Model–based Testing of Cryptographic Protocols

Dean Rosenzweig[1,2], Davor Runje[2], and Wolfram Schulte[1]

[1] Microsoft Research
schulte@microsoft.com
[2] University of Zagreb
dean@math.hr, davor.runje@fsb.hr

Abstract. Modeling is a popular way of representing the behavior of a system. A very useful type of model in computing is an abstract state machine which describes transitions over first order structures. The general purpose model-based testing tool SpecExplorer (used within Microsoft, also available externally) uses such a model, written in AsmL or Spec#, to perform a search that checks that all reachable states of the model are safe, and also to check conformance of an arbitrary .NET implementation to the model. Spec Explorer provides a variety of ways to cut down the state space of the model, for instance by finitizing parameter domains or by providing predicate abstraction. It has already found subtle bugs in production software.

First order structures and abstract state machines over them are also a useful way to think about cryptographic protocols, since models formulated in these terms arise by natural abstraction from computational cryptography.

In this paper we explain this abstraction process, 'experiments as structures', and argue for its faithfulness. We show how the Dolev–Yao intruder model fits into SpecExplorer. In a word, the actions of the Dolev–Yao intruder are the 'controllable' actions of the testing framework, whereas the actions of protocol participants are the 'observable' actions of the model. The unsafe states are the states violating say Lowe's security guarantees. Under this view, the general purpose software testing tool quickly finds known attacks, such as Lowe's attack on the Needham–Schroeder protocol.

Introduction: Why Yet Another Formal Model

A new 'behavioral' theory of algorithms has been developed in recent years in a series of papers by Y.Gurevich, A.Blass [Gur00,BG03,BG04a,BG04b,Gur05], and also B.Rossman and the authors [RR05]. The gist is that algorithms can be mathematically captured at their own native level of abstraction - ex. the native level of abstraction of the Euclidean algorithm is that of Euclidean rings. Algorithms operate over abstract first-order structures, well studied and familiar in mathematical logic, algebra and abstract mathematics in general.

The techniques developed for behavioral theory suggest a natural representation of Dolev-Yao assumptions in first-order structures, and a natural mapping of ad-hoc notations present in abstract models of cryptography. Unlike the

static abstract models, which necessarily invoke additional proof-theoretic devices to capture dynamic aspects, the behavioral theory explicitly targets the dynamic behavior of algorithms semantically. By recent work on behavioral theory [BG04a,BG04b,RR05,Gur05], this also includes interactive algorithms talking to an environment between steps, and within a step, allowing us to represent the abstract content of oracle algorithms and adversary games typical of computational cryptography directly. In the framework of intra-step interactive algorithms exact abstract representations of computational security notions, defined in terms of adversary games, emerge clearly. The experiments of asymptotic computational cryptography can be naturally represented in terms of interactive algorithms over first-order structures, this is our experiments-as-structures paradigm, providing a setting for soundness/completeness proofs. The abstract content of these proofs gets more clearly separated from the probabilistic aspects.

In this paper we execute a small initial segment of this program, in case of confusion-free symmetric encryption. Abstract models for the standard asymptotic security notions in this case are provided, with proofs of their soundness (under the assumption of acyclicity) and completeness. The relation of these proofs to proofs in the literature [AR02,MW04a,AJ01,Ban04,ABS05] can best be described as extraction of abstract content. We also briefly indicate how the assumptions of confusion-freeness and acyclicity can be relaxed in our setting. Partially establishing the exact relation to existing models of abstract cryptography, we show how a variant of Abadi-Rogaway expressions with explicit coins naturally embeds into our framework.

Section 1 is a (necessarily cursory) overview of the behavioral theory of algorithms, essentially referring the reader to the literature. Section 2 is a brief summary of the relevant assumptions of asymptotic computational cryptography in the asymmetric (public key) case. Section 3 presents the experiments-as-structures paradigm and our abstract model of cryptographic adversary games. Section 4 contains sketches of soundness and completeness proofs, and how the Abadi-Rogaway expression language variant embeds into our framework. Testing model for public key protocols is in Section 5, together with an example of rediscovery of Lowe's attack on the Needham–Schroeder protocol by SpecExplorer.

In addition to quoted cryptographic literature, some understanding of the framework as presented in [RR05] is expected of the reader.

1 Behavioral Theory of Algorithms

The behavioral theory of algorithms is *not* an attempt to question the Church-Turing thesis, saying that every computable function over natural numbers can be computed by a Turing machine, or the stronger implicit thesis, actually argued for by Turing, that every algorithm can be simulated by a Turing machine. The aim of the behavioral theory is to make semantical distinctions finer than that precise.

While algorithms get implemented (simulated) exclusively over bits these days, they are often intended to operate over much more abstract objects, ab-

stract data-structures of algebraic or geometric or analytic or even not explicitly mathematical character. The behavioral theory aims to capture algorithms as they are intended, at their own level of abstraction.

The requirement of "capturing algorithms at their own level of abstraction" is made precise as the requirement of simulation step-by-step. The technology to achieve this is using first-order structures, well known to capture faithfully arbitrary static mathematical situations, as states of algorithms. The dynamics, the step, is also defined in terms of the abstract state.

This philosophy leads to a sharp mathematical definition, technically developed in [Gur00] and overviewed in [BG03], computationally realized in the theoretical programming language of Abstract State Machines [Gur00] and the implemented programming languages AsmL [AsmL] and Spec# [Spec#]. Models written in these modelling languages are used by a model-based software testing tool SpecExplorer, also developed at Microsoft Research [SpecExp].

1.1 Interactive Algorithms

Interactive algorithms issue *queries* to the environment, which contain labels and data, and receive replies, which are data, elements of algorithm's state, within a step. This mechanism allows a clean separation of computational (the algorithm) and declarative (the environment) aspects, and naturally models nondeterminism, function calls, interaction with oracles, input and output,...The full theory of (ordinary) interactive algorithms is developed in [BG04a,BG04b]; overviews are given in [Gur05] and [RR05].

All algorithms in this paper are assumed to be small-step ordinary interactive algorithms in the sense of [BG04a,BG04b,Gur05].

1.2 Accessibility, Reachability and Indistinguishability

The notions of accessibility of objects, reachability and indistinguishability of states, as introduced in [RR05], will be important here. An object is *accessible* at a state if it is the value of a term there. A state Y is *reachable* from a state X if there is an algorithm turning X to Y. Two states X, Y are *distinguishable* if there is an algorithm turning them into states distinct by values of a specific term. Structures X, Y of the same vocabulary are *similar*, written $X \sim Y$ if they induce the same equivalence on ground terms:

$$Val(t_1, X) = Val(t_2, X) \text{ iff } Val(t_1, Y) = Val(t_2, Y)$$

Precise definitions and the theory behind these notions can be found in [RR05]. Here we shall repeatedly use the following results from [RR05](where $Y - X$ is the set of differences of two states over the same carrier, see [Gur00,BG04a,RR05] for definitions):

Theorem 1. *State Y is reachable from state X iff*

  – *X and Y have the same base set; and*

– $Y - X$ is finite and every element in $Y - X$ is accessible.

Theorem 2. *State X and Y are indistinguishable by small–step algorithm iff $X \sim Y$.*

1.3   Background Structures and Importing/Creating

An algorithms often needs to create a new object. A Turing machine often needs to access a new tape location never used before.

In the TM case it obviously doesn't matter whether we conceive its tape as finite, creating new locations as needed, or as infinite, with all locations possibly needed given in advance. In the latter case locations get activated as the TM visits them for the first time.

The case of a first-order structure is the same, a reserve pool ("the heap") of sufficiently many fresh amorphous objects can be given in advance, to be accessed as needed. For interactive algorithms, they are available to the environment to be returned in reply to an appropriate query (get me a new ...). The reserve elements are amorphous in the sense that no "significant" functions are defined on them, or denote them as values. For abstract cryptography the amorphous reserve objects will represent random coins.

But if we have some infrastructure defined on all objects, such as ordered pairs and/or finite sets and/or encryptions, it would be both unnatural and very boring to have to establish all the infrastructure over a new element each time one is introduced, brought forward from the reserve.

The notions of *background structure* and *background class* [BG00] serve exactly this purpose: the axioms for a background class of [BG00] specify what kind of structure can exist over amorphous atoms without imposing any specific properties on them except for identity.

See [BG00,RR05] for definitions of background classes, background of algorithms, exposed elements, active part, reserve.

A structure X is *explicitly atom–generated* if the smallest substructure of X that includes all atoms is X itself. All background structures in the paper are assumed to be explicitly atom-generated. *Atomic support* of a set S of elements of a structure X from a background class $\mathcal{K}$ is the set of atoms of the *envelope* of S, the smallest $\mathcal{K}$-substructure containing S.

Corollary 1. *If the atomic support $Sup_X(\{x\})$ of an element x is accessible in a state X, then x is accessible in X.*

We assume that the set of exposed elements is finite, but not necessary uniformly bounded, in every state. Remember that the foreground of an algorithm is its (generalized) memory, storing input data and results of previous calculations. As such, after a finite number of algorithm steps, only a finite number of locations can be changed.

Let $\mathbf{0}_X$ denote the reduct of X to the background vocabulary, the structure obtained by "forgetting" all foreground functions in state X. We assume that

4

all states have an infinite but countable reserve. It follows immediately from the axioms of [BG00] that if X and Y are $\mathcal{K}$-states over the same carrier, then their background reducts are isomorphic $\mathbf{0}_X \cong \mathbf{0}_Y$.

Theorem 3. *Let X be a state with background BC. Then there is an algorithm A and an injective answer function α appropriate for $\mathbf{0}_X$ with only reserve elements in its codomain such that $X = A(\mathbf{0}_X, \alpha)$.*

2 Computational Cryptography

2.1 Encryption Schemes

An *asymmetric encryption scheme Π* is a tuple of polytime algorithms $(\mathcal{K}, \mathcal{I}, \mathcal{E}, \mathcal{D})$

$$\mathcal{K} : \texttt{Parameter} \times \texttt{Coins} \longrightarrow \texttt{DecryptionKey}$$
$$\mathcal{I} : \texttt{DecryptionKey} \longrightarrow \texttt{EncryptionKey}$$
$$\mathcal{E} : \texttt{EncryptionKey} \times \texttt{String} \times \texttt{Coins} \longrightarrow \texttt{Ciphertext} \cup \{\bot\}$$
$$\mathcal{D} : \texttt{DecryptionKey} \times \texttt{String} \longrightarrow \texttt{Plaintext} \cup \{\bot\}$$

where $\texttt{String}$ denotes the set of finite strings over $\{0, 1\}$, domains $\texttt{EncryptionKey}$, $\texttt{DecryptionKey}$, $\texttt{Ciphertext}$, $\texttt{Plaintext}$ are subsets of $\texttt{String}$, $\bot$ is a distinguished string representing failure of the algorithm, and $\texttt{Coins}$ is the set of all infinite strings over $\{0, 1\}$. The polytime assumption for $\mathcal{K}$ means time polynomial in η (not the size of its string representation) and ignores the $\texttt{Coins}$ argument representing random coin flips. Suppressing the $\texttt{Coins}$ argument $\mathcal{K}, \mathcal{E}$ become probabilistic polytime algorithms, and $\mathcal{K}(\eta, c), \mathcal{E}(k, m, c), \mathcal{D}(K, m)$ are, according to tradition, often written as $\mathcal{K}(\eta), \mathcal{E}_k(m), \mathcal{D}_K(m)$ respectively.

The key-inversion algorithm $\mathcal{I}$ returns an encryption key matching the decryption key.

Remark 1 (Usual Assumptions). We require that

- $\mathcal{D}_K(\mathcal{E}_k(m)) = m$ whenever $k = \mathcal{I}(K)$, for every key K sampled from $\mathcal{K}(\eta)$ and every plaintext m such that $\mathcal{E}_k(m)$ doesn't fail;
- the $\texttt{Plaintext}$ domain is the set of all m for which, for some $\texttt{EncryptionKey}$ k, $\mathcal{E}_k(m)$ doesn't fail; $\texttt{Ciphertext}$ is the corresponding codomain;
- if K, K' are two outputs of $\mathcal{K}(\eta)$ for the same η, then
  - K, K' have the same length;
  - $k = \mathcal{I}(K), k' = \mathcal{I}(K')$ have the same length;
  - if m, m' are strings of the same length, then $\mathcal{E}_k(m)$ doesn't fail if and only if $\mathcal{E}_{k'}(m')$ doesn't fail, and then the encryptions have the same length.

Remark 2. Syntax of an asymmetric encryption scheme is usually defined as a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ returns a pair of both encryption and decryption keys [BDPR98]. But then $\mathcal{I}$ is simply a projection and the decryption algorithm simply ignores one of the parameters. We find our variant more convenient for the purpose of abstract modeling.

We also assume a (polytime) encoding of ordered pairs, which means a triple of functions $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$, where $\mathcal{P}$ is a binary pairing function on strings, and $\mathcal{F}$ and $\mathcal{S}$ are unary projections, with the usual properties. We also assume a type-flaw preventing tagging scheme, ensuring that the codomains of $\mathcal{K}, \mathcal{E}, \mathcal{I}, \mathcal{P}$ are pairwise disjoint, and that neither of them contains $\bot$.

2.2 Notions of Security

Notions of security of encryption schemes are typically based on a notion of indistinguishability, represented by two sequences of oracles of the same length, the good-oracles $O_1^{\mathcal{G}}, \ldots, O_n^{\mathcal{G}}$ and the fake-oracles $O_1^{\mathcal{F}}, \ldots, O_n^{\mathcal{F}}$. Each of the sequences gets initialized by randomly generating a sequence of keys to be used by respective oracles, good-init and fake-init. The oracles and the initializations are implicitly paremeterized by the encryption scheme Π and possibly the pairing scheme Σ, but we shall drop this from the notation. Some data resulting from the initialization can be passed to the adversary algorithm as parameters—we consider this to be a part of the initialization. Let us call the initialization and oracle data just ATT, and let the notion of security defined by ATT be IND-ATT. The idea is that no PPT-limited adversary can distinguish whether she is working with the good or the fake oracles:

Definition 1. Let A be an algorithm working with n oracles. Its *advantage* for IND-ATT is

$$
\begin{aligned}
\mathsf{Adv}_\Pi^{\text{ind-att}}(A) = \Pr[\mathsf{good\text{-}init} :\ A(\ldots)^{O_1^{\mathcal{G}}, \ldots, O_n^{\mathcal{G}}} = 1] - \\
\Pr[\mathsf{fake\text{-}init} :\ A(\ldots)^{O_1^{\mathcal{F}}, \ldots, O_n^{\mathcal{F}}} = 1]
\end{aligned}
$$

The encryption scheme Π is IND-ATT secure if no probabilistic polytime algorithm A can guess which set of oracles it is provided with probability negligible in the security parameter η: $\mathsf{Adv}_\Pi^{\text{ind-att}}(A)$ is negligible.

The $A(\ldots)$ notation denotes the adversary algorithm called with any parameters that the initialization chooses to provide. Thus the notion of security is completely characterized by the initializations and the oracles selected.

By *negligible* we mean, throughout this paper, *polynomially negligible* functions: $f(n)$ such that for every c for all sufficiently large n we have $f(n) \leq \frac{1}{n^c}$, and by *overwhelming* those negligibly close to 1.

We define oracles characterizing notions of securities called *indistinguishability under chosen–plaintext attack* and indistinguishability under adaptive chosen–ciphertext attack, denoted with IND-CPA and IND-CCA, respectively.

Example 1 (IND-CPA).

- Let good-init be $K \leftarrow \mathcal{K}(\eta)$, passing along to the adversary algorithm $k = \mathcal{I}(K)$.
- Let good-oracles be $\mathcal{O}$ with $\mathcal{O}(m_1, m_2) = \mathcal{E}_k(m_1)$.
- Let fake-init be as good-init.

– Let fake-oracles be $\mathcal{O}$ with $\mathcal{O}(m_1, m_2) = \mathcal{E}_k(m_2)$, where $k = \mathcal{I}(K)$.

This defines the notion of security known as IND-CPA, "security under known plaintext attack".

Example 2 (IND-CCA).

– Let good-init, fake-init be as for IND-CPA.
– Let good-oracles be $\mathcal{O}, \mathcal{O}_d$ where $\mathcal{O}$ is as good-oracles of IND-CPA, and $\mathcal{O}_d(e) = \mathcal{D}_K(e)$ given that e is not an output obtained from $\mathcal{O}$; if it is, then $\mathcal{O}_d(e)$ fails.
– Let fake-oracles be $\mathcal{O}, \mathcal{O}_d$, with $\mathcal{O}$ as fake-oracle of IND-CPA and $\mathcal{O}_d$ as in good-init.

This defines a strictly stronger notion of security known as IND-CCA or IND-CCA2, "security under known ciphertext attack".

Nonces Nonces are random values enclosed with some formating data generated with a nonce generation algorithm $\mathcal{N}$. They serve as a source of fresh, unguessable data exchanged in protocols. Nonce generation algorithms can be stateful, which somewhat complicates the appropriate definition of their security. We define the advantage of an arbitrary algorithm A of breaking the security of nonce generation algorithm $\mathcal{N}$ as a probability of succeeding in the following game: $k+l+1$ nonces are sequentially generated with $\mathcal{N}$ and then the algorithm A is run on the first k and the last l nonces with the task to guess the value of $k + 1$-th nonce:

$$\mathsf{Adv}_{\mathcal{N}}^{\mathrm{nonce}}(A) = \Pr[\boldsymbol{m}, n, \boldsymbol{p} \xleftarrow{\$} \mathcal{N}(\eta) : A(\boldsymbol{m}, \boldsymbol{p}) = n]$$

If this advantage is negligible in η for every PPT algorithm A, then $\mathcal{N}$ is secure.

In practice, this type of security is achieved by simply enclosing η long uniformly sampled string with formating data.

2.3 Confusion Freeness and Weak Key Authenticity

Neither the syntax of an encryption scheme nor the typical notions of security, such as the one defined above, say much about what happens if we attempt to decrypt an encryption with a key distinct from the decryption key. Syntax of an encryption scheme allows for such decryption to fail, but it does not insist on it. If it does not fail, notions of security forbid that the result is in any *meaningful way* related to the underlying plaintext — a PPT algorithm has no way of distinguishing it from any other potential plaintext with non-negligible probability.

As a reader might already suspect, a failure to detect such situations would affect the completeness of an abstract model of cryptography. It is implicitly assumed that an abstract agent recognizes undecryptable encryptions in most

if not all abstract models; if a PPT agent in the computational model is strictly weaker, then the abstract model would be incomplete.

We might require that decrypting an encryption with independently generated fresh key fails with all but negligible probability (as a function of security parameter η). This property was defined in [MW04a] and called *confusion freeness*. It is sufficient to prove the completeness of an abstract model. Similar and independent definition can also be found in [AJ01].

However, confusion freeness is a quite strong requirement on an encryption scheme. It turned out not to be a necessary one: a strictly weaker notion called *weak key authenticity* was defined and shown to be both necessary and sufficient for proving completeness [HG03]. Weak key authenticity requires only that an attempt to decrypt an encryption with incorrect decryption key fails with non-negligible probability.

3 The Abstract Model

3.1 Messages as Experiments

The act of creating a cryptographic message, in view of the probabilistic character of cryptographic algorithms, is a probabilistic experiment. Say the message is $\mathcal{E}_k(\mathcal{P}(n,0))$. Without any contextual assumptions on the key k and nonce n, meaning that they should be freshly generated, this implies the following cryptographic experiment:

$$[K \xleftarrow{\$} \mathcal{K}(\eta); k \longleftarrow \mathcal{I}(K); n \xleftarrow{\$} \mathcal{N}(\eta); m \longleftarrow \mathcal{P}(n,0); e \xleftarrow{\$} \mathcal{E}(k,m) : e]$$

While it is easy to formalize the above notation for experiments directly, we skip it here. It should suffice to say that an experiment is a sequence of actions delimited with semicolon; if the experiment has an output, then it is separated from preceding actions by a colon. Left arrows are assignment operators, sometimes decorated with $ to emphasize the use of randomized algorithms on the right hand side.

Expanding the shorthand for probabilistic algorithms, the above experiment would take the form of

$$[c_1 \xleftarrow{\$} \texttt{Coins}; K \longleftarrow \mathcal{K}(\eta, c_1); k \longleftarrow \mathcal{I}(K); c_2 \xleftarrow{\$} \texttt{Coins};$$
$$n \longleftarrow \mathcal{N}(c_2); m \longleftarrow \mathcal{P}(n,0); c_3 \xleftarrow{\$} \texttt{Coins}; e \longleftarrow \mathcal{E}(k,m,c_3) : e]$$

We shall in the sequel assume that all experiments are so expanded, that $\xleftarrow{\$}$ appears only at the left of `Coins`.

3.2 Experiments as Terms

Here we develop a more systematic notation for representing cryptographic probabilistic experiments, with well-known and widely used *terms* of first-order logic.

In logic every function symbol comes equiped with its arity and, optionally, can be marked as relational. We in addition mark some function symbols as *probabilistic* and some as *parameterized*.

Here we list all vocabularies that will be used throughout this paper.

Vocabularies:

- Υ_{log} is the vocabulary of logical constants, containing nullary symbols true, false and undef, the usual boolean operators and the equality =.
- Υ_{exp} contains unary symbols key, inv, fst, snd and nonce, binary decrypt and pair, and ternary encrypt. Symbols key, nonce and encrypt are marked as probabilistic and symbols key and nonce are also marked as parameterized.
- Υ_{const} contains nullary symbols for some constants, at least for bits 0 and 1.
- Υ_{fun} contains unary relation symbols PriKey, PubKey, Ciphertext, Pair, unary len and a binary relation symbol sameKey.
- $\Upsilon = \Upsilon_{log} \cup \Upsilon_{exp} \cup \Upsilon_{const} \cup \Upsilon_{fun}$.

For experiment-representing terms the vocabulary $\Upsilon_{exp} \cup \Upsilon_{const} \cup \{\mathsf{undef}\}$ will suffice, together with some set of additional constants to denote some coins.

Definition 2. Let C be a set of constants. The set of *experiment-representing terms*, in short e-terms, of vocabulary $\Upsilon_{const} \cup \Upsilon_{exp} \cup \{\mathsf{undef}\}$ over C, is defined inductively as:

- nullary symbols in Υ_{const} and undef are e-terms;
- if n-ary symbol $f \in \Upsilon_{exp}$ is not marked as probabilistic and $t_1, \ldots, t_n$ are e-terms, then $f(t_1, \ldots, t_n)$ is an e-term; and
- if n-ary symbol $f \in \Upsilon_{exp}$ is marked as probabilistic, $t_1, \ldots, t_{n-1}$ are e-terms and $c \in C$, then $f(t_1, \ldots, t_{n-1}, c)$ is an e-term.

Given an assignment of infinite strings to constants in C and a concrete value of security parameter η, we can assign a concrete string to every e-term.

Definition 3. Let t be an experiment-representing term of vocabulary $\Upsilon_{const} \cup \Upsilon_{exp} \cup \{\mathsf{undef}\}$ over C, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ an encryption scheme, $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ a pairing scheme, $\mathcal{N}$ a nonce generation algorithm, and σ an assignment of infinite strings to constants in C. Then a string $[\![t]\!]_{\eta,\sigma}^{\Pi,\Sigma,\mathcal{N}}$ is defined inductively as follows (when $\Pi, \Sigma, \mathcal{N}$ are known, we drop them from the notation):

- undef is interpreted as the failure string

$$[\![\mathsf{undef}]\!]_{\eta,\sigma} = \bot$$

- if g is neither marked as probabilistic nor marked as parameterized, then

$$[\![g(t_1, \ldots, t_n)]\!]_{\eta,\sigma} = \mathcal{G}\left([\![t_1]\!]_{\eta,\sigma}, \ldots, [\![t_n]\!]_{\eta,\sigma}\right)$$

- if g is marked as probabilistic but not as parameterized, then

$$[\![g(t_1, \ldots, t_{n-1}, c)]\!]_{\eta,\sigma} = \mathcal{G}\left([\![t_1]\!]_{\eta,\sigma}, \ldots, [\![t_{n-1}]\!]_{\eta,\sigma}, \sigma(c)\right)$$

– if g is marked as both probabilistic and parameterized, then

$$\llbracket g(t_1, \ldots, t_{n-1}, c) \rrbracket_{\eta,\sigma} = \mathcal{G}\left(\eta, \llbracket t_1 \rrbracket_{\eta,\sigma}, \ldots, \llbracket t_{n-1} \rrbracket_{\eta,\sigma}, \sigma(c)\right)$$

for every $(g, \mathcal{G}) \in \{(\mathsf{key}, \mathcal{K}), (\mathsf{encrypt}, \mathcal{E}), (\mathsf{decrypt}, \mathcal{D}), (\mathsf{pair}, \mathcal{P}), (\mathsf{fst}, \mathcal{F}), (\mathsf{snd}, \mathcal{S}), (\mathsf{nonce}, \mathcal{N})\}$ and every $c \in C$.

Thus taking any e-term t, sampling for σ from the uniform distribution we obtain a probability distribution $\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \llbracket t \rrbracket_{\eta,\sigma}\right]$; varying η we obtain an ensemble.

The assumptions on the encryption scheme force that

$$\Pr\left[c, c' \xleftarrow{\$} \mathcal{U} : \mathcal{D}(\mathcal{K}(\eta, c), \mathcal{E}(\mathcal{I}(\mathcal{K}(\eta, c)), m, c')) = m\right] = 1$$

must hold for any message string m, while the confusion-freeness assumption forces

$$\Pr\left[c, c', c'' \xleftarrow{\$} \mathcal{U} : \mathcal{D}(\mathcal{K}(\eta, c), \mathcal{E}(\mathcal{I}(\mathcal{K}(\eta, c')), m, c'')) = \perp\right]$$

to be overwhelming for every message string m. Similar equivalences are forced by assumptions on the pairing function and projections.

We show that these equivalences carry over to formalization by e-terms, for instance that

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \llbracket \mathsf{decrypt}(\mathsf{key}(c_1), \mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c_1)), t, c_2)) \rrbracket_{\eta,\sigma} = \llbracket t \rrbracket_{\eta,\sigma}\right]$$

must be overwhelming for every e-term t.

Definition 4 (Equivalence of E-Terms). Let T be a set of e-terms of vocabulary $\Upsilon_{const} \cup \Upsilon_{exp} \cup \{\mathsf{undef}\}$ over C. Then $\doteq$ is the smallest equivalence over T induced by the clauses

– for every pairwise distinct $c_1, c_2, c_3 \in C$ and every e-term t

$$\mathsf{decrypt}(t_k, t_e) = \begin{cases} t_m & \text{if } t_k = \mathsf{key}(c_k) \wedge t_e \doteq \mathsf{encrypt}(\mathsf{inv}(t_k), t_m, c_e) \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

– for all e-terms t_1, t_2, t, where $t \neq \mathsf{pair}(x, y)$ for all x, y, then

$$\mathsf{fst}(t) = \begin{cases} t_f & \text{if } t \doteq \mathsf{pair}(t_f, t_s) \\ \mathsf{undef} & \text{otherwise} \end{cases} \quad \mathsf{snd}(t) = \begin{cases} t_s & \text{if } t \doteq \mathsf{pair}(t_f, t_s) \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

Let $[t]_{\doteq}$ be the standard notation for the class of $\doteq$ equivalent terms. The above definition justifies the common representation of cryptographic messages with terms without $\mathsf{decrypt}, \mathsf{fst}$ and snd symbols:

Corollary 2. *For every e-term t there is an e-term t_0 in which $\mathsf{decrypt}, \mathsf{fst}$ and snd do not occur and $t_0 \doteq t$.*

Finally, we will show that the equivalence just introduced is justified by its computational interpretation.

Lemma 1. *Let t_1, t_2 be experiment-representing terms of vocabulary $\Upsilon_{const} \cup \Upsilon_{exp} \cup \{\mathsf{undef}\}$ over C, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ a confusion-free encryption scheme, $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ a pairing scheme, $\mathcal{N}$ a nonce generation algorithm. If $t_1 \doteq t_2$, then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![t_1]\!]_{\eta, \sigma} = [\![t_2]\!]_{\eta, \sigma}\right]$$

is overwhelming in η.

Proof. The proof is by induction on the definition of equivalence $\doteq$. Use the assumptions on Π, Σ, the confusion freeness property of Π and the fact that negligible functions are closed under addition. □

Remark 3. A corresponding statement can be made in the case of weak key authenticity. Under this assumption, the statement of the above lemma becomes

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![t_1]\!]_{\eta, \sigma} = [\![t_2]\!]_{\eta, \sigma}\right]$$

is not negligible in η. The proof is (almost) the same.

We shall often assume the following properties of encryption schemes:

$$\Pr\left[c_1, c_2 \xleftarrow{\$} \mathcal{U} : \mathcal{K}(c_1) = \mathcal{K}(c_2)\right]$$
$$\Pr\left[c_k, c_1, c_2 \xleftarrow{\$} \mathcal{U} : \mathcal{E}(\mathcal{K}(\mathcal{I}(c_k)), m, c_1) = \mathcal{E}(\mathcal{K}(\mathcal{I}(c_k)), m, c_2)\right]$$

are both negligible in η. We shall name these properties, which easily follow from the usual security notions such as IND-CPA, but are themselves much weaker, as "random keys" and "random encryption" properties.

Lemma 2. *Let Π be a confusion-free encryption scheme with random keys and random encryption properties, Σ a pairing scheme, $\mathcal{N}$ a secure nonce generation algorithm and t_1, t_2 e-terms. If $[t_1]_{\doteq} \neq [t_2]_{\doteq}$ then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \ [\![t_1]\!]_{\eta, \sigma}^{\Pi, \Sigma} = [\![t_2]\!]_{\eta, \sigma}^{\Pi, \Sigma}\right]$$

is negligible in η.

Proof. By Lemma 1 and Corollary 2, it suffices to show that

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \ [\![t_1]\!] = [\![t_2]\!]\right]$$

is negligible for terms t_1 and t_2 in which $\mathsf{decrypt}, \mathsf{fst}$ and snd do not occur. The rest of the proof is straightforward simultaneous induction on the structure of construction terms t_1 and t_2. □

3.3 Experiments as Structures

Given a set of abstract representatives of coins to interprete constants from C, we can organize the e-terms modulo $\doteq$ to a first-order structure. What it buys us is possibility to harness the well-developed theory of interactive algorithms of [BG04a,BG04b], which operate over such structures as their states.

If elements of the structure are essentially equivalence classes of e-terms, and $\doteq$ is closed under substitution, the interpretation of any function g in the vocabulary $\Upsilon_{const} \cup \Upsilon_{exp} \cup \{\mathsf{undef}\}$ is naturally defined as

$$g([t_1]_{\doteq}, \ldots, [t_n]_{\doteq}) = [g(t_1, \ldots, t_n)]_{\doteq}$$

The logical part of the structure is defined in the usual way. Some additional relations are added to the interpretation, reflecting the assumptions on the tagging scheme $\mathcal{T}$, holding in codomains of functions $\mathsf{key}, \mathsf{encrypt}, \mathsf{pair}$ and nonce.

We proceed with a verbose definition of an isomorphism-closed classes of structures $\mathcal{K}$.

Definition 5. Let $\mathcal{K}$ be an isomorphism closed class of Υ–structures such that $X \in \mathcal{K}$ if and only if there is a uniquely defined set Coins_X such that:

- $\mathsf{true}, \mathsf{false}$ and undef denote distinct elements; elements in domains and codomains of all logical constants except equality are *logical elements* in X; the interpretations of logical connectives in Υ_{log} are the usual ones, and
- each $k \in \Upsilon_{const}$ denotes a unique non-logical element, we denote the set of such elements Const;
- domains and codomains of functions in Υ_{const} and Υ_{exp}, and the set Coins_X contain non-logical elements only;
- the non-logical part of X is freely generated with functions $\mathsf{key}, \mathsf{inv}, \mathsf{encrypt}, \mathsf{pair}$ and nonce from $\mathsf{Coins}_X \cup \mathsf{Const}$;
- $\mathsf{PriKey}, \mathsf{PubKey}, \mathsf{Ciphertext}, \mathsf{Pair}$ and Nonce hold on codomains of functions $\mathsf{key}, \mathsf{inv}, \mathsf{encrypt}, \mathsf{pair}$ and nonce respectively,
- sets $\mathsf{Coins}_X, \mathsf{Const}, \mathsf{PriKey}, \mathsf{PubKey}, \mathsf{Ciphertext}, \mathsf{Pair}, \mathsf{Nonce}$ are pairwise disjoint, and we define $\mathsf{Msg} = \mathsf{Const} \cup \mathsf{PriKey} \cup \mathsf{PubKey} \cup \mathsf{Ciphertext} \cup \mathsf{Pair} \cup \mathsf{Nonce}$;
- functions $\mathsf{key}, \mathsf{inv}, \mathsf{encrypt}, \mathsf{pair}, \mathsf{nonce}$ are injective, with the domains Coins_X, $\mathsf{PriKey}, \mathsf{PubKey} \times \mathsf{Msg} \times \mathsf{Coins}_X, \mathsf{Msg} \times \mathsf{Msg}, \mathsf{Coins}_X$, respectively;
- $\mathsf{decrypt}, \mathsf{fst}$ and snd are defined as

$$
\begin{aligned}
\mathsf{fst}(\mathsf{pair}(m_1, m_2)) &= m_1 \\
\mathsf{snd}(\mathsf{pair}(m_1, m_2)) &= m_2 \\
\mathsf{decrypt}(\mathsf{key}(c_1), \mathsf{encrypt}(\mathsf{key}(\mathsf{inv}(c_1)), m, c_2)) &= m
\end{aligned}
$$

for every $m, m_1, m_2 \in \mathsf{Msg}$ and every $c_1, c_2 \in \mathsf{Coins}_X$; elsewhere these functions take the value undef;

– function len assigns an integer to each $m \in$ Msg such that, assuming $\mathsf{len}(m_1) = \mathsf{len}(m'_1), \mathsf{len}(m_2) = \mathsf{len}(m'_2)$, we have:

$$\mathsf{len}(\mathsf{pair}(m_1, m_2)) = \mathsf{len}(\mathsf{pair}(m'_1, m'_2))$$
$$\mathsf{len}(\mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c_1)), m_1, c_2)) = \mathsf{len}(\mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c'_1)), m'_1, c'_2))$$
$$\mathsf{len}(m_1) + \mathsf{len}(m_2) \leq \mathsf{len}(\mathsf{pair}(m_1, m_2))$$
$$\mathsf{len}(m_1) \leq \mathsf{len}(\mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c_1)), m_1, c_2));$$

for every $m_1, m'_1, m_2, m'_2 \in$ Msg and every $c_1, c'_1, c_2, c'_2 \in$ Coins$_X$;if the argument is not in Msg, len takes the value undef.
– relation sameKey holds in e_1, e_2 iff $e_1 = \mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c)), m_1, c_1)$ and $e_2 = \mathsf{encrypt}(\mathsf{inv}(\mathsf{key}(c)), m_2, c_2)$ for some $c, c_1, c_2 \in$ Coins$_X$, $m_1, m_2 \in$ Msg.

Defining the structure, we have used e-terms with set of constants $C = $ Coins$_X$.

What exactly is the relation of e-terms and structures just defined? Elements of Coins$_X$ are not accessible by ground terms in a structure $X \in \mathcal{K}$, and therefor e-terms cannot be directly evaluated in X. But if we expand the structure X with constant symbols denoting Coins$_X$, then non-logical elements can be seen as a class of $\doteq$ equivalent terms. For $X \in \mathcal{K}$, we will denote with X^+ its unique expansion with constants Coins$_X$ denoting themselves in X^+. Since the non-logical part of X is freely generated by key, inv, encrypt, pair, nonce from Const $\cup$ Coins$_X$, there is a unique ground term t_x^X of vocabulary $\{$key, inv, encrypt, pair, nonce$\} \cup \Upsilon_{const} \cup$ Coins$_X$ denoting every non-logical x in X^+. Denote with T_x^X the set of all ground terms denoting x in X^+. Then T_x^X is exactly $[t_x^X]_{\doteq}$. This reading of the definition allows us to attach the computational interpretation to elements of structures as well.

Definition 6. *Let $X \in \mathcal{K}$, x a non-logical element in X and σ an assignment of infinite strings to Coins$_X$. Then*

$$[\![x]\!]_{X,\eta,\sigma} = [\![t_x^X]\!]_{\eta,\sigma}.$$

If any of the parameters is determined by the context, we might suppress it and ultimately write $[\![t]\!]$ and $[\![x]\!]$ if all parameters are understood from the context.

By Lemma 1 and Lemma 2, both equality and inequality on non-logical part are preserved with overwhelming probability. If we fix some distinct coding of the logical elements, then we can extend the computational interpretation to all elements of the structure. The abstract interpretation will be preserved with overwhelming probability by the computational representation.

Corollary 3. *Let $\Pi = (\mathcal{K}, \mathcal{I}, \mathcal{E}, \mathcal{D})$ be a confusion-free encryption scheme with random keys and random encryption properties, $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ a pairing scheme, $\mathcal{N}$ a secure nonce generation algorithm, $X \in \mathcal{K}$ and t_1, t_2 terms of X^+. Then $Val(t_1, X^+) = Val(t_2, X^+)$ if and only if $[\![t_1]\!]_{\eta,\sigma} = [\![t_2]\!]_{\eta,\sigma}$ with overwhelming probability.*

3.4 Experiments and Algorithms

If we wanted to capture full static logic of asymptotic computational cryptography, we would need much more involved logical constructions. But full static logic is not what we are after, capturing equality and inequality suffices for our purposes. Equality and inequality, which means similarity, suffices to determine the behavior of abstract interactive algorithms of [BG04a,BG04b].

Under the computational interpretation, concrete PPT Turing machines operating on concrete cryptographic messages can simulate abstract algorithms operating over structures representing such messages. A concrete PPT Turing Machine, run on a tape containing a finite set of cryptographic messages, can analyze the messages by running deterministic algorithms such as decryption $\mathcal{D}$ and projections of pairs $\mathcal{F}$ and $\mathcal{S}$, testing parts of analyzed messages for equality etc. It can also create new messages by running probabilistic key and nonce generation algorithms $\mathcal{K}$ and $\mathcal{N}$, encryption algorithm $\mathcal{E}$, or deterministic algorithms such as the pairing algorithm $\mathcal{P}$.

The fact that concrete PPT Turing machines can do essentially no more than the abstract algorithms will be forced by security assumptions on the encryption schemes.

Abstract algorithms represent all possible internal actions of an algorithm with evaluation of terms, and external actions, such as receiving of input messages, with an answer function attached to a state. Internal memory of the abstract algorithm will be modeled with additional functions expanding the structures. The modeling choices we just made are quite obvious and sufficient for everything but coin flipping, e.g. creation of fresh nonces, encryptions etc.

The behavioral theory of algorithms has a well developed theory of importing of fresh objects. Almost every non-trivial application of the theory use importing over a background structure. There is nothing fundamentally different in extending the working space of an algorithm with a fresh atom used to build hereditarily finite sets, or with a fresh atom representing a fresh coin flip used for probabilistic functions. Only atoms that are not used in any meaningful way in the state can be imported, and the exact choice of the atom imported is irrelevant since they all produce isomorphic states.

The isomorphism-closed class of structures $\mathcal{K}$ is a background classes with $Atoms(X) = \mathsf{Coins}_X$ for every $X \in \mathcal{K}$. We will denote it by BC_{CPA}.

Let A be an ordinary interactive small–step algorithm with background BC_{CPA}. In every state X, A evaluates a finite set of terms, possibly using results of interaction with its environment α, and finally, based of the result of the evaluation, generates an update set $\Delta_A^+(X, \alpha)$. We will make no limitations on coins that can be imported by α, except the usual one that an imported coin must be a reserve atom.

Definition 7. *Let A be an ordinary interactive small-step algorithm with background BC_{CPA} and X its state. Let Π an encryption scheme, Σ a pairing scheme and $\mathcal{N}$ a nonce generation algorithm. Then*

- $[\![X]\!]_{\eta,\sigma}$ *is a concatenation of strings $[\![x]\!]_{\eta,\sigma}$ for all accessible $x \in X$.*

– $[\![A]\!]_\eta$ *is a Turing machine that evaluates computational interpretations of abstract terms evaluated by A.*

Example 3. An abstract algorithm modeling the first action of responder B in the Needham–Schroeder protocol:

$$A \xrightarrow[\{n_A, n_B\}_{k_A}]{\{k_A, n_A\}_{k_B}} B$$

is given with the following ASM program:

```
let  p = decrypt(B, in), k_A = fst(p),  n_A = snd(p)  in
   if  PubKey(k_A) and Nonce(n_A)
   then
      import c_1, c_2 in
         let  n_B = nonce(c_1)  in
            a := k_A
            n := n_A
            m := n_B
            out(encrypt(k_A, pair(n_A, n_B), c_2))
```

The program is executed in a state X with background BC_{CPA} and a context α. State X contains a constant B denoting the private key K_B, and undefined constants n, m, and a. Context α is

$$\alpha = \{(c_1, c_1), (c_2, c_2), (in, e_1), (out[e_2], ack)\}$$

for some $c_1, c_2 \in Reserve(X)$ and encryptions $e_1, e_2 \in X$. Queries in and out are used for communicating for the environment, while c_1 and c_2 represent internal coin flips made by the algorithm—environment replies to queries c_1, c_2.

The corresponding Turing machine $[\![A]\!]_\eta$ operates on three tapes. The first tape represents the internal memory of the algorithm and contains the string $[\![K_B]\!]_{\eta,\sigma}$. The second one represents internal randomness needed by the algorithm, it is an infinite sequence of random bits. The third tape represents interaction with an environment, containing $[\![e_1]\!]_{\eta,\sigma}$ at beginning and $[\![e_2]\!]_{\eta,\sigma}$ at the end of the calculation.

The interpretation of actions of an abstract algorithm with experiments deserve some additional attention. The (abstract) work performed by an algorithm is measured in ground terms it evaluates. Evaluation of a term is inductively defined as:

1. interpretation of a background function,
2. interpretation of a foreground function, and
3. querying and receiving an answer from the environment.

Work performed in (1) amounts to evaluation of the appropriate function $\mathcal{G}$ represented by a background function g. Foreground functions represent the internal memory of an algorithm, and therefore (2) is usually a simple memory lookup. Work performed by the environment (3) is not done by the algorithm, as it simply creates queries and uses the answers provided. This view has the following simple consequence when A is instantiated with a concrete implementation working on strings:

Corollary 4. *Let A be a small–step algorithm with background BC_{CPA}, Π an encryption scheme, and Σ an pairing scheme. Then $[\![A]\!]_\eta$ is a PPT Turing machine.*

As already said, we will assume that every algorithm is capable of performing experiments on its own, and therefore we will pose no restriction on importing of fresh coins from the reserve of a state. However, algorithms might also receive, within a step, results of experiments from its environment, such as answers obtained by oracles. Specific modeling circumstances, such as a specific notion of security, will determine our restrictions on such answer functions.

Theorem 4 (Completeness). *Let $\Pi = (\mathcal{K}, \mathcal{I}, \mathcal{E}, \mathcal{D})$ be a confusion-free encryption scheme with random keys and random encryptions, $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ a pairing scheme and $\mathcal{N}$ a secure nonce generation algorithm. Furthermore, assume that for all but a finite number of values of η:*

- *$\mathsf{len}(x) = \mathsf{len}(y)$ in X iff $|[\![x]\!]_{\eta,\sigma}| = |[\![y]\!]_{\eta,\sigma}|$ for every state X with background BC_{CPA};*
- *encryption scheme Π is equipped with a PPT algorithm that can distinguish two encryptions created with different keys.*

Let X and Y be states with background BC_{CPA}. If $X \not\sim Y$ then there is a PPT algorithm distinguishing $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ with overwhelming probability.

Proof. If $X \not\sim Y$, then there are ground terms t_1 and t_2 such that $Val(t_1, X) = Val(t_2, X)$ and $Val(t_1, Y) \neq Val(t_2, Y)$. By Corollary 3, the equality and inequality of terms is preserved with overwhelming probability by the computational interpretation. Let A be an algorithm outputting true when $t_1 = t_2$ and false otherwise. Then $[\![A]\!]_\eta$ distinguishes $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ with overwhelming probability. By Corollary 4, $[\![A]\!]_\eta$ is a PPT algorithm. □

3.5 Abstract Notions of Security

Abstract interactive algorithms of [BG04a,BG04b] allow us to model the oracle adversary games defining cryptographic security notions directly on the abstract level!

The security of an encryption scheme is completely characterized with the corresponding oracles attached to a PPT algorithm trying to break the security of an encryption scheme, as described in section 2.2. The oracles attached

to the algorithm also perform experiments, and the results of the experiments are already representable by elements of the base set of the background classes BC_{CPA}. From the algorithm's point of view, actions of oracles are actions of its environment. Interaction of an algorithm and environment is well studied in the behavioral theory of algorithm. It is represented with a collection of answer functions attached to a state completely characterizing all possible reactions of environment.

We will show how to view definitions of the notions of security from section 2.2 abstractly as a set of abstract answer functions attached to a state of an algorithm. Recall the IND-CPA notion of security given by Definition 2. Its abstract representation is as follows, minding that $\mathsf{n}, \mathsf{pk}, \mathsf{eo}(x, y)$ are *queries* asking the environment for a new nonce, a new private key, an encryption of one of the two messages respecitvely:

Definition 8. Let X be a state with background BC_{CPA}. Then

- a context α of an ordinary interactive small–step algorithm A in X is *IND-CPA good* if there are distinct elements $c^k, c_1, \ldots, c_n, c_1^e, \ldots, c_k^e$ in the reserve of X such that

$$\alpha(\mathsf{n}_i) = c_i$$
$$\alpha(\mathsf{pk}) = \mathsf{inv}_X(\mathsf{key}_X(c^k))$$
$$\alpha(\mathsf{eo}_j(x_j, y_j)) = \begin{cases} \mathsf{encrypt}_X(\mathsf{inv}_X(\mathsf{key}_X(c^k)), x_j, c_j^e) & \text{if } \mathsf{len}_X(x_j) = \mathsf{len}_X(y_j) \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

for some $x_j, y_j \in X$, $i = 1, \ldots, n$, $j = 1, \ldots, k$; and
- a context β of an ordinary interactive small–step algorithm A in X is *IND-CPA fake* if there are distinct elements $c^k, c_1, \ldots, c_n, c_1^e, \ldots, c_k^e$ in the reserve of X such that

$$\alpha(\mathsf{n}_i) = c_i$$
$$\alpha(\mathsf{pk}) = \mathsf{inv}_X(\mathsf{key}_X(c^k))$$
$$\alpha(\mathsf{eo}_j(x_j, y_j)) = \begin{cases} \mathsf{encrypt}_X(\mathsf{inv}_X(\mathsf{key}_X(c^k)), y_j, c_j^e) & \text{if } \mathsf{len}_X(x_j) = \mathsf{len}_X(y_j) \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

for some $x_j, y_j \in X$, $i = 1, \ldots, n$, $j = 1, \ldots, k$.

Let $\mathcal{A}$ be the set of all IND-CPA good contexts and $\mathcal{B}$ the set of all IND-CPA fake contexts in state X. Then $\mathcal{A}, \mathcal{B}$ is *the abstract model of IND-CPA oracle interaction* in X.

Instantiations of these answer functions with concrete encryption schemes are exactly the experiments defined with the IND-CPA notion of security in Definition 2.

Both IND-CCA good and fake contexts are extension of IND-CPA good and fake contexts with additional queries and answers representing the decryption oracles

$$\alpha(\mathsf{do}(x_l)) = \beta(\mathsf{do}(x_l)) = \mathsf{decrypt}_X(\mathsf{key}_X(c^k), x_l)$$

for some x_l such that x_l *is not* one of the answers to encrypt queries in α or β, for $l = 1, \ldots, m$.

The abstract model of interaction of IND-CPA notion of security induces an equivalence relation on states in the following way. We say that a small–step algorithm A *reduces* state X to state Y for answer functions α, β if $X = A(\mathbf{0}_X, \alpha)$ and $Y = A(\mathbf{0}_Y, \beta)$. State X is *reducible* to Y for α, β if such a small–step algorithm exists.

Definition 9. Let $\mathcal{A}, \mathcal{B}$ be the abstract model of IND-CPA interaction. Then X is reducible to Y for IND-CPA, denoted with $X \xrightarrow{\text{CPA}} Y$, if X is reducible to Y for some $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$. If both α and β are parameterized with the same oracle encryption key K, then we also write $X \xrightarrow{K} Y$.

The equivalence induced by the reducibility relation for IND-CPA relation, its transitive and symmetric closure, is denoted with $X \overset{\text{CPA}}{=} Y$.

Lemma 3. *Let Π be a confusion-free IND-CPA secure encryption scheme, Σ a pairing scheme and $\mathcal{N}$ a secure nonce generation algorithm. Let X and Y states with background BC_{CPA}. If $X \overset{\text{CPA}}{=} Y$, then $[\![X]\!]_\eta$ is indistinguishable from $[\![Y]\!]_\eta$ by probabilistic polynomial time algorithms with all but negligible probability.*

Proof. Since computational indistinguishability is an equivalence, it is sufficient to show that $[\![X]\!]_\eta \approx [\![Y]\!]_\eta$ when $X \xrightarrow{\text{CPA}} Y$.

We argue by contradiction. Suppose there is a PPT algorithm $\mathcal{A}$ distinguishing $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ with non-negligible probability. We will use this algorithm to distinguish oracles characterizing IND-CPA security.

Denote with α an IND-CPA good context and with β an IND-CPA fake context such that for some algorithm A we have $X = A(\mathbf{0}_X, \alpha)$ and $Y = A(\mathbf{0}_Y, \beta)$. Run the algorithm $[\![A]\!]_\eta$ with an IND-CPA oracle to create $[\![X]\!]_{\eta,\sigma}$ or $[\![Y]\!]_{\eta,\sigma}$, depending on whether you are provided with a good oracle or a fake one. Run $\mathcal{A}$ on the resulting state. If $\mathcal{A}$ can distinguish $[\![X]\!]_{\eta,\sigma}$ from $[\![Y]\!]_{\eta,\sigma}$ with non-negligible probability, then we can break IND-CPA security of the encryption scheme used. $\square$

The above lemma tells us that certain challenges are indistinguishable as a simple consequence of the notion of security. If two inputs of a challenge can be generated by the same abstract algorithm, but using two different oracles, then it is clear that this algorithm, if successful, would break security of the underlying encryption scheme.

Remark 4. Given some enumeration of coins $c_1, c_2, \ldots$ in a state X, we will often use the following notation for elements of X: n_i for $\mathsf{nonce}_X(c_i)$, K_i for $\mathsf{key}_X(c_i)$, k_i for $\mathsf{inv}_X(\mathsf{key}_X(c_i))$, $\langle m_1, m_2 \rangle$ for $\mathsf{pair}_X(m_1, m_2)$ and $\{m\}_{K_j}^i$ for $\mathsf{encrypt}_X(k_j, m, c_i)$. If a state has a single nullary foreground symbol, then we will identify the state with the unique element the symbol is denoting in it. E.g. a state X with f denoting $\mathsf{encrypt}_X(\mathsf{inv}_X(\mathsf{key}_X(c_2)), \mathsf{key}_X(c_1), c_3)$ for some coins $c_1, c_2, c_3 \in Reserve(X)$ is identified with $\{K_1\}_{K_2}^3$.

Example 4. We will show that

$$\{\{K_1\}^4_{K_2}, K_3\}^5_{K_1} \xrightarrow{K_2} \{\{0\}^4_{K_2}, K_3\}^5_{K_1} \xrightarrow{K_1} \{0\}^5_{K_1}$$

for IND-CPA security (assuming that 0 denotes a zero string of an appropriate length). Let A_1 and A_2 be algorithms with programs Π_1 and Π_2:

$$
\begin{aligned}
\Pi_1 \quad = \quad & \texttt{import } \mathsf{c_1, c_3, c_5} \\
& \texttt{let } K_1 = \mathsf{key}(\mathsf{c_1}),\ K_3 = \mathsf{key}(\mathsf{c_3}),\ k_1 = \mathsf{inv}(K_1)\ \texttt{in} \\
& \quad \mathsf{f} := \mathsf{encrypt}(K_1, \mathsf{pair}(\mathsf{eo}(K_1, 0), K_3), \mathsf{c_5}) \\
\Pi_2 \quad = \quad & \texttt{import } \mathsf{c_2, c_3, c_4} \\
& \texttt{let } K_2 = \mathsf{key}(\mathsf{c_2}),\ K_3 = \mathsf{key}(\mathsf{c_3}),\ k_2 = \mathsf{inv}(K_2)\ \texttt{in} \\
& \quad \mathsf{f} := \mathsf{eo}(\mathsf{pair}(\mathsf{encrypt}(k_2, 0, \mathsf{c_4}), K_3), 0)
\end{aligned}
$$

and let IND-CPA positive α_1, α_2 and IND-CPA negative β_1, β_2 be

$$
\begin{aligned}
\alpha_1 &= \{(\mathsf{c_1}, c_1), (\mathsf{c_3}, c_3), (\mathsf{c_5}, c_5), (\mathsf{eo}[K_1, 0], \{K_1\}^4_{K_2})\} \\
\beta_1 &= \{(\mathsf{c_1}, c_1), (\mathsf{c_3}, c_3), (\mathsf{c_5}, c_5), (\mathsf{eo}[K_1, 0], \{0\}^4_{K_2})\} \\
\alpha_2 &= \{(\mathsf{c_2}, c_2), (\mathsf{c_3}, c_3), (\mathsf{c_4}, c_4), (\mathsf{eo}[\langle\{0\}^4_{K_2}, K_3\rangle, 0], \{\langle\{0\}^4_{K_2}, K_3\rangle\}^5_{K_1})\} \\
\beta_2 &= \{(\mathsf{c_2}, c_2), (\mathsf{c_3}, c_3), (\mathsf{c_4}, c_4), (\mathsf{eo}[\langle\{0\}^4_{K_2}, K_3\rangle, 0], \{0\}^5_{K_1})\}
\end{aligned}
$$

Then

$$
\begin{array}{llll}
A_1(0, \alpha_1) & = & \{\{K_1\}^4_{K_2}, K_3\}^5_{K_1} \qquad\qquad & A_1(0, \beta_1) \quad = \quad \{\{0\}^4_{K_2}, K_3\}^5_{K_1} \\
A_2(0, \alpha_2) & = & \{\{0\}^4_{K_2}, K_3\}^5_{K_1} & A_2(0, \beta_2) \quad = \quad \{0\}^5_{K_1}
\end{array}
$$

By Lemma 3, we can conclude that

$$
\begin{aligned}
\mathsf{Adv}(\mathcal{A}) = &\Pr\left[K_1, K_2, K_3 \xleftarrow{\$} \mathcal{K}(\eta); e \xleftarrow{\$} : \mathcal{E}_{k_1}(\mathcal{P}(\mathcal{E}_{k_2}(K_1), K_3)) :\ \mathcal{A}(e) = 1\right] \\
& - \Pr\left[K_1 \xleftarrow{\$} \mathcal{K}(\eta); e \xleftarrow{\$} \mathcal{E}_{k_1}(0^\ell) :\ \mathcal{A}(e) = 1\right]
\end{aligned}
$$

is negligible for every PPT algorithm $\mathcal{A}$ (we use $k_i = \mathcal{I}(K_i)$).

4 Soundness and Completeness of the Abstract Model

4.1 Indistinguishability

If we prove that state X and Y are indistinguishable by small–step algorithms, what have we proved? We hope that than there is no probabilistic polytime algorithm that can distinguish strings produced by experiments encoded by X and Y with all but negligible probability.

For the IND-CPA notion of security, the background class BC_{CPA}, an encryption scheme Π, a pairing scheme Σ and a nonce generation algorithm $\mathcal{N}$, we have three equivalence relations on abstract states with background BC_{CPA} representing experiments

1. computational indistinguishability;
2. abstract indistinguishability; and
3. abstract reducibility.

If $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ are indistinguishable by PPT algorithms, we write $X \overset{\Pi}{\approx} Y$. Computational indistinguishability is the semantical relation on states, defined independently from our formalism in terms of capabilities of probabilistic polynomial time Turing machines.

If X and Y are indistinguishable by small–step algorithms, we write $X \sim Y$. Abstract indistinguishability articulates our intention about what an encryption scheme should achieve. It can also be seen as the power explicitly given to an agent by an encryption scheme: if an agent can distinguish two states with an abstract algorithm, then she can use an instantiation of the program to distinguish instantiations of the states, all with the concrete instantiated encryption scheme. This property is usually called *completeness*, and it can be phrased as "whatever an abstract algorithm can do, a concrete instantiation can do with overwhelming probability as well". The proof given in Theorem 4 is quite straightforward, but it involves some simple reasoning about probabilities. This is necessary, since it relates an abstract relation with semantics defined in terms of PPT algorithms. We get

$$X \not\sim Y \;\Rightarrow\; X \overset{\Pi}{\not\approx} Y,$$

or equivalently

$$X \overset{\Pi}{\approx} Y \;\Rightarrow\; X \sim Y. \tag{1}$$

Abstract reducibility tells us what a concrete PPT algorithm *cannot do* as a direct consequence of the notion of security. The proof given in Lemma 3 is again very simple, it is nothing more than expressing what is the true meaning of a particular notion of security. We get

$$X \overset{\mathrm{CPA}}{=} Y \;\Rightarrow\; X \overset{\Pi}{\approx} Y \tag{2}$$

From equations (1) and (2), we have

$$X \overset{\mathrm{CPA}}{=} Y \;\Rightarrow\; X \overset{\Pi}{\approx} Y \;\Rightarrow\; X \sim Y \tag{3}$$

If we could relate *abstract notions of equivalence* by showing that

$$X \sim Y \;\Rightarrow\; X \overset{\mathrm{CPA}}{=} Y, \tag{4}$$

we could, using (3), conclude that all three notions are equivalent

$$X \sim Y \;\Leftrightarrow\; X \overset{\mathrm{CPA}}{=} Y \;\Leftrightarrow\; X \overset{\Pi}{\approx} Y.$$

The theorem establishing (4) is the essence of the computational soundness of abstract wrt computational cryptography. It is also the most difficult one to prove. However, it is expressed and proved completely in abstract terms, with

no mention of Turing machines and their probabilities to distinguish concrete strings in PPT time.

A state X is said to be in (IND-CPA) *normal form* if every accessible encryption with inaccessible decryption key has zero string (of appropriate length) as a subject.

Let $\mathcal{A}, \mathcal{B}$ be the model of interaction for IND-CPA security in $\mathbf{0}_X$. If an inaccessible key k is not a subject of any used message in a state X, then X can be constructed from $\mathbf{0}_X$ by a small–step algorithm A and some $\alpha \in \mathcal{A}$. If A is run on $\mathbf{0}_X, \beta$ for some $\beta \in \mathcal{B}$, we can produce a state X' in which k encrypts only zeros, like we did in Example 4.

Lemma 4. *Let X be a state with background BC_{CPA} and K a decryption key not occurring as a submessage of any used encryption in X. Then $X \xrightarrow{\mathrm{CPA}} X'$ for some state X' with the same background reduct and the same accessibility of nonces and keys such that*

- *all keys and nonces are accessible with the same terms in both X and X';*
- *key K encrypts only zero strings in X'; and*
- *if encryption key k_1 encrypts decryption key K_2 or nonce n in X', then k_1 encrypts K_2 or n in X as well.*

Now we have everything we need to prove that acyclic states are reducible to normal form.

Lemma 5. *Let X be an acyclic state with background BC_{CPA} and accessible all exposed elements. Then X is IND-CPA reducible to its normal form.*

Proof. Enumerate inaccessible decryption keys such that encryption key k_j does not encrypt K_i if $i \leq j$. Since the state is acyclic, such numeration is possible.

We will reduce X in n steps to a state X_n such that 0 is the only used subject encrypted by an inaccessible encryption key. The proof is by induction on the enumeration of inaccessible decryption keys. Key K_1 is inaccessible and does not occur as a submessage of subject of any used encryption in X. By Lemma 4, then there is a state X_1 such that $X_0 \xrightarrow{\mathrm{CPA}} X_1$, k_1 does not encrypt any decryption key in X_1 and encrypts in X_1 is a subrelation of encrypts in X. Hence, the enumeration of keys in X is good for X_1. Since key k_1 does not encrypts any decryption key in X_1, there is no key in X_1 that encrypts K_2 in X_1. But then K_2 satisfies the condition of Lemma 4 in X_1 and we can make another step of the induction.

In X_n, subject of every undecryptable encryption is zero. Thus X_n is in the normal form. □

An example of a reduction of a (cyclic) state to its normal form is given in Example 4.

Since a normal form is a representative of its similarity class, we have:

Corollary 5. *Let X and Y be acyclic states with background BC_{CPA} and accessible all exposed elements. If $X \sim Y$ then $X \stackrel{\mathrm{CPA}}{=} Y$.*

Theorem 5 (Soundness). *Let Π be an IND-CPA secure encryption scheme, Σ a pairing scheme, $\mathcal{N}$ a nonce generation algorithm, and X and Y acyclic states with background BC_{CPA}. If $X \sim Y$, then $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ are indistinguishable by probabilistic polynomial time algorithms.*

Proof. We will assume that all exposed elements are accessible in both states. If a state contains exposed but inaccessible elements, replace it with a state obtained by undefining all foreground functions on such elements. The resulting state is clearly computationally indistinguishable from the original one, it provides the same information to the intruder. By Corollary 5, we have $X \stackrel{\mathrm{CPA}}{=} Y$. Finally, by Lemma 3, $[\![X]\!]_\eta$ and $[\![Y]\!]_\eta$ are indistinguishable. □

4.2 Accessibility

If an element x is not accessible by a term in X, α for some state X with background BC_{CPA} and α whose codomain is in the reserve of the state, can we conclude that no PPT algorithm can output $[\![x]\!]_\eta$ when run on $[\![X]\!]_\eta$ with non-negligible probability? The similar theorem was proved for an Abadi–Rogaway language in [MW04b]. We will extend the soundness result of the previous subsection to accessibility here.

Lemma 6. *Let X be a state with background BC_{CPA} and $x \in X$ with non-empty support in X: $Sup_X(\{x\}) \neq \emptyset$. Then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \mathcal{A} = [\![x]\!]_{X,\eta,\sigma}\right]$$

is negligible for every PPT algorithm $\mathcal{A}$.

Intuitively, this means that a PPT algorithm cannot guess an independently created key, nonce or encryption if no data is provided to it.

Lemma 7. *Let Π be an IND-CCA secure encryption scheme and X, X' non-isomorphic states with background BC_{CPA} such that $X \xrightarrow{\mathrm{K}} X'$ for IND-CCA contexts α and β. If some PPT algorithm $\mathcal{A}$ can produce $[\![x]\!]_{X,\eta,\sigma}$ with non-negligible probability when run on $[\![X]\!]_{\eta,\sigma}$, then*

- *decryption key K is not as a submessage of x in X (key $\mathsf{inv}_X(K)$ can occur as a submessage or as an encryption key);*
- *there is a term t such that $x = Val(t, \mathbf{0}_X, \alpha)$ and $\mathcal{A}$ can produce $[\![x']\!]_{X',\eta,\sigma}$ with non-negligible probability when run on $[\![X']\!]_{\eta,\sigma}$ for $x' = Val(t, \mathbf{0}_{X'}, \beta)$;*
- *x is accessible in X iff x' is accessible in X'.*

Proof. We argue by contradiction. Suppose that K is a submessage of x in X. All decryption keys are accessible in $\mathbf{0}_X, \alpha$ except K, and every encryption created by encryption oracle in α does not contain K in the subject. Hence we can use IND-CCA decryption oracle and decryption with accessible keys to retrieve $[\![K]\!]_{X,\eta,\sigma}$ from $[\![x]\!]_{X,\eta,\sigma}$. But then we can distinguish $[\![X]\!]_{\eta,\sigma}$ and $[\![X']\!]_{\eta,\sigma}$ with non-negligible probability, which is a contradiction by Lemma 3.

Since K can only occur in x as an encryption key, there is a term t such that $x = Val(t, \alpha)$. Suppose that $\mathcal{A}$ can produce $[\![x']\!]_{X',\eta,\sigma}$ with negligible probability only. Then

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![t]\!]_{\eta,\sigma} = \mathcal{A}([\![X]\!]_{\eta,\sigma})\right] - \Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![t]\!]_{\eta,\sigma} = \mathcal{A}([\![X']\!]_{\eta,\sigma})\right]$$

is non-negligible and can be used to distinguish states X and X', which is a contradiction.

The last part is a simple consequence of X and X' being indistinguishable. $\square$

Lemma 8. *Let Π be an IND-CCA secure encryption scheme and X a state with background BC_{CPA} in the IND-CPA normal form. If x is not accessible in X, then $[\![x]\!]_{X,\eta,\sigma}$ is not accessible with non-negligible probability to PPT algorithms.*

Proof. If x is not accessible in X, then some inaccessible key K must occur as a submessage in x. Use IND-CCA decryption oracle and keys not used in X to retrieve it and break the IND-CCA security. $\square$

Theorem 6. *Let Π be an IND-CCA secure encryption scheme and X a state reducible to its normal form. If $x \in X$ is not accessible in X to a small-step importing algorithm, then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : \mathcal{A}([\![X]\!]_{\eta,\sigma}) = [\![x]\!]_{X,\eta,\sigma}\right]$$

is negligible for every PPT algorithm $\mathcal{A}$.

Proof. Let X_n be the normal form of X and

$$X \xrightarrow{K_1} X_1 \xrightarrow{K_2} X_2 \xrightarrow{K_3} \ldots \xrightarrow{K_n} X_n$$

for some inaccessible decryption keys $K_1, \ldots, K_n$. Let $x, x_1, x_2, \ldots, x_n$ be elements from Lemma 7. Then x_n is inaccessible in X_n. By Lemma 8, $[\![x_n]\!]_{X_n,\eta,\sigma}$ is not accessible by PPT algorithms with non-negligible probability, which is a contradiction by Lemma 7.

5   Model–based Testing of Protocols

In this section we will show how to encode ASM programs working over BC_{CPA} in Spec# and how to use SpecExplorer to explore all possible execution traces for a bounded number of roles and agents.

The interactive algorithms of [BG04a,BG04b] are implemented in AsmL and Spec#. SpecExplorer is a tool developed at Microsoft Research for exhaustive exploration of (finitized) state spaces of specifications written in AsmL or Spec#, in order to test an implementation for conformance and to generate unit tests.

We have found that it can be effectively used to explore state spaces of protocol adversary situations, given a finite number of roles ensuring that the state space is finite. Our analysis shows that this exploration also has direct computational significance.

```
class Coins {}

structure Message{
  public virtual int len(){ return 1; }
  case Nonce{
    private Coins c;
  }
  case Pair{
    public Message fst;
    public Message snd;
    public override int len(){ return fst.len() + snd.len(); }
  }
  case PrivateKey{
    private Coins c;
    public PublicKey inv(){ return PublicKey(this); }
  }
  case PublicKey{
    private PrivateKey sk;
  }
  case Encryption{
    public PublicKey pk;
    private Message subject;
    private Coins c;
    public Message decrypt(PrivateKey sk)
      require sk.inv() == pk;
    {
      return subject;
    }
    public bool sameKey(Encryption e){ return pk == e.pk; }
    public override int len(){ return subject.len() + 1; }
  }
  public PrivateKey key(){ return PrivateKey(new Coins()); }
  public Encryption encrypt(PublicKey pk, Message subject) {
    return Encryption(pk, subject, new Coins());
  }
  public Pair pair(Message f, Message s){ return Pair(f, s); }
}
```

Fig. 1. BC_{CPA} background encoding in Spec#

5.1 Encoding in Spec# and SpecExplorer

The original idea of modeling abstract properties of cryptographic primitives by an object–oriented programming language is from [RRS03]. An encoding of the BC_{CPA} background in Spec# is given in Figure 1. The encoding should be clear to a reader with some basic understanding of accessibility modifiers `private` and `public` in OO programming languages. A more elaborate discussion of the similar encoding in AsmL language can be found in [RRS03].

An honest role will be represented with a Spec# encoding of an ASM program operating over BC_{CPA} background, such as the one given in example 3. On the other hand, the intruder will not be represented explicitly in the model with a concrete program. We will use the exploration capabilities of SpecExplorer to explore all possible execution paths with a given set of honest roles. It is not very difficult to teach SpecExplorer to completely analyze a message created by an honest role, but we have a dramatically different situation when it comes to creating a message that would be accepted by a role, forcing it to make a step and possibly output a fresh message. The set of states of an honest role is closed under isomorphisms and therefore infinite. We will look into a very common class of protocols in which the set of messages that can be created by an intruder and accepted by an honest role is infinite, but representable with a finite set of messages. Every step that a role can make will produce a state isomorphic to one of the states obtained by running the role with one of the representative messages.

A role of a protocol exposes a very simple interface to the outside world. It analyzes an input message and, if certain conditions are satisfy, outputs a message. If conditions are not met, it typically hangs, not producing a new messages regardless of any future inputs. If A is an action with a message m given as input in a state X and an answer function α with only reserve elements in its codomain, then $\tau_A(X, m, \alpha)$ is a state

An action of such role is called *simple* if it checks the type of all submessages of an input message. This means that any encryption must be decrypted, any pair must be analyzed, and type of any nonce and key must be checked. A protocol is simple if all actions of its roles are simple.

Theorem 7. *Let A be a simple action and I an intruder with a finite set of messages accessible by ground terms. Then there is a finite set of messages M accessible to I in X, α such that for every message m' accessible to I in X, β there is $m \in M$ such that*

$$\tau_A(X, m, \alpha) \cong \tau_A(X, m', \beta)$$

Proof. If action A is simple, then atomic support of m' in X $Sup_X(m')$ is finite and bounded. It suffice to import $|Sup_X m'|$ fresh coins from the reserve of X and create all messages of a fixed submessage structure using freshly created coins and known messages. □

We use the above fact, together with theorems 1 and 6, to produce abstract representations of all non-negligible computational traces representable by the BC_{CPA} background for a bounded number of honest roles.

Fix a simple protocol P. Let k be a maximum number of coins in support of any message accepted by an action of an honest role in P. At the initialization phase, SpecExplorer creates a fixed number of honest agents and corrupted keys. In each subsequent step, SpecExplorer imports k fresh coins from the reserve, and creates all messages from fresh coins and already know messages that could potentially be accepted by some honest role. Each role is ran with each such message used as its input by SpecExplorer, thus producing a set of all reachable states in the model. The exact order in which states are explored is non-deterministic, although the tool allows different priorities to be assigned to states. A role accepting an input message, can possibly output a message. The output message gets analyzed by the tool, thus updating the internal memory of the intruder. At the end of each step, the protocol guarantees are checked in new states by the tool. If any of protocol guarantees is not satisfied, the exploration is aborted and a graph with explored states, including the one in which the guarantee is not fulfilled, is rendered by the tool. The trace that resulted with the bad state can be explored and studied using the tool.

One optimization of the exploring process can be achieved by a grouping of states and further exploration of a single state representative of a group. The exploration space can be dramatically reduced for an appropriate grouping, but an optimal grouping is not always easy to find. In our case, isomorphism seems like a good choice of grouping relation on states. Since SpecExplorer does not have a built-in option of grouping of isomorphic states, we use an ad–hoc coding of states resulting in a grouping relation finer than isomorphism, but still significantly reducing the exploration space.

Example 5 (Lowe's attack on Needham–Schroeder protocol). One example of a simple protocol in the above sense is the public–key variant of the Needham–Schroeder authentication protocol. The flaw found by Lowe is easily (re)discovered by SpecExplorer, usually in less than 100 explored states.

The model is initially in a state marked as `Initial` in Figure 2. The first step of initialization is performed by calling `CreateAgentFactory`, which creates an object capable of creating honest agents of the protocol. The next invocation of `CreateAgents(2,1)` creates two honest agents and one corrupted private key. The internal memory of the intruder is enriched with public keys of the honest agents and the private corrupted key. The resulting state is marked with `Agents Created`. The exploration process starts here. SpecExplorer now can create fresh roles of already created honest agents using `CreateInitiatorRole` and `CreateresponderRole`, or run already an created role using `RunRole`. The parameters for `RunRole` are picked up from the finite set of representative of messages using the Theorem 7.

The Lowe's attack on the protocol is found after creating one initiator and one responder role, and then calling `RunRole` four times with the appropriate parameters. In the resulting state, protocol guarantee is violated and the exploration process is terminated. The resulting state is clearly marked and the trace leading to the state is included in its description in the exploration graph in Figure 2.
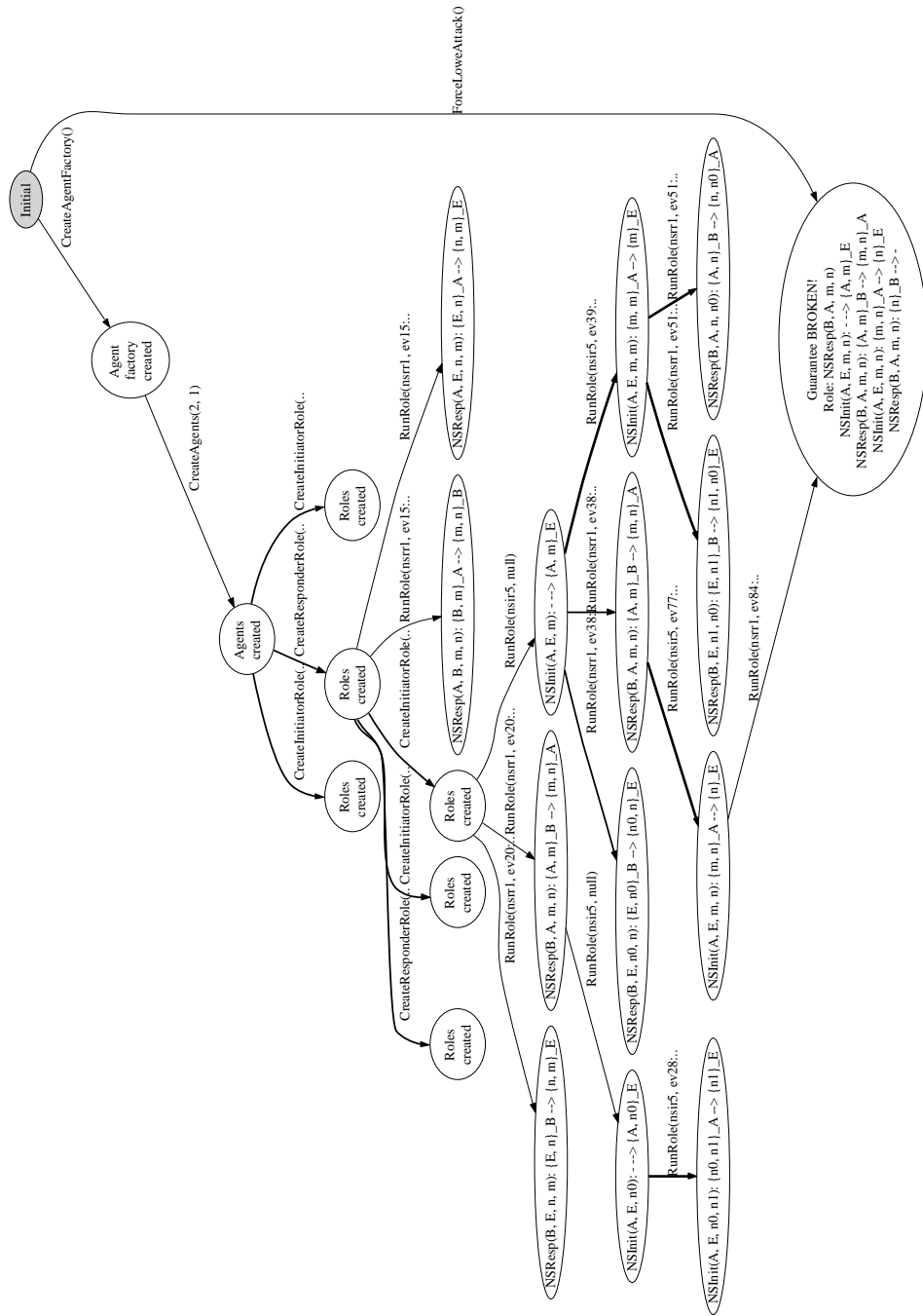
Fig. 2. SpecExplorer discovers Lowe's attack on the Needham–Schroeder protocol

References

[ABS05]   Pedro Adão, Gergei Bana, and Andre Scedrov. Computational and infor-
           mation theoretic soundness and completeness of formal encryption. In *18th
           IEEE Computer Security Foundations Workshop – CSFW 2005*, 2005.

[AJ01]   Martin Abadi and Jan Jürjens. Formal eavesdropping and its computational
           interpretation. In *Theoretical Aspects of Computer Software (4th Interna-
           tional Symposium, TACS '01)*, volume 2215 of *LNCS*, 2001.

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

[Ban04]   Gergei Bana. *Soundness and Completeness of Formal Logics of Symmetric
           Encryption*. PhD thesis, University of Pennsylvania, 2004.

[BDPR98]   M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among
           notions of security for public–key encryption schemes. In *CRYPTO '98*,
           volume 1462 of *LNCS*, 1998.

[BG00]   Andreas Blass and Yuri Gurevich. Background, reserve, and Gandy ma-
           chines. In *Proceedings of CSL '00*, volume 1862 of *LNCS*, 2000.

[BG03]   Andreas Blass and Yuri Gurevich. Algortihms: A quest for absolute def-
           initions. *Bulletin of the European Association for Theoretical Computer
           Science*, (81):195–225, October 2003.

[BG04a]   Andreas Blass and Yuri Gurevich. Ordinary interactive small–step algo-
           rithms I. Technical Report MSR-TR-2004-16, Microsoft Research, 2004.

[BG04b]   Andreas Blass and Yuri Gurevich. Ordinary interactive small–step algo-
           rithms II. Technical Report MSR-TR-2004-88, Microsoft Research, 2004.

[Gur00]   Yuri Gurevich. Sequential abstract state machines capture sequential al-
           gorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July
           2000.

[Gur05]   Yuri Gurevich. Interactive algorithms 2005. Technical Report MSR-TR-
           2005-73, Microsoft Research, 2005.

[HG03]   Omer Horvitz and Virgil Gligor. Weak key authenticity and the computa-
           tional completeness of formal encryption. In *Crypto 2003*, volume 2729 of
           LNCS, 2003.

[MW04a]   Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the
           Abadi-Rogaway language of encrypted expressions. *Journal of Computer
           Security*, 12(1):99–130, 2004.

[MW04b]   Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption
           in the presence of active adversaries. In *Theory of cryptography conference
           - Proceedings of TCC 2004*, volume 2951 of *LNCS*, 2004.

[RR05]   Dean Rosenzweig and Davor Runje. Some things algorithms cannot do.
           Technical Report MSR-TR-2005-52, Microsoft Research, 2005.

[RRS03]   Dean Rosenzweig, Davor Runje, and Neva Slani. Privacy, abstract encryp-
           tion and protocols: an ASM model – Part I. In *ASM 2003*, volume 2589 of
           LNCS. Springer-Verlag, 2003.

[AsmL]   The AsmL webpage. `http://research.microsoft.com/asml/`.

[Spec#]   The Spec# webpage. `http://research.microsoft.com/specsharp/`.

[SpecExp] The SpecExplorer webpage. `http://research.microsoft.com/specexplorer/`.