ASM Model of Computational Cryptography

Dean Rosenzweig and Davor Runje

University of Zagreb

Abstract. We present an ASM model of cryptography based of the recently developed behavioral algorithm theory. The model has two interpretations: an abstract interpretation and a computational interpretation. In the abstract interpretation, agents are ordinary interactive small-step algorithm operating over states with a carefully chosen background. In the computational interpretation, agents are probabilistic polynomial time Turing machines working over strings. The common ASM model allows us to easily relate those two abstraction levels and to prove soundness and completeness of the abstract model with respect to the computational one:

- an attack of a small-step algorithm in the abstract interpretation is, when interpreted computationally, an attack in the computational interpretation;
- a violation of postulates of small-step algorithms in the abstract interpretation would break the security of an underlying encryption scheme in the computational interpretation.

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,BG06,BGa,BGb,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 [BG06,BGa,BGb,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,MW04,AJ01,Ban04] 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 brief summary of the relevant assumptions of asymptotic computational cryptography. Section 2 presents the experiments-as-structures paradigm and our abstract model of cryptographic adversary games. Section 3 contains soundness and completeness proofs, and section 4 shows how the Abadi-Rogaway expression language variant embeds into our framework.

In addition to quoted cryptographic literature, understanding of the framework for establishing negative results for abstract algorithms as presented in [RR05] is expected of the reader. Due to space constraints, we were unable to include it here.

1 Computational model of cryptography

1.1 Syntax of encryption schemes

A symmetric encryption scheme Π is a triple of probabilistic polytime algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where

$$\begin{aligned}\mathcal{K} &: \texttt{Parameter} \times \texttt{Coins} \longrightarrow \texttt{Key}\\ \mathcal{E} &: \texttt{Key} \times \texttt{String} \times \texttt{Coins} \longrightarrow \texttt{Ciphertext} \cup \{\bot\}\\ \mathcal{D} &: \texttt{Key} \times \texttt{String} \longrightarrow \texttt{Plaintext} \cup \{\bot\}\end{aligned}$$

`String` denotes the set of finite strings over $\{0,1\}$. Key generation algorithm $\mathcal{K}$ takes as input a security parameter η typically representing the key length, flips some coins internally, and outputs a *key*. Random coin flips are represented with elements of `Coins`. To avoid talking about sufficient length we will assume `Coins` is the set of all infinite strings over $\{0,1\}$. Encryption algorithm $\mathcal{E}$ takes a key and a string as parameters, also flips some coins, and either outputs an *encryption* or a special symbol $\bot$ denoting its failure. Decryption algorithm $\mathcal{D}$ is deterministic, taking a key and a string as input, and outputting a string or $\bot$ if it fails. The set of strings for which $\mathcal{E}$ does not fail is called *plaintext* and its codomain is called *ciphertext*. We require that $\mathcal{D}_k(\mathcal{E}_k(m)) = m$ for every key k sampled from $\mathcal{K}(\eta)$ and every plaintext m.

We will also assume the existence of functions on strings needed for pairing of strings and appropriate projections on such pairs. A pairing scheme is a triple of deterministic polynomial time algorithms $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$, where $\mathcal{P}$ is a binary pairing function, and $\mathcal{F}$ and $\mathcal{S}$ unary projection. For every two strings x and y, we have $\mathcal{F}(\mathcal{P}(x,y)) = x$ and $\mathcal{S}(\mathcal{P}(x,y)) = y$.

Practical encryption and pairing schemes are often provided with a tagging scheme uniquely determining the identity of the algorithm used to generate the string. We will assume that our encryption scheme and pairing functions use such encoding scheme,

thus providing a mechanism for distinguishing pairs from encryptions, and from other types of messages possibly used. This assumption is not crucial for our analysis, but it will make it simpler.

1.2 Notions of security

Many different notions of security for encryptions are defined and widely used today. They articulate our intention that a ciphertext created with an unknown key should reveal no partial information on the underlying plaintext to an intruder, for a given specific attack model. More precisely, security is defined in terms of capabilities of an arbitrary probabilistic polynomial time algorithm to distinguish the following two types of situations: an algorithm is typically provided with one out of two finite sets of oracles, denoted with $O_1^{\mathcal{A}}, \ldots, O_n^{\mathcal{A}}$ and $O_1^{\mathcal{B}}, \ldots, O_n^{\mathcal{B}}$, and its task is to determine which one it is given to. The sets of oracles completely characterize the notion of security.

Definition 1. *Let IND-SEC be a notion of security characterized by the sets of oracles $O_1^{\mathcal{A}}, \ldots, O_n^{\mathcal{A}}$ and $O_1^{\mathcal{B}}, \ldots, O_n^{\mathcal{B}}$, Π an encryption scheme and A an algorithm. The* advantage $\mathsf{Adv}_{\Pi}^{\text{ind-sec}}(A)$ *is defined when oracles are instantiated with the scheme Π as:*

$$\mathsf{Adv}_{\Pi}^{\text{ind-sec}}(A) = \mathsf{Pr}[A^{O_1^{\mathcal{A}}(\Pi),\ldots,O_n^{\mathcal{A}}(\Pi)} = 1] - \mathsf{Pr}[A^{O_1^{\mathcal{B}}(\Pi),\ldots,O_n^{\mathcal{B}}(\Pi)} = 1]$$

If no probabilistic polytime algorithm A can guess which set of oracles it is provided with with all but negligible probability for the security parameter η:

$$\mathsf{Adv}_{\Pi}^{\text{ind-sec}}(A) \leq \nu(\eta)$$

for all but finite number of η and a negligible function $\nu(\eta)$, then we say the encryption scheme Π is IND-SEC secure.

Different oracles in the definition above will give us different standard notions of security. E.g. type-0, type-1 and type-3 [AR02] advantages are defined as:

$$\mathsf{Adv}_{\Pi}^{\text{type-0}}(A) = \mathsf{Pr}[k_1, k_2 \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_{k_1}(\cdot),\mathcal{E}_{k_2}(\cdot)} = 1] - \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(0),\mathcal{E}_k(0)} = 1]$$
$$\mathsf{Adv}_{\Pi}^{\text{type-1}}(A) = \mathsf{Pr}[k_1, k_2 \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_{k_1}(\cdot),\mathcal{E}_{k_2}(\cdot)} = 1] - \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(0^{|\cdot|}),\mathcal{E}_k(0^{|\cdot|})} = 1]$$
$$\mathsf{Adv}_{\Pi}^{\text{type-3}}(A) = \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(\cdot)} = 1] - \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(0^{|\cdot|})} = 1]$$

Intuitively, this means that a type-0 encryption scheme conceals repetitions of the same subject, message length, and the encryption key. Type-1 security conceals both key and repetitions but possibly not the length of the message, while type-3 only conceals repetitions repetitions but not necessary the length of the message or the encryption key. The notion of type-3 security is equivalent to the widely used IND-CPA notion of security:

$$\mathsf{Adv}_{\Pi}^{\text{ind-cpa}}(A) = \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(LR(\cdot,\cdot,1))} = 1] - \mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(LR(\cdot,\cdot,0))} = 1]$$

where $LR(x, y, b) = \begin{cases} x & \text{if } b = 1 \\ y & \text{otherwise} \end{cases}$

1.3 Confusion freeness and weak key authenticity

Neither the syntax of an encryption scheme nor 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 a ciphertext with independently generated fresh key fails with all but negligible probability (as a function of security parameter η). This property was defined in [MW04] 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.

1.4 Cryptographic protocols

Messages exchanged in cryptographic protocols are created by running PPT algorithms such as algorithms of an encryption scheme, pairing and its projections, etc. Although messages are strings, a run of a protocol can be best described as an *experiment* where strings are obtained in a prescribed way by tossing coins and running the algorithms.

The best way to clarify this point, is an example. Observe the following protocol between two agents, denoted with A and B, sharing a long term secret key k_S and exchanging two freshly generated session keys k_A, k_B, shown in the familiar informal but self-explaining notation:

$$A(k_S) \xrightarrow{\{k_A\}_{k_S}} B(k_S)$$
$$A(k_S) \xleftarrow{\{k_A, k_B\}_{k_S}} B(k_S)$$

The protocol such as this simply describes the way messages are created as a result of an experiment. In this case, the experiment R producing a concrete run of this

protocol $[m_1, m_2]$ is:

$$\begin{aligned}
A_1(k_S) &= [k_A \xleftarrow{\$} \mathcal{K}(\eta); e \xleftarrow{\$} \mathcal{E}_{k_S}(k_A) : k_A, e] \\
B_1(k_S, m) &= [k_A \longleftarrow \mathcal{D}_{k_S}(m); k_B \xleftarrow{\$} \mathcal{K}_\eta; e \xleftarrow{\$} \mathcal{E}_{k_S}(\mathcal{P}(k_A, k_B)) : k_B, e] \\
A_2(k_S, k_A, m) &= [\text{fail if } k_A \neq \mathcal{F}(\mathcal{D}_{k_S}(m)); k_B \longleftarrow \mathcal{S}(\mathcal{D}_{k_S}(m)); k_B] \\
R &= [k_S \xleftarrow{\$} \mathcal{K}(\eta); k_A, m_1 \xleftarrow{\$} A_1(k_S); k_B, m_2 \xleftarrow{\$} B(k_S, m_1); \\
&\quad A_2(k_S, k_A, m_2) : m_1, m_2]
\end{aligned}$$

We have no intention to formalize the above notation, even though it is not particulary difficult to do so. Informally, it suffices to say that an experiment is a sequence of actions delimited with semicolon; if 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.

In the next section, we will develop the idea of representing such experiments with first order structures.

2 Experiments as structures

2.1 Experiments as terms

A nondeterministic function can be seen as a deterministic one with one additional parameter containing all the randomness needed for the calculation. Computable function use only a finite amount of randomness for the calculation, but there might not be a uniform bound for all inputs on the amount of randomness needed. To avoid talking about sufficient lengths, it is convenient to assign infinite strings to such random parameters. Experiments are composed of nondeterministic and deterministic functions, or equivalently, functions assigning a concrete value to the assignment of its random parameters. Performing an experiment amounts to assigning infinite strings from a given distribution to random parameters and simply performing a deterministic calculation.

Definition 2. Probabilistic vocabulary *is a vocabulary in which function symbols can be additionally marked as probabilistic and parameterized. The set of terms of a probabilistic vocabulary over a set* Coins *is defined with the following induction: if f is an n-ary function symbol, $t_1, \ldots t_n$ terms, and $c \in$ Coins, then*

- *$f(t_1, \ldots, t_n)$ is a term if f is not marked as probabilistic,*
- *$f(t_1, \ldots, t_n, c)$ is a term if f is marked as probabilistic.*

We define a probabilistic vocabulary for our model of symmetric cryptography: Υ_0 contains nullary function symbols true, false, undef and key, unary function symbols fst and snd, and binary function symbols pair, encrypt and decrypt. Symbols key and encrypt are marked probabilistic in Υ_0 and key is additionally marked as parameterized. Examples of terms of Υ_0 over $\{c_1, c_2, c_3\}$ are $\mathsf{key}(c_1)$ and $\mathsf{encrypt}(\mathsf{key}(c_1), \mathsf{pair}(\mathsf{key}(c_2), \mathsf{true}), c_3)$.

We will provide a computational interpretation to each term of vocabulary Υ_0:

Definition 3. *Let t be a term of probabilistic vocabulary Υ over set* Coins*, σ a function from* Coins *into infinite strings over $\{0,1\}$, η an integer, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ a symmetric encryption scheme and $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ a pairing function. Then*

– $[\![t]\!]^{\Pi,\Sigma}_{\eta,\sigma}$ *is a finite string inductively defined over a structure of t as follows:*

$$
\begin{array}{lcl}
[\![\mathsf{true}]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & 1 \\
[\![\mathsf{false}]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & 0 \\
[\![\mathsf{undef}]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \perp \\
[\![\mathsf{key}(c)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{K}(\eta, \sigma(c)) \\
[\![\mathsf{encrypt}(t_1, t_2, c)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{E}([\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma}, [\![t_2]\!]^{\Pi,\Sigma}_{\eta,\sigma}, \sigma(c)) \\
[\![\mathsf{decrypt}(t_1, t_2)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{D}([\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma}, [\![t_2]\!]^{\Pi,\Sigma}_{\eta,\sigma}) \\
[\![\mathsf{pair}(t_1, t_2)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{P}([\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma}, [\![t_2]\!]^{\Pi,\Sigma}_{\eta,\sigma}) \\
[\![\mathsf{fst}(t_1)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{F}([\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma}) \\
[\![\mathsf{snd}(t_1)]\!]^{\Pi,\Sigma}_{\eta,\sigma} & = & \mathcal{S}([\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma})
\end{array}
$$

– $[\![t]\!]^{\Pi,\Sigma}_{\eta}$ *is the distribution of finite strings obtained by sampling σ from the uniform distribution and calculating $[\![t]\!]^{\Pi,\Sigma}_{\eta,\sigma}$.*
– $[\![t]\!]^{\Pi,\Sigma}$ *is the ensamble of η parameterized distributions $[\![t]\!]^{\Pi,\Sigma}_{\eta,\sigma}$ (a function associating $[\![t]\!]^{\Pi,\Sigma}_{\eta}$ to each η).*

Notice that we are using $\perp$ to denote a failure.

By the syntax of the encryption and pairing schemes, the computational interpretations of some terms will always yield the same value for the same parameters. E.g. $[\![\mathsf{decrypt}(\mathsf{key}(c_1), \mathsf{encrypt}(\mathsf{key}(c_1), t, c_2))]\!]^{\Pi,\Sigma}_{\eta,\sigma} = [\![t]\!]^{\Pi,\Sigma}_{\eta,\sigma}$ for every term t.

We proceed to definition of an equivalence relation articulating that intuition. Notice that so far it is only our intuition, we will later show that computational interpretations of equivalent terms are computationally equivalent (indistinguishable by PPT algorithms).

Binary relation $\approx$ is the smallest equivalence relation on terms of vocabulary Υ_0 over Coins such that:

– $\mathsf{decrypt}(\mathsf{key}(c_1), \mathsf{encrypt}(\mathsf{key}(c_1), t, c_2)) \approx t$
– $\mathsf{fst}(\mathsf{pair}(t_1, t_2)) \approx t_1$
– $\mathsf{snd}(\mathsf{pair}(t_1, t_2)) \approx t_2$

for every term t, t_1, t_2 and $c_1, c_2 \in$ Coins. Equivalence $\approx$ partitions terms into equivalence classes $[t]_{\approx}$, where $[t]_{\approx} = \{t' | t \approx t'\}$.

Lemma 1. *Let Π be a confusion free encryption scheme, Σ a pairing scheme, t_1 and t_2 terms of Υ_0 vocabulary.*

– *If $t_1 \approx t_2$, then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![t_1]\!]^{\Pi,\Sigma}_{\eta,\sigma} = [\![t_2]\!]^{\Pi,\Sigma}_{\eta,\sigma}\right] \geq 1 - \nu(\eta)$$

for some polynomial function $\nu(\eta)$ for all but a finite number of η.

– *if* $t_1 \not\approx t_2$, *then*

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

for some polynomial function $\nu(\eta)$ *for all but a finite number of* η.

The *minimal term* of equivalence class $[t]_\approx$ is the unique term $t_{min} \in [t]_\approx$ not containing decrypt, fst and snd. Lemma 1 justifies the following definition of computational interpretation of $[t]_\approx$:

Definition 4. *Let* t_{min} *be a minimal term of probabilistic vocabulary* Υ *over set* Coins, σ *a function from* Coins *into infinite strings over* $\{0,1\}$, η *an integer,* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *a symmetric encryption scheme and* $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{S})$ *a pairing function. Then* $[\![[t]_\approx]\!]^{\Pi,\Sigma}_{\eta,\sigma} = [\![t_{min}]\!]^{\Pi,\Sigma}_{\eta,\sigma}$, $[\![[t]_\approx]\!]^{\Pi,\Sigma}_{\eta} = [\![t_{min}]\!]^{\Pi,\Sigma}_{\eta}$ *and* $[\![[t]_\approx]\!]^{\Pi,\Sigma} = [\![t_{min}]\!]^{\Pi,\Sigma}$.

Definition 5. *The structure* $X(\mathsf{Coins})$ *of over* Coins *is defined as follows:*

– *the base set of* $X(\mathsf{Coins})$ *contains* Coins *and all* $\approx$*-equivalence class of terms:*

$$|X| = \mathsf{Coins} \ \cup \ \{[t]_\approx \mid t \text{ is a term of } \Upsilon_0\},$$

– *logical functions are defined in the usual way,*
– *unary relational symbol Atomic holds in every* $c \in \mathsf{Coins}$,
– *the interpretation of every function symbol* $f \in \Upsilon_0$ *in* $X(\mathsf{Coins})$ *is the smallest function such that*

$$f_{X(\mathsf{Coins})}([t_1]_\approx, \ldots, [t_n]_\approx) = [f(t_1, \ldots, t_n)]_\approx.$$

$BC_{\text{TYPE-0}}$ *is the isomorphism closed class of structures containing* $X(\mathsf{Coins})$ *for every set* Coins.

Lemma 2. $BC_{\text{TYPE-0}}$ *is a background class.*

2.2 Experiments and Algorithms

Structures are the natural working environment of abstract algorithms. A background structure representing an abstraction of experiments is the basis of defining algorithm that performs experiment on their natural level of abstraction.

Let A be an ordinary interactive small–step algorithm with background expanding $BC_{\text{TYPE-0}}$. In every state X, A evaluates a finite set of terms, while (possibly) using results of interaction with its environment α in the process, and finally, based of the result of the evaluation, generates an update set $\Delta^+_A(X, \alpha)$. Since every element in X and α represents an experiment, we can naturally extend the intended meaning of experiments to algorithms.

Definition 6. *Let* A *be a small–step algorithm with background* $BC_{\text{TYPE-0}}$, Π *an encryption scheme, and* Σ *an pairing scheme. Then* $[\![A]\!]^{\Pi,\Sigma}_{\eta}$ *is an algorithm that takes as input strings* $[\![x]\!]^{\Pi,\Sigma}_{\eta,\sigma}$ *for all accessible* $x \in X, \alpha$ *and outputs strings* $[\![y]\!]^{\Pi,\Sigma}_{\eta,\sigma}$ *for all accessible* $y \in \tau_A(X, \alpha)$, *for every* σ *sampled from the uniform distribution.*

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 $\mathcal{F}$ represented by a background function f. Foreground functions typically represent internal memory of an algorithm, and therefor (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 1. *Let A be a small–step algorithm with background expanding $BC_{\text{TYPE-0}}$, Π an encryption scheme, and Σ an pairing scheme. Then $[\![A]\!]_\eta^{\Pi,\Sigma}$ is a* PPT *algorithm.*

We will assume that every algorithm is capable of performing experiments on its own, and therefor 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. Specific modelling circumstances, such as a specific notion of security, will determine our restrictions on such answer functions.

2.3 Abstract Notions of Security

The security of an encryption scheme is completely characterized with the corresponding set of oracles attached to a PPT algorithm trying to break the security of an encryption scheme, as described in section 1.1. The oracles attached to the algorithm also perform experiments, and these experiments are already representable by elements of the base set of the background classes introduced. 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. A collection of answer functions is attached to a state completely characterizing possible reactions of environment.

Recall the definition of type–0 security

$$\begin{aligned}\mathsf{Adv}_{\Pi}^{\text{type-0}}(A) = \mathsf{Pr}[k_1, k_2 \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_{k_1}(\cdot),\mathcal{E}_{k_2}(\cdot)} = 1] \\ -\mathsf{Pr}[k \leftarrow \mathcal{K}(\eta) : A^{\mathcal{E}_k(0),\mathcal{E}_k(0)} = 1]\end{aligned}$$

Definition 7. *Let X be a state with background $BC_{\text{TYPE-0}}$. Then*

– *a context α of an ordinary interactive small–step algorithm A in X is* type-0 positive *if there are distinct elements in the reserve of X*

$$c^l, c^r, c_1, \ldots, c_n, c_1^l, \ldots, c_k^l, c_1^r, \ldots, c_k^r$$

and α is the smallest functions such that

$$\begin{aligned}\alpha(\hat{\mathsf{n}}_i) &= c_i \\ \alpha(\hat{\mathsf{ol}}_j(x_j)) &= \mathsf{encrypt}_X(\mathsf{key}_X(c^l), x_j, c_j^l) \\ \alpha(\hat{\mathsf{ol}}_j(y_j)) &= \mathsf{encrypt}_X(\mathsf{key}_X(c^r), y_j, c_j^r)\end{aligned}$$

for some elements $x_1, \ldots, x_n, y_1, \ldots, y_n \in X$, $i = 1, \ldots, n$, $j = 1, \ldots, k$; and

– *a context β of a ordinary interactive small-step algorithm A in X is* type-0 negative *if there are distinct elements in the reserve of X*

$$c, c_1, \ldots, c_n, c_1^l, \ldots, c_k^l, c_1^r, \ldots, c_k^r$$

and β is the smallest functions such that

$$\begin{aligned}\beta(\hat{\mathsf{n}}_i) &= c_i\\ \beta(\hat{\mathsf{ol}}_j(x_j)) &= \mathsf{encrypt}_X(\mathsf{key}_X(c), 0, c_j^l)\\ \beta(\hat{\mathsf{ol}}_j(y_j)) &= \mathsf{encrypt}_X(\mathsf{key}_X(c), 0, c_j^r)\end{aligned}$$

for some elements $x_1, \ldots, x_n, y_1, \ldots, y_n \in X$, $i = 1, \ldots, n$, $j = 1, \ldots, k$.

Let $\mathcal{A}$ be a set of all type-0 positive functions and $\mathcal{B}$ a set of all type-0 negative functions in state X. Then $\mathcal{A}, \mathcal{B}$ is the abstract model of type-0 interaction *in X.*

Instantiations of the answer functions with concrete encryption scheme are exactly the experiments defined with type-0 notion of security.

Abstract models of type-1, type-3 and IND-CPA interactions are defined in the same manner; due to the space constraints, we will not spell them out explicitly here.

The abstract model of interaction of a 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 8. *Let IND-SEC be a notion of security and $\mathcal{A}, \mathcal{B}$ the abstract model of IND-SEC interaction. Then X is reducible to Y for IND-SEC, denoted with $X \overset{\text{IND-SEC}}{\longrightarrow} Y$, if X is reducible to Y for some $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$.*

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

3 Computational Soundness and Completeness of the Abstract Model

If we prove that structures 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 an IND-SEC notion of security, a background class $\mathcal{K}$ expanding $BC_{\text{TYPE-0}}$, and an encryption scheme Π we have three equivalences on abstract states with background $\mathcal{K}$ representing experiments

1. computational indistinguishability;
2. abstract indistinguishability (by small-step algorithms); and
3. abstract reducibility.

If $[\![X]\!]_\eta^\Pi$ and $[\![Y]\!]_\eta^\Pi$ 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 articulate our intention what an encryption scheme should achieve. It can also be seen as the power explicitly given to an agent by the syntax of an encryption scheme: if an agent can distinguish two state with an abstract algorithm, then it can use instantiation of the program to distinguish instantiations of the states, all with the concrete 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 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 is very simple, it is nothing more than expressing what is the true meaning of a particular notion of security. We get

$$X \overset{\text{IND-SEC}}{=} Y \;\Rightarrow\; X \overset{\Pi}{\approx} Y \tag{2}$$

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

$$X \overset{\text{IND-SEC}}{=} Y \;\Rightarrow\; X \overset{\Pi}{\approx} Y \;\Rightarrow\; X \sim Y \tag{3}$$

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

$$X \sim Y \;\Rightarrow\; X \overset{\text{IND-SEC}}{=} Y, \tag{4}$$

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

$$X \sim Y \;\Leftrightarrow\; X \overset{\text{IND-SEC}}{=} Y \;\Leftrightarrow\; X \overset{\Pi}{\approx} Y.$$

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 Turning machines and their probabilities to distinguish concrete strings in PPT time. Our view of abstract cryptography is top-down, rather than bottom-up. Computational notions of security are assumed not to be fixed, and abstract models are not seen as simplified abstractions of computational ones. We think of a choice of the concrete background class, and a resulting notion of abstract indistinguishability, as an expression of our intention of what encryption scheme should achieve. The choice of a concrete computational notion of security is simply our attempt to fulfill that intention.

3.1 Equivalence Induced by Security Notion

Lemma 3. *Let IND-SEC be a notion of security, Π an IND-SEC secure encryption scheme, and states with background $\mathcal{K}$ expanding $BC_{\text{TYPE-0}}$. If X is reducible to Y for IND-SEC, then $[\![X]\!]^{\Pi}_{\eta}$ is indistinguishable from $[\![Y]\!]^{\Pi}_{\eta}$ by probabilistic polynomial time algorithms with all but negligible probability.*

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.

Since the computational indistinguishability is an equivalence relation preserved by the abstract reducibility by Lemma 3, then it is also preserved by the equivalence relation induced by the abstract reducibility:

Corollary 2. *Let IND-SEC be a notion of security, Π an IND-SEC secure encryption scheme, and X and Y states with background $\mathcal{K}$ expanding $BC_{\text{TYPE-0}}$. If $X \overset{\text{IND-SEC}}{=} Y$ then $[\![X]\!]_\eta^\Pi$ and $[\![Y]\!]_\eta^\Pi$ are indistinguishable by probabilistic polytime algorithms.*

3.2 Soundness Theorem

The structural counterpart of the Abadi–Rogaway notion of pattern [AR02] is indistinguishability by small–step algorithms: the *pattern* of a state X is the (isomorphism closed) class of all states indistinguishable from X by small–step algorithms with import. Pattern is an equivalence class, but for many notions of security some states are more appropriate for representing the class than other. More specifically, type-0, type-1 and type-3 notions suggest representing a pattern with a state with a null string as a subject of every used undecryptable message.

A state X with background $BC_{\text{TYPE-0}}$ is *normalized* if all used undecryptable encryptions have 0 as a subject and the same encryption key. Normalized states with background $BC_{\text{TYPE-1}}$ and $BC_{\text{TYPE-3}}$ are defined in the similar way: they both use $|0|^\ell$ as a subject of all undecryptable encryptions, while $BC_{\text{TYPE-3}}$ normalized form exceptionally does not require the same encryption key to be used in undecryptable encryptions.

The first step in establishing equivalence of the notion of indistinguishability and reducibility it to show that reducibility preserves indistinguishability.

Lemma 4. *Let X and Y be states with background $BC_{\text{TYPE-0}}$. If $X \overset{\text{TYPE–0}}{\longrightarrow} Y$, then $X \sim Y$.*

Let $\mathcal{A}, \mathcal{B}$ be the model of interaction for type-0 security in $\mathbf{0}_X$. If an inaccessible key k is 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 would get a state X' where k would not encrypt any key. In Abadi–Rogaway terms, type–0 reduction allows us to substitute all subjects encrypted with k with 0.

Lemma 5. *Let X be a state with background $BC_{\text{TYPE-0}}$ and k an inaccessible and unexposed key in X such that no key k' encrypts k in X. Then $X \overset{\text{TYPE-0}}{\longrightarrow} X'$ for some state X' with the same background reduct and the same key accessibility relation such that*

- *key k does not encrypt any key in X'; and*
- *if key k_1 encrypts key k_2 in X', then k_1 encrypts k_2 in X.*

If all encryptions with inaccessible keys k_1 and k_2 are encryptions of 0, then the type–0 reduction also allows us to substitute all encryptions with k_2 with encryptions with k_1.

Lemma 6. *Let X be a state with background $BC_{\text{TYPE-0}}$ and k_1, k_2 inaccessible and unexposed keys in X such that if k_1 or k_2 encrypts m in X, then $m = 0$. Then $X \overset{\text{TYPE-0}}{\longrightarrow} X'$ for some state X' with the same background reduct and the same key accessibility relation such that*

- *key k_2 is not used in X'; and*
- *key k encrypts key k' in X iff k encrypts k' in X'.*

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

Lemma 7. *Let X be an acyclic state with background $BC_{\text{TYPE-0}}$ and accessible all exposed elements. Then X is type–0 reducible to its normal form.*

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

Corollary 3. *Let X and Y be acyclic states with background $BC_{\text{TYPE-0}}$ and accessible all exposed elements. If $X \sim Y$ then $X \overset{\text{TYPE-0}}{=} Y$.*

Theorem 1 (Computational soundness). *Let Π be type-0 secure encryption scheme and X and Y acyclic states with background $BC_{\text{TYPE-0}}$. If $X \sim Y$, then challenges $[\![X]\!]_\eta^\Pi$ and $[\![Y]\!]_\eta^\Pi$ 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 3, we have $X \overset{\text{TYPE–0}}{=} Y$. Finally, by Corollary 2, $[\![X]\!]_\eta^\Pi$ and $[\![Y]\!]_\eta^\Pi$ are computationally indistinguishable. □

Modify the proof of Lemma 5 by using (term denotation of) $|0|^\ell$ instead of 0 to get proofs for type–1 and type–3 reducibilities when paired with $BC_{\text{TYPE-1}}$ and $BC_{\text{TYPE-3}}$ backgrounds, respectively. Lemma 6 holds for type–1 reducibility and $BC_{\text{TYPE-1}}$ background, when 0 is replaced with $|0|^\ell$ both in the wording and the proof. Use this to prove Lemma 7, and it consequence the computational soundness theorem, for type–1 and type–3 notions of security. Notice that the same lemmas with only slightly modified proofs would work if we used IND-CPA instead of type–3 security, but that is not surprising since

$$\left(\overset{\text{TYPE-0}}{\longrightarrow}\right) \subset \left(\overset{\text{TYPE-0}}{=}\right) = \left(\overset{\text{IND-CPA}}{\longrightarrow}\right) = \left(\overset{\text{IND-CPA}}{=}\right).$$

3.3 Completeness Theorem

The consequence of the syntax of encryption and pairing schemes, together with a tagging scheme and the confusion freeness property, is preservation of equality with overwhelming probability by experiments represented with (expansions of) $BC_{\text{TYPE-0}}$ background, as shown in Lemma 1. Here we show the type-0 encryption schemes also preserve the inequality with overwhelming probability:

Lemma 8. *Let Π be a type-0 encryption scheme, Σ a pairing scheme, X an acyclic state with background $BC_{\text{TYPE-0}}$, and x_1, x_2 exposed elements in X. If $x \neq x_2$, then*

$$\Pr\left[\sigma \xleftarrow{\$} \mathcal{U} : [\![x_1]\!]^{\Pi,\Sigma}_{\eta,\sigma} = [\![x_2]\!]^{\Pi,\Sigma}_{\eta,\sigma}\right] \leq \nu(\eta)$$

for some polynomial function $\nu(\eta)$ for all but a finite number of η.

Theorem 2 (Computational completeness). *Let Π be a type-0 encryption scheme, and X, Y states with the background $BC_{\text{TYPE-0}}$. If $X \overset{\Pi}{\approx} Y$ then $X \sim Y$.*

Proof. We argue by contradiction. Assume $X \not\sim Y$. Then there are terms t_1, t_2 such that

$$Val(t_1, X) = Val(t_2, X), \quad Val(t_1, Y) \neq Val(t_2, Y).$$

By Lemma 1 and Lemma 8, an algorithm computing $[\![t_1]\!]^{\Pi,\Sigma}_{\eta}$ and $[\![t_2]\!]^{\Pi,\Sigma}_{\eta}$ can distinguish $[\![X]\!]^{\Pi,\Sigma}_{\eta}$ from $[\![Y]\!]^{\Pi,\Sigma}_{\eta}$ with overwhelming probability. □

The same construction works for type-1 and type-3 notions of security when paired with $BC_{\text{TYPE-1}}$ and $BC_{\text{TYPE-3}}$ backgrounds. We leave the proofs out of this paper due to the space constraints.

4 Relation with Abadi–Rogaway Languages

Interpretations of our abstract representation of experiments with structures include not just computational interpretations but other abstract models as well. Recall that a background class is closed under isomorphisms. We will show that one concrete structure, with a language of expressions like the ones commonly used in other abstract models as a base set, is in $BC_{\text{TYPE-0}}$. Moreover, the notions of accessibility and indistinguishability on such expressions are equivalent to its counterparts in our model. Such interpretations are important, as they allow us to extended the results proved in the ASM model to other abstract models of cryptography.

A language of expressions in the Abadi-Rogaway style [AR02] is defined over a set of coins Coins with the following grammar:

$$m ::= 0 \mid 1 \mid K_c \mid \langle m_1, m_2 \rangle \mid \{m\}^{c_1}_{K_{c_2}}$$

for $c, c_1, c_2 \in \mathsf{Coins}$.

The notion of accessibility is modeled with a binary *entailment* relation $\vdash$ over such expressions. It is defined as the smallest reflexive relation closed under the following rules:

- $m \vdash 1$ and $m \vdash 0$
- if $m \vdash \langle m_1, m_2 \rangle$ then $m \vdash m_1$ and $m \vdash m_2$
- if $m \vdash \{m_s\}^{c_1}_{K_{c_2}}$ and $m \vdash K_{c_2}$ then $m \vdash m_s$
- if $m \vdash m_1$ and $m \vdash m_2$ then $m \vdash \langle m_1, m_2 \rangle$
- if $m \vdash m_s$ and $m \vdash K(c_2)$ then $m \vdash \{m_s\}^{c_1}_{K_{c_2}}$

It is assumed that coins in practical algorithms are not reused, since that would most certainly break any security. We will request appropriate coin freshness for an expression m:

– if K_c is a subexpression of m, then K_c is an subexpression of any subexpression m' of m in which c occurs; and
– if $\{m_s\}^{c_1}_{K_{c_2}}$ is a subexpression of m, then $\{m_s\}^{c_1}_{K_{c_2}}$ is a subexpression of any subexpression m' of m in which c_1 occurs.

We obtain the language of *patterns* by adding the symbol $\square$ decorated with a coin, intuitively representing an undecryptable encryption, to the language of expressions:

$$p ::= 0 \mid 1 \mid K_c \mid \langle m_1, m_2 \rangle \mid \{m\}^{c_1}_{K_{c_2}} \mid \square^c$$

Each expression is assigned a pattern:

$$\mathit{pattern}(m) = p(m, \{K(c) \mid m \vdash K(c)\})$$

where

$$p(\{m\}^{c_1}_{K(c_2)}, T) = \begin{cases} \{p(m,T)\}^{c_1}_{K(c_2)} & \text{if } K(c_2) \in T \\ \square^{c_1} & \text{otherwise} \end{cases}$$
$$p(\langle m_1, m_2 \rangle, T) = \langle p(m_1, T), p(m_2, T) \rangle$$

and $p(m, T) = m$ for other expressions. Two expressions of the language are *equivalent up to renaming* if there is a bijection on coins σ such that their patterns are equal: $m \cong n$ iff there is a bijection σ such that $\mathit{pattern}(m) = \mathit{pattern}(n\sigma)$.

Let X be the structure with base set $|X|$ containing Abadi–Rogaway expressions enriched with the distinct element undef, and the following functions:

$$\begin{aligned} \mathsf{key}_X(c) &= K_c \\ \mathsf{encrypt}_X(K_{c_2}, m, c_1) &= \{m\}^{c_1}_{K_{c_2}} \\ \mathsf{pair}_X(m_1, m_2) &= \langle m_1, m_2 \rangle \\ \mathsf{decrypt}_X(K_{c_2}, \{m\}^{c_1}_{K_{c_2}}) &= m \\ \mathsf{fst}_X(\langle m_1, m_2 \rangle) &= m_1 \\ \mathsf{snd}_X(\langle m_1, m_2 \rangle) &= m_2 \end{aligned}$$

Everywhere else, the functions are undefined.

It is not difficult to see that X is in $BC_{\text{TYPE-0}}$. Moreover, the notions of accessibility and indistinguishability defined on expressions are matching the corresponding ones on structures. Denote with X_m an expansion of X with an additional nullary foreground symbols in denoting m in it.

Theorem 3. *If m_1 and m_2 are Abadi–Rogaway expressions, then*

– *$m_1 \vdash m_2$ iff m_2 is accessible by creation in X_{m_1},*
– *$m_1 \cong m_2$ iff $X_{m_1} \sim X_{m_2}$.*

Some encryption schemes might reveal additional information about the ciphertext, such as the encryption key or the length. Patterns for such schemes are obtained by "decorating" the square symbol with some additional information. Abadi and Jürjens used this approach first to distinguish two different encryption of the same subject with the same key in the same fashion as we did above [AJ01]. The same idea was used later by Micciancio and Warinschi in [MW04], where the information related to the encryption key and the length was attached to the ciphertext. In his PhD thesis

[Ban04], Bana pushed the principle to its limits by defining an abstract decorating function. Any information could be possibly used to decorate undecryptable "boxes"; e.g. an encryption created with DES algorithm could be marked differently from the ones created with AES algorithm.

The Theorem 3 also holds when an appropriate background class is matched with an appropriate box–decorating function. Background $BC_{\text{TYPE-1}}$ matches the decorating function additionally attaching the length to an encryption: $\Box_{\ell}^{c_1}$ is the pattern of encryption $\{m\}_{K(c_2)}^{c_1}$ of length ℓ. Similarly, $BC_{\text{TYPE-3}}$ matches the function attaching both length and the key used to encryption: $\Box_{\ell,c_2}^{c_1}$ is the pattern of encryption $\{m\}_{K(c_2)}^{c_1}$.

References

[AJ01] Martin Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Theoretical Aspects of Computer Software (4th International 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.

[BGa] Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms, II. To appear in *ACM Transactions on Computational Logic*.

[BGb] Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms, III. To appear in *ACM Transactions on Computational Logic*.

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

[BG06] Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms, I. *ACM Transactions on Computational Logic*, 7(2):363–419, 2006.

[Gur00] Yuri Gurevich. Sequential abstract state machines capture sequential algorithms. *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 computational completeness of formal encryption. In *Crypto 2003*, volume 2729 of *LNCS*, 2003.

[MW04] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *Journal of Computer Security*, 12(1):99–130, 2004.

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