Sound Computational Interpretation of Formal Encryption with Composed Keys

Peeter Laud*
Tartu University, Liivi 2, Tartu, Estonia
peeter_l@math.ut.ee
and Ricardo Corin**
University of Twente, P.O.Box 217, 7500AE The Netherlands.
corin@cs.utwente.nl

Abstract. The formal and computational views of cryptography have been related by the seminal work of Abadi and Rogaway. In their work, a formal treatment of encryption that uses atomic keys is justified in the computational world. However, many proposed formal approaches allow the use of *composed keys*, where any arbitrary expression can be used as encryption key. In this paper we consider an extension of the formal model presented by Abadi and Rogaway, in which it is allowed to use composed keys in formal encryption. We then provide a computational interpretation for expressions that allow us to establish the computational soundness of formal encryption with composed keys.

1 Introduction

Usually, it is necessary to adopt an abstract view of cryptographic operations (such as message encryption) to make the design and analysis of cryptographic protocols more manageable.

Two different, but still related abstract views of cryptographic operations –the formal and the computational– have developed separately in the last years. In the former, the exchanged messages of the protocol are modelled as formal expressions of a term algebra. The (cryptographic) operations, such as message pairing and encryption, are modelled as term constructors. In this setting, an adversary and its abilities can be modelled in terms of the messages the adversary knows; see for e.g. [11]. Furthermore, the security properties a protocol is supposed to achieve are also modelled formally [9, 19]. On the other hand, in the computational model, messages are considered to be (more realistically) bit-strings, while cryptographic operations are seen as functions over these bit-strings. Here, an

* Supported by Estonian Science Foundation, grant No. 5279.
** Funded by the LicenseScript project and the Telematica Instituut.

adversary is modelled as any efficient algorithm, while the security properties of a cryptographic protocol are defined on terms of the probability of the adversary to perform a succesful attack [13, 5].

Both of the two above models have advantages and disadvantages. On the one hand, the formal model allows to reason about cryptographic protocols more easily and generally. However, such benefits arise from the adoption of fairly strong assumptions (such as freeness of the term algebra, and fixing the adversary model). On the other hand, the computational model, by considering messages as bit-strings and modelling the adversary as any efficient algorithm, provides a more realistic model and thus offers more convincing security guarantees. However, proving protocols correct in the computational model is more difficult and less general than in the formal model.

In the work of Abadi and Rogaway [3], it is shown that if two formal expressions are similar to a formal adversary, then their corresponding computational interpretations, represented as bit-strings in the computational model, are also indistinguishable to any computational adversary. This result comprises a very important step into relating the formal and computational model.

Composed Keys. In [3], formal encryption is modelled by using atomic keys: that is, a formal expression $\{M\}_K$ represents encryption of message M with key K, where M is again a formal expression and K is an atomic symbol representing the cryptographic key. However, considering only atomic keys in encryption is not sufficient, and sometimes we need to be able to allow encryption with *composed keys*, representing non-atomic, constructed keys. In that setting, the formal language would need to be able to consider expressions of the form $\{M\}_N$, where both M and N are expressions. Considering composed keys as possible encryption keys is important due to that, in protocol design, it is fairly common to construct symmetric keys from shared secrets and other exchanged data as part of the protocol run. Examples of this can be found in the work of Gong [14], and, more recently, in a proposed protocol for achieving private authentication [1]. Moreover, many "real-world" cryptographic protocols use composed keys —see, for example SSL 3.0 [12]. Furthermore, in the formal model, some approaches based on constraint solving have been designed with specific support of composed keys [18] (this work was subsequently improved in [10]) and, more recently [4].

This paper defines a computational interpretation $[\![\cdot]\!]$ for the operation $\{M\}_N$. Briefly, the interpretation $[\![\{M\}_N]\!]$ consists of encrypting $[\![M]\!]$ —

the interpretation of M with a key obtained by applying the *random oracle* to $[\![N]\!]$. So, the interpretation of $\{M\}_N$ is quite intuitive. On the other hand, this forces us to use the *random oracle model* as the computational model. Using a random oracle seems to be necessary, since otherwise the goodness of keys might be questioned, as well as the independence of different keys.

We also define a relation $\cong$ over formal expressions and, as our main contribution, we show that $M \cong N$ implies the computational indistinguishability of $[\![M]\!]$ and $[\![N]\!]$.

Related Work. The work of Abadi and Rogaway [3] was later extended in Abadi and Jürjens [2] and Laud [15]. In these works, similar soundness results were obtained for richer formal languages, where instead of considering values of formal expressions, it is dealt with *outputs* of programs. However, differently from the formal language presented in this paper, both of these extended languages still treat the encryption operation as using atomic keys.

Micciancio and Warinschi [17] considered the converse of the soundness result (i.e., completeness of the formal language of [3]). In their work, it is shown that a completeness result can be obtained by considering a stronger encryption scheme, namely an authenticated encryption scheme.

Further extensions of the seminal work [3] deal with *encryption cycles* in expressions. For instance, the expression $\{K\}_K$ contains a trivial cycle: key K is immediately encrypted with itself. In the computational model, the security of a traditional encryption scheme can be compromised if an adversary gets hold of a message containing an encryption cycle. Thus, in the original work of Abadi and Rogaway, formal expressions were restricted to be cycle free. However, further work of Black et al. [8] and Laud [16] has shown that, in fact, this discrepancy can be addressed in two different ways: either by considering a new, stronger security definition of the encryption scheme [8], or by strengthening the adversary model of the formal model, such that it can be able to "break" encryption cycles [16].

Recently, Bellare and Kohno [6] have studied the security of cryptosystems against *related-key* attacks and also provided a construction of a secure cryptosystem against a certain kind of such attacks. Related keys are different from composed keys — a related key is something that is constructed from an already existing good key and some non-key data, whereas a composed key is constructed from non-key data only.

Plan of the paper. In Section 2 we present the formal language. Then, in Section 3 we introduce some basic notions of the computational model that are needed in the sequel, and also present an algorithm for translating formal expressions into computational [distributions of] bit-strings. In Section 4, we introduce an equivalence relation $\cong$ over formal expressions. This equivalence relation $\cong$ is elaborated and illustrated with some examples in Section 5. After that, in Section 6 we present the main contribution, a soundness result that relates the formal and computational models. Finally, Section 7 concludes the paper.

2 Expressions and patterns

Let **Bool** be the set $\{0,1\}$ and let **Keys** be the set of *formal keys* — this is a fixed, infinite set of symbols. Intuitively, elements of **Keys** represent cryptographic keys. Also, let **Rnd** be the set of *formal random numbers* — again a fixed, infinite set of symbols disjoint from **Keys**. The use of **Rnd** is needed since usually some of the constructors of formal expressions (such as encryption) represent probabilistic operations. This means that if such an operation is executed twice, even with the same arguments, the results will be different. Thus, the elements of **Rnd** are used to keep track which subexpressions of an expression represent the same invocation of that operation and which subexpressions represent different invocations. Our set of formal expressions **Exp** is defined by the following grammar:

$$\begin{array}{rlll} M, N ::= & b & \text{(bit)} \\ \mid & K & \text{(key)} \\ \mid & (M, N) & \text{(pair)} \\ \mid & \{M\}_N^r & \text{(encryption)} & . \end{array}$$

Here, $b \in \mathbf{Bool}$, $K \in \mathbf{Keys}$ and $r \in \mathbf{Rnd}$. Clearly, we can see that composed keys are allowed in the encryption operation. As the labels r are identifiers of invocations of the encryption algorithm, we demand that whenever we consider two expressions $\{M\}_N^r$ and $\{M'\}_{N'}^r$ with the same label r, then also $M = M'$ and $N = N'$.

Even though Abadi and Rogaway [3] did not use formal random numbers, they assumed that each occurrence of the encryption constructor represents a different invocation of the encryption operation. Furthermore, the later work of Abadi and Jürjens [2] considered a richer language in which they also needed to keep track of different invocations. This was done, similarly to the present work, by using formal random numbers.

Let us define some notation related to the structure of formal expressions. The *subexpression relation* $\sqsubseteq$ is the smallest reflexive transitive relation over **Exp** containing $M \sqsubseteq (M,N)$, $N \sqsubseteq (M,N)$, $M \sqsubseteq \{M\}_N^r$ and $N \sqsubseteq \{M\}_N^r$ for all $M,N \in \mathbf{Exp}$ and $r \in \mathbf{Rnd}$. For an expression M, we denote:

$$\begin{aligned} \mathit{keys}(M) &:= \{K \in \mathbf{Keys} \,:\, K \sqsubseteq M\} \\ \mathit{rns}(M) &:= \{r \in \mathbf{Rnd} \,:\, \{N'\}_N^r \sqsubseteq M \text{ for some } N', N \in \mathbf{Exp}\} \\ \mathit{atoms}(M) &:= \mathit{keys}(M) \cup \mathit{rns}(M) \end{aligned}$$

We call the elements of $\mathit{atoms}(M)$ the *atoms* of M.

Intuitively, a *formal pattern* describes what an adversary is able to see when looking at an expression. The elements P, Q of the set of *formal patterns* **Pat** is defined by the following grammar:

$$\begin{aligned} P, Q ::= &\; b && \text{(bit)} \\ \mid &\; K && \text{(key)} \\ \mid &\; (P,Q) && \text{(pair)} \\ \mid &\; \{P\}_Q^r && \text{(encryption)} \\ \mid &\; \Box^r && \text{(undecryptable)}\ . \end{aligned}$$

Here, $\Box^r$ denote ciphertexts that are encrypted with a key that the adversary does not know, and thus can not "see" inside. We use formal random numbers to differentiate between these ciphertexts, and therefore we require that the formal random numbers used at encryptions be different from formal random numbers used at undecryptables. Now, the relation $\sqsubseteq$, as well as the functions *keys*, *rns* and *atoms* are extended to **Pat**. Finally, note that the sets $\mathit{rns}(P)$ and $\mathit{atoms}(P)$ also contain formal random numbers at the undecryptables.

3 Computational interpretation

In the computational model, an encryption system is a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ working with bit-strings. Here, $\mathcal{G}$ and $\mathcal{E}$ are probabilistic algorithms while $\mathcal{D}$ is deterministic. The *key generation algorithm* $\mathcal{G}$ takes as input the security parameter n, represented in unary, and returns a new key. The *encryption algorithm* $\mathcal{E}$ takes as input the security parameter, a key and a plaintext and produces a corresponding ciphertext. Since $\mathcal{E}$ is probabilistic, different invocations of $\mathcal{E}$ may return different ciphertexts. Lastly, the *decryption algorithm* $\mathcal{D}$ takes as input the security parameter, a key and a ciphertext and returns the corresponding plaintext.

Let $\mathbf{0}$ be a fixed bit-string. We say that the encryption system $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is *type-0 secure* [3] if, for all probabilistic polynomial-time (PPT) algorithms $\mathcal{A}^{(\cdot),(\cdot)}$ (with interfaces to two oracles), the difference of probabilities

$$\Pr[\mathcal{A}^{\mathcal{E}(1^n,k,\cdot),\mathcal{E}(1^n,k',\cdot)}(1^n) = 1 \,:\, k, k' \leftarrow \mathcal{G}(1^n)] - \\ \Pr[\mathcal{A}^{\mathcal{E}(1^n,k,\mathbf{0}),\mathcal{E}(1^n,k,\mathbf{0})}(1^n) = 1 \,:\, k \leftarrow \mathcal{G}(1^n)]$$

is negligible in n. A function is negligible if its reciprocal grows faster than any polynomial. In [3], Abadi and Rogaway showed that type-0 security is achievable under standard cryptographic assumptions.

Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a type-0 secure encryption system, such that the distribution $\mathcal{G}(1^n)$ is the uniform probability distribution over $\{0,1\}^{\ell(n)}$, where ℓ is a fixed polynomial. Now, being type-0 secure guarantees that the algorithm $\mathcal{E}$ is probabilistic, and thus we denote by $\mathcal{E}^{\mathbf{r}}$ the invocation of $\mathcal{E}$ with random coin-flips $\mathbf{r} \in \{0,1\}^*$. Thus, if we fix $\mathbf{r}$, the algorithm $\mathcal{E}^{\mathbf{r}}$ is now deterministic. In the security definition, we assume the uniform distribution of $\mathbf{r}$.

Now, let x be a bit-string. A *random oracle* R is a machine that, on query $(1^m, x)$, first checks whether it has been queried with the same values before. If this is the case, then it returns the same answer as before. Otherwise, it proceeds as follows. First, the random oracle creates, uniformly and randomly, a bit-string y of length m. Then, the random oracle records the query $(1^m, x)$ together with y, and then finally y is returned. In the *random oracle model* [7], there is a single random oracle in the world, while all other algorithms and machines are allowed to query this oracle. To be able to translate a model in the random oracle world into a real system, the random oracle needs to be replaced with some "random-looking" function h. Thus, there is a leap of faith involved in applying the results proved in the random oracle model to a real system. Nevertheless, we can still be sure that if the real system is insecure, then this must be caused by h not being a good approximation of R.

Now we are ready to give a computational interpretation to expressions and patterns. With each $P \in \mathbf{Pat}$ we associate a family (indexed by the security parameter) of probability distributions over bit-strings. We denote that family by $[\![P]\!]$. Fig. 1 depicts the algorithm sampling the n-th distribution in that family. First, $\textsc{Initialize}(1^n, P)$ is run and then $\textsc{Convert}(1^n, P)$ is invoked.

```
algorithm INITIALIZE(1^n, P)
    for all K ∈ keys(P) do τ(K) ← G(1^n)
    τ_bb ← G(1^n)
    for all r ∈ rns(P) do τ(r) ∈_R {0,1}*

algorithm CONVERT(1^n, P)
    if P is K ∈ Keys
        return ⟨τ(K), "key"⟩
    else if P is b ∈ Bool
        return ⟨b, "bit"⟩
    else if P is (P_1, P_2)
        let x = CONVERT(1^n, P_1)
        let y = CONVERT(1^n, P_2)
        return ⟨x, y, "pair"⟩
    else if P is {P_2}^r_{P_1}
        let x = CONVERT(1^n, P_1)
        let y = CONVERT(1^n, P_2)
        let z = E^{τ(r)}(1^n, R(1^{ℓ(n)}, x), y)
        return ⟨z, "ciphertext"⟩
    else: P is □^r
        let z = E^{τ(r)}(1^n, R(1^{ℓ(n)}, τ_bb), 0)
        return ⟨z, "ciphertext"⟩
```

Fig. 1. Algorithm sampling $[\![P]\!]$

Note that if $P_1 \neq P_2$ then if we sample $\langle x, y, \text{“pair”}\rangle \leftarrow [\![(P_1, P_2)]\!]_n$, then the probability for $x = y$ is negligible.

In fact, the existence of the random oracle is itself sufficient for the existence of type-0 encryption systems. In particular, we could have fixed the encryption system, for example, to the one given in [8]. However, we would like to use the random oracle as little as possible and thus we have not fixed it.

Two families of probability distributions over bit-strings D and D' are *indistinguishable* (denoted $D \approx D'$) if for all PPT algorithms $\mathcal{A}$, the difference of probabilities

$$\Pr[\mathcal{A}(1^n, x) \,:\, x \leftarrow D_n] - \Pr[\mathcal{A}(1^n, x) \,:\, x \leftarrow D'_n]$$

is negligible in n. In fact, indistinguishability is the computational equivalent of sameness.

4 Equivalence relation on Pat

We would like to define an equivalence relation $\cong$ over formal expressions (and more generally, over patterns), such that $M \cong N$ implies $[\![M]\!] \approx [\![N]\!]$. Similarly to Abadi and Rogaway, we define a function *pattern* : **Exp** $\longrightarrow$ **Pat** and state $M \cong N$ iff *pattern*(M) and *pattern*(N) can be obtained from each other by an α-conversion over keys and formal random numbers. Even though we could also define the function *pattern* similarly to [3], that is by giving the entailment relation $\vdash$ (this relation describes which formal expressions the Dolev-Yao attacker may obtain from a given expression) and replacing the undecryptable encryptions in the expressions by the corresponding "blobs" $\Box$, we chose to give a different equivalence definition. The reason for this is that, if we followed Abadi and Rogaway, then we would have to assume that the expressions M and N do not contain *encryption cycles*. With atomic keys only, an encryption cycle in an expression M is a sequence of keys $K_1, \ldots, K_m$, where K_i is encrypted under K_{i+1} (possibly indirectly) for all $i \in \{1, \ldots, m-1\}$ and K_m is encrypted under K_1, all in the expression M. Even though the definition of type-0 security does not say anything about the security of encryption cycles; in systems where the Abadi and Rogaway results can be applied, encryption cycles cannot occur. However, when considering composed keys, the definition of encryption cycles is likely much more contrieved, because the different parts of the same key have to be kept track of. Therefore, we avoid defining the encryption cycles at all, and thus our definition of $\cong$ applies to all expressions.

Let $P, Q \in$ **Pat**. The operation $\mathsf{box}_Q(P)$ replaces all encryptions of the form $\{\cdot\}_Q^r$ occurring in P with undecryptables. Formally, the operation $\mathsf{box}_Q(P)$ is defined by

$$\begin{aligned}
\mathsf{box}_Q(b) &= b \\
\mathsf{box}_Q(K) &= K \\
\mathsf{box}_Q((P_1, P_2)) &= (\mathsf{box}_Q(P_1), \mathsf{box}_Q(P_2)) \\
\mathsf{box}_Q(\{P_2\}_{P_1}^r) &= \begin{cases} \Box^r, & \text{if } P_1 = Q \\ \{\mathsf{box}_Q(P_2)\}_{\mathsf{box}_Q(P_1)}^r, & \text{if } P_1 \neq Q \end{cases} \\
\mathsf{box}_Q(\Box^r) &= \Box^r \ .
\end{aligned}$$

We are looking for sufficient conditions for $[\![P]\!] \approx [\![\mathsf{box}_Q(P)]\!]$. In particular, we are going to prove that the following is a sufficient condition. Let T_P be the set of all atoms occurring in P, except that if P has subexpressions of the form $\{\cdot\}_Q^r$, then the keys and random numbers inside that Q do not count. The sufficient condition that we are looking for is $atoms(Q) \nsubseteq T_P$. To state this formally, we define the sets $\mathcal{B}_Q(P)$ for all $P \in \mathbf{Pat}$ in the following way:

$$\begin{aligned}
\mathcal{B}_Q(b) &= \emptyset \\
\mathcal{B}_Q(K) &= \{K\} \\
\mathcal{B}_Q((P_1, P_2)) &= \mathcal{B}_Q(P_1) \cup \mathcal{B}_Q(P_2) \\
\mathcal{B}_Q(\{P_2\}^r_{P_1}) &= \begin{cases} \{r\} \cup \mathcal{B}_Q(P_2), & \text{if } P_1 = Q \\ \{r\} \cup \mathcal{B}_Q(P_1) \cup \mathcal{B}_Q(P_2), & \text{if } P_1 \neq Q \end{cases} \\
\mathcal{B}_Q(\Box^r) &= \{r\}
\end{aligned}$$

and set $T_P := \mathcal{B}_Q(P)$.

If the above condition is fulfilled we say that $P \cong \mathsf{box}_Q(P)$. We also say $P \cong Q$ whenever Q can be obtained from P through some α-conversion applied to its formal keys and random numbers. Finally, we extend $\cong$ to an equivalence relation.

Now we are ready to define the function *pattern*. Let $P \in \mathbf{Pat}$. If there exists some $Q \in \mathbf{Pat}$, such that $P \neq \mathsf{box}_Q(P)$ (i.e. Q occurs as an encryption key somewhere in P) and $P \cong \mathsf{box}_Q(P)$ then we put $pattern(P) := pattern(\mathsf{box}_Q(P))$. Otherwise we put $pattern(P) := P$. It is easy to check that *pattern* is well-defined. Furthermore, the function *pattern* is efficiently computable.

5 Examples

Before going to the proof that equivalence implies indistinguishability of interpretations, let us see some examples. We are not going to repeat the examples given by Abadi and Rogaway [3][1]. In our examples we intend to illustrate and clarify what constitutes an "encryption cycle" in an expression and what does not.

– $\{K_1\}^r_{(K_1,K_2)} \cong \Box^r$. This is so since the atom K_2 of the encryption key (K_1, K_2) does not occur anywhere else.

[1] The reader checking out these examples should keep in mind that:

– [3] uses no formal random numbers; each occurrence of the encryption constructor is assumed to have a different formal random number attached to it;
– in [3], $M \cong N$ does not imply $[\![M]\!] \approx [\![N]\!]$, if M or N contains encryption cycles.

- $\{(K_1, K_2)\}^r_{(K_1,K_2)} \not\cong \Box^r$. This expression is a clear-cut encryption cycle. However, encryption cycles can be more subtle, as the next two examples show.
- $\{(K_2, K_1)\}^r_{(K_1,K_2)} \not\cong \Box^r$.
- $(\{K_1\}^{r_1}_{(K_1,K_2)}, \{K_2\}^{r_2}_{(K_1,K_2)}) \not\cong (\Box^{r_1}, \Box^{r_2})$.
- $\{\{K_2\}^{r_1}_{K_1}\}^r_{\{K_2\}^{r_1}_{K_1}} \not\cong \Box^r$, but $\{\{K_2\}^{r_2}_{K_1}\}^r_{\{K_2\}^{r_1}_{K_1}} \cong \Box^r$. The first example contains an encryption cycle. The second example, however, does not, because the atom r_1 of the key does not occur anywhere else. These two examples show the importance of formal random numbers.
- $(\{K_1\}^r_{(K_1,K_2)}, K_2) \not\cong (\Box^r, K_2)$. Compared to the first example, the addition of K_2 means that now all atoms of the encryption key occur somewhere else.

6 Correctness proof

Theorem. *Let $P, Q \in \mathbf{Pat}$, such that $atoms(Q) \not\subseteq \mathcal{B}_Q(P)$. Then $[\![P]\!] \approx [\![\mathsf{box}_Q(P)]\!]$.*

Proof. Assume the contrary, i.e. there is an algorithm $\mathcal{A}$ that can distinguish the families of probability distributions $[\![P]\!]$ and $[\![\mathsf{box}_Q(P)]\!]$. Fig. 2 shows an algorithm (first call INITIALIZE and then CONVERT) sampling either $[\![P]\!]$ or $[\![\mathsf{box}_Q(P)]\!]$, depending on the values of the oracles f and g. If f and g are $\mathcal{E}(1^n, k, \cdot)$ and $\mathcal{E}(1^n, k', \cdot)$, then the algorithm in Fig. 2 samples $[\![P]\!]$. If f and g are both $\mathcal{E}(1^n, k, \mathbf{0})$ then this algorithm samples $[\![\mathsf{box}_Q(P)]\!]$. Composing this algorithm with algorithm $\mathcal{A}$ allows us to break the type-0 security of the encryption system.

We have to show that the algorithm CONVERT$^{f(\cdot),g(\cdot)}$ can complete its job, i.e. it does not have to access τ at a point where it is undefined. If a key K occurs in P but does not belong to $\mathcal{B}_Q(P)$ then this key only occurs as a subexpression of Q, where Q is used as an encryption key in P. But CONVERT$^{f(\cdot),g(\cdot)}(1^n, Q, Q)$ is never needed in this context, the oracle f is used instead of encrypting with it. Therefore we do not need the value of K there. Similarly, if $r \in \mathbf{Rnd}$ occurs in P but not in $\mathcal{B}_Q(P)$ then we would need it only for computing the interpretation of the encryption key Q, which is not necessary to compute at all. If r belongs to the set subtracted from $\mathcal{B}_Q(P)$ in the algorithm INITIALIZE, then $\tau(r)$ is not used, but the oracles f or g generate some random numbers of their own.

We have to argue that the algorithm in Fig. 2 indeed samples the claimed families of distributions. Clearly, if f and g are both $\mathcal{E}(1^n, k, \mathbf{0})$, then CONVERT$^{f(\cdot),g(\cdot)}(1^n, P, Q)$ samples $[\![\mathsf{box}_Q(P)]\!]$. If f is $\mathcal{E}(1^n, k, \cdot)$ and

algorithm $\textsc{Initialize}(1^n, P, Q)$
    for all $K \in \mathcal{B}_Q(P)$ do $\tau(K) \leftarrow \mathcal{G}(1^n)$
    for all r in the set
        $\mathcal{B}_Q(P) \backslash \big(\{r' \in \mathbf{Rnd} \mid \Box^{r'} \text{ occurs in } P\} \cup \{r' \in \mathbf{Rnd} \mid \{\cdot\}_Q^{r'} \text{ occurs in } P\}\big)$
                do $\tau(r) \stackrel{R}{\in} \{0,1\}^*$

algorithm $\textsc{Convert}^{f(\cdot),g(\cdot)}(1^n, P, Q)$
    if $\textsc{Convert}^{f(\cdot),g(\cdot)}(1^n, P, Q)$ has been invoked before
        return the value returned previously
    if P is $K \in \mathbf{Keys}$
        return $\langle \tau(K), \text{“key”} \rangle$
    else if P is $b \in \mathbf{Bool}$
        return $\langle b, \text{“bit”} \rangle$
    else if P is (P_1, P_2)
        let $x = \textsc{Convert}^{f,g}(1^n, P_1, Q)$
        let $y = \textsc{Convert}^{f,g}(1^n, P_2, Q)$
        return $\langle x, y, \text{“pair”} \rangle$
    else if P is $\{P_2\}_{P_1}^r$
        if $P_1 = Q$
            let $y = \textsc{Convert}^{f,g}(1^n, P_2, Q)$
            let $z \leftarrow f(y)$
        else
            let $x = \textsc{Convert}^{f,g}(1^n, P_1, Q)$
            let $y = \textsc{Convert}^{f,g}(1^n, P_2, Q)$
            let $z = \mathcal{E}^{\tau(r)}(1^n, R(1^{\ell(n)}, x), y)$
        return $\langle z, \text{“ciphertext”} \rangle$
    else: P is $\Box^r$
        let $z \leftarrow g(\mathbf{0})$
        return $\langle z, \text{“ciphertext”} \rangle$

Fig. 2. Algorithm sampling either $[\![P]\!]$ or $[\![\mathsf{box}_Q(P)]\!]$

g is $\mathcal{E}(1^n, k', \cdot)$ then we have to show that the key k used by f is indistinguishable from $R(1^{\ell(n)}, \textsc{Convert}(1^n, Q, Q))$ even when we are given the values of τ on $\mathcal{B}_Q(P)$. The key used by f is independent of all the given values of τ. Also, these values of τ do not uniquely determine $\textsc{Convert}^{f,g}(1^n, Q, Q)$ yet, because $atoms(Q) \nsubseteq \mathcal{B}_Q(P)$. Even more, with given values of τ we can guess the value of $\textsc{Convert}^{f,g}(1^n, Q, Q)$ only with negligible success probability. Therefore the application of the random oracle to this value gives us a random bit-string that is independent of the given values of τ. The key used by f and the bit-string $R(1^{\ell(n)}, \textsc{Convert}(1^n, Q, Q))$ are identically distributed, therefore they are indistinguishable under given conditions. Hence $\textsc{Convert}^{f(\cdot),g(\cdot)}(1^n, P, Q)$ samples $[\![P]\!]$. $\square$

7 Conclusions

In this paper we have considered an extension of the work of Abadi and Rogaway [3]. This extension is mainly constituted by considering the use of composed, non-atomic keys in the encryption operator of the formal language. Briefly, we proceeded as follows: First, we related formal expressions in our language with an equivalence relation $\cong$. By providing an intuitive computational interpretation, and then showing that each time two formal expressions that are equivalent according to $\cong$ are also indistinguishable in the computational world, we have lifted the work of Abadi and Rogaway [3] to the case of composed keys.

As we already mentioned, support for encryption with composed keys is important since many cryptographic protocols use them [14, 1, 12]. Thus, having the soundness result for the case of formal encryption with composed keys provides further faithfulness in the verification results of formal approaches that support composed keys (such as [18, 10, 4].)

While giving the computational interpretation, we needed to use the random oracle. Thus, our approach gives less security guarantees than the original work of Abadi and Rogaway, based on standard security assumptions. However, we believe the use of the random oracle is necessary to guarantee the goodness and independence of the constructed keys. Usage of the random oracle allow us to model the situation in which a user generates keys in a completely secure manner, which is in accordance with the existing definitions in the computational model. However, in some situations (e.g. when considering composed keys), the key generation process may *not* be a so private activity. In this new setting, an adversary might have some knowledge about the randomness used during the key generation. Furthermore, a stronger and *active* adversary may even have some control over the key generation process. We believe it would be interesting to study such a new scenario, where new and proper definitions (and constructions) would be needed.

References

1. M. Abadi. Private authentication. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies*, pages 27–40. Springer-Verlag, 2003.
2. M. Abadi and J. Jurjens. Formal eavesdropping and its computational interpretation. In *Fourth International Symposium on Theoretical Aspects of Computer Software (TACS2001)*, LNCS. Springer-Verlag, 2001.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Journal of Cryptology*, number 15, pages 103–127. Springer-Verlag, 2000.
4. D. Basin, S. Modersheim, and L. Vigano. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Workshop on Security Protocol Verification. CONCUR 2003*, September 2003.
5. M. Bellare. Practice-oriented provable security. In *Information Security, First International Workshop, ISW '97*, LNCS 1396, pages 221–231, 1997.
6. M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT 2003*, LNCS 2656, pages 491–506, 2003.
7. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
8. J. Black, P. Rogaway, and T. Shrimpton. Encryption scheme security in the presence of key-dependent messages. In *Selected Areas in Cryptography — SAC'02*, LNCS. Springer-Verlag, 2002.
9. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.
10. R. Corin and S. Etalle. An Improved Constraint-based system for the verification of security protocols. *9th Int. Static Analysis Symp. (SAS)*, LNCS 2477:326–341, september 2002.
11. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. A. Freier, P. Karlton, and P. Kocher. The ssl protocol. version 3.0.
13. O. Goldreich. On the foundations of modern cryptography. *Lecture Notes in Computer Science*, 1294:46–74, 1997.
14. L. Gong. Using one-way functions for authentication. *ACM SIGCOMM Computer Communication Review*, 19(5):8–11, 1989.
15. P. Laud. Semantics and program analysis of computationally secure information flow. In *Programming Languages and Systems: 10th European Symposium on Programming, ESOP 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genoa, Italy, April 2-6, 2001*, volume 2028 of *LNCS*, pages 77–91. Springer-Verlag, 2001.
16. P. Laud. Encryption cycles and two views of cryptography. In *NORDSEC 2002 - Proceedings of the 7th Nordic Workshop on Secure IT Systems (Karlstad University Studies 2002:31)*, pages 85–100, 2002.

17. D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway language of encrypted expressions. To appear.
18. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communication Security*, volume Proc. 2001, pages 166–175. ACM press, 2001.
19. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.