Formal Eavesdropping and its Computational Interpretation

Martín Abadi[1] and Jan Jürjens[2]

[1] InterTrust, Strategic Technologies and Architectural Research Laboratory[⋆⋆⋆]
[2] Computing Laboratory, University of Oxford

Abstract. We compare two views of symmetric cryptographic primitives in the context of the systems that use them. We express those systems in a simple programming language; each of the views yields a semantics for the language. One of the semantics treats cryptographic operations formally (that is, symbolically). The other semantics is more detailed and computational; it treats cryptographic operations as functions on bitstrings. Each semantics leads to a definition of equivalence of systems with respect to eavesdroppers. We establish the soundness of the formal definition with respect to the computational one. This result provides a precise computational justification for formal reasoning about security against eavesdroppers.

1 Introduction

This work is part of an effort to bridge the gap between the formal and the computational views of cryptography and cryptographic protocols. The formal view is based on ideas and methods from logic and programming languages (e.g., [1, 8, 16, 18, 22]). It has led, in particular, to techniques for high-level reasoning (even automated reasoning) about security protocols. The computational view relies on the vocabulary of complexity theory (e.g., [5, 7, 10, 12, 13, 26]). It has developed sophisticated foundations and proof techniques. Several recent projects have made progress in the direction of linking these two views [3, 17, 23–25]. These projects differ in their approaches and their technical details. Overall, however, they aim to refine and to provide foundations for formal treatments of cryptography, and to combine the benefits of both treatments for reasoning about systems that use cryptography.

More specifically, this work extends the recent work of Abadi and Rogaway [3], which compares a formal account of symmetric (shared-key) encryption [19] with a computational account. Both of these accounts are fairly standard in the respective communities. That work deals

⋆⋆⋆ This work was started while the author was at Bell Labs Research.

with expressions that represent pieces of data: booleans, pairs, randomly generated keys, and symmetric encryptions. Each expression induces an ensemble (that is, for each choice of a security parameter, a probability distribution over the bitstrings that the expression represents). The main result there is that if two expressions are equivalent in the formal model then the ensembles that they induce are computationally indistinguishable. The result means that formal reasoning about equivalence of expressions is sound with respect to the computational model. Furthermore, the hypotheses of the result indicate a few significant discrepancies between the two accounts (for example, in the treatment of encryption cycles).

Here we go further by considering systems that use symmetric encryption and decryption. The systems may in particular represent cryptographic protocols with several parties. We describe systems in a simple programming language, a small but expressive calculus of the kind common in the programming-language literature. We present the two models as two semantics for the language. Thus, we rely on basic concepts and techniques from programming-language theory [21]. In both the formal model and the computational model, systems generate traces, traces are sequences of tuples of values (one value for each communication channel at each time instant), and two systems are equivalent if an eavesdropping adversary cannot distinguish a trace generated by one from a trace generated by the other. In the formal model, a value is a formal expression; the adversary is allowed only simple symbolic computations over expressions. In the computational model, a value is a bitstring; the adversary is a resource-bounded Turing machine that computes over bitstrings.

We obtain a soundness result analogous to that of Abadi and Rogaway: if two systems are equivalent in the formal model then the probability ensembles that they induce are computationally indistinguishable. This means that formal reasoning about equivalence of systems is sound with respect to the computational model, and that computational eavesdroppers have no more power than the simpler formal eavesdroppers.

This study of eavesdroppers presents difficulties and offers some insights, beyond those encountered in previous work. In particular, we treat situations where exactly the same ciphertext appears twice in a piece of data, and the adversary can notice a repetition (cf. [3]). In addition, we analyze programs where the control flow depends on the properties of the underlying cryptosystem, for example a program that generates two keys, compares them, and branches on the result of the comparison. In such examples, the formal model and the computational model do not give rise to exactly corresponding traces, but we show that the differences are negligible. Finally, we consider a substantial difference between the formal and the computational accounts of decryption. Formal models that treat symmetric encryption commonly assume that decrypting a message with a key K succeeds only if the message was encrypted under K, and that the success or failure of a decryption is evident to whoever performs it. This property is convenient in both protocol design and analysis; one may even argue that "encryption without integrity checking is all but useless" [6]. The property can be implemented by means of checkable redundancy. Computational models typically do not build it in. One outcome of our work is a precise computational counterpart of this property.

The next section presents the programming language. Sections 3 and 4 give formal semantics and computational semantics of the language, respectively, each with a notion of equivalence. Section 5 establishes our main soundness result; for this purpose, it develops some hypotheses. Section 6 concludes. A longer version of this paper [2] contains proofs and secondary results.

While the present work focuses on eavesdroppers and symmetric cryptosystems, we hope that it will serve as a stepping stone toward treating active adversaries and other cryptographic primitives. In these respects, there might be an opportunity for some convergence with the work of Lincoln et al. [17], which develops a process calculus with some features of the computational approach, and with the recent work of Pfitzmann and Waidner [24, 25], which investigates the secure composition of reactive systems with a simulatability approach. However, those works—unlike ours—consider formal models with probabilistic aspects; we would prefer to avoid this interesting complication as long as possible. (We refer to previous papers [3, 24] for further discussion of background and other, less closely related work.)

2 The Language (Syntax and Informal Semantics)

This section defines the basic programming language that serves as setting for our work. We let **Bool** be $\{0, 1\}$. We let **Keys**, **Coins**, and **Var** be fixed, infinite sets of symbols and **Channels** be a finite set of symbols (which may vary from system to system), all disjoint from **Bool** and each other. We partition **Channels** into $\mathbf{Channels}_i$ and $\mathbf{Channels}_e$, intuitively sets of internal (private) channels and external (public) channels.

A *system* is a collection of programs that communicate synchronously (in rounds) through channels, with the constraint that for each channel

c the system contains exactly one program P_c that outputs on c. This program P_c may take input from any channels, including c. Intuitively, the program is a description of a value to be output on the channel c in round $n+1$, computed from values found on channels in round n. Local state can be maintained through the use of feedback channels, and used for iteration (for instance, for coding *while* loops).

More precisely, the set of *programs* is defined by the following grammar:

$P, Q ::=$	programs
ε	null value
K	key ($K \in \mathbf{Keys}$)
b	bit ($b \in \mathbf{Bool}$)
$\perp$	error
a	input on channel ($a \in \mathbf{Channels}$)
x	variable ($x \in \mathbf{Var}$)
(P, Q)	pair
$(\nu K)P$	"new" ($K \in \mathbf{Keys}$)
$(\nu r)\{P\}_Q^r$	shared-key encryption ($r \in \mathbf{Coins}$)
if $P = Q$ *then* P' *else* Q'	conditional
case P *of* (x, y) *do* Q *else* R	case: pair ($x, y \in \mathbf{Var}$)
case P *of* $\{x\}_{P'}$ *do* Q *else* R	case: encryption ($x \in \mathbf{Var}$)

In other words, the set of programs is defined inductively to contain a null value, keys, bits, pairs of programs, etc.; and we typically use letters like P and Q for programs. The expression ε represents the null value; it is the initial value on all channels. An occurrence of a channel name a refers to the value found on a at the previous instant. Variables are introduced in case constructs, which determine their values. The "new" construct (ν), borrowed from the pi calculus [20] (see also [1]), introduces a fresh key K; the symbol K is bound. The "new" notation also appears in $(\nu r)\{P\}_Q^r$, which represents the result of encrypting the value of P using the value of Q as key and randomizing the encryption with the fresh coins r. Informally, we may just write $\{P\}_Q$ instead of $(\nu r)\{P\}_Q^r$; this notation is unambiguous since each encryption operation uses fresh coins. The first case construct tests whether P is a pair, of the form (x, y); if so, Q is evaluated, using the actual values of (x, y); if not, R is evaluated. The second case construct tests whether P is a ciphertext under the key represented by P' with contents x; if so, Q is evaluated, using the actual value of x; if not, R is evaluated. This construct reflects the assumption (discussed in the introduction) that decrypting a message with a key succeeds only if the message was encrypted under the key, and that the success or failure of a decryption is evident. In the case constructs, x and y are bound variables. For example, suppose that $c_0, c_1 \in \mathbf{Channels}$ and $K_0, K_1 \in \mathbf{Keys}$. Then the following is a program:

$$\textit{case } c_0 \textit{ of } \{x\}_{K_0} \textit{ do } (0,x) \textit{ else case } c_0 \textit{ of } \{x\}_{K_1} \textit{ do } (1,x) \textit{ else } \varepsilon$$

This program looks at the value in c_0. If it is a ciphertext with plaintext x under K_i then it outputs (i, x); otherwise, it outputs ε. We may form a system for example by letting this program output on c_1 and adding the trivial program $\{0\}_{K_1}$ to output on c_0. We may write the resulting system as:

$$c_0 := \{0\}_{K_1}$$
$$c_1 := \textit{case } c_0 \textit{ of } \{x\}_{K_0} \textit{ do } (0,x) \textit{ else case } c_0 \textit{ of } \{x\}_{K_1} \textit{ do } (1,x) \textit{ else } \varepsilon$$

(See [14, 15] for further examples in a fragment of this calculus.)

Our choice of a synchronous model avoids consideration of scheduling, and the delicate mix of non-deterministic and probabilistic choices (cf. [17]). While scheduling issues are interesting, we believe that they are best addressed in the context of active adversaries.

Other aspects of the language are somewhat more arbitrary. Our main concern is to have a tractable language with which we can express examples (though not always in the most convenient way), identify problematic issues, and develop precise solutions. The exact details of the language are secondary.

3 Formal Model

This section gives the first precise semantics for programs, and a corresponding notion of equivalence between programs.

Expressions. As indicated in the introduction, this semantics views the values communicated on channels as formal expressions. The set of *expressions* **Exp** is generated by the grammar:

$M, N ::=$	expressions
$\quad\varepsilon$	null value
$\quad K$	key ($K \in \mathbf{Keys}$)
$\quad b$	bit ($b \in \mathbf{Bool}$)
$\quad\bot$	error
$\quad(M, N)$	pair
$\quad\{M\}_K^r$	encryption ($K \in \mathbf{Keys}, r \in \mathbf{Coins}$)

We fix a set $\mathbf{Plain} \subseteq \mathbf{Exp} \setminus \{\bot\}$, which represents the set of plaintexts.

$$
\begin{array}{ll}
[\![\varepsilon]\!](\boldsymbol{M}) = \varepsilon & \\
[\![K]\!](\boldsymbol{M}) = K & (K \in \mathbf{Keys}) \\
[\![b]\!](\boldsymbol{M}) = b & (b \in \mathbf{Bool}) \\
[\![\perp]\!](\boldsymbol{M}) = \perp & \\
[\![u]\!](\boldsymbol{M}) = \boldsymbol{M}_u & (u \in \mathbf{Channels} \cup \mathbf{Var}) \\
[\![(P,Q)]\!](\boldsymbol{M}) = ([\![P]\!](\boldsymbol{M}), [\![Q]\!](\boldsymbol{M})) & \\
[\![(\nu v)P]\!](\boldsymbol{M}) = [\![P]\!](\boldsymbol{M}) & (v \in \mathbf{Keys} \cup \mathbf{Coins}) \\
[\![\{P\}_Q^r]\!](\boldsymbol{M}) = \{[\![P]\!](\boldsymbol{M})\}^r_{[\![Q]\!](\boldsymbol{M})} & \text{if } [\![P]\!](\boldsymbol{M}) \in \mathbf{Plain} \text{ and } [\![Q]\!](\boldsymbol{M}) \in \mathbf{Keys} \\
[\![\{P\}_Q^r]\!](\boldsymbol{M}) = \{0\}^r_{[\![Q]\!](\boldsymbol{M})} & \text{if } [\![P]\!](\boldsymbol{M}) \notin \mathbf{Plain} \text{ and } [\![Q]\!](\boldsymbol{M}) \in \mathbf{Keys} \\
[\![\{P\}_Q^r]\!](\boldsymbol{M}) = \perp & \text{if } [\![Q]\!](\boldsymbol{M}) \notin \mathbf{Keys} \\
[\![\textit{if } P = Q \textit{ then } P' \textit{ else } Q']\!](\boldsymbol{M}) = [\![P']\!](\boldsymbol{M}) & \text{if } [\![P]\!](\boldsymbol{M}) = [\![Q]\!](\boldsymbol{M}) \\
[\![\textit{if } P = Q \textit{ then } P' \textit{ else } Q']\!](\boldsymbol{M}) = [\![Q']\!](\boldsymbol{M}) & \text{if } [\![P]\!](\boldsymbol{M}) \neq [\![Q]\!](\boldsymbol{M}) \\
[\![\textit{case } P \textit{ of } (x,y) \textit{ do } Q \textit{ else } R]\!](\boldsymbol{M}) = [\![Q]\!](\boldsymbol{M}[x \to M_1, y \to M_2]) & \\
 & \text{if } [\![P]\!](\boldsymbol{M}) = (M_1, M_2) \\
[\![\textit{case } P \textit{ of } (x,y) \textit{ do } Q \textit{ else } R]\!](\boldsymbol{M}) = [\![R]\!](\boldsymbol{M}) & \text{if } [\![P]\!](\boldsymbol{M}) \text{ is not a pair} \\
[\![\textit{case } P \textit{ of } \{x\}_{P'} \textit{ do } Q \textit{ else } R]\!](\boldsymbol{M}) = [\![Q]\!](\boldsymbol{M}[x \to M]) & \text{if } [\![P]\!](\boldsymbol{M}) = \{M\}^r_{[\![P']\!](\boldsymbol{M})} \\
\quad \text{for } M \in \mathbf{Plain} \text{ and } r \in \mathbf{Coins} \text{ and if } [\![P']\!](\boldsymbol{M}) \in \mathbf{Keys} & \\
[\![\textit{case } P \textit{ of } \{x\}_{P'} \textit{ do } Q \textit{ else } R]\!](\boldsymbol{M}) = [\![R]\!](\boldsymbol{M}) & \text{otherwise}
\end{array}
$$

Fig. 1. Definition of $[\![P]\!](\boldsymbol{M})$ in the formal semantics.

Generating traces. Without loss of generality, we assume that keys and coins are renamed so that in a system $\mathcal{P}$ each key or coin symbol is bound at most once, and any bound key does not also occur free. We write $\mathbf{new}(P)$ for the set of bound keys in a program P. We also fix injective functions $\psi_k : \mathbb{N} \times \mathbf{Keys} \to \mathbf{Keys}$ and $\psi_c : \mathbb{N} \times \mathbf{Coins} \to \mathbf{Coins}$, for renaming keys and coins.

For any program P and $\boldsymbol{M} : \mathbf{Channels} \cup \mathbf{Var} \to \mathbf{Exp}$, we define $[\![P]\!](\boldsymbol{M})$ in Figure 1, so that $[\![P]\!](\boldsymbol{M})$ is the expression that results from running P once, when the variables and channels have the initial values given in $\boldsymbol{M}$. In the definition, $\boldsymbol{M}[x \to M]$ is the function in $\mathbf{Channels} \cup \mathbf{Var} \to \mathbf{Exp}$ that coincides with $\boldsymbol{M}$ except on x, which it maps to M.

A *trace* is a sequence of mappings $\mathbf{Channels} \to \mathbf{Exp}$ (that is, a sequence of $\mathbf{Channels}$-indexed tuples of expressions). Suppose that $\mathcal{P}$ is a system that consists of a program P_c for each channel c, all without free variables. We define the semantics $[\![\mathcal{P}]\!]$ of $\mathcal{P}$ as an infinite trace $([\![\mathcal{P}]\!]_n)_{n \in \mathbb{N}}$:

- $[\![\mathcal{P}]\!]_1(c) = \ulcorner [\![P_c]\!] \urcorner^1(\varepsilon, \ldots, \varepsilon)$
- $[\![\mathcal{P}]\!]_{n+1}(c) = \ulcorner [\![P_c]\!] \urcorner^{n+1}([\![\mathcal{P}]\!]_n)$

where $\ulcorner \ldots \urcorner^n$ substitutes any occurrence of a key $K \in \mathbf{new}(P_c)$ in P_c by $\psi_k(n, K)$, any occurrence of a key $K \notin \mathbf{new}(P_c)$ in P_c by $\psi_k(0, K)$, and any occurrence of a coin r in P_c by $\psi_c(n, r)$. The substitutions guarantee that all fresh keys and coins are represented with different symbols. Intuitively, the trace $[\![\mathcal{P}]\!]$ is the result of running $\mathcal{P}$ forever.

Equivalence. Abadi and Rogaway define an equivalence relation on formal expressions; it is intended to relate two expressions if they look the same to an adversary. We adapt and extend their definition.

First, we define an *entailment* relation $\mathcal{M} \vdash N$, where $\mathcal{M}$ is a set of expressions and N is an expression. Intuitively, $\mathcal{M} \vdash N$ means that the adversary can derive N from $\mathcal{M}$. Formally, we define the relation inductively, as the least relation with the following properties:

- $\mathcal{M} \vdash M$ for any $M \in \mathcal{M}$,
- if $\mathcal{M} \vdash (N_1, N_2)$ then $\mathcal{M} \vdash N_1$ and $\mathcal{M} \vdash N_2$,
- if $\mathcal{M} \vdash \{N\}_K^r$ (for any $r \in \mathbf{Coins}$) and $\mathcal{M} \vdash K$ then $\mathcal{M} \vdash N$.

Next, we introduce the box symbol $\square$, which represents a ciphertext that an attacker cannot decrypt. Since the attacker can test bitwise equality of ciphertexts, we index boxes by coins: $\square_r$ and $\square_{r'}$ represent the same ciphertext if and only if $r = r'$ (basically, because we do not permit coin reuse in programs). The set **Pat** of *patterns* is generated by the grammar:

$M, N ::=$	patterns
ε	null value
K	key ($K \in \mathbf{Keys}$)
b	bit ($b \in \mathbf{Bool}$)
$\bot$	error
(M, N)	pair
$\{M\}_K^r$	encryption ($K \in \mathbf{Keys}, r \in \mathbf{Coins}$)
$\square_r$	undecryptable ($r \in \mathbf{Coins}$)

We define a function that, given a set of keys T and an expression M, reduces M to a pattern. Intuitively, this is the pattern that an attacker can see in M if the attacker has the keys in T.

$$
\begin{aligned}
p(\varepsilon, T) &= \varepsilon \\
p(K, T) &= K && \text{(for } K \in \mathbf{Keys}\text{)} \\
p(b, T) &= b && \text{(for } b \in \mathbf{Bool}\text{)} \\
p(\bot, T) &= \bot \\
p((M, N), T) &= (p(M, T), p(N, T)) \\
p(\{M\}_K^r, T) &= \begin{cases} \{p(M, T)\}_K^r & \text{if } K \in T \\ \square_r & \text{otherwise} \end{cases}
\end{aligned}
$$

For any trace $\boldsymbol{t}$ we define a corresponding sequence $pattern(\boldsymbol{t})$ of tuples of patterns, using the set of keys obtained from the trace itself and projecting out the internal channels:

$$
\begin{aligned}
(pattern(\boldsymbol{t})) \;&=\; p(\boldsymbol{t}\lfloor_{\mathbf{Channels}_e}, T)\\
&\text{where } T = \{K \in \mathbf{Keys} \mid \{\boldsymbol{t}_i(c) : 1 \le i \le |\boldsymbol{t}|, c \in \mathbf{Channels}_e\} \vdash K\}
\end{aligned}
$$

where $\boldsymbol{t}\lfloor_{\mathbf{Channels}_e}$ is the projection of $\boldsymbol{t}$ onto $\mathbf{Channels}_e$ and $|\boldsymbol{t}| \le \infty$ the length of the sequence $\boldsymbol{t}$. Roughly, this is the pattern that an attacker can see in $\boldsymbol{t}$ using the set of keys obtained from $\boldsymbol{t}$. We say that two traces are *equivalent* ($\equiv$) if they yield the same pattern, and *equivalent up to renaming* ($\cong$) if they are equivalent modulo renaming of keys and coins:

$$
\begin{aligned}
&\boldsymbol{t} \equiv \boldsymbol{s} \;\text{ if and only if }\; pattern(\boldsymbol{t}) = pattern(\boldsymbol{s})\\
&\boldsymbol{t} \cong \boldsymbol{s} \;\text{ if and only if }\; \text{there exist bijections } \sigma \text{ on } \mathbf{Keys} \text{ and } \sigma' \text{ on } \mathbf{Coins}\\
&\qquad\qquad\qquad\qquad \text{such that } \boldsymbol{t} \equiv \boldsymbol{s}\sigma\sigma'
\end{aligned}
$$

where $\boldsymbol{s}\sigma\sigma'$ is the result of applying σ and σ' as substitutions to $\boldsymbol{s}$.

Suppose that two systems $\mathcal{P}$ and $\mathcal{Q}$ have the same set of external channels (so we can compare them). They are *equivalent* if they generate equivalent traces:

$$
\mathcal{P} \cong \mathcal{Q} \;\text{ if and only if }\; [\![\mathcal{P}]\!] \cong [\![\mathcal{Q}]\!]
$$

Our intent is that $\mathcal{P} \cong \mathcal{Q}$ holds when an eavesdropper cannot distinguish $\mathcal{P}$ and $\mathcal{Q}$ by observing the external channels, with no a priori knowledge.

In the following simple examples, we consider pairs of systems that consist each of only one program that outputs on an external channel, and we identify the systems with the corresponding programs.

- First, $\{0\}_K$ and $\{1\}_K$ differ only in the bit that they send encrypted under a key that the eavesdropper does not have. According to the definition of equivalence, we obtain $\{0\}_K \cong \{1\}_K$. Thus, in the formal model there is no need to consider the possibility that the eavesdropper can guess the key.
- We also obtain $(\mathit{if}\ (\nu K)K = K'\ \mathit{then}\ P\ \mathit{else}\ Q) \cong Q$. Thus, in the formal model there is no need to consider the possibility that the keys K and K' are equal "by chance".
- Finally, we also obtain $(\mathit{case}\ (\nu K)\{M\}_K\ \mathit{of}\ \{x\}_{K'}\ \mathit{do}\ Q\ \mathit{else}\ R) \cong R$. Thus, in the formal model there is no need to consider the possibility that a ciphertext under K can be decrypted with K'.

Such simplifications contribute greatly to the convenience and practicality of formal reasoning.

4 Computational Model

This section gives a second semantics for programs, with a corresponding notion of equivalence between programs. The semantics relies on bit-strings, rather than formal expressions. The limitations of the adversary are expressed in terms of computational complexities and probabilities.

Elements of encryption schemes and other basics. We let $\mathsf{String} = \{0,1\}^*$ be the set of all finite strings, let $|m|$ be the length of string m, and let Cipher and Key be non-empty sets of finite strings. $\mathsf{Coins} = \{0,1\}^\omega$, the set of infinite strings, is endowed with a probability distribution. The set of *security parameters* $\mathsf{Parameter}$ is 1^* (the set of finite strings of 1 bits). For each $\eta \in \mathsf{Parameter}$, we let the set of *plaintexts* Plain_η be a finite non-empty set of finite strings, with $\mathsf{Plain}_\eta \subseteq \mathsf{Plain}_{\eta 1}$ for each η, and we write $\mathsf{Plain} = \bigcup_\eta \mathsf{Plain}_\eta$. In allowing the set of plaintexts to depend on a security parameter we follow for example Goldwasser and Bellare [11, p.105]. We let $0, 1, \varepsilon$ be particular strings in Plain_0 and $\perp$ a particular string in $\mathsf{String} \setminus (\mathsf{Plain} \cup \mathsf{Cipher})$. We assume that, for all η, if $m \in \mathsf{Plain}_\eta$ and $|m| = |m'|$ then $m' \in \mathsf{Plain}_\eta$.

An *encryption scheme*, Π, is a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where

$$\begin{aligned} &\mathcal{K}\colon \mathsf{Parameter} \times \mathsf{Coins} \to \mathsf{Key} \\ &\mathcal{E}\colon \mathsf{Key} \times \mathsf{String} \times \mathsf{Coins} \to \mathsf{Cipher} \\ &\mathcal{D}\colon \mathsf{Key} \times \mathsf{String} \to \mathsf{Plain} \cup \{\perp\} \end{aligned}$$

and each algorithm is computable in time polynomial in the size of its input (but without consideration for the size of Coins input). Algorithm $\mathcal{K}$ is the *key-generation* algorithm, $\mathcal{E}$ is the *encryption* algorithm, and $\mathcal{D}$ is the *decryption* algorithm. We usually write the first argument to $\mathcal{E}$ or $\mathcal{D}$, the key, as a subscript, and the last argument to $\mathcal{E}$, the coin, as a superscript. When we omit the final argument to $\mathcal{K}$ or $\mathcal{E}$, this indicates the corresponding probability space, or the support set of this space. We require that for all $\eta \in \mathsf{Parameter}$, $k \in \mathcal{K}(\eta)$, and $r \in \mathsf{Coins}$, if $m \in \mathsf{Plain}_\eta$ then $\mathcal{D}_k(\mathcal{E}_k^r(m)) = m$, while if $m \notin \mathsf{Plain}_\eta$ then $\mathcal{E}_k^r(m) = \mathcal{E}_k^r(0)$. We insist that $|\mathcal{E}_k^r(m)|$ depends only on η and $|m|$ when $k \in \mathcal{K}(\eta)$.

A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is *negligible* if for all $c > 0$ there exists N_c such that $\epsilon(\eta) \leq \eta^{-c}$ for all $\eta \geq N_c$. A $\mathsf{Parameter}$-indexed family $D = (D_\eta)$ of distributions on a set S is called an *ensemble* on S. We write $x \stackrel{R}{\leftarrow} D_\eta$ to indicate that x is sampled from D_η. Let $D = \{D_\eta\}$ and $D' = \{D'_\eta\}$ be ensembles on a set S. We say that D and D' are (computationally)

$$
\begin{array}{ll}
\llbracket \varepsilon \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \varepsilon, \text{“null”} \,\rangle & \\
\llbracket K \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \tau(K), \text{“key”} \,\rangle & (K \in \mathbf{Keys}) \\
\llbracket b \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, b, \text{“bool”} \,\rangle & (b \in \mathbf{Bool}) \\
\llbracket \bot \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \bot, \text{“error”} \,\rangle & \\
\llbracket u \rrbracket^{\tau}(\boldsymbol{m}) = \boldsymbol{m}_u & (u \in \mathbf{Channels} \cup \mathbf{Var}) \\
\llbracket (P, Q) \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \llbracket P \rrbracket^{\tau}(\boldsymbol{m}), \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}), \text{“pair”} \,\rangle & \\
\llbracket (\nu v) P \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) & (v \in \mathbf{Keys} \cup \mathbf{Coins}) \\
\llbracket \{P\}^{r}_{Q} \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \mathcal{E}^{\tau(r)}_{\llbracket Q \rrbracket^{\tau}(\boldsymbol{m})}(\llbracket P \rrbracket^{\tau}(\boldsymbol{m})), \text{“ciphertext”} \,\rangle & \text{if } \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}) \in \mathbf{Keys}\ (r \in \mathbf{Coins}) \\
\llbracket \{P\}^{r}_{Q} \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, \bot, \text{“error”} \,\rangle & \text{otherwise } (r \in \mathbf{Coins}) \\
\llbracket \mathit{if}\ P = Q\ \mathit{then}\ P'\ \mathit{else}\ Q' \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket P' \rrbracket^{\tau}(\boldsymbol{m}) & \text{if } \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}) \\
\llbracket \mathit{if}\ P = Q\ \mathit{then}\ P'\ \mathit{else}\ Q' \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket Q' \rrbracket^{\tau}(\boldsymbol{m}) & \text{if } \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) \neq \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}) \\
\llbracket \mathit{case}\ P\ \mathit{of}\ (x, y)\ \mathit{do}\ Q\ \mathit{else}\ R \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}[x \to m_1, y \to m_2]) & \\
 & \text{if } \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, m_1, m_2, \text{“pair”} \,\rangle \\
\llbracket \mathit{case}\ P\ \mathit{of}\ (x, y)\ \mathit{do}\ Q\ \mathit{else}\ R \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket R \rrbracket^{\tau}(\boldsymbol{m}) & \\
\qquad \text{if there are no } m_1, m_2 \text{ with } \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, m_1, m_2, \text{“pair”} \,\rangle & \\
\llbracket \mathit{case}\ P\ \mathit{of}\ \{x\}_{P'}\ \mathit{do}\ Q\ \mathit{else}\ R \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket Q \rrbracket^{\tau}(\boldsymbol{m}[x \to \mathcal{D}_k(m)]) & \\
\qquad \text{if } \llbracket P' \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, k, \text{“key”} \,\rangle \text{ and } \llbracket P \rrbracket^{\tau}(\boldsymbol{m}) = \langle\, m, \text{“ciphertext”} \,\rangle & \\
\qquad \text{with } k \in \mathsf{Key} \text{ and } \mathcal{D}_k(m) \neq \bot & \\
\llbracket \mathit{case}\ P\ \mathit{of}\ \{x\}_{P'}\ \mathit{do}\ Q\ \mathit{else}\ R \rrbracket^{\tau}(\boldsymbol{m}) = \llbracket R \rrbracket^{\tau}(\boldsymbol{m}) & \text{otherwise}
\end{array}
$$

Fig. 2. Definition of $\llbracket P \rrbracket^{\tau}(\boldsymbol{m})$ in the computational semantics.

indistinguishable, and write $D \approx D'$, if for every probabilistic polynomial-time adversary $A : \mathsf{Parameter} \times S \times \mathsf{Coins} \to \{0, 1\}$, the following function ϵ is negligible:

$$\epsilon(\eta) \stackrel{\text{def}}{=} \Pr\left[x \stackrel{R}{\leftarrow} D_\eta : A(\eta,\ x) = 1\right] - \Pr\left[x \stackrel{R}{\leftarrow} D'_\eta : A(\eta,\ x) = 1\right]$$

Generating traces. Again we assume that keys and coins are renamed so that in a system $\mathcal{P}$ each key or coin symbol is bound at most once, and any bound key does not also occur free. We write $\mathbf{keys}(P)$ and $\mathbf{coins}(P)$ for the sets of all key symbols and coin symbols that occur in P.

A function $\tau : \mathbf{keys}(P) \cup \mathbf{coins}(P) \to \mathsf{Key} \cup \mathsf{Coins}$ mapping $\mathbf{keys}(P)$ to Key and $\mathbf{coins}(P)$ to Coins is called a *choice function*. For any program P, any $\boldsymbol{m} \in \mathbf{Channels} \cup \mathbf{Var} \to \mathsf{String}$, and any choice function τ, we define $\llbracket P \rrbracket^{\tau}(\boldsymbol{m})$ in Figure 2, so that $\llbracket P \rrbracket^{\tau}(\boldsymbol{m})$ is the bitstring that results from running P once, when the variables and channels have the initial values given in $\boldsymbol{m}$ and keys and coins are chosen according to τ. In the definition, we write $\langle\, a_1, \ldots, a_n \,\rangle$ for a bitstring representation of the tupling of $a_1, \ldots, a_n$. We assume that bitstring representations are such that, for each η, all and only expression plaintexts in $\mathbf{Plain}$ are mapped to bitstring plaintexts in Plain_η (up to negligible probability). We write

$\boldsymbol{m}[x \to m]$ for the function in **Channels** $\cup$ **Var** $\to$ String that coincides with $\boldsymbol{m}$ except on x, which it maps to m.

Despite superficial similarities, the definitions $[\![P]\!](\boldsymbol{M})$ and $[\![P]\!]^{\tau}(\boldsymbol{m})$ can assign different behaviors to programs. For example, in the formal model we have $[\![\textit{if } K = K' \textit{ then } 1 \textit{ else } 0]\!](\boldsymbol{M}) = 0$, while in the computational model we have $[\![\textit{if } K = K' \textit{ then } 1 \textit{ else } 0]\!]^{\tau}(\boldsymbol{m}) = \langle 1, \text{“bool”} \rangle$ or $= \langle 0, \text{“bool”} \rangle$ depending on τ. Part of the proof of our main theorem consists in showing that these differences are negligible.

A *trace* is a sequence of mappings **Channels** $\to$ String (that is, a sequence of **Channels**-indexed tuples of bitstrings). Suppose that $\mathcal{P}$ is a system that consists of a program P_c for each channel c, all without free variables. For any sequence $\tau = (\tau_n)_{n \in \mathbb{N}}$ of choice functions that agree on the free keys in $\mathcal{P}$, we define the result of running $\mathcal{P}$ with τ, inductively:

- $[\![\mathcal{P}]\!]_1^{\tau}(c) = [\![P_c]\!]^{\tau_1}(\varepsilon, \ldots, \varepsilon)$
- $[\![\mathcal{P}]\!]_{n+1}^{\tau}(c) = [\![P_c]\!]^{\tau_{n+1}}([\![\mathcal{P}]\!]_n^{\tau})$

where P_c is the unique program having c as its output channel.

Computational indistinguishability. Suppose $\mathcal{P}$ and $\mathcal{Q}$ are systems with the same external channels. We let:

$$\mathcal{P} \approx_{\Pi} \mathcal{Q} \ \text{ if and only if } \ \text{for all } n \in \mathbb{N},\ [\![\mathcal{P}]\!]_{\leq n}^{e} \approx [\![\mathcal{Q}]\!]_{\leq n}^{e}$$

where $[\![\mathcal{P}]\!]_{\leq n}^{e}$ is:

$$\{([\![\mathcal{P}]\!]_1^{\tau}, \ldots, [\![\mathcal{P}]\!]_n^{\tau}) \lfloor_{\mathbf{Channels}_e} : \tau \leftarrow \textsc{Init}_{\eta,n}(\mathbf{coins}(\mathcal{P}), \mathbf{keys}(\mathcal{P}), \mathbf{new}(\mathcal{P}))\}$$

and $\textsc{Init}_{\eta,n}(\mathbf{coins}(\mathcal{P}), \mathbf{keys}(\mathcal{P}), \mathbf{new}(\mathcal{P}))$ chooses a sequence $\tau = (\tau_1, \ldots, \tau_n)$ of choice functions $\tau_i : \mathbf{keys}(\mathcal{P}) \cup \mathbf{coins}(\mathcal{P}) \to \mathsf{Key} \cup \mathsf{Coins}$ that agree on $\mathbf{keys}(\mathcal{P}) \setminus \mathbf{new}(\mathcal{P})$ (the set of free keys in $\mathcal{P}$), generating keys by $\mathcal{K}(\eta)$ and coins according to the distribution on Coins.

Again, our intent is that $\mathcal{P} \approx_{\Pi} \mathcal{Q}$ holds when an eavesdropper cannot distinguish $\mathcal{P}$ and $\mathcal{Q}$ by observing external channels. However, the definition of $\mathcal{P} \approx_{\Pi} \mathcal{Q}$ relies on a computational view of the semantics of systems and of the powers of the eavesdropper, quite different from the formal view of section 3. The next section relates these two views.

Note that this definition of $\mathcal{P} \approx_{\Pi} \mathcal{Q}$ includes a quantification over all lengths n. One might have expected some bound on n, perhaps as a function of the security parameter η. The quantification over all n yields a simpler relation. It may be rather strong; it is nevertheless useful for our purposes.

5 Soundness

While the semantics of Section 4 does not specify security assumptions on the underlying encryption scheme, such assumptions are necessary in order to relate the two semantics. We use *type-0 security* (a variant of the standard notion of semantic security [4, 12], defined and justified in [3]):

Definition 1 (Type-0 security). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an encryption scheme, let* $\eta \in \mathsf{Parameter}$ *be a security parameter. An encryption scheme* Π *is* type-0 secure *if, for every probabilistic polynomial-time adversary* $A^{-,-}$ *with two oracles, the function*

$$\epsilon(\eta) \stackrel{\text{def}}{=} \Pr\left[k, k' \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(\cdot),\, \mathcal{E}_{k'}(\cdot)}(\eta) = 1\right] - \Pr\left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(0),\, \mathcal{E}_k(0)}(\eta) = 1\right]$$

is negligible.

In addition, we need a computational counterpart to the assumption that decrypting a message with the "wrong" key is a noticeable error, as this assumption is built into the formal model. We use the following concept of *confusion-free* encryption:

Definition 2 (Confusion-free encryption scheme). *The encryption scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is* confusion-free *if for all* $m \in \mathsf{String}$ *the probability* $\Pr[k, k' \stackrel{R}{\leftarrow} \mathcal{K}(\eta), x \stackrel{R}{\leftarrow} \mathcal{E}_k(m) : \mathcal{D}_{k'}(x) \neq \perp]$ *is negligible.*

M. Fischlin's related concept of committing encryption requires the encryption function to be essentially injective (to map different plaintexts to different ciphertexts) [9]. Confusion-freedom differs in that it allows some collisions and puts requirements on the decryption function. The long version of this paper [2] describes a construction of confusion-free encryption schemes.

Finally, we need to rule out encryption cycles (such as encrypting a key with itself), as in [3]. These cycles are not supported in standard computational definitions of security (such as semantic security), which for example allow $\{K\}_K^r$ to reveal K. These cycles may however be acceptable in the random-oracle model, and they are allowed (silently!) in formal methods. According to [3], K *encrypts* K' in the expression M if there exist N and r such that $\{N\}_K^r$ is a subexpression of M and K' occurs in N. Here we adopt the slightly more liberal definition that ignores occurrences of K' as a subscript (that is, as an encryption key). For each M, this defines a binary relation on keys. We say that M is *cyclic* if this relation is cyclic, and is *acyclic* otherwise. Similarly, we say that a trace is cyclic or acyclic. A system $\mathcal{P}$ does not generate encryption cycles if the formal trace $[\![\mathcal{P}]\!]$ is acyclic.

Our main theorem says that if two systems are equivalent with respect to formal eavesdroppers, then they are also equivalent with respect to computational eavesdroppers. Thus, computational eavesdroppers do not have essentially more power than formal eavesdroppers, so we may reason with the higher-level formal eavesdroppers without loss of generality. (We believe that a converse also holds; it is interesting but less important.)

Theorem 1. *Let $\mathcal{P}$ and $\mathcal{Q}$ be two systems that do not generate encryption cycles. Suppose Π is a type-0 secure and confusion-free encryption scheme. If $\mathcal{P} \cong \mathcal{Q}$ then $\mathcal{P} \approx_{\Pi} \mathcal{Q}$.*

The proof of this theorem has several components. One of them is analogous to the main proof in Abadi and Rogaway's paper [3], with a few twists and extensions (for example, dealing with repetitions). Others deal with concepts not present in Abadi and Rogaway's work, for instance the control flow of processes. (See [2] for details.)

6 Conclusion

This paper relates two views of symmetric cryptographic primitives, in the context of the systems that use them. We express the systems in a simple programming language. This language has two semantics. Each of the semantics leads to a notion of equivalence of systems; roughly, two systems are equivalent if an eavesdropper cannot tell them apart. In one of the semantics, cryptographic operations are interpreted symbolically. In the other, they are interpreted as operations on bitstrings; this more concrete interpretation leads to reasoning with probabilities and computational complexities. Therefore, the formal semantics may seem attractively simpler but also naive. Nevertheless, under suitable hypotheses, formal equivalence implies computational equivalence. This result provides a computational justification for high-level, formal reasoning about security against eavesdroppers, and another significant and promising link between the formal and the computational views of cryptography and cryptographic protocols.

Acknowledgements

Discussions with Phillip Rogaway and John Mitchell contributed to the writing of this paper. This work was partly done while the second author was visiting Bell Labs Research, Palo Alto, whose hospitality is gratefully acknowledged.

References

[1] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

[2] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation, 2001. Longer version of this paper, available at http://www.jurjens.de/jan/lambdaweb.ps.

[3] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). In *Proceedings of the First IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, August 2000.

[4] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: analysis of the DES modes of operation. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 97)*, pages 394–403, 1997.

[5] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—CRYPTO '94*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.

[6] Steven M. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the Sixth Usenix Unix Security Symposium*, pages 1–16, July 1996.

[7] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 82)*, pages 112–117, 1982.

[8] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.

[9] Marc Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In *Advances in Cryptology—Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 429–444. Springer-Verlag, 1999.

[10] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[11] S. Goldwasser and M. Bellare. Lecture notes on cryptography, 1999.

[12] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, April 1984.

[13] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing*, 17:281–308, 1988.

[14] Jan Jürjens. Composability of secrecy. In *International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM 2001)*, volume 2052 of *Lecture Notes in Computer Science*, pages 28–38. Springer-Verlag, 2001.

[15] Jan Jürjens. Secrecy-preserving refinement. In J. Fiadeiro and P. Zave, editors, *Formal Methods Europe*, volume 2021 of *Lecture Notes in Computer Science*, pages 135–152. Springer-Verlag, 2001.

[16] R. Kemmerer, C. Meadows, and J. Millen. Three system for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, Spring 1994.

[17] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[18] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[19] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[20] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[21] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[22] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[23] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems (extended abstract). *Electronic Notes in Theoretical Computer Science*, 32, April 2000.

[24] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, November 2000.

[25] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184–200, May 2001.

[26] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 82)*, pages 80–91, 1982.