Since A is reactively polynomial-time, and ZC_p polynomial-time, there is a polynomial q_C and a negligible function μ'_C such that $ZC_p + A$ runs at most q_C steps with probability at least $1 - \mu'_C$. Thus $\Pr[ZC_p + A \Downarrow_{q_C} a] \geq \Pr[ZC_p + A \Downarrow_\infty a] - \mu'_C$. Since ZC_p simulates $Z + C$ and either outputs on *ifc* what $Z + C$ would output, or outputs `stop` $\neq a$, and since ZC_p does not run faster than $Z + C$, we have $\Pr[Z + C + A \Downarrow_{q_C} a] \geq \Pr[ZC_p + A \Downarrow_{q_C} a]$. Combining the inequalities derived so far, we get

$$\Pr[Z + C + A \Downarrow_{q_C} a] \geq \Pr[Z + C + B \Downarrow_p a] - \mu_C - \mu'_C. \tag{17}$$

Analogously, we derive that

$$\Pr[Z + D + A \Downarrow_{q_D} b] \geq \Pr[Z + D + B \Downarrow_p b] - \mu_D - \mu'_D \tag{18}$$

for some polynomial q_D and negligible functions μ_D, μ'_D.

Furthermore, for $q := q_C + q_D$, we have $\Pr[Z + C + A \Downarrow_q a] \geq \Pr[Z + C + A \Downarrow_{q_C} a]$ and $\Pr[Z + D + A \Downarrow_q b] \geq \Pr[Z + D + A \Downarrow_{q_D} b]$. With (16), (17), (18) we then get

$$\Pr[Z + C + A \Downarrow_q a] + \Pr[Z + D + A \Downarrow_q b] \geq 1 + \nu'.$$

with $\nu' := \nu - \mu_C - \mu'_C - \mu_D - \mu'_D$. Since ν' is non-negligible, this implies that $C + A \approx_{tic} D + A$ does not hold. This is a contradiction to our assumptions. Thus $C + B \approx_{tic} D + B$ holds. $\square$

Proof of Lemma 24. Assume that $A \approx_{stat} B$. We wish to show that $A \approx_{tic}^{\mathfrak{M}} B$ for all $\mathfrak{M}$. Fix a machine model $\mathfrak{M}$, an I_A-closing machine $Z \in \mathfrak{M}$, a polynomial p, sequences $z_\eta, a_\eta, b_\eta \in \{0,1\}^*$ with $a_\eta \neq b_\eta$ for all η. We have to show that there is a negligible function μ such that

$$\Pr[(Z + A)(\eta, z_\eta) \Downarrow_{p(\eta)} a_\eta] + \Pr[(Z + B)(\eta, z_\eta) \Downarrow_{p(\eta)} b_\eta] \leq 1 + \mu(\eta)$$

for all η. We abbreviate this as

$$\Pr[Z + A \Downarrow_p a] + \Pr[Z + B \Downarrow_p b] \leq 1 + \mu.$$

Let Z_b be the machine that simulates Z, but when Z gives some output x, Z_b outputs 1 if $x = b_\eta$ and 0 otherwise. (We do not claim that $Z_b \in \mathfrak{M}$.) Then

$$\begin{aligned}
&\Pr[Z + A \Downarrow_p a] + \Pr[Z + B \Downarrow_p b] \\
&\leq \Pr[Z + A \Downarrow_\infty a] + \Pr[Z + B \Downarrow_\infty b] \\
&= \Pr[Z + A \Downarrow_\infty a] + \Pr[Z_b + B \Downarrow_\infty 1] \\
&\overset{(*)}{\leq} \Pr[Z + A \Downarrow_\infty a] + \Pr[Z_b + A \Downarrow_\infty 1] + \mu \\
&= \Pr[Z + A \Downarrow_\infty a] + \Pr[Z + A \Downarrow_\infty b] + \mu \\
&\overset{(**)}{\leq} 1 + \mu.
\end{aligned}$$

for some negligible μ. Here $(*)$ uses the statistical indistinguishability $A \approx_{stat} B$, and $(**)$ uses the fact that $a_\eta \neq b_\eta$ and hence that $Z + A \Downarrow_\infty a$ and $Z + A \Downarrow_\infty b$ are exclusive events. $\square$

C Postponed definitions for Section 5 (computational soundness)

In the definition of processes, we treat $|$ as commutative and associative, and 0 as the neutral element for $|$. (I.e., $0|A = A$.) And we consider two processes as equal if they are α-convertible (i.e., equal up to renaming of bound variables).

We assume an efficiently computable function $\ell : \mathit{terms} \to \mathbf{L}$. (We assume that both *terms* and $\mathbf{L}$ have some canonical encoding into bitstrings so that speaking of efficient computability of L makes sense.) $\ell(M)$ represents some formal abstraction of the computational concept of a length. Examples for length functions are: $\ell(M)$ is the term M with every nonce replaced by $*$ (in this case, ℓ is an overapproximation of the leakage produced by knowing the computational length of a bitstring, since from $\ell(M)$ we can compute the length of the bitstring corresponding to M.) Or $\ell(M)$ could be the actual length of the bitstring corresponding to M.

We define $\mathit{minrand}(P)$ and $\mathit{minnonce}(P)$ to be the smallest randomness symbol R or protocol nonce N, respectively, that is not contained in P.

Given a sequence S of terms, let φ_S be the substitution that maps x_i to S_i where x_i are fixed, distinct variables. An adversary term is a term containing no protocol nonces and no randomness symbols. We define $\vdash$ as follows: $S \vdash M$ iff there is an adversary term M' such that $M = M'\varphi_S$.

We define the following reduction relation $\longrightarrow$ (where S denotes a list of terms and U a set of nonces and randomness symbols):

$$
\begin{array}{rll}
S;U \parallel \bar{c}\langle M\rangle.P \mid c(x).Q \longrightarrow S;U \parallel P \mid Q\{M/x\} & & (c \text{ private})\\
S;U \parallel \bar{c}\langle M\rangle.P \mid !c(x).Q \longrightarrow S;U \parallel P \mid !c(x).Q \mid Q\{M/x\} & & (c \text{ private})\\
S;U \parallel \mathsf{if}\ M = M\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel P & & \\
S;U \parallel \mathsf{if}\ M = M'\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel Q & & (M \neq M')\\
S;U \parallel \mathsf{let}\ (x,y) = (M,M')\ \mathsf{in}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel P\{M/x, M'/y\} & & \\
S;U \parallel \mathsf{let}\ (x,y) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel Q & & (M \text{ not a pair})\\
S;U \parallel \mathsf{let}\ x = \mathsf{Dec}_K(\mathsf{enc}(K,M,R))\ \mathsf{in}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel P\{M/x\} & & \\
S;U \parallel \mathsf{let}\ x = \mathsf{Dec}_K(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q \longrightarrow S;U \parallel Q & & (M \neq \mathsf{enc}(K,\cdot,\cdot))\\
S;U \parallel \mathsf{let}\ x = \mathsf{Enc}_K(M)\ \mathsf{in}\ P \longrightarrow S;U\cup\{R\} \parallel P\{\mathsf{enc}(K,M,R)/x\} & & (R := \mathit{minrand}(U))\\
S;U \parallel \mathsf{let}\ x = (M_1,M_2)\ \mathsf{in}\ P \longrightarrow S;U \parallel P\{(M_1,M_2)/x\} & & \\
S;U \parallel \nu x.P \longrightarrow S;U\cup\{N\} \parallel P\{N/x\} & & (N := \mathit{minnonce}(U))\\
S;U \parallel \overline{\mathsf{net}}\langle M\rangle.P \xrightarrow{\bar{M}} S,M;U \parallel P & & \\
S;U \parallel \mathsf{net}(x).P \xrightarrow{M} S;U \parallel P\{M/x\} & & (S \vdash M)\\
S;U \parallel P \mid Q \xrightarrow{\alpha} S';U' \parallel P' \mid Q & & (\xrightarrow{\alpha} \in \{\xrightarrow{M}, \longrightarrow\},\\
 & & S;U \parallel P \xrightarrow{\alpha} S';U' \parallel P')
\end{array}
$$

We call a process *honest* if every term in P is a variable.

Definition 31 *The set of* deterministic *processes is the largest set such that:*

- *If P is deterministic and $P \longrightarrow P'$, then P' is deterministic.*
- *If P is deterministic and $P \longrightarrow Q$, $P \longrightarrow Q'$, then $Q = Q'$.*
- *If P is deterministic and $P \xrightarrow{\bar{M}} Q$ and $P \xrightarrow{\bar{M}'} Q'$, then $M = M'$ and $Q = Q'$.*
- *If P is deterministic and $P \xrightarrow{M} Q$ and $P \xrightarrow{M} Q'$, then $Q = Q'$.*
- *If P is deterministic, then the following three statements are mutually exclusive for all Q_1, Q_2, Q_3, M_2, M_3: $P \longrightarrow Q_1$, $P \xrightarrow{\bar{M}_2} Q_2$, and $P \xrightarrow{M_3} Q_3$.*

Notice that determinism does not exclude $P \xrightarrow{M_1} Q_1$ and $P \xrightarrow{M_2} Q_2$ for $M_1 \neq M_2$. This is because the incoming message M_i is chosen by the adversary, not by the protocol.

Definition 32 (Adversary term) *A* destructor term *D is a term of the following grammar:*

$$D ::= x \mid N \mid (D,D) \mid \mathsf{enc}(K,D,R) \mid \mathsf{garb}^L(N) \mid \mathsf{fst}(D) \mid \mathsf{snd}(D)$$

An adversary term *is a destructor term that does not contain protocol nonces (but may contain arbitrary keys and adversary nonces).*

The evaluation eval(D) *of a closed destructor term is defined recursively as follows:* eval$(N) := N$. eval$(D_1,D_2) :=$ (eval(D_1), eval(D_2)) *or* $\perp$ *if* eval(D_1) *or* eval(D_2) *return*

$\perp$. $\mathrm{eval}(\mathsf{enc}(K,D,R)) := \mathsf{enc}(K,\mathrm{eval}(D),R)$ *or* $\perp$ *if* $\mathrm{eval}(D)$ *returns* $\perp$. $\mathrm{eval}(\mathsf{garb}^L(N)) :=$ $\mathsf{garb}^L(N)$. $\mathrm{eval}(\mathsf{fst}(D)) := m_1$ *and* $\mathrm{eval}(\mathsf{snd}(D)) = m_2$ *if* $\mathrm{eval}(D) = (m_1,m_2)$, *and* $\mathrm{eval}(\mathsf{fst}(D)), \mathrm{eval}(\mathsf{snd}(D)) :=$ $\perp$ *if* $\mathrm{eval}(D)$ *is not a pair.*

Definition 33 (Static equivalence) *Two substitutions* φ_S *and* φ_T *are* statically equivalent *iff the following holds:*

- $\mathrm{dom}\,\varphi_S = \mathrm{dom}\,\varphi_T$.
- *For all adversary terms* D *with* $fv(D) \subseteq \mathrm{dom}\,\varphi_S$ *we have that* $\mathrm{eval}(D\varphi_S) = \perp$ *iff* $\mathrm{eval}(D\varphi_T) = \perp$.
- *For all adversary terms* D *with* $fv(D) \subseteq \mathrm{dom}\,\varphi_S$ *we have that* $\ell(\mathrm{eval}(D\varphi_S)) = \ell(\mathrm{eval}(D\varphi_T))$.
- *For all adversary terms* D *with* $fv(D) \subseteq \mathrm{dom}\,\varphi_S$ *we have that* $\mathrm{eval}(D\varphi_S)$ *and* $\mathrm{eval}(D\varphi_T)$ *have the same type (where the type of a term is one of encryption, pair, nonce, or garbage).*
- *For all adversary terms* D *with* $fv(D) \subseteq \mathrm{dom}\,\varphi_S$ *and* $\mathrm{eval}(D\varphi_S) = \mathsf{enc}(K,M,R)$ *and* $\mathrm{eval}(D\varphi_T) = \mathsf{enc}(K',M',R')$, *we have* $K = K'$.
- *For all adversary terms* D, D' *with* $fv(D), fv(D') \subseteq \mathrm{dom}\,\varphi_S$, *we have that* $\mathrm{eval}(D\varphi_S) = \mathrm{eval}(D'\varphi_S)$ *iff* $\mathrm{eval}(D\varphi_T) = \mathrm{eval}(D'\varphi_T)$.

Definition 34 (Labeled bisimilarity) $\cong_{bisi}$ *is the largest symmetric relation such that* $S;U\|P \cong_{bisi} T;V\|Q$ *implies:*

- φ_S *and* φ_T *are statically indistinguishable.*
- *If* $S;U\|P \longrightarrow S';U'\|P'$, *then* $T;V\|Q \longrightarrow{}^* T';V'\|Q'$ *and* $S';U'\|P' \cong_{bisi} T';V'\|Q'$ *for some* T',V',Q'.
- *For all adversary terms* D, *if* $S;U\|P \xrightarrow{D\varphi_S} S';U'\|P'$ *then there are* T',V',Q' *such that* $T;V\|Q \longrightarrow{}^* \xrightarrow{D\varphi_T}\longrightarrow{}^* T';V'\|Q'$ *and* $S';U'\|P' \cong_{bisi} T';V'\|Q'$.
- *If* $S;U\|P \xrightarrow{\bar{M}} S';U'\|P'$ *then there are* T',V',Q',M' *such that* $T;V\|Q \longrightarrow{}^* \xrightarrow{\bar{M}'}\longrightarrow{}^* T';V'\|Q'$ *and* $S';U'\|P' \cong_{bisi} T';V'\|Q'$.

We abbreviate $\varnothing;\varnothing\|P \cong_{bisi} \varnothing;\varnothing\|Q$ *as* $P \cong_{bisi} Q$.

Given a partial function ξ from variables to bitstrings with finite domain, let $\min(\xi)$ denote the smallest variable $x \notin \mathrm{dom}\,\xi$. Let $\min_2(\xi)$ denote the pair consisting of the two smallest variables $x, y \notin \mathrm{dom}\,\xi$. Let $pair : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ denote the pair building functions, and let $(KeyGen, Enc, Dec)$ be an IND-CCA secure public-key encryption scheme.

Definition 35 (Computational execution) *The computational execution of a closed honest deterministic process* P *is a machine* M_P *with interfaces* $I_{M_P} = \{\mathsf{net}\}$ *that performs the following steps:*

- *At any point, when getting a message* $(\mathtt{getpk}, K)$ *on* net *(while waiting for some other message, e.g.,* $(\mathtt{continue})$ *or* $(\mathtt{input}, m)$*), send* $(\mathtt{pk}, pk_K)$ *on* net.

- *Let η denote the security parameter. We maintain values pk_K, sk_K for all keys K which are initialized as $(pk_K, sk_K) \leftarrow KeyGen(1^\eta)$ upon first use. Set $P_{curr} := P$ and $\xi := \varnothing$. (ξ is a partial function from variables to bitstrings, initially empty.) Wait for a message* (continue) *on* net. *Then proceed with the main loop.*
- *Main loop: Repeat the following ad infinitum.*
  - *If $P_{curr} = \bar{c}\langle x\rangle.P_1|c(y).P_2|P_3$ with $y = \min(\xi)$:*
    Let $P_{curr} := P_1|P_2|P_3$ and $\xi(x) := \xi(y)$.
  - *If $P_{curr} = \bar{c}\langle x\rangle.P_1|!c(y).P_2|P_3$ with $y = \min(\xi)$:*
    Let $P_{curr} := P_1|!c(y).P_2|P_2|P_3$ and $\xi(x) := \xi(y)$.
  - *If $P_{curr} = (\mathsf{if}\ x = y\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2)|P_3$:*
    If $\xi(x) = \xi(y)$, let $P_{curr} := P_1|P_3$. Otherwise, let $P_{curr} := P_2|P_3$.
  - *If $P_{curr} = (\mathsf{let}\ (x, y) = z\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2)|P_3$ with $(x, y) = \min_2(\xi)$:*
    If $\xi(z) = pair(m_1, m_2)$ for some m_1, m_2, then let $\xi(x) := m_1$, $\xi(y) := m_2$, and $P_{curr} := P_1|P_3$. Otherwise let $P_{curr} := P_2|P_3$.
  - *If $P_{curr} = (\mathsf{let}\ x = \mathsf{Dec}_K(y)\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2)|P_3$ with $x = \min(\xi)$:*
    Compute $m := Dec(sk_K, \xi(y))$. If Dec fails, let $P_{curr} := P_2|P_3$. Otherwise let $\xi(x) := m$ and $P_{curr} := P_1|P_3$.
  - *If $P_{curr} = (\mathsf{let}\ x = (y, z)\ \mathsf{in}\ P_1)|P_2$ with $x \in \min(\xi)$:*
    Let $P_{curr} := P_1|P_2$ and $\xi(x) := pair(\xi(y), \xi(z))$.
  - *If $P_{curr} = (\mathsf{let}\ x = \mathsf{Enc}_K(z)\ \mathsf{in}\ P_1)|P_2$ with $x = \min(\xi)$:*
    Let $P_{curr} := P_1|P_2$ and $\xi(x) := Enc(pk_K, \xi(z))$.
  - *If $P_{curr} = P_1|\mathsf{net}(x).P_2$ with $x = \min(\xi)$:*
    Send (`wantinput`) *on* net. *Wait for a message* (`input`, m) *on* net, *let $\xi(x) := m$ and set $P_{curr} := P_1|P_2$.*
  - *If $P_{curr} = P_1|\overline{\mathsf{net}}\langle x\rangle.P_2$:*
    Send (`output`, $\xi(x)$) *on* net. *Wait for a message* (`continue`) *on* net. *Set $P_{curr} := P_1|P_2$.*
  - *If $P_{curr} = P_1|\nu x.P_2$ with $x = \min(\xi)$:*
    Pick $m \overset{\$}{\leftarrow} \{0,1\}^\eta$. Let $\xi(x) := m$ and set $P_{curr} := P_1|P_2$.
  - *All other cases:*
    Do nothing (i.e., loop).

References

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

[2] Michael Backes, Dennis Hofheinz, and Dominique Unruh. CoSP: A general framework for computational soundness proofs. In *ACM CCS 2009*, pages 66–78. ACM Press, November 2009. Full version on IACR ePrint 2009/080.

[3] 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.

[4] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 255–269, June 2008. Preprint on IACR ePrint 2008/152. To appear in the Journal of Computer Security.

[5] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proc. ACM CCS*, pages 109–118, 2008.

[6] Véronique Cortier, Steve Kremer, Ralf Küsters, 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.

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

[8] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Polynomial runtime and composability. IACR ePrint 2009/023, 2009.

Index

Symbol index

$\mathfrak{M}$	Usually denotes a machine model	
M	Usually denotes a machine	
η	Security parameter	
σ	Usually denotes state of a machine	
ifc	Usually dentoes an interface of a machine	
M_0	Machine that never terminates	6
$\dot{\cup}$	Disjoint union of sets	
$\triangle$	Symmetric difference of sets	
I_M	Interfaces of machine M	8
σ_M	Initial state of machine M	8
δ_M	State transition function of machine M	8
$\mathfrak{M}_{\text{Turing}}$	Machine model with Turing machines	8
$\mathfrak{M}_0$	Machine model with zero running time	8
$+$	Composition of two machines	8
$\Downarrow_t$	Termination after t steps	9
$\sim$	Intermediate notion of indistinguishability	9
$\approx_{tic}^{\mathfrak{M}}$	Termination-insensitive indistinguishability w.r.t. machine model $\mathfrak{M}$	9
$\lesssim_{time}$	Is time-majorized by	10
$\approx_{time}$	Time-equivalence	10
$\approx_{tic}$	Termination-insensitive computational indistinguishability (short for $\approx_{tic}^{\mathfrak{M}_{\text{Turing}}}$)	10
$\approx_{comp}$	Computational indistinguishability	11
$\approx_{stat}$	Statistical indistinguishability	11
M_P	Computational execution of process P	12
M_P^*	Computational execution of P with outsourced operations	12
M_{cmd}	Machine for outsourcing bitstring operations	12
M_{sym}	Variant of M_{cmd}, working with terms	12
Sim	Simulator in computational soundness proof	13
$\longrightarrow$	Reduction relation of the process calculus	20
$\cong_{bisi}$	Labeled bisimilarity	21
ξ	Substitution storing process's environment	22