Implementing Cryptographic Primitives in the Symbolic Model*

Peeter Laud

Cybernetica AS and Tartu University
peeter@cyber.ee

Abstract. When discussing protocol properties in the symbolic (Dolev-Yao; term-based) model of cryptography, the set of cryptographic primitives is defined by the constructors of the term algebra and by the equational theory on top of it. The set of considered primitives is not easily modifiable during the discussion. In particular, it is unclear what it means to define a new primitive from the existing ones, or why a primitive in the considered set may be unnecessary because it can be modeled using other primitives. This is in stark contrast to the computational model of cryptography where the constructions and relationships between primitives are at the very foundation of the theory. In this paper, we explore how a primitive may be constructed from other primitives in the symbolic model, such that no protocol breaks if an atomic primitive is replaced by the construction. As an example, we show the construction of (symbolic) "randomized" symmetric encryption from (symbolic) one-way functions and exclusive or.

1 Introduction

One of the main tasks of cryptographic research is the building of secure and efficient protocols needed in various systems, and the construction of primitives that these protocols need. In the computational model [12, 21] of cryptography, where messages are modeled as bit-strings and the adversary as a probabilistic polynomial-time adversary, these primitives are constructed from simpler primitives, all the way down to certain base primitives (one-way functions or trapdoor one-way functions). The security properties of constructed primitives are derived from the properties of the base primitives. In the further development, only derived properties are important, making the whole approach modular (in theory).

The research in the symbolic model of cryptography (also known as the *Dolev-Yao model*, or *perfect cryptography assumption*) [10] has so far almost fully concentrated on the construction and analysis of cryptographic protocols. The messages are modeled as elements of some term algebra where the constructors of that algebra are seen as abstractions of cryptographic algorithms. In these

* Supported by Estonian Science Foundation, grant #8124, by European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and by EU FP7-ICT Project HATS

treatments, the set of constructors has been fixed, causing the set of primitives (in our treatment, a primitive is a set of cryptographic algorithms) also to be fixed. There is no notion of implementing a primitive using some already existing primitives. This can hinder the generalization of certain kinds of results. For example, as stated, the impossibility result of [13] only applies to hash functions and XOR operations.

Another obvious application of our result is the modularization of security proofs of protocols in the symbolic model. While the symbolic model is generally more amenable to automatic analysis, certain commonly-used operations (exclusive or, and to lesser extent, Diffie-Hellman computation) are only handled with difficulty. Certain other operations (addition and multiplication, equipped with the theory for rings) are not handled at all. If these primitives are only used in a certain manner (e.g. to define the session key) then the construction of messages containing the uses of those primitives can be seen as a primitive itself, which may have properties that are simpler to handle by the automatic analysis.

The main difficulty in defining that a primitive has been securely implemented by a set of messages with variables is the difference in signatures. In general, the implementation satisfies different (more) equalities than the primitive. Hence the set of meaningful operations is richer and a simple observational equivalence is not a useful definition of security. In this paper, we give a suitable definition that in our opinion precisely captures the intuition of the equivalence of processes using the primitive operation and the processes using the implementation. We propose a technique for proving the security of the implementation. The technique does not require the prover to universally quantify over processes and contexts, but just provide an observationally equivalent process to a specific, the "most powerful" attacker against processes using the implementation of the primitive. We apply the technique by providing a secure implementation for the randomized symmetric encryption primitive in terms of one-way hash functions and exclusive or, thereby generalizing our impossibility result [13].

The paper is structured as follows. Sec. 2 gives the necessary background on process calculi, introducing the applied pi-calculus that we will be working with. Sec. 3 provides the main definitions and proof techniques, while Sec. 4 applies them to the randomized symmetric encryption primitive. Finally, Sec. 5 reviews related work and Sec. 6 concludes.

2 Applied pi-calculus

Let us recall the syntax and semantics of the applied pi-calculus [2], in which our results will be stated. We have a countable set *Vars* of *variables*, ranged over by $x, y, \ldots$, and a countable set *Names* of *names*, ranged over by $m, n, \ldots$. We let $u, v, \ldots$ range over both names and variables. Additionally, we have a set Σ of *function symbols*, ranged over by $f, g, \ldots$, each with a fixed arity. Function symbols abstract cryptographic and other operations used by protocols. In the current paper, more often occurring function symbols besides tupling and projections are the ternary *randomized encryption* $\mathsf{Enc}(R, K, X)$ and binary *de-*

cryption $\mathsf{Dec}(K,Y)$, as well as the unary one-way (or hash) function $\mathsf{H}(X)$ and the binary XOR (written using infix notation) $X \oplus Y$ together with its nullary neutral element $\mathsf{0}$. A *term* in signature Σ, ranged over by $M, N, \ldots$ is either a name, a variable, or a function application $f(M_1, \ldots, M_k)$, where the arity of f is k. Let $\mathbf{T}_{\Sigma}(\mathit{Vars} \cup \mathit{Names})$ denote all terms over the signature Σ, where the atomic terms belong to the set $\mathit{Vars} \cup \mathit{Names}$.

An *equational theory* E is a set of pairs of terms in signature Σ. It defines a relation $=_E$ on terms which is the smallest congruence containing E and is closed under the substitution of terms for variables and bijective renaming of names [20]. Equational theories capture the relationships between primitives defined in Σ. The properties of tupling are captured by the equations $\pi_i^n((x_1, \ldots, x_n)) =_E x_i$ for all i and n (we let π_i^n denote the i-th projection from an n-tuple). Encryption and decryption are related by $\mathsf{Dec}(k, \mathsf{Enc}(r, k, x)) =_E x$. The XOR-operation has its own set of equations capturing commutativity ($x \oplus y =_E y \oplus x$), associativity ($(x \oplus y) \oplus z =_E x \oplus (y \oplus z)$), unit ($x \oplus \mathsf{0} =_E x$) and cancellation ($x \oplus x = \mathsf{0}$). No equations are necessary to capture the properties of H. If E is clear from the context, we abbreviate $M =_E N$ as $M = N$.

Processes, ranged over by $P, Q, \ldots$, *extended processes*, ranged over by $A, B, \ldots$, and their structural equivalence are defined in Fig. 1. We use $\overrightarrow{x}$ and $\overrightarrow{M}$ to denote a sequence of variables or terms. In the if-statement, the symbol $=$ denotes equality modulo the theory E, not syntactic equality. The extended process $\{^M/_x\}$ represents a process that has previously output M which is now available to the environment through the handle x. Variable x is free in $\{^M/_x\}$. As indicated by the structural equivalence, $\{^M/_x\}$ can replace the variable x with M in any process it comes to contact with under the scope of νx. Here $A[x \leftarrow M]$ denotes the process A where all free occurrences of the variable x have been replaced with the term M, without capturing any free variables in M. If $\{^M/_x\}$ is outside the scope of νx then we say that the variable x is *exported*. The *domain* of an extended process is the set of variables it exports. An extended process is *closed* if it exports all its free variables. The internal reduction relation describes a single step in the evolution of a process. As usual, we consider only *well-sorted* processes: all variables and names have *sorts* and all operations (conditional checking, communication, substitution) must obey them. In our sort system, there is a sort Data; the inputs and output of any function symbol have this sort. For any sequence of sorts $T_1, \ldots, T_l$ there is also a sort $\mathcal{C}\langle T_1, \ldots, T_l \rangle$ for channels communicating values of that sort. Let $\mathbf{Proc}(\Sigma)$ and $\mathbf{Ctxt}(\Sigma)$ denote the sets of all extended processes and evaluation contexts with function symbols from the set Σ. We refer to [2, 20] for details and justifications.

In this paper we want to state that two processes, where the second has been obtained from the first by replacing in it certain term constructors with their "implementations", are somehow indistinguishable. *Observational equivalence*, denoted $\approx$ is the standard notion capturing indistinguishability by all environments. For defining it, we denote with $A \Downarrow c$ the existence of an evaluation context C not binding c, a term M and a process P, such that $A \rightarrow^* C[\overline{c}\langle M \rangle.P]$.

$$P ::= \mathbf{0} \mid P \mid Q \mid\ !P \mid \nu n.P \mid \textbf{if } M = N \textbf{ then } P \textbf{ else } Q \mid u(\overrightarrow{x}).P \mid \overline{u}\langle\overrightarrow{M}\rangle.P$$
$$A ::= P \mid A \mid B \mid \nu n.A \mid \nu x.A \mid \{^M/_x\} \qquad C ::= [\,] \mid A \mid C \mid C \mid A \mid \nu n.C \mid \nu x.C$$

$$A \equiv A \mid \mathbf{0} \qquad A \mid (B \mid C) \equiv (A \mid B) \mid C \qquad A \mid B \equiv B \mid A \qquad !P \equiv P|!P$$
$$\nu n.\mathbf{0} \equiv \mathbf{0} \qquad \nu u \nu v.A \equiv \nu v \nu u.A \qquad A \mid \nu u.B \equiv \nu u.(A \mid B) \text{ if } u \text{ not free in } A$$
$$\nu x.\{^M/_x\} \equiv \mathbf{0} \qquad \{^M/_x\} \mid A \equiv \{^M/_x\} \mid A[x \leftarrow M] \qquad \{^M/_x\} \equiv \{^N/_x\} \text{ if } M =_E N$$
$$\overline{c}\langle\overrightarrow{x}\rangle.P \mid c(\overrightarrow{x}).Q \to P \mid Q \qquad \textbf{if } N = N \textbf{ then } P \textbf{ else } Q \to P$$
$$\textbf{if } M = N \textbf{ then } P \textbf{ else } Q \to Q \text{ for ground terms } M \text{ and } N \text{, where } M \neq_E N$$
$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q} \qquad \frac{A \to B}{C[A] \to C[B]}$$

Fig. 1. Applied pi calculus processes and extended processes, evaluation contexts, structural equivalence $\equiv$, and internal reduction $\to$ [20]

Definition 1. *The* observational equivalence *is the largest symmetric relation $\mathcal{R}$ over closed extended processes with the same domain such that $A \mathcal{R} B$ implies (a) if $A \Downarrow c$ for some c, then $B \Downarrow c$; (b) if $A \to^* A'$, then $B \to^* B'$ and $A' \mathcal{R} B'$ for some B'; (c) $C[A] \mathcal{R} C[B]$ for all closing evaluation contexts C. [20]*

3 Secure Implementation of Primitives

We start by defining the notions of *cryptographic primitive* and *implementing* a cryptographic primitive. A cryptographic primitive Prim is a subset of Σ, e.g. the randomized symmetric encryption primitive is $\mathsf{R\text{-}ENC} = \{\mathsf{Enc}, \mathsf{Dec}, \mathsf{e\_r}, \mathsf{is}_{enc}\}$. Beside the encryption and decryption function we also have a unary *randomness extraction* function with the equality $\mathsf{e\_r}(\mathsf{Enc}(r, k, x)) =_E r$ and the unary *type verifier* with the equality $\mathsf{is}_{enc}(\mathsf{Enc}(r, k, x)) =_E \mathsf{true}$, where true is a nullary operation. Many implementations of symmetric randomized encryption (in the computational model) allow the randomness used in encryption to be recovered from the ciphertext, and our intended implementation has the same property. The possibilities it gives to the adversary must be reflected at the primitive level. Another reason for including $\mathsf{e\_r}$ and is_{enc} in $\mathsf{R\text{-}ENC}$ is, that the security proof of our implementation in Sec. 4 makes significant used of these symbols.

An implementation assigns to each function symbol $f \in \mathsf{Prim}$ a term f^{i} over Σ with no free names and with free variables $x_1, \ldots, x_{\mathrm{arity}(f)}$. The implementation defines a mapping (a second-order substitution) tr from terms to terms, replacing each occurrence of each $f \in \mathsf{Prim}$ in the term with f^{i}. Formally,

$$\begin{aligned} tr(u) &= u \\ tr(f(M_1, \ldots, M_n)) &= f(tr(M_1), \ldots, tr(M_n)) && \text{if } f \notin \mathsf{Prim} \\ tr(f(M_1, \ldots, M_n)) &= f^{\mathsf{i}}[x_1 \leftarrow tr(M_1), \ldots, x_n \leftarrow tr(M_n)] && \text{if } f \in \mathsf{Prim}. \end{aligned}$$

To make the translation well-behaved with respect to the equational theory E we require $tr(M) =_E tr(N)$ for each $(M, N) \in E$. The mapping tr can be straightforwardly extended to processes, extended processes and evaluation contexts. When defining secure implementations, we want to state that A and $tr(A)$ are somehow indistinguishable for all extended processes A.

Example 1. We can give the following implementation to the randomized symmetric encryption primitive $\mathsf{R\text{-}ENC}$. Let $\mathsf{eq?}$ be a ternary function symbol. Let $\mathsf{eq?}(x, x, y) =_E y$ be a pair of terms in the equational theory E. Let the implementation of $\mathsf{R\text{-}ENC}$ be

$$\begin{aligned}
\mathsf{Enc}^{\mathsf{i}} &= (x_1, H(x_2, H(x_2, x_1, x_3)) \oplus x_3, H(x_2, x_1, x_3))\\
\mathsf{Dec}^{\mathsf{i}} &= \mathsf{eq?}(H(x_1, \pi^3_1(x_2), H(x_1, \pi^3_3(x_2)) \oplus \pi^3_2(x_2)), \pi^3_3(x_2), H(x_1, \pi^3_3(x_2)) \oplus \pi^3_2(x_2))\\
\mathsf{e\_r}^{\mathsf{i}} &= \pi^3_1(x_1)\\
\mathsf{is}^{\mathsf{i}}_{enc} &= \mathsf{eq?}((\pi^3_1(x_1), \pi^3_2(x_1), \pi^3_3(x_1)), x_1, \mathsf{true})
\end{aligned}$$

(recall that the arguments of Enc were the formal randomness x_1, the key x_2 and the plaintext x_3, while the arguments of Dec were the key x_1 and the ciphertext x_2). The application of H to several arguments denotes the application of H to the tuple of these arguments.

Our implementation of $\mathsf{Enc}(r, k, x)$ is similar to the OFB- or CTR-modes of operation of block ciphers [11]. The randomness r (the initialization vector IV) is included in the ciphertext and used, together with the key k, to generate a random-looking sequence which is then reversibly combined with the plaintext x. Hence the result of the OFB- or CTR-mode can be modelled as $(r, H(k, r) \oplus x)$. Formal encryption also provides integrity of the plaintext. Thus we add the third component $H(k, r, x)$ as a formal *message authentication code*. Finally, we note that if the randomness r is reused (this case is ruled out in computational definitions, but in this paper we are considering the most general way of using the primitives), then the adversary is able to find $x \oplus x'$ from the implementations of $\mathsf{Enc}(r, k, x)$ and $\mathsf{Enc}(r, k, x')$ by XOR-ing their second components. Such recovery of $x \oplus x'$ is not possible with primitive Enc. We rule out the reuse of randomness by making it depend also on the key and the plaintext — we replace r in the second component $H(k, r) \oplus x$ by $H(k, r, x)$. In this construction, similarities to the *resettable encryption* [22] can be seen. But the use of $H(k, r, x)$ in two roles seems to be novel to our construction.

The decryption $\mathsf{Dec}(k, y)$ recovers the plaintext by extracting $H(k, r, x)$ from y, XOR-ing the second component of y with $H(k, H(k, r, x))$, and checking the authentication tag.

The only pair in E relating Enc and Dec to other primitives (or each other) is $\mathsf{Dec}(k, \mathsf{Enc}(r, k, x)) =_E x$. It is simple to verify that $\mathsf{Dec}^{\mathsf{i}}(k, \mathsf{Enc}^{\mathsf{i}}(r, k, x) =_E x$. Also, obviously $\mathsf{e\_r}^{\mathsf{i}}(\mathsf{Enc}^{\mathsf{i}}(r, k, x)) =_E r$ and $\mathsf{is}^{\mathsf{i}}_{enc}(k, \mathsf{Enc}^{\mathsf{i}}(r, k, x)) =_E \mathsf{true}$. The implementation of is_{enc} is unsatisfactory because it declares all triples to be ciphertexts, but in the following we'll see how to improve it.

Defining the secureness of the implementation as $A \approx tr(A)$ for all extended processes A immediately leads to problems. Consider e.g. the following process

$$\nu r \nu k \nu x.(\bar{c}\langle \mathsf{Enc}(r,k,x)\rangle | c(y).\mathbf{if}\ y = (\pi_1^3(y), \pi_2^3(y), \pi_3^3(y))\ \mathbf{then}\ P_{Impl}\ \mathbf{else}\ P_{Prim})\ . \tag{1}$$

It generates a ciphertext and then proceeds to check whether it is a triple or not. A triple means that we are dealing with the implementation, while a non-triple is a sign of using primitive encryption. Clearly, we have to restrict the processes A if we want to have a meaningful definition. There should be a set of function symbols $\Sigma_{\mathsf{fb}} \subseteq \Sigma$ that it is forbidden to apply. These function symbols express the "internal details" of the implementation of the primitive. A process just using the primitive should have no need to use them. Denote $\Sigma_{\mathsf{ok}} = \Sigma \backslash \Sigma_{\mathsf{fb}}$.

Example 2. It is unreasonable to restrict A from constructing and decomposing triples. We thus introduce $\bar{(}\cdot,\cdot,\cdot\bar{)}$ as the *tagged* [9, 6] version of tripling and $\bar{H}$ as the tagged version of hashing (these can be thought of as normal operations with one extra argument that is fixed as a constant that is used nowhere else). We use these operations to implement R-ENC. The use of these operations, as well as the projections $\bar{\pi}_1$, $\bar{\pi}_2$ and $\bar{\pi}_3$ from tagged triples, should be unnecessary for any process A whose security we care about (and for which we desire $A \approx tr(A)$).

Continuing our running example, we redefine

$$\begin{aligned}
\mathsf{Enc}^{\mathsf{i}} &= \bar{(}x_1, \bar{H}(x_2, \bar{H}(x_2,x_1,x_3)) \oplus x_3, \bar{H}(x_2,x_1,x_3)\bar{)} \\
\mathsf{Dec}^{\mathsf{i}} &= \mathsf{eq?}(\bar{H}(x_1, \bar{\pi}_1(x_2), \bar{H}(x_1, \bar{\pi}_3(x_2)) \oplus \bar{\pi}_2(x_2)), \bar{\pi}_3(x_2), \bar{H}(x_1, \bar{\pi}_3(x_2)) \oplus \bar{\pi}_2(x_2)) \\
\mathsf{e\_r}^{\mathsf{i}} &= \bar{\pi}_1(x_1) \\
\mathsf{is}^{\mathsf{i}}_{enc} &= \mathsf{eq?}(\bar{(}\bar{\pi}_1(x_1), \bar{\pi}_2(x_1), \bar{\pi}_3(x_1)\bar{)}, x_1, \mathsf{true})
\end{aligned}$$

The forbidden set of function symbols is $\Sigma_{\mathsf{fb}} = \{\bar{H}, \bar{\pi}_1, \bar{\pi}_2, \bar{\pi}_3, \bar{(}\cdot\bar{)}\}$. The newly introduced symbols are related to each other by $\bar{\pi}_i(\bar{(}x_1,x_2,x_3\bar{)}) =_E x_i$. No other $(M,N) \in E$ contains those function symbols. We see that the tagging of triples induces the tagging also for encryptions.

In the definition of observational equivalence, the usage of function symbols in Σ_{fb} has to be restricted in evaluation contexts, too, or the test (1) can still be performed in cooperation with the process A (generating the ciphertext) and the context (testing whether it is a tagged triple). We see that the contexts must also be translated — if A is enveloped by $C[\,]$, then $tr(A)$ should be enveloped by $tr(C)[\,]$. This models the fact that both A and C implement the primitive Prim in the same manner (otherwise they would be different primitives). Thus we modify the notion of observational equivalence as follows.

Definition 2. *The* observational equivalence modulo implementation, *denoted $\approx_{tr}$, is the largest relation $\mathcal{R}$ over closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies*

1. *$A \Downarrow c$ if and only if $B \Downarrow c$, for all channel names c;*
2. *if $A \rightarrow^* A'$, then there exists a process B', such that $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$;*

3. if $B \to^* B'$, then there exists a process A', such that $A \to^* A'$ and $A' \mathcal{R} B'$;
4. $C[A] \mathcal{R}\ tr(C)[B]$ for all closing evaluation contexts $C \in \mathbf{Ctxt}(\Sigma_{\mathsf{ok}})$

While we can show that $A \approx_{tr} tr(A)$ for all extended processes in the randomized symmetric encryption example, the relation $\approx_{tr}$ also does not satisfactorily capture the meaning of secure implementation. Namely, the context is restricted in the operations it can perform; as it cannot use the function symbols in Σ_{fb}, it cannot attack the implementation of the cryptographic primitive. We would like to have a simulation-based definition — for any attacker D attacking $tr(A)$ there is an attacker S attacking A, such that $A \mid S$ and $tr(A) \mid D$ are indistinguishable [7, 17]. This motivates our definition of secure implementation.

Definition 3. *Let Σ be a signature and E an equational theory over it. An* implementation *of a cryptographic primitive* $\mathsf{Prim} \subseteq \Sigma = \Sigma_{\mathsf{ok}} \dot\cup \Sigma_{\mathsf{fb}}$ *with forbidden symbols Σ_{fb} is* secure *if for any closed process $A \in \mathbf{Proc}(\Sigma_{\mathsf{ok}})$ and any closed process $D \in \mathbf{Proc}(\Sigma)$ (the adversary) there exists a closed process $S \in \mathbf{Proc}(\Sigma_{\mathsf{ok}})$ (the simulator), such that $A \mid S \approx_{tr} tr(A) \mid D$. Here the mapping tr is induced by the implementation.*

The definition captures the notion of $tr(A)$ being *at least as secure as A* — anything that the environment $tr(C)$ can experience when interacting with $tr(A)$, it (as C) can also experience when interacting with A. Hence, if nothing bad can happen to C when running together with A, then nothing bad can happen to $tr(C)$ when running together with $tr(A)$. The first can be established by analyzing A (and possibly C), without considering the implementation details of the cryptographic primitive.

An immediate consequence of the definition is, that the process A we're trying to protect does not have to be quantified over.

Proposition 1. *Let an implementation of a cryptographic primitive* $\mathsf{Prim} \subseteq \Sigma = \Sigma_{\mathsf{ok}} \dot\cup \Sigma_{\mathsf{fb}}$ *with forbidden symbols Σ_{fb} be given; let the mapping tr be induced by the implementation. If for any closed process $D \in \mathbf{Proc}(\Sigma)$ there exists a closed process $S \in \mathbf{Proc}(\Sigma_{\mathsf{ok}})$, such that $S \approx_{tr} D$, then the implementation is secure.*

Proof. Let $A \in \mathbf{Proc}(\Sigma_{\mathsf{ok}})$ and $D \in \mathbf{Proc}(\Sigma)$ be closed processes. By the premise of the proposition, there exists a closed process $S \in \mathbf{Proc}(\Sigma_{\mathsf{ok}})$, such that $S \approx_{tr} D$. Consider the context $C[\,] = A \mid [\,]$. It does not use symbols in Σ_{fb}. Item 4 of the definition of $\approx_{tr}$ implies that $A|S = C[S] \approx_{tr} tr(C)[D] = tr(A)|D$. □

We propose the following method for showing the security of a certain implementation. We rewrite any process D as $\nu c_{\mathsf{q}}.(D_{\mathsf{ctrl}} \mid VM)$ where VM does not depend on D (it only depends on Σ) and D_{ctrl} does not contain any function symbols. Intuitively, the process D_{ctrl} sends computation requests to the process VM (the "virtual machine") which performs those computations and stores their results in its database, responding with *handles* (new names) that the process D_{ctrl} can later use to refer to them. The channel c_{q} is used for communication between the two processes. We then construct a process VM_{sim} and show that $VM_{\mathsf{sim}} \approx_{tr} VM$ (this construction is primitive-specific). As $tr(D_{\mathsf{ctrl}}) = D_{\mathsf{ctrl}}$, we deduce that $\nu c_{\mathsf{q}}.(D_{\mathsf{ctrl}} \mid VM_{\mathsf{sim}}) \approx_{tr} \nu c_{\mathsf{q}}.(D_{\mathsf{ctrl}} \mid VM)$.

By defining a suitable bisimulation [2], it will be straightforward to show that $D \approx \nu c_{\mathsf{q}}.(D_{\mathsf{ctrl}} \mid VM)$. To complete the security proof, we only need refer to the following proposition that is given here in a somewhat more general form.

Proposition 2. *Let A_1, A_2, B_1, B_2 be four closed extended processes with the same domain. If $A_1 \approx A_2$, $A_2 \approx_{tr} B_1$ and $B_1 \approx B_2$, then $A_1 \approx_{tr} B_2$.*

Proof. Co-induction over the definition of $\approx_{tr}$. Consider the relation $\approx \circ \approx_{tr} \circ \approx$. It is easy to verify that it satisfies the requirements put on relations $\mathcal{R}$ in Def. 2. Hence $(\approx \circ \approx_{tr} \circ \approx) \subseteq \approx_{tr}$. $\square$

The process VM is depicted in Fig. 2. We use syntactic sugar $u(=\text{key}, \overrightarrow{x}).P$ for the process $u(z, \overrightarrow{x}).\mathbf{if}\ z = \text{key}\ \mathbf{then}\ P\ \mathbf{else}\ \overline{u}\langle z, \overrightarrow{x}\rangle$ that reads a tuple of values from the channel u and continues as P, with the restriction that the first component of the tuple must be equal to key. The VM process can "obey" the commands for putting a new value in the database (input: the value; output: a handle to it), getting a value from the database (input: handle; output: corresponding value) and applying a function symbol f to the values in the database (input: handles to arguments; output: handle to result). Here "put", "get", and "comp$_f$" for each function symbol $f \in \Sigma$ are fixed free names. The process VM gives its output on a channel c_b that is given together with the input.

$$\begin{aligned}VM = \nu c_{\mathsf{int}}.\big(!\big(&c_{\mathsf{q}}(=\text{put}, x, c_b).\nu n.(\overline{c_b}\langle n\rangle \mid !c_{\mathsf{int}}(=n, c_o).\overline{c_o}\langle x\rangle)\big) \mid \\ !\big(&c_{\mathsf{q}}(=\text{get}, x, c_b).\overline{c_{\mathsf{int}}}\langle x, c_b\rangle\big) \mid \\ !\big(&c_{\mathsf{q}}(=\text{comp}_f, (x_1, \ldots, x_k), c_b).\nu c_o.\overline{c_{\mathsf{int}}}\langle x_1, c_o\rangle.c_o(v_1) \ldots \overline{c_{\mathsf{int}}}\langle x_k, c_o\rangle.c_o(v_k). \\ &\qquad \nu n.(\overline{c_b}\langle n\rangle \mid !c_{\mathsf{int}}(=n, c_o).\overline{c_o}\langle f(v_1, \ldots, v_k)\rangle)\big)\big)\end{aligned}$$

Fig. 2. The process *VM*

The translation from D to $D_{\mathsf{ctrl}} = [\![D]\!]^{fn(D)}$ is given in Fig. 3. The process $[\![M]\!]_c^{\mathcal{N}}$ causes the handle to the value of M to be sent on the channel c if run in parallel with VM. Here $\mathcal{N}$ is a set of names of sort Data that are supposed to be free in the transformed process. In the transformed process, the values of the names in $\mathcal{N}$ will be the same as in the original process, while the variables and the names not in $\mathcal{N}$ will contain handles to their values in the original process. The notations $c(\overrightarrow{u}, \overrightarrow{c})$ and $\overline{c}\langle\overrightarrow{M}, \overrightarrow{c}\rangle$ indicate the inputs and outputs of sort Data and $\mathcal{C}(\ldots)$, respectively. We see that data is handled by the virtual machine, while the values of sort "channel" are not affected by the translation.

The bisimilarity relates each closed process P to a process $\hat{P} = ([\![P]\!]^{\mathcal{N}} \mid VM \mid Store)$ where $\mathcal{N}$ is a subset of free names in P and $Store$ is a parallel composition

$$\begin{aligned}
[\![u]\!]_c^{\mathcal{N}} &= \overline{c}\langle u\rangle \quad \text{if } u \notin \mathcal{N}\\
[\![n]\!]_c^{\mathcal{N}} &= \overline{c_{\mathsf{q}}}\langle \mathrm{put}, n, c\rangle \quad \text{if } n \in \mathcal{N}\\
[\![f(M_1,\ldots,M_k)]\!]_c^{\mathcal{N}} &= \nu c_1 \ldots \nu c_k.\big([\![M_1]\!]_{c_1}^{\mathcal{N}} \mid \cdots \mid [\![M_k]\!]_{c_k}^{\mathcal{N}} \mid c_1(x_1)\ldots c_k(x_k).\overline{c_{\mathsf{q}}}\langle \mathrm{comp}_f, (x_1,\ldots,x_k), c\rangle\big)
\end{aligned}$$

$$\begin{aligned}
[\![\mathbf{0}]\!]^{\mathcal{N}} &= \mathbf{0}\\
[\![P \mid Q]\!]^{\mathcal{N}} &= [\![P]\!]^{\mathcal{N}} \mid [\![Q]\!]^{\mathcal{N}}\\
[\![!P]\!]^{\mathcal{N}} &= ![\![P]\!]^{\mathcal{N}}\\
[\![\nu u.P]\!]^{\mathcal{N}} &= \nu n \nu c_b.\overline{c_{\mathsf{q}}}\langle \mathrm{put}, n, c_b\rangle.c_b(u).[\![P]\!]^{\mathcal{N}\setminus\{u\}} \quad \text{if } u \text{ is data}\\
[\![\nu c.P]\!]^{\mathcal{N}} &= \nu c.[\![P]\!]^{\mathcal{N}} \quad \text{if } c \text{ is channel}\\
[\![\textbf{if } M = N \textbf{ then } P \textbf{ else } Q]\!]^{\mathcal{N}} &= \nu c_M \nu c_N.\big([\![M]\!]_{c_M}^{\mathcal{N}} \mid [\![N]\!]_{c_N}^{\mathcal{N}} \mid c_M(x_M).c_N(x_N).\\
&\quad \nu c_b.(\overline{c_{\mathsf{q}}}\langle \mathrm{get}, x_M, c_b\rangle.c_b(y_M).\overline{c_{\mathsf{q}}}\langle \mathrm{get}, x_N, c_b\rangle.c_b(y_N).\textbf{if } y_M = y_N \textbf{ then } [\![P]\!]^{\mathcal{N}} \textbf{ else } [\![Q]\!]^{\mathcal{N}})\big)\\
[\![c(u_1,\ldots,u_k,\overrightarrow{c}).P]\!]^{\mathcal{N}} &= c(x_1,\ldots,x_k,\overrightarrow{c}).\\
&\quad \nu c_b.\overline{c_{\mathsf{q}}}\langle \mathrm{put}, x_1, c_b\rangle.c_b(u_1)\ldots\overline{c_{\mathsf{q}}}\langle \mathrm{put}, x_k, c_b\rangle.c_b(u_k).[\![P]\!]^{\mathcal{N}}\\
[\![\overline{c}\langle M_1,\ldots,M_k,\overrightarrow{c}\rangle.P]\!]^{\mathcal{N}} &= \nu c_1 \cdots \nu c_k.\big([\![M_1]\!]_{c_1}^{\mathcal{N}} \mid \cdots \mid [\![M_k]\!]_{c_k}^{\mathcal{N}} \mid c_1(x_1)\ldots c_k(x_k).\nu c_b.\\
&\quad \overline{c_{\mathsf{q}}}\langle \mathrm{get}, x_1, c_b\rangle.c_b(y_1)\ldots\overline{c_{\mathsf{q}}}\langle \mathrm{get}, x_k, c_b\rangle.c_b(y_k).\overline{c}\langle y_1,\ldots,y_k,\overrightarrow{c}\rangle.[\![P]\!]^{\mathcal{N}}\big)
\end{aligned}$$

Fig. 3. Transforming out computations

of processes of the form $!c_{\mathsf{int}}(=n, c_o).\overline{c_o}\langle M\rangle$ associating the names n to values M. Moreover, the terms occurring in P (except for names in $\mathcal{N}$) must correspond to names in $[\![P]\!]^{\mathcal{N}}$ that are mapped to the same terms by *Store*. One transition step of P may correspond to several internal steps of $\hat{P}$. The processes at intermediate steps are also related to P.

4 Security of the Implementation of Randomized Symmetric Encryption

We have to present a process VM_{sim}, such that $VM_{\mathsf{sim}} \approx_{tr} VM$. The process VM performs computations on behalf of the processes knowing the channel name c_{q}. Given handles to values $v_1,\ldots,v_k$, it returns the handle to value $f(v_1,\ldots,v_k)$, where $f \in \Sigma = \Sigma_{\mathsf{ok}} \dot\cup \Sigma_{\mathsf{fb}}$. The process VM_{sim} must respond to the same computation (and put/get) queries, but it may not use the operations in Σ_{fb}. These queries must be handled in some other way.

For the R-ENC primitive, the set Σ of function symbols contains at least tupling, projections, Enc, Dec and the operations in Σ_{fb} outlined in Example 2. Any other operations must be handled by VM_{sim}, too. In the following, we are not going to present VM_{sim} as precisely as VM in Fig. 2, but we explain in detail the operations it performs and appeal to the Turing-completeness of π-calculus [16] in order to convince ourselves that such VM_{sim} exists.

The process VM_{sim} responds to the same commands as VM in the same manner (receives a channel for sending its output as part of the input). It keeps a table T_{val} of values it has received or constructed. Each entry (row) R in T_{val} has the fields "handle" (denoted $R.\mathsf{hnd}$), "value" (denoted $R.\mathsf{v}$) and extra arguments for bookkeeping (denoted $R.\mathit{args}$). For the rows R where $R.\mathsf{v}$ has been computed by VM_{sim} after a request to apply a symbol in Σ_{fb}, the extra arguments record that request.

The process VM_{sim} also keeps a second table T_{ct} of ciphertexts it has seen or constructed. Each row R in this table has the fields $R.\mathit{ct}$ (the ciphertext), $R.\mathit{snd}$, $R.\mathit{thd}$ (the second and third component of the ciphertext, considered as a tagged triple), $R.k$ (the correct key) and $R.\mathit{pt}$ (the corresponding plaintext). The field ct is a unique identifier for rows in this table. We let $T_{\mathsf{ct}}[M]$ denote the row of the table T_{ct} where the field ct equals M.

The process VM_{sim} handles the commands as follows.

Storing. To a store a value M, the process VM_{sim} generates a new name n and a new row R in the table T_{val} with $R.\mathsf{hnd} = n$, $R.\mathsf{v} = M$ and $R.\mathit{args} = \bot$. If M is a valid ciphertext (checked by comparing $\mathsf{is}_{enc}(M)$ to true) and T_{ct} does not yet contain a row $T_{\mathsf{ct}}[M]$, then this row is added, the field ct is initialized to M and other fields to $\bot$. The command returns n.

Retrieving. To retrieve a value by handle n, the process VM_{sim} locates the row R of T_{val} with $R.\mathsf{hnd} = n$, and replies with $R.\mathsf{v}$. If there is no such R, there will be no answer (This is similar to the behavior of VM).

Computing. To apply a function symbol f to values with handles $n_1, \ldots, n_k$, the process VM_{sim} locates the rows $R_1, \ldots, R_k$ of T_{val} with $R_i.\mathsf{hnd} = n_i$ for $i \in \{1, \ldots, k\}$. If some R_i cannot be located, or if k is different from the arity of f, there will be no answer. Otherwise, VM_{sim} generates a new name n and a new row R in T_{val} with $R.\mathsf{hnd} = n$, and replies with n. Before replying, it also defines $R.\mathsf{v}$ and $R.\mathit{args}$ as follows.

- If $f \in \Sigma_{\mathsf{ok}}$, then $R.\mathsf{v} = f(R_1.\mathsf{v}, \ldots, R_k.\mathsf{v})$ and $R.\mathit{args} = \bot$. Additionally,
  - If the operation was Enc and the row $T_{\mathsf{ct}}[R.\mathsf{v}]$ was not present, then it is added (and the field ct initialized with $R.\mathsf{v}$). The field k of this row is set to $R_2.\mathsf{v}$ and the field pt to $R_3.\mathsf{v}$.
  - If the operation was Dec, the row $T_{\mathsf{ct}}[R_2.\mathsf{v}]$ exists (recall that ciphertext was the second argument of Dec), and $R_1.\mathsf{v}$ was the correct key (checked by comparing $\mathsf{Enc}(\mathsf{e\_r}(R_2.\mathsf{v}), R_1.\mathsf{v}, R.\mathsf{v})$ to $R_2.\mathsf{v}$) then the field k of this row is updated to $R_1.\mathsf{v}$ and the field pt is updated to $R.\mathsf{v}$.
- If $f \in \Sigma_{\mathsf{fb}}$ and there exists a row R', such that $R'.\mathit{args}$ indicates that the same computation query has been made to VM_{sim} before (i.e. $R'.\mathit{args}$ names the operation f and the arguments $R_1.\mathsf{v}, \ldots, R_k.\mathsf{v}$), then let $R.\mathsf{v} = R'.\mathsf{v}$ and $R.\mathit{args} = R'.\mathit{args}$. In the following cases, we assume that the same query has not been made before.
- If f is $\bar{H}$ and the argument $R_1.\mathsf{v}$ is a triple (x, y, z) then check whether T_{ct} contains a row $T_{\mathsf{ct}}[M]$, where $M = \mathsf{Enc}(y, x, z)$. If such row exists, then its fields k and pt are updated to x and z. If such row does not exist, then it is created and its fields k and pt likewise set to x and z. If $T_{\mathsf{ct}}[M].\mathit{thd}$ is

not $\bot$ then VM_{sim} sets $R.\mathsf{v}$ to $T_{\mathsf{ct}}[M].thd$, otherwise it generates new names $n', \bar{n}$, sets both $R.\mathsf{v}$ and $T_{\mathsf{ct}}[M].thd$ to n', and adds a new row $\bar{R}$ to T_{val} with $\mathsf{hnd} = \bar{n}$, $\mathsf{v} = n'$ and $args = (\bar{\pi}_3, M)$. Next, it generates new names $\tilde{n}, \hat{n}$ and adds a two new rows $\tilde{R}, \hat{R}$ to the table T_{val} with $\tilde{R}.\mathsf{hnd} = \tilde{n}$, $\hat{R}.\mathsf{hnd} = \hat{n}$, $\tilde{R}.\mathsf{v} = (x, R.\mathsf{v})$, $\tilde{R}.args = \bot$, $\hat{R}.args = (\bar{H}, \tilde{R}.\mathsf{v})$. If $T_{\mathsf{ct}}[M].snd$ is not $\bot$ then VM_{sim} sets $\hat{R}.\mathsf{v}$ to $T_{\mathsf{ct}}[M].snd \oplus z$. Otherwise it generates a new name n'', sets $\hat{R}.\mathsf{v}$ to n'' and $T_{\mathsf{ct}}[M].snd$ to $n'' \oplus z$, and adds a new row to T_{val} with $\mathsf{hnd} = \bot$, $\mathsf{v} = n'' \oplus z$ and $args = (\bar{\pi}_2, M)$.
  We see that if VM_{sim} is requested to create a value that may serve as the third component of a ciphertext then this ciphertext will also appear in the table T_{ct} and the entire row corresponding to it will be initialized. Also, all entries in that row will also appear in T_{val}.
- If f is $\bar{H}$ and the argument $R_1.\mathsf{v}$ is a pair (x, y) then check whether there exists a row $T_{\mathsf{ct}}[M]$, such that $T_{\mathsf{ct}}[M].thd = y$ and x is the correct key for M. If there is no such row then y cannot have the form $\bar{H}(a, b, c)$; hence VM_{sim} generates a new name n and sets $R.\mathsf{v} = n$. Otherwise check whether $T_{\mathsf{ct}}[M].snd$ is not $\bot$. If this is the case, then set $R.\mathsf{v} = T_{\mathsf{ct}}[M].snd \oplus \mathsf{Dec}(x, M)$. Otherwise generate a new name n', set $R.\mathsf{v} = n'$ and $T_{\mathsf{ct}}[M].snd = n' \oplus \mathsf{Dec}(x, M)$, and add a new row to T_{val} with $\mathsf{hnd} = \bot$, $\mathsf{v} = n' \oplus \mathsf{Dec}(x, M)$ and $args = (\bar{\pi}_2, M)$.
- If f is $\bar{H}$ and the argument $R_1.\mathsf{v}$ is neither a pair nor a triple then generate a new name n' and set $R.\mathsf{v} = n'$.
- If f is $\overline{(\cdot, \cdot, \cdot)}$ then check whether there exists a row $T_{\mathsf{ct}}[M]$, such that $\mathsf{e\_r}(M) = R_1.\mathsf{v}$, $T_{\mathsf{ct}}[M].snd = R_2.\mathsf{v}$ and $T_{\mathsf{ct}}[M].thd = R_3.\mathsf{v}$. If such row exists then set $R.\mathsf{v} = M$. Otherwise generate new names n_k, n_x and add a new row $T_{\mathsf{ct}}[\mathsf{Enc}(R_1.\mathsf{v}, n_k, n_x)]$ to the table T_{ct}. Initialize the field snd of this row to $R_2.\mathsf{v}$ and field thd to $R_3.\mathsf{v}$. Also set $R.\mathsf{v} = \mathsf{Enc}(R_1.\mathsf{v}, n_k, n_x)$.
  We see that the result of applying the symbol $\overline{(\cdot)}$ is always a ciphertext. If the three components would result in an invalid ciphertext, then we generate this ciphertext using a throw-away key, thereby making its decryption impossible.
- If f is $\bar{\pi}_1$ then set $R.\mathsf{v} = \mathsf{e\_r}(R_1.\mathsf{v})$.
- If f is $\bar{\pi}_2$ and there is a row $T_{\mathsf{ct}}[R_1.\mathsf{v}]$ in T_{ct} and $T_{\mathsf{ct}}[R_1.\mathsf{v}].snd$ is not $\bot$, then let $R.\mathsf{v} = T_{\mathsf{ct}}[R_1.\mathsf{v}].snd$. If $T_{\mathsf{ct}}[R_1.\mathsf{v}].snd$ is $\bot$ then generate a new name n' and let both $R.\mathsf{v}$ and $T_{\mathsf{ct}}[R_1.\mathsf{v}].snd$ equal it. If the row $T_{\mathsf{ct}}[R_1.\mathsf{v}]$ does not exist then $R_1.\mathsf{v}$ is not a ciphertext, because this fact would have been noticed at the time when the row R_1 was added to T_{val}. Generate a new name n' and let $R.\mathsf{v} = n'$.
- If f is $\bar{\pi}_3$ then behave similarly to the case $f = \bar{\pi}_2$.

In all cases of handling a function symbol f from Σ_{fb}, the process VM_{sim} sets the fields $args$ of the newly created row R of T_{val} to $(f, R_1.\mathsf{v}, \ldots, R_k.\mathsf{v})$, where $R_1.\mathsf{hnd}, \ldots, R_k.\mathsf{hnd}$ were the arguments given to f.

Proposition 3. $VM_{\mathsf{sim}} \approx_{tr} VM$.

Proof (Sketch). Both VM and VM_{sim} maintain a database that maps from handles to values, update it according to certain rules, and reveal the values of queried elements. A context $C \in \mathbf{Ctxt}(\Sigma_{\mathsf{ok}})$ trying to distinguish VM and

VM_{sim} (i.e. having a channel c, such that $C[VM_{\mathsf{sim}}] \Downarrow c$, but $tr(C)[VM] \not\Downarrow c$) will at some point query for certain elements of the database and then perform a test (check the equality of two terms built from the queried elements), the result of which determines whether the communication on c happens. We will show that at no point in the execution there exists a test that can tell apart the databases of VM and VM_{sim}. Formally, a *test* is a pair of *test messages* $M, N \in \mathbf{T}_{\Sigma_{\mathsf{ok}}}(Names \cup Refs)$, where *Refs* is the set of "references" to the cells of the database of VM_{sim}. A reference $r \in Refs$ can either be $n.\mathsf{v}$, where n is the handle of a row R in T_{val}, or $T_{\mathsf{ct}}[M_e].\mathit{field}$, where the ciphertext M_e identifies a row in T_{ct} and the value of $\mathit{field} \in \{snd, thd\}$ in this row is different from $\bot$. The context C can access these references by using get-queries, possibly preceded by a $\mathrm{comp}_{\bar{\pi}_2}$ or $\mathrm{comp}_{\bar{\pi}_3}$ query. The names that a test message may contain are free names generated by C.

If the context $C[\,]$ encapsulating VM_{sim} evaluates a test message M, then the value $\langle\!\langle M \rangle\!\rangle^{\mathsf{sim}}$ it learns is obtained by replacing the references to database cells in M with their actual values, as kept by VM_{sim}. If $C[\,]$ encapsulates VM, then it learns the value $\langle\!\langle M \rangle\!\rangle^{\mathsf{real}}$ instead, where the reference $n.\mathsf{v}$ is replaced with the value VM associates with the handle n. In this case, the references $T_{\mathsf{ct}}[M_e].snd$ and $T_{\mathsf{ct}}[M_e].thd$ are replaced with $\bar{\pi}_2(tr(M_e))$ and $\bar{\pi}_3(tr(M_e))$, respectively.

Example 3. Suppose that the context $C[\,]$ issues the following commands: $n_1 \leftarrow \mathrm{put}(r)$; $n_2 \leftarrow \mathrm{put}(k)$; $n_3 \leftarrow \mathrm{put}(x)$; $n_4 \leftarrow \mathrm{comp}_{(,,)}(n_2, n_1, n_3)$; $n_5 \leftarrow \mathrm{comp}_{\bar{H}}(n_4)$; $n_6 \leftarrow \mathrm{comp}_{(,)}(n_2, n_5)$; $n_7 \leftarrow \mathrm{comp}_{\bar{H}}(n_6)$; $n_8 \leftarrow \mathrm{comp}_{\oplus}(n_7, n_3)$; and finally $n_9 \leftarrow \mathrm{comp}_{\overline{(,,)}}(n_1, n_8, n_5)$. Through these queries, the handle n_9 will correspond to the ciphertext $\mathsf{Enc}(r, k, x)$. Let $M = (n_9.\mathsf{v}, n_5.\mathsf{v}, T_{\mathsf{ct}}[\mathsf{Enc}(r, k, x)].thd)$ and n^{hash} be a new name generated by VM_{sim}.. Then $\langle\!\langle M \rangle\!\rangle^{\mathsf{sim}} = (\mathsf{Enc}(r, k, x), n^{\mathsf{hash}}, n^{\mathsf{hash}})$ and $\langle\!\langle M \rangle\!\rangle^{\mathsf{real}} = (\overline{(}r, \bar{H}(k, \bar{H}(k, r, x)) \oplus x, \bar{H}(k, r, x)\overline{)}, \bar{H}(k, r, x), \bar{H}(k, r, x))$.

We show that the following claim holds.

Claim (*) At any step of the computation of VM_{sim}, the equivalence $\langle\!\langle M \rangle\!\rangle^{\mathsf{sim}} = \langle\!\langle N \rangle\!\rangle^{\mathsf{sim}} \Leftrightarrow \langle\!\langle M \rangle\!\rangle^{\mathsf{real}} = \langle\!\langle N \rangle\!\rangle^{\mathsf{real}}$ holds for all $M, N \in \mathbf{T}_{\Sigma_{\mathsf{ok}}}(Names \cup Refs)$.

If **(*)** would not hold for some M, N at some step of VM_{sim}, then we can show that for some $M_{\mathsf{prev}}, N_{\mathsf{prev}} \in \mathbf{T}_{\Sigma_{\mathsf{ok}}}(Names \cup Refs)$ the claim **(*)** would not hold at the previous step. We will not present the full analysis of cases in this paper, but as an example, consider the computation step where f is $\bar{H}$ and the argument $R_1.\mathsf{v}$ is a triple (x, y, z). If M and N do not refer to any newly created entries of T_{val} or T_{ct}, then we can set $M_{prev} = M$ and $N_{prev} = N$. Otherwise we obtain M_{prev} and N_{prev} by replacing the new entries in M and N as follows. Let $M_e = \mathsf{Enc}(y, x, z)$. We consider four possibilities, depending on whether $T_{\mathsf{ct}}[M_e].thd$ is defined (1&2) or not (3&4), and whether $T_{\mathsf{ct}}[M_e].snd$ is defined (1&3) or not (2&4). Fig. 4 outlines the replacements for references to possibly new entries in T_{ct} and T_{val}. An empty cell indicates that the reference existed before the current computation step, or the row was not created in this computation step. We refer to the description of VM_{sim} for the meaning of new rows and definitions of new names. The table in Fig. 4, describing how M_{prev} is constructed from M, should be understood as follows: if e.g. M contains the reference $\tilde{n}.\mathsf{v}$, and the cells $T_{\mathsf{ct}}[M_e].\mathit{thd}$ and $T_{\mathsf{ct}}[M_e].\mathit{snd}$ are not defined (4th case), then $\tilde{n}.\mathsf{v}$ should be substituted with the pair (x, n'), where n' is a new name.

ref.	replacement			
$T_{\mathsf{ct}}[M_e].\mathit{snd}$		$n'' \oplus z$		$n'' \oplus z$
$T_{\mathsf{ct}}[M_e].\mathit{thd}$			n'	n'
$n.\mathsf{v}$	$T_{\mathsf{ct}}[M_e].\mathit{thd}$	$T_{\mathsf{ct}}[M_e].\mathit{thd}$	n'	n'
$\bar{n}.\mathsf{v}$			n'	n'
$\tilde{n}.\mathsf{v}$	$(x, T_{\mathsf{ct}}[M_e].\mathit{thd})$	$(x, T_{\mathsf{ct}}[M_e].\mathit{thd})$	(x, n')	(x, n')
$\hat{n}.\mathsf{v}$	$T_{\mathsf{ct}}[M_e].\mathit{snd} \oplus z$	n''	$T_{\mathsf{ct}}[M_e].\mathit{snd} \oplus z$	n''

Fig. 4. Replacement of new references in simulating $\bar{H}(x, y, z)$

The replacement gives us M_{prev} and N_{prev}, such that $\langle\!\langle M\rangle\!\rangle^{\mathsf{sim}} = \langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{sim}}$ and $\langle\!\langle N\rangle\!\rangle^{\mathsf{sim}} = \langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{sim}}$. The induction assumption states that $\langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{sim}} = \langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{sim}}$ if and only if $\langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{real}} = \langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{real}}$. The values $\langle\!\langle M\rangle\!\rangle^{\mathsf{real}}$ and $\langle\!\langle N\rangle\!\rangle^{\mathsf{real}}$ are obtained from $\langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{real}}$ and $\langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{real}}$ by substituting some names with (possibly more complex) values. If $\langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{real}} = \langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{real}}$ then also $\langle\!\langle M\rangle\!\rangle^{\mathsf{real}} = \langle\!\langle N\rangle\!\rangle^{\mathsf{real}}$. If $\langle\!\langle M_{prev}\rangle\!\rangle^{\mathsf{real}} \neq \langle\!\langle N_{prev}\rangle\!\rangle^{\mathsf{real}}$ then we also have $\langle\!\langle M\rangle\!\rangle^{\mathsf{real}} \neq \langle\!\langle N\rangle\!\rangle^{\mathsf{real}}$, because the structure of substituted values cannot be explored using only the function symbols in Σ_{ok}.

In such manner we obtain a *bisimilarity modulo implementation* between VM_{sim} and VM even for the case where their entire databases are public. Hence also $VM_{\mathsf{sim}} \approx_{tr} VM$. □

5 Related Work

The implementation of cryptographic primitives is a certain case of *process refinement*. While various aspects of refinement have been explored [3, 19, 18], mostly concerned with the refinement of possible behaviors of a process, the work reported in this paper has primarily been inspired by the notions of universal composability [7] and (black-box) reactive simulatability [17], both originating in the computational model of cryptography. The notion of indifferentiability by Maurer et al. [15] is similar. These definitions have been recently carried over to the symbolic model by Delaune et al. [9]. In all these definitions, there is a notion of two interfaces — one for the "legitimate" user and one for the adversary — of the process under investigation. Two processes can be equivalent (or in the refinement relation) only if the user's interface stays the same. The adversary's interface can change and a simulator process is used to translate between different interfaces. This is in contrast to our problem, where the user's interface is also naturally considered as changing, and the replacement of the primitive with the implementation is more invasive for the user process. While we think that the definition of secure implementation could be based on the notion of strong simulatability by Delaune et al. [9], the setup would be less natural and possibly the virtual machine process VM has to be included even in the definition.

The question of secure protocol composition is related to the issues of implementability of abstract processes or primitives. In recent work, Ciobâcă and Cortier [8] give sufficient conditions for the security of composition of two protocols using arbitrary primitives to follow from the security of stand-alone protocols. Interestingly, they have a similar restriction on primitives used by different protocols — the sets of primitives have to be disjoint.

Regarding the study of implementability of primitives, it is worth mentioning that in the same paper where they introduced applied pi-calculus, Abadi and Fournet [2, Sec 6.2] also considered an implementation of the MAC primitive, inspired from the HMAC construction [5]. Still, the construction is *ad hoc* and puts restrictions on how the process uses certain values.

6 Conclusions

We have explored the notion of securely implementing a cryptographic primitive in the symbolic model, and presented definitions that are more general and convenient to use than definitions that could be obtained from the application of existing treatment of process refinement and simulation. We have shown the usefulness of the proposed definition by demonstrating a secure implementation for the randomized symmetric encryption primitive. Future work in this topic would involve a systematic treatment of the implementability of common cryptographic primitives from each other. Obtained reductions and simulations may also give new insights to the security proofs in the computational model. The first step in this direction would be the analysis of the Luby-Rackoff construction [14] for constructing pseudorandom permutations ("deterministic" symmetric encryption, where $\mathsf{Enc}(k, \cdot)$ and $\mathsf{Dec}(k, \cdot)$ are inverses of each other) from random functions (modeled in the symbolic model as hashing) and exclusive or. Another line of future work is the application of the results of this paper, as well as [9], to the analysis of complex protocols.

References

1. *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010.* IEEE Computer Society, 2010.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001.
3. Luca Aceto and Matthew Hennessy. Towards action-refinement in process algebras. *Inf. Comput.*, 103(2):204–269, 1993.
4. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A Universally Composable Cryptographic Library. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, October 2003. ACM Press. Extended version available as Report 2003/015 of Cryptology ePrint Archive.

5. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.
6. Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In Andrew D. Gordon, editor, *FoSSaCS*, volume 2620 of *Lecture Notes in Computer Science*, pages 136–152. Springer, 2003.
7. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
8. Stefan Ciobâca and Véronique Cortier. Protocol composition for arbitrary primitives. In *CSF* [1], pages 322–336.
9. Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In Ravi Kannan and K. Narayan Kumar, editors, *FSTTCS*, volume 4 of *LIPIcs*, pages 169–180. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.
10. Danny Dolev and Andrew Chi-Chih Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
11. Morris Dworkin. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A, 2001.
12. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
13. Madeline González Muñiz and Peeter Laud. On the (Im)possibility of Perennial Message Recognition Protocols without Public-Key Cryptography. In *26th ACM Symposium On Applied Computing*, volume 2, pages 1515–1520, March 2011.
14. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
15. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.
16. Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
17. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.
18. Steve Reeves and David Streader. Comparison of data and process refinement. In Jin Song Dong and Jim Woodcock, editors, *ICFEM*, volume 2885 of *Lecture Notes in Computer Science*, pages 266–285. Springer, 2003.
19. Markus Roggenbach. CSP-CASL - a new integration of process algebra and algebraic specification. *Theor. Comput. Sci.*, 354(1):42–71, 2006.
20. Mark D. Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2010.
21. Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 1982. IEEE Computer Society Press.
22. Scott Yilek. Resettable public-key encryption: How to encrypt on a virtual machine. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 41–56. Springer, 2010.