# Decision Procedures for Simulatability*

Charanjit S. Jutla[1] and Arnab Roy[2]

[1] IBM T.J. Watson Research Center,
Yorktown Heights, NY 10598, USA
[2] Fujitsu Laboratories of America,
Sunnyvale, CA 94058, USA

**Abstract.** We address the question of automatically proving security theorems in the universally composable (UC) model for ideal and real functionalities composed of if-then-else programs with uniform random number generation and data objects from the additive group of $\mathbb{F}_{2^m}$. We prove that for this restricted yet powerful language framework, there is an effective procedure to decide if a real functionality realizes an ideal functionality, and this procedure is in computational time independent of $m$, which is essentially the security parameter.

To this end, we consider multivariate *pseudo-linear* functions, which are functions computed by branching programs over data objects from the additive group of $\mathbb{F}_{2^m}$. The conditionals in such programs are built from equality constraints over linear expressions, closed under negation and conjunction.

Let $f_1, f_2, ..., f_k$ be $k$ pseudo-linear functions in $n$ variables, and let $f$ be another pseudo-linear function in the same $n$ variables. We show that if $f$ is a function of the given $k$ functions, then it must be a *pseudo-linear* function of the given $k$ functions. This generalizes the straightforward claim for just linear functions. Proceeding further, we generalize the theorem to randomized pseudo-linear functions. We also prove a more general theorem where the $k$ functions can in addition take further arguments, and prove that if $f$ can be represented as an iterated composition of these $k$ functions, then it can be represented as a probabilistic pseudo-linear iterated composition of these functions. Additionally, we allow $f$ itself to be a randomized function, i.e. we give a procedure for deciding if $f$ is a probabilistic sub-exponential (in $m$) time iterated function of the given $k$ randomized functions. The decision procedure runs in computational time independent of $m$.

## 1 Introduction

Security primitives and protocols are deceptively concise. While they may have a short description, a very specialized level of expertise is required for developing these protocols and reasoning as to why they meet a specific security goal. In the last few decades, the field of cryptography has come a long way in understanding

* Authors were supported in part by the Department of Homeland Security under grant FA8750-08-2-0091.

S. Foresti, M. Yung, and F. Martinelli (Eds.): ESORICS 2012, LNCS 7459, pp. 573–590, 2012.

fundamental principles and laying down the framework of specifying security goals and proof methods on a firm formal footing.

Concomitant to the maturity of this field, a substantial research community has grown around attempting to consolidate and automate these reasoning principles so that that the manual need to provide tedious and cumbersome proofs is removed. Importantly, automation provides greater assurance since all possible corner cases are also considered, eliminating subtle errors.

*Related Work.* Majority of previous work in formal methods [1–4] has focused on abstracting some of the fundamental primitives of cryptography, *viz.*, encryption, signatures, hashes and so on, as part of the language of specifying the protocols. Though some of these methodologies have even led to automation of the verification procedure [5], all these systems have only been proven sound, and there has been very limited work on completeness of these logics (cf. [6]). Another drawback has been the coarseness of specification required by these lines of work. The work by [7] is a promising approach to reason about fine-grained primitives by leveraging axiomatization of bounded arithmetic [8], but because of its still rather general number-theoretic approach, it may still be incomplete (see [9]). Similarly, newer works like CertiCrypt [10] and EasyCrypt [11] show sequence of human-generated games to be indistinguishable by employing general purpose theorem-provers, which clearly do not possess a completeness property. Important works [12, 13] have also tried to leverage the UC paradigm (also see [14] for decision procedures for equality of probabilistic terms), but still the simulators in these works have to be human-generated. In particular, these approaches focus on proving indistinguishability of games/processes, but if there is a simulator required to act as a wrapper around the ideal functionality, then the simulator is built by humans.

*Our Contributions.* In this paper, we take a purely algebraic approach to the problem and focus on using the UC paradigm to build an appropriate specification and definition language which is fairly general and yet complete (and even efficiently decidable; specifically with complexity independent of the security parameter). In particular, we seek an automated procedure which finds a simulator if it exists and returns failure if the protocol is not simulatable in the ideal world. To start with, in this work we focus on specification and definition languages which are general enough to capture a rich class of high level protocols that use cryptographic primitives as ideal functionalities.

In the UC framework [15], which builds on the ideal process emulation paradigm of [16], the specification of the target (multi-party) protocol is given by an *ideal functionality* which is handed all the inputs, and which computes the respective outputs all the while interacting with an adversary in a specified way. Thus, the *specification language* is just the language in which the ideal functionality can be defined. The actual protocol is a set of definitions, one for each of the parties for their internal computation and interaction with other parties and/or the adversary. Thus the *definition language* is a language in which the code of each party in the real protocol can be defined. A protocol is (informally)

considered *secure* if for every adversary in the real protocol, there is an adversary interacting with the ideal functionality, such that it is impossible to efficiently distinguish between the two[1]. Most such proofs of security are obtained by a black box simulation paradigm, i.e. by obtaining a Simulator that simulates the "view" of the real world adversary, while accessing the latter only in a black-box manner. In such a black box paradigm (which in the case of UC security is known to be sufficient (see [15] Sec. 4.3.2)), the question of security of the real protocol boils down to deciding if the view of the adversary (and/or the Environment) in the real protocol can be efficiently simulated by accessing the ideal functionality's interface.

For our base language, both the specification language and the definition language are just if-then-else or **branching programs** involving data objects from the additive group of fields of characteristic two. The conditionals are built from equality constraints of linear expressions, closed under negation and conjunction. The multivariate functions computed by such programs are called *pseudo-linear* functions, because they are piece-wise linear over different linear subspaces. Before we define these functions more formally, we state the completeness theorem for such pseudo-linear functions, which allows us to claim efficient decidability of simulatability.

While for linear multivariate functions a *completeness theorem* which states that if a linear function $f$ of $n$ variables is a function of $k$ other linear functions (in the same $n$ variables), then $f$ must be a *linear function* of the $k$ linear functions, is well known and rather easy to prove, a similar completeness result for pseudo-linear functions is novel and not so easy to prove.

Thus, one of the main results of this paper is a theorem which states that if a pseudo-linear multivariate function $f$ of $n$ variables is a *function* of $k$ pseudo-linear functions $f_1, f_1, ..., f_k$ (in the same $n$ variables), then $f$ must be a *pseudo-linear* function (say, $g$) of $f_1, f_2, ..., f_k$. Note that it is given that $f$ itself is a pseudo-linear function in the original $n$ variables. Thus in this context, if a simulator must simulate a real-world pseudo-linear function $f$ using pseudo-linear ideal functionality subroutines, then we can restrict the search for simulators to the space of pseudo-linear functions. Since the size of this space is independent of the security parameter[2], this search can be efficient.

Proceeding further, we include random number generation as an additional primitive in both the specification and the definition language, and *extend* the procedures to decide if a *probabilistic poly time* simulator exists for the given set of *randomized* ideal functionalities and *randomized* target function. To this end, we first establish that any randomized pseudo-linear function is statistically equivalent to a randomized pseudo-linear function that generates a *single*

[1] Technically, in the UC framework, the distinguisher is just the Environment, which also gets to control and see the inputs and outputs of the honest parties and can control the Adversary as well.

[2] This space can be of size exponential or more in the size of the function descriptions, but since the functions in cryptographic applications tend to be simple and small, we focus on the security parameter as the real complexity parameter.

random number. We call the later class of functions, Simplified Randomized Pseudo-Linear (SRPL) functions.

To model the fact that a simulator can iteratively compose various calls to the different functions in the ideal functionality, we *prove a more general theorem* involving arbitrary iterations of $k$ randomized pseudo-linear functions $f_1(\boldsymbol{z}, \boldsymbol{y}), f_2(\boldsymbol{z}, \boldsymbol{y}), ..., f_k(\boldsymbol{z}, \boldsymbol{y})$, where $\boldsymbol{y}$ are arguments which the simulator can supply. We construct iterated compositions of these functions and establish that they are bounded in number. The completeness theorem in this setting shows that if there is no simulator which is an iterated composition of these functions, then all probabilistic poly-time simulators are distinguishable from the target function on more than a certain non-negligible fraction of the probability space for any input. This is a much harder theorem to prove because (i) the compositions are distributions rather than fixed quantities and (ii) we are ruling out all simulators which are "close enough" to the target function rather than being exactly same.

For cryptographic applications, this means that an algorithmic search for a simulator in proving that a protocol in this language realizes an ideal functionality (also in this language) is *independent* of the security parameter, as the bounds in the completeness theorem are independent of the field size (the security parameter is usually related to the field size). Since the program sizes in cryptographic protocols are usually small, this can lead to efficient theorem proving.

*Open Problems and Scope.* We note here that if the real protocol is given in the hybrid model, i.e. by making calls to some other ideal functionalities (to avoid confusion, we will call them hybrid functionalities), our decision procedure still works as long as these hybrid functionalities are also in the same language. To address the situation that the adversary, which may have access to these hybrid functionalities, may call these functionalities an indefinite number of times, we do need to prove a (meta-) theorem which essentially says that it suffices to prove simulatability for adversaries restricted to making a constant number of calls to each of these hybrid functionalities. Such theorems may be easy to prove on a protocol basis, but there is also scope for general theorems based on the structure of these functionalities. This also implies that hybrid functionalities (e.g. Random Oracle or public key encryption (PKE)) that can have indefinitely many table entries, need only be in-lined in the target function with the tables restricted to constant number of entries. Additionally, some hybrid functionalities (e.g. PKE) may require function symbols in their specification – in case of PKE these are the encryption and decryption functions $e$ and $d$. However, because these function symbols are uninterpreted, i.e. have no constraints on them, an easy extension of our theorems handles such function symbols.

In the future, similar extension to realization of ideal functionalities (i.e. instead of just hybrid functionalities) which support simple tables and uninterpreted functions can be envisioned. Since, most such functionalities, e.g. PKE, are realized using specific algebraic structures, this would require appropriate axiomatization of their underlying operations and computational assumptions

(see e.g. [17]). While most cryptographic functionalities which support tables with the size of the entries fixed and with operations limited to keyword search (e.g. PKE, Random Oracle, Ideal Cipher etc.) lead to, or are expected to lead to decidable simulatability, if we allow arbitrary sized entries in tables, and further allow some non-trivial operations on the entries, the question of simulatability becomes *undecidable* [18]. Finally, we remark that our completeness results require sufficiently large fields (as a function of the number of variables in the functionalities), but given that most UC proofs only seek proofs of simulatability which do not depend on the security parameters, our completeness theorem covers all such UC proofs.

While the decision procedures in this work can be exponential or even double-exponential time in the program sizes, the exact complexity of these problems remains open. Further, for cryptographic protocols one does not expect the worst case exponential blowup in state-space (space of pseudo-linear functions) and further work is required in this direction. Finally, there is a tantalizing possibility of extending our work to synthesis (human-assisted or otherwise) of real protocols given an ideal functionality as specification.

The inability to handle arbitrary sequence of adversary calls in the real world limits the applicability of our current results to practical cryptographic protocols. However, we believe that our results open up a new and exciting line of research in this area with promising directions to explore. We believe this is the first work that seeks complete decision procedures for the problem of statistical simulatability.

*Organization.* The next section formally defines pseudo-linear functions, and proves a basis, an interpolation theorem and the Completeness theorem for pseudo-linear functions. Section 3 extends the set of primitives to include random number generation and proves a completeness theorem for deciding simulatability. Section 4 considers iterated composition of randomized pseudo-linear functions and proves a completeness theorem. Section 5 relates the results in this paper to proof automation in the UC model. Finally, Section 6 concludes with a discussion on work in progress and open problems.

Due to space constraints, most of the proofs are deferred to the full version of the paper [18].

## 2 Pseudo-Linear Functions

In this section we introduce and formally define pseudo-linear functions. We begin with examples of how pseudo-linear polynomials relate to branching programs with bit-wise exclusive-or operations (which is just addition in fields of characteristic two). So, consider a finite field $\mathbb{F}_q$, where $q = 2^m$. We adopt the convention that $l_*(\boldsymbol{x})$ denote linear polynomials over the components of $\boldsymbol{x}$ which are elements of $\mathbb{F}_{2^m}$. To start with, an equality constraint of the form $l_1(\boldsymbol{x}) = l_2(\boldsymbol{x})$ can then be written as

$$1 + (l_1(\boldsymbol{x}) + l_2(\boldsymbol{x}))^{q-1}$$

which evaluates to 1 if $l_1(\boldsymbol{x}) = l_2(\boldsymbol{x})$, and evaluates to zero otherwise. Similarly, $l_1(\boldsymbol{x}) = 0$ *and* $l_2(\boldsymbol{x}) = 0$ can be written as

$$(1 + l_1(\boldsymbol{x})^{q-1}) \cdot (1 + l_2(\boldsymbol{x})^{q-1})$$

As a final example, an expression "if ($l_1(\boldsymbol{x}) = 0$ *and* $l_2(\boldsymbol{x}) = 0$) then $l_3(\boldsymbol{x})$ else $l_4(\boldsymbol{x})$" can be written as

$$(1 + l_1(\boldsymbol{x})^{q-1}) \cdot (1 + l_2(\boldsymbol{x})^{q-1}) \cdot (l_3(\boldsymbol{x}) + l_4(\boldsymbol{x})) \; + \; l_4(\boldsymbol{x})$$

Formally, a **pseudo-linear** multivariate polynomial defined over sub-field $\mathbb{F}_2$ is then a polynomial which is a sum of guarded linear-terms; a *guarded linear-term* is a polynomial which is the product of a linear (over $\mathbb{F}_2$) polynomial[3] and zero or more linear-guards; a *linear-guard* is a linear (over $\mathbb{F}_2$) polynomial raised to the power $(q-1)$. Since, in this paper we will only be dealing with pseudo-linear polynomials defined over $\mathbb{F}_2$, from now on we will implicitly assume that. A pseudo-linear polynomial in $n$ variables and defined over $\mathbb{F}_2$, however does yield a function from $(\mathbb{F}_q)^n$ to $\mathbb{F}_q$, which we call a **pseudo-linear function**. Thus, even though the polynomial is defined over $\mathbb{F}_2$, the underlying field will be $\mathbb{F}_q$, and hence the algebra of the polynomials is modulo $(x_i^q = x_i)$ (for $i$ ranging from 1 to $n$). They are also further restricted by the fact that all expressions in the guards are linear instead of affine, but we can also introduce constant additive terms from $\mathbb{F}_q$ [18].

We observe that pseudo-linear polynomials are closed under pseudo-linear transformations, i.e. given a pseudo-linear polynomial, raising it to the power $(q-1)$, and multiplying it by another pseudo-linear polynomial yields just another pseudo-linear polynomial. More importantly, the branching programs mentioned in the introduction compute exactly the pseudo-linear functions. We make this connection more formal in Section 5.

### 2.1 A Basis for Pseudo-linear Functions

Let $X$ denote a set of $n$ variables $\{x_1, x_2, \cdots, x_n\}$ from $\mathbb{F}_q$. Let $\mathcal{L}$ stand for the set of all linear expessions, including zero, over elements of $X$. We define the set of **elementary pseudo-linear (**EPSELIN**)** polynomials to be all polynomials of the form

$$\prod_{l \in J}(1 + l(\boldsymbol{x})^{q-1}) \; \cdot \; \prod_{l \in \mathcal{L} \setminus J} l(\boldsymbol{x})^{q-1} \; \cdot \; p(\boldsymbol{x})$$

where $p(\boldsymbol{x})$ is in $\mathcal{L}$, and $J$ is any subset of $\mathcal{L}$ which is closed under addition, i.e. $J$ is a subspace of $\mathcal{L}$. We also include the zero polynomial amongst the elementary pseudo-linear polynomials. Note that if $\mathcal{L} \setminus J$ included a linearly-dependent term of $J$, then the above polynomial reduces to zero in $\mathbb{F}_q$.

Related to the earlier definition of a linear-guard, we will refer to expressions of the form

$$\prod_{l \in J}(1 + l(\boldsymbol{x})^{q-1}) \; \cdot \; \prod_{l \in \mathcal{L} \setminus J} l(\boldsymbol{x})^{q-1}$$

[3] Which is just a sum of variables for a field of characteristic 2.

as just **guards**. We will mainly be focusing on guards corresponding to $J$ which are linear subspaces.

For the next definition, we will require that the $n$ variables be ordered by their indices. Thus $x_1$ is considered to be of lesser index than $x_2$, and so on. This also induces a lexicographic ordering on all equal-sized subsets of the $n$ variables $X$. An elementary pseudo-linear polynomial with the above notation will be called a **reduced elementary pseudo-linear (**REPSELIN**)** polynomial if it satisfies the following:

1. Let $r$ be the rank of $J$ ($r \leq \min(n, |J|)$).
2. Let $R$ be the lexicographically greatest set of $r$ variables occuring in $J$ which can be expressed in terms of smaller indexed variables (or just zero) when the elements of $J$ are set to zero. This for example, can be accomplished by considering a row-echelon normal form of $J$. As a simple instance, if $J = \{x_1 + x_2, x_2 + x_3, x_3 + x_1\}$, then $R = \{x_2, x_3\}$, since setting the elements of $J$ to 0 will imply $x_3 = x_2 = x_1$.
3. None of the variables in $R$ occur in $p(\boldsymbol{x})$.

To justify this definition, we note that if an elementary pseudo-linear polynomial is not reduced, then it is equivalent to a reduced one. One implication of the above definition is that if $p(\boldsymbol{x})$ is non-zero then it itself cannot be in $J$. Recall, $J$ is closed under addition, by definition of EPSELIN-polynomials. Let $r$ be the rank of $J$. Let $\bar{J}$ be the $r$ sized subset of $J$ which forms a basis of $J$, and which define the variables $R$ by the row-echelon normal form of $J$. Thus, all $l(\boldsymbol{x})$ in $J$ *must have at least one variable from* $R$. Thus, $p(\boldsymbol{x})$ cannot be in $J$.

Finally, we define a REPSELIN-polynomial to be a **basic pseudo-linear** polynomial if the linear term $p(\boldsymbol{x})$ is just a variable from $X$. Note that since the basic polynomial is REPSELIN, from item (3) above it follows that this variable is not from $R$.

**Lemma 1.** *Every pseudo-linear polynomial can be expressed as a sum of basic pseudo-linear polynomials.*

We will now show that the basic pseudo-linear polynomials actually form a *basis* for pseudo-linear polynomials. Before that we need some more notation. Let $\mathcal{Q}(X)$ be the set of all basic pseudo-linear polynomials in variables $X$. Further, let $\mathcal{G}(X)$ be the set of all guards *which form a part of these polynomials* $\mathcal{Q}(X)$. Let $|\mathcal{G}(X)| = t$. The guards can then be named w.l.o.g. $g_1$, $g_2$, ...,$g_t$. Recall, for each guard $g_i$, there is associated a subset of variables $X$, namely $R$, that do not occur in any linear terms $p(\boldsymbol{x})$. We refer to all linear combinations of $X \backslash R$ as $\mathcal{P}_i(X)$, including the linear term zero. Let $|\mathcal{P}_i(X)| = s_i + 1$. Note that $(s_i + 1)$ is two to the power of the size of the subset of variables associated with $g_i$. The linear terms in $\mathcal{P}_i(X)$ can be named $p_i^j(\boldsymbol{x})$, $j$ ranging from 0 to $s_i$ (not to be confused with exponent). W.l.o.g., zero will always be $p_i^0(\boldsymbol{x})$.

Thus, any pseudo-linear function $\phi(\boldsymbol{x})$ can be represented as a sum (over $\mathbb{F}_2$) of polynomials from $\mathcal{Q}(X)$, i.e.,

$$\phi(\boldsymbol{x}) \;=\; \sum_{i \in T} g_i(\boldsymbol{x}) \cdot p_i^{j(\phi,i)}(\boldsymbol{x})$$

where $T$ is a subset of $[1..t]$, and each $p_i^{j(\phi,i)}(\boldsymbol{x}) \in \mathcal{P}_i(X)$. In fact, we do not even need to take a subset $T$ of $[1..t]$; all zero terms just imply that $j(\phi, i) = 0$, by our notation above that $p_i^0(\boldsymbol{x})$ is always taken to be zero. Thus the above representation of $\phi(\boldsymbol{x})$ is totally defined by the map $j(\phi, \cdot)$.

While we state and prove the following theorem only for large fields, as only for such fields do the basic pseudo-linear polynomials form a basis, a slightly more complicated characterization can be given for smaller fields.

**Lemma 2 (Basis).** *For fields of size $q > 2^n$, the basic pseudo-linear polynomials in $n$ variables form a basis for pseudo-linear polynomials in $n$ variables.*

Detailed proofs can be found in the full version of the paper [18].

**Lemma 3 (Homomorphism).** *For any pseudo-linear functions $\phi_1(\boldsymbol{x})$ and $\phi_2(\boldsymbol{x})$, and for all $i \in [1..t]$,*

$$p_i^{j(\phi_1+\phi_2,i)} \;=\; p_i^{j(\phi_1,i)} + p_i^{j(\phi_2,i)}$$

*Proof.* Follows from the fact that the basic pseudo-linear polynomials form a basis for pseudo-linear polynomials.

### 2.2 Interpolation Property and the Completeness Theorem

Before we prove the main theorem, we need a few more definitions and related lemmas. Let $f_1, f_2, ..., f_k$ be $k$ pseudo-linear functions in $n$ variables $X$, over a field $\mathbb{F}_q$ $(q = 2^m)$. Collectively, we will refer to these polynomials as $F$. For any pseudo-linear polynomial $f(\boldsymbol{x})$ in $X$, let its representation in terms of the basis be given by $j(f, \cdot)$. Since each of the polynomials from $F$, i.e. $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ...., f_k(\boldsymbol{x})$ is pseudo-linear, it can be represented by $j(f_s, \cdot)$ $(s \in [1..k])$. Further, each linear combination of $F$ is represented similarly.

We say that two guards $g_a(\boldsymbol{x})$ and $g_b(\boldsymbol{x})$ are $F$-**equivalent** if for every linear combination $\phi$ of functions from $F$, it is the case that $j(\phi, a) = 0$ iff $j(\phi, b) = 0$. In this case, we write $a \cong_F b$, which is an equivalence relation.

**Lemma 4.** *If $a$ and $b$ are $F$-equivalent then if for some subset $S \subseteq [1..k]$, the linear combination $\sum_{s\in S} p_a^{j(f_s,a)}$ is identically zero , then so is $\sum_{s\in S} p_b^{j(f_s,b)}$.*

The lemma follows by Lemma 3. Thus, if $k'$ is the rank of $p_a^{j(f_s,a)}$ $(s \in [1..k])$, then it is also the rank of $p_b^{j(f_s,b)}$. In fact, we can take the exact same $k'$ indices from $(s \in [1..k])$, w.l.o.g. $[1..k']$, to represent the basis for the $k$ linear expressions, for both $a$ and $b$.

Let $\mathcal{L}(F)$ denote the set of all linear combinations of functions in $F$. For any pseudo-linear function $f(\boldsymbol{x})$, and any set $F$ of pseudo-linear functions in $X$, we say that $f(\boldsymbol{x})$ has the $F$-**interpolatable** property if it satisfies the following two conditions:

(i) $\forall i \in [1..t] : \exists \phi_\star \in \mathcal{L}(F) \; : \; j(f, i) = j(\phi_\star, i)$ , and

(ii) For every $a, b \in [1..t]$ such that $a$ and $b$ are $F$-equivalent, w.l.o.g. by Lemma 4, let the first $k'$ functions out of ($k$ functions) $p_a^{j(f_s,a)}$ (out of $p_b^{j(f_s,b)}$), represent their basis (resp. for $b$). Then, if the $\phi_\star$ in (i) is given by $\sum c_s^a p_a^{j(f_s,a)}$ and $\sum c_s^b p_b^{j(f_s,b)}$ , respectively for $a$ and $b$, then for all $s \in [1..k']$, $c_s^a = c_s^b$.

**Lemma 5.** *If $f$ is a pseudo-linear function in $X$, and $f$ satisfies the $F$- interpolatable property for some set $F$ of pseudo-linear polynomials in $X$, then $f$ is a pseudo-linear function of $F$.*

While the main completeness theorem below is stated for only large finite fields, it holds for all finite fields of characteristic two.

**Theorem 1.** *Let $f_1, f_2, ..., f_k$ be $k$ pseudo-linear functions in $n$ variables $X$, over a field $\mathbb{F}_q$ ($q = 2^m$), such that $q > 2^n$. Collectively, we will refer to these polynomials as $F$. Let $f$ be another pseudo-linear function in $X$. Then, if $f$ is a function of $F$, then $f$ is a pseudo-linear function of $F$.*

## 3 Randomized Pseudo-linear Functions

In this section we consider randomized pseudo-linear functions, or distributions over pseudo-linear families of pseudo-linear functions. A pseudo-linear family of pseudo-linear functions is given by a pseudo-linear function $f'$ in variables $\boldsymbol{x}$ and $\boldsymbol{r}$, where the variables $\boldsymbol{r}$ parametrize the family. Given such an $f'$, a randomized pseudo-linear function $f$ (in $\boldsymbol{x}$) is given by choosing $\boldsymbol{r}$ uniformly and randomly. The simulation question then becomes whether one can generate the target function distribution by sampling the input function distributions.

When we regard the $\boldsymbol{r}$ as formal variables, we can apply Lemma 2 to deduce that $f'$ is expressible in terms of the basic pseudo-linear polynomials in $(\boldsymbol{x}, \boldsymbol{r})$. In particular,

$$f'(\boldsymbol{x}, \boldsymbol{r}) = \sum_{i \in T} g_i(\boldsymbol{x}, \boldsymbol{r}) \cdot p_i^{j(f',i)}(\boldsymbol{x}, \boldsymbol{r})$$

where $T$ is the set of indices of all the guards over $(\boldsymbol{x}, \boldsymbol{r})$.

Consider a guard $g_i$ in just the space of the input variables $\boldsymbol{x}$, with associated set $J$, i.e. $g_i = \prod_{l \in \mathcal{L} \setminus J} l(\boldsymbol{x})^{q-1} \cdot \prod_{l \in J}(1 + l(\boldsymbol{x})^{q-1})$. Consider the set of super-guards $\mathbb{I}_i$ which extend $J$ to $\mathcal{L} \cup \mathcal{L}(\boldsymbol{r})$ and each super-guard $I \in \mathbb{I}_i$ corresponds to a different subspace $J_r \subseteq \mathcal{L}(\boldsymbol{r})$ added to the subspace $J$ (and then taking closure). Thus, we get a set of guards $g_I$ ($I \in \mathbb{I}_i$) corresponding to each guard $g_i$. From now on, when clear from context, we will refer to the randomized function as $f(\boldsymbol{x}, \boldsymbol{r})$, to signify the random variables over which the distribution is defined.

We now show that given any randomized pseudo-linear function $f(\boldsymbol{x}, \boldsymbol{r})$, there is a randomized pseudo-linear function in just one random variable $\hat{r}$, such that it is statistically indistinguishable from $f$. The *new* randomized pseudo-linear function $\hat{f}$ in just one random variable $\hat{r}$ and the same input variable set $\boldsymbol{x}$, is defined in the following way:

- The function $\hat{f}$ will have the same $p$ as $f$ for guards involving only $\boldsymbol{x}$. For guards involving $\hat{r}$, $p$ will be set to zero.
- For each guard $g_i$ (with associated $J$), consider its extension super-guard $I_0 \in \mathbb{I}_i$ corresponding to $J_r = \{0\}$, In this case, $J_{I_0}$ is just $J$. Suppose $p_{I_0}^{j(f,I_0)}(\boldsymbol{x},\boldsymbol{r}) = l_1(\boldsymbol{x}) + l_2(\boldsymbol{r})$. If $l_2$ is not identically 0, then set $p_i^{j(\hat{f},i)}(\boldsymbol{x}) = \hat{r}$, otherwise set $p_i^{j(\hat{f},i)}(\boldsymbol{x}) = l_1(\boldsymbol{x})$.

**Lemma 6.** *Let* $\log q > 2(\rho+\chi)$*, where* $\rho$ *is the number of random variables and* $\chi$ *is the number of input variables in* $f$*. The distribution* $f(\boldsymbol{x})$ *is statistically indistinguishable from* $\hat{f}(\boldsymbol{x})$ *with advantage* $< 1/\sqrt{q}$*.*

We will refer to the functions of the form of $\hat{f}$ as *Simplified Randomized Pseudo-Linear* functions (SRPL). These are functions which can be expressed with guards from $\boldsymbol{x}$ only and just one random variable. Lemma 6 indicates that we can just focus on SRPL functions since any randomized pseudo-linear function is statistically close to an SRPL function.

**Lemma 7 (Homomorphism).** *For any SRPL functions* $\phi_1(\boldsymbol{x})$ *and* $\phi_2(\boldsymbol{x})$*, and for all* $i \in [1..t]$*,*

$$p_i^{j(\phi_1+\phi_2,i)} \;=\; p_i^{j(\phi_1,i)} + p_i^{j(\phi_2,i)}$$

*with the rule that* $\hat{r} + \cdot$ *is re-written as* $\hat{r}$*.*

*Proof.* Follows from the fact that for a fixed $\boldsymbol{x}$, exactly one of the guards evaluates to 1 and the rest evaluate to 0. Also, adding a uniformly distributed random number to any quantity yields a uniformly distributed random number.

**Theorem 2.** *Let* $f_1, f_2, ..., f_k$ *be* $k$ *SRPL functions in* $n$ *variables* $X$*, over a field* $\mathbb{F}_q$ *(*$q = 2^m$*), such that* $q > 2^n$*. Collectively, we will refer to these polynomials as* $F$*. Let* $f$ *be another SRPL function in* $X$*. Then if there exists a probabilistic poly-time (in* $\lg q$*) algorithm* $S^F$ *which makes oracle calls to* $F$*, such that the distribution* $f(X)$ *is statistically indistinguishable from the distribution* $S^F(X)$*, then* $f$ *is an SRPL function of the set of functions* $F$*.*

## 4 Completeness Theorem for Randomized Simulators and Iterated Composition of SRPL Functions

In this section, we consider SRPL functions which can take arguments, modeling oracles which are SRPL functions of secret values and arguments. Thus, for instance it may be required to find if there exists a *randomized simulator* which given access to functionalities which are SRPL functions of secret parameters $X$ and arguments supplied by simulator/adversary, can compute a given SRPL function.

This generalizes the problem from the previous sections, where the simulator could not pass any arguments to the given functions. For simplicity, we will deal

here with functions which only take a single argument, and thus all the functions can be written as $f_i(\boldsymbol{x}, y)$, each SRPL in $\boldsymbol{x}$ and $y$.

So, given a collection of $k$ SRPL functions $F(X, y)$, we now define an **iterated composition of** $F$. Let $\mathbb{F}_q$ be the underlying field as before. An iterated composition $\sigma$ of $F$ is a length $t$ sequence of pairs ($t$ an arbitrary number), the first component of the $s$-th ($s \in [1..t]$) pair of $\sigma$ being a function $\phi_s$ from $F$, and the second component an arbitrary *randomized* function $\gamma_s$ of $s-1$ arguments (over $\mathbb{F}_q$).

Given an iterated composition $\sigma$ of $F$, one can associate a function $f^\sigma$ of $X$ with it as follows by induction. For $\sigma$ of length one, $f^\sigma$ is just $\phi_1(\boldsymbol{x}, \gamma_1())$, recalling that $\phi_1 \in F$. For $\sigma$ of length $t$,

$$f^\sigma(\boldsymbol{x}) = \phi_t(\boldsymbol{x}, \gamma_t(f^{\sigma_{|1}}(\boldsymbol{x}), f^{\sigma_{|2}}(\boldsymbol{x}), ..., f^{\sigma_{|t-1}}(\boldsymbol{x})))$$

where $\sigma_{|j}$ is the prefix of $\sigma$ of length $j$.

Since, SRPL functions in $n$ variables over $\mathbb{F}_q$ are just polynomials in $n$ variables, there is a finite bound on $t$, after which no iterated composition of $F$ can produce a new SRPL function of the $n$ variables. The collection of all functions that can be obtained by iterated composition of $F$ will be referred to as $\mathbf{terms}(F)$. The expression $\mathbf{terms}_T(F)$ will stand for the collection of functions obtained by iterated compositions of $F$ of length less than $T$. In particular we will be interested in $T$ which is bounded by polynomials in $\log q$ and/or $n$, the number of variables in $X$.

Recall the functions in $F$ now have an additional argument $y$. As before, $\mathcal{L}(G)$, for any set of functions $G$ will denote the set of all linear combinations (over $\mathbb{F}_2$) of functions from $G$. Below we define the class $\mathcal{I}^i(F)$ of SRPL functions in $X$, for $i$ an arbitrary natural number. In fact, since the inductive definition will sometimes use functions in both $X$ and $y$, we will just define this class as SRPL functions in $X$ and $y$, though for different $y$, they would evaluate to the same value. In other words, for an arbitrary guard $g_a(\boldsymbol{x})$, which corresponds to a subset $J \subseteq \mathcal{L}(X)$ ($J$ is closed under addition), there are many **super-guards** when viewed as a function of $X$ and $y$, namely with subsets $J' \subseteq \mathcal{L}(X, y)$ ($J'$ closed under addition) such that $J \subseteq J'$ *and* $(\mathcal{L}(X) \backslash J) \subseteq (\mathcal{L}(X, y) \backslash J')$. Thus, for all these super-guards, a SRPL function $\phi(X)$ will have the same $j(\phi, \cdot)$ value (see Section 2).

However, and more importantly, with $y$ set to some linear expression $l(\boldsymbol{x}) \in \mathcal{P}_a(X)$ (including zero), *exactly one* of these (super-)guards has the property that $J'_{y|l(\boldsymbol{x})} = J$ (Note the subscript $y|l(\boldsymbol{x})$ means $l(\boldsymbol{x})$ is substituted for every occurrence of $y$ in $J'$). This particular $J'$ is given by

$$J' \;=\; \mathcal{L}(J, \{y + l(\boldsymbol{x})\})$$

In this case we say that this super-guard of $g_s$ is consistent with $y + l(\boldsymbol{x}) = 0$. The super-guard corresponding to $J' = J$ will be called the **degenerate super-guard** of $g_a(\boldsymbol{x})$.

Now we define the SRPL function which is the composition of $f_s$ and $h$, i.e. $f_s \circ h$, where $f_s$ is a SRPL function in $X$ and $y$, and $h$ is a SRPL function in $X$,

by defining its components in the basis for SRPL functions. For any guard $g_i(\boldsymbol{x})$ (of functions in $X$), let $g_I(\boldsymbol{x}, y)$ be the unique (super-) guard, mentioned in the previous paragraph, which is consistent with $y$ set to $p_i^{j(h,i)}$ (note the map $j$ here is for guards corresponding to $X$, and in general it will be clear from context whether we are referring to map $j$ for guards corresponding to $X$ or $X, y$). Then, define

$$p_I^{j(f_s \circ h, I)}(\boldsymbol{x}, y) = p_I^{j(f_s, I)}(\boldsymbol{x}, p_i^{j(h,i)}(\boldsymbol{x}, y))$$

Further, for all $I'$ which are super-guards of $i$, we set $p_{I'}^{j(f_s \circ h, I')}$ to be the same value (as $f_s \circ h$ is only a function of $X$). Note that since each $p$ is just a linear function, this implies that each component of $f_s \circ h$ is a linear function of $X$ (and hence $X, y$). In particular, $(f_s \circ h)(\boldsymbol{x}) = f_s(\boldsymbol{x}, h(\boldsymbol{x}))$.

Define **Compose**$(F(X, y), H(X))$, where $F(X, y)$ are a set of SRPL functions in $X, y$ and $H(X)$ is a set of SRPL functions in $X$, to be the set of all functions $f_s \circ h$, where $f_s \in F(X, y)$ and $h \in H(X)$.

For each SRPL function $f_s$ of $X$ and $y$, we also need to define a SRPL function (in $X$ called **degenerate**$(f_s)$, which for each guard $g_a(\boldsymbol{x})$, defines the corresponding $p$ function using its degenerate super-guard. Thus,

$$p_a^{j(\text{degenerate}(f_s),a)}(\boldsymbol{x}) = p_I^{j(f_s,I)}(\boldsymbol{x}, 0),$$

where $I$ is the degenerate super-guard of $g_a$.

Now, we are ready to define the iterated SRPL functions. Define

$$\begin{aligned} \mathcal{I}^0(F) &= \mathcal{L}(\text{Compose}(F, \text{degenerate}(F))) \\ \mathcal{I}^{i+1}(F) &= \mathcal{L}(\mathcal{I}^i(F) \cup \text{Compose}(F, \mathcal{I}^i(F))), \text{ for } i \geq 0. \end{aligned}$$

Since, these functions are just polynomials over finite fields (in fact defined over $\mathbb{F}_2$), the above iteration reaches a fixed-point at an $i$ bounded by a function only of $n$. We will denote the fixed-point by just $\mathcal{I}(F)$. Now, we generalize the definitions of $F$-equivalence and $F$-interpolatable from Section 2.2. Two guards $g_a(\boldsymbol{x})$ and $g_b(\boldsymbol{x})$ are said to be $F^*$-equivalent if for every $\phi(\boldsymbol{x})$ in $\mathcal{I}(F)$, it is the case that $j(\phi, a) = 0$ iff $j(\phi, b) = 0$ and $j(\phi, a) = \$$ iff $j(\phi, b) = \$$, where $j(\phi, \cdot) = \$$ indicates the special index for the single random variable. The definition of $F^*$**-interpolatable** property is same as the $F$-interpolatable property except that $\mathcal{L}(F)$ is replaced by $\mathcal{I}(F)$ and the random variable is accounted for.

**Lemma 8.** *If $f$ is an SRPL function of $n$ variables $X$ over a field $\mathbb{F}_q$, and $f$ satisfies the $F^*$-interpolatable property, for some set $F$ of SRPL polynomials in $X, y$, then there exists a probabilistic poly-time (in $\lg q$) algorithm $S^F$, such that the distribution $f(X)$ is statistically indistinguishable from the distribution $S^F(X)$, with error at most $2^n/q$.*

**Theorem 3.** *Let $f_1, f_2, ..., f_k$ be $k$ SRPL functions in $n$ variables $X$ and an additional variable $y$, over a field $\mathbb{F}_q$ such that $q > 2^{4n}$. Collectively, we will refer to these polynomials as $F(X, y)$. Let $T$ be a positive integer less than $2^n (< q^{1/4})$. Let $f$ be another SRPL function in $X$. Then if there exists a probabilistic*

*poly-time (in* $\lg q$*) algorithm* $S^{\boldsymbol{terms}_T(F(X,y))}$*, such that the distribution* $f(X)$ *is statistically indistinguishable from the distribution* $S^{\boldsymbol{terms}_T(F(X,y))}(X)$*, then* $f$ *is* $F^*$*-interpolatable.*

## 5 Proof Automation in the Universally Composable Model

A proof of security in the Universally Composable (UC) model boils down to the following: as input, we are given two sets of algorithms: one called an *Ideal Functionality* which is a set of algorithms $F = \{F_1, F_2, \cdots\}$, and another *Real Protocol* which is a set of algorithms $P = \{P_1, P_2, \cdots\}$. We say that $P$ realizes $F$ if it is possible to construct an algorithm $S$, called ideal world adversary (usually built as a simulator), that invokes the functions in $F$, such that the following holds: *For any PPT algorithm A, there exists a PPT algorithm S, such that for any PPT environment Z, the execution of A with calls to P is indistinguishable from the execution of S with calls to F.*

We now formally describe the language $L^{\$,\oplus,\mathsf{if}}$ in Table 1 which corresponds to the branching programs over data objects from fields of characteristic two as mentioned in the introduction.

**Table 1.** Grammar for the Language $L^{\$,\oplus,\mathsf{if}}$

| | | | | |
|---|---|---|---|---|
| (expressions) | $AE$ | ::= | $x_1 \mid x_2 \mid \cdots$ | variables |
| | $XE$ | ::= | $AE \mid AE \oplus XE$ | bitwise xor expression |
| | $BE$ | ::= | $\mathtt{true} \mid (XE == XE) \mid BE \wedge BE \mid \neg BE$ | boolean expression |
| (assignments) | $a$ | ::= | $x \leftarrow \$$ | assign new random no. |
| | | | $x := XE$ | assign xor expression |
| (program) | $\pi$ | ::= | $a;$ | single action |
| | | | $\pi a;$ | sequence of actions |
| | | | $\mathtt{if}\ BE\ \mathtt{then}\ \pi\ \mathtt{else}\ \pi$ | conditional |

**Definition 1** ($L^{\$,\oplus,\mathsf{if}}$)**.** *An Ideal Functionality* $F$ *and a real protocol* $P$ *are in the language* $L^{\$,\oplus,\mathsf{if}}$ *if*

- $F$ *is a set of programs* $\{f_1(\boldsymbol{x}, \boldsymbol{y}), f_2(\boldsymbol{x}, \boldsymbol{y}), \cdots\}$.
- $P$ *is a single program* $\{f(\boldsymbol{x})\}$.

*such that* $f_1(\boldsymbol{x}, \boldsymbol{y}), f_2(\boldsymbol{x}, \boldsymbol{y}), \cdots$ *and* $f(\boldsymbol{x})$ *are all described as* $L^{\$,\oplus,\mathsf{if}}$ *programs, as defined in Table 1.*

The **semantics** of this language is that $\boldsymbol{x}$ is a set of inputs passed by the environment at the outset of execution and $\boldsymbol{y}$ is a set of parameters that the simulator

is allowed to pass to the functionalities. All the parameters and random numbers are represented as $\lg q$-bit strings, corresponding to elements in $\mathbb{F}_q$. The programs in $F$ can be called *in any order* and an *arbitrary number of times*, whereas $P$ is called *only once*. This means that the real world adversary $A$ does not send any message to the protocol - however, the environment $Z$ initializes the arguments $\boldsymbol{x}$ as per its choice. At the end, the protocol sends a value $f(\boldsymbol{x})$ to the adversary $A$. The challenge for the simulator $S$ is to generate a statistically indistinguishable value using the functions in the ideal functionality $F$. It can call these functions in any order and an arbitrary number of times with arguments of its choice. It doesn't know the values $\boldsymbol{x}$, but observes the results of the ideal functionality calls. The simulatability question is then whether a probabilistic poly-time simulator exists which can produce an acceptable result for any choice of the environment supplied arguments.

The following lemma connects the language $L^{\$,\oplus,\mathsf{if}}$ to our results in the previous sections.

**Lemma 9.** *All the variables in an $L^{\$,\oplus,\mathsf{if}}$ program $f(\boldsymbol{z})$ are randomized pseudo-linear in $\boldsymbol{z}$.*

We now proceed to the main theorem.

**Theorem 4 (Completeness of $L^{\$,\oplus,\mathsf{if}}$).** *There is a decision procedure, which given an Ideal Functionality $F$ and a Real Protocol $P$ described in the language $L^{\$,\oplus,\mathsf{if}}$, decides if $P$ realizes $F$ in the Universally Composable model.*

*Proof (Theorem 4).* By Lemma 9, all the functions in $P$ and $F$ compute randomized pseudo-linear functions in the inputs. By Lemma 6, with negligible error, we can assume that these are given as SRPL functions. Observe that a simulator employing $T$ calls to the ideal functionality computes over values in $\mathbf{terms}_T(F(X, y))$.

Now, by Theorem 3, if $f$ is simulatable using $\mathbf{terms}_T(F(X, y))$, with $T < q^{1/4}$, then $f$ is $F^*$-interpolatable. $F^*$-interpolatability can be decided by computing $\mathcal{I}(F)$, which can be computed in time independent of $\lg q$. Further, by Lemma 8, if $f$ is $F^*$-interpolatable, then there exists a probabilistic polynomial time (in $\lg q$) simulator.

## 6 Work in Progress and Open Problems

Consider an extension of the language $L^{\$,\oplus,\mathsf{if}}$ , where we add a fixed number of variables to the Ideal world that are persistent across subroutine calls. Let us call this language $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$. We describe the key ideas for developing a decision procedure for $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$, which is a work in progress.

We first construct stateful iterated compositions of $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$ subroutines. We construct a tree where each node is such a composition and its subtrees denote further compositions extending its own computation. The key observation is that there is only a finite number of such nodes which are distinct modulo renaming of uniformly random quantities. In other words, the nodes fall into a finite number

of equivalence classes modulo permutation of uniformly random quantities and hence represent the *same randomized algorithm*. An important property of these equivalence classes is that two members of a class lead to subtrees which are equivalent as sets.

This leads to the conclusion that there is a finite set of stateful iterated compositions. A considerably harder theorem is to prove completeness: if there is a probabilistic poly-time simulator, then there is a simulator which is a stateful iterated composition of the subroutines. Based on our work so far this result seems plausible, and we pose a formal proof of such to be an *open problem*.

**Key Ideas for Encryption and Signatures.** The UC formulation of the encryption and signature primitives makes them expressible in very abstract terms. We leverage the fact that standard notions of security of these primitives have been shown to be equivalent to the UC formulation. Specifically, we express protocols in the *hybrid model*, where the concrete operations for encryptions and signatures are replaced by their ideal counterpart. The proofs of security translate due to the *Composition Theorem* supported by the UC framework.
*Signatures.* The UC formulation of signatures can be found in [15]. In addition to the operations in $L^{\$,\oplus,\mathsf{if}}$, we need function variables (*viz.*, $s$ and $v$) and storage (for the records). For a single session of a protocol, the honest protocol participants only do a bounded number of signatures. However, the adversary may make an unbounded number of calls to the verification function - but this does not create any requirement for more storage. Hence the language $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$ suffices for the storage part.

As regards the function variables $s, v, v'$, observe that they do not have to satisfy any equation. Hence they can be treated as *uninterpreted* function symbols. For example $s(m)$ can be represented as the tuple $\langle \text{“}s\text{”}, m \rangle$, where “$s$” is a constant string. To support these entities, we only need to define tuples, constants and equality of tuples.

*Public-Key Encryption.* The UC formulation of PKE can be found in [15]. The discussion on signatures carries over to PKE. In addition, we need to distinguish the ciphertexts being output on separate invocations of the Encryption subroutine. This can be done by tagging the ciphertexts with a uniformly random quantity generated at each invocation: $[r \leftarrow \$;\ c := \langle e', \mathbf{0}, r \rangle\ ;]$.

However, in contrast to the signature functionality, the adversary can induce a requirement for unbounded storage by calling the Encryption subroutine multiple times. The language $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$ is only able to support a bounded number of such calls - hence we only have a conditional security proof. While for individual protocols it can be rather simple to prove that it suffices to show realizability for an adversary which makes only a bounded (or even single) number of calls to the hybrid encryption functionality, it is an interesting open problem to prove a structural meta-theorem which establishes the sufficiency of a bounded number of calls for a general class of protocols.

### 6.1 Password-Based Key Exchange

Password-based key exchange is an important security problem which has been studied extensively in cryptographic research [19], and which brings out the power of the UC framework particularly well. Canetti et al [20] proposed an Ideal Functionality for password-based key exchange. (See [20] for a formal description of this ideal functionality $\mathcal{F}_{\text{PWKE}}$).

Consider two parties $P_i$ and $P_j$ that wish to come up with a common cryptographically strong key based on the fact that they share the same password. The idea is to capture the fact that modulo the adversary outright guessing the password exactly during an active session between the parties, it has no control (or information) on the key being generated. It is allowed to interrupt sessions by tampering with the messages being exchanged, but doing so only results in the parties ending up with different uniformly randomly distributed keys. If, however, the session is not interrupted, the parties end up with the same key which is distributed uniformly and randomly and is not controlled by the adversary.

We describe a protocol $\Pi_{\text{ICPWKE}}$ in the ideal cipher model (Figure 1), which is a candidate to realize $\mathcal{F}_{\text{PWKE}}$. In the ideal cipher model, the results of two decryptions are the same if the key is identical. Otherwise, the results are uniformly and independently random. The protocol is symmetric from the perspective of both the participants - so we describe the actions of just one party $P_i$. Both parties get a password from the environment $\mathcal{E}$. Party $P_i$ generates a random number $r_1$, encrypts it and sends the ciphertext $c_1$ to the peer. When it receives a response $c'_2$, it first checks whether its own message was reflected. If so, it outputs a random key to the environment. Otherwise, it decrypts the response using its password $pw_1$ and xors the plaintext with $r_1$. The resulting quantity is output as its key to the environment.

| Party $P_i$ | Adv | Party $P_j$ |
|---|---|---|
| $\mathcal{E}$ | | $\mathcal{E}$ |
| $\downarrow pw_1$ | | $\downarrow pw_2$ |
| $r_1 \leftarrow \$$ | | $r_2 \leftarrow \$$ |
| $c_1 \leftarrow enc_{pw_1}(r_1)$ | | $c_2 \leftarrow enc_{pw_2}(r_2)$ |
| | $\xrightarrow{c_1}$ $\xleftarrow{c_2}$ | |
| | $\xleftarrow{c'_2}$ $\xrightarrow{c'_1}$ | |
| `if` $(c'_2 == c_1)$ `then` $sk_1 \leftarrow \$$ `else` $d_1 \leftarrow dec_{pw_1}(c'_2)$ $sk_1 \leftarrow r_1 \oplus d_1$ | | `if` $(c'_1 == c_2)$ `then` $sk_2 \leftarrow \$$ `else` $d_2 \leftarrow dec_{pw_2}(c'_1)$ $sk_2 \leftarrow r_2 \oplus d_2$ |
| $\downarrow sk_1$ | | $\downarrow sk_2$ |
| $\mathcal{E}$ | | $\mathcal{E}$ |

**Fig. 1.** Protocol for Password-based Key Exchange using Ideal Cipher

Consider the following *ideal functionality for the ideal cipher* primitive. The functionality takes two arguments: a key and a plaintext. It has a table where each entry is a triplet (key, plaintext, ciphertext). The table is initially empty. It supports two subroutines: encrypt(key, plaintext) and decrypt(key, ciphertext). The **encrypt** subroutine, given input $(key, plaintext)$, generates a random number $r$, stores $(key, plaintext, r)$ in the table and outputs $r$. The **decrypt** subroutine, given input $(key, ciphertext)$, looks up if there is an entry $(key, p, ciphertext)$ in the table. If so, it outputs $p$. Otherwise, it generates a random number $r$, stores $(key, r, ciphertext)$ in the table and outputs $r$.

Now consider the real-world scenario where the adversary intercepts the first message $c_1$ and changes it to $c_1'$ before transmitting to $P_j$. The adversary's action may involve querying the ideal cipher in the hybrid model. More importantly, if the password is weak, the adversary maybe able to guess the password, and hence a proper simulation would require the simulator to extract this password guess from the call to the ideal cipher, and use that in the TestPwd subroutine of $\mathcal{F}_{\text{PWKE}}$.

We now describe how an extension of our decision procedure might automatically figure out such a simulator, as the languages for the real protocol (in the ideal cipher hybrid model) and for the ideal functionality $\mathcal{F}_{\text{PWKE}}$ are covered by the language $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$. First, however we need a theorem which states that it suffices to consider an adversary which makes only a single call to the hybrid ideal cipher functionality. It is plausible that such meta-theorems can be proven as general structural theorems and that is an interesting open problem. We can then in-line a single ideal cipher call in a serialization of the protocol (the variables input by the adversary can be named, say $kk$ and $rr$). Observe that the following operations are sufficient to describe the ideal cipher functionality: equality testing, conditional branches, random number generation and table storage and lookups. When we consider a constant number of calls to the ideal cipher, the table operations reduce to assignment statements and equality testing. The ideal functionality $\mathcal{F}_{\text{PWKE}}$ is clearly supported by the language $L^{\$,\oplus,\mathsf{if},\mathsf{state}}$. Finally, notice that the variables $c_1'$ and $c_2'$ are also available to the simulator. Thus the decision problem is whether the serialization of the protocol (with a single inlined ideal cipher call) can be obtained as a randomized iterated composition of $\mathcal{F}_{\text{PWKE}}$, where the simulator also has access to variables $kk$, $rr$, $c_1'$ and $c_2'$.

**Acknowledgements.** The authors would like to thank Daniele Micciancio, Russell Impagliazzo and referees for helpful comments.

## References

1. Abadi, M., Rogaway, P.: Reconciling Two Views of Cryptography. In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 3–22. Springer, Heidelberg (2000)

2. Cortier, V., Warinschi, B.: Computationally Sound, Automated Proofs for Security Protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
3. Canetti, R., Herzog, J.C.: Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg (2006)
4. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (pcl). Electr. Notes Theor. Comput. Sci. 172, 311–358 (2007)
5. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: IEEE Symposium on Security and Privacy, pp. 140–154. IEEE Computer Society (2006)
6. Micciancio, D., Warinschi, B.: Completeness theorems for the abadi-rogaway language of encrypted expressions. Journal of Computer Security 12(1), 99–130 (2004)
7. Impagliazzo, R., Kapron, B.M.: Logics for reasoning about cryptographic constructions. In: FOCS, pp. 372–383. IEEE Computer Society (2003)
8. Parikh, R.: Existence and feasibility in arithmetic. J. Symb. Log. 36(3), 494–508 (1971)
9. Buss, S.R.: Bounded arithmetic, proof complexity and two papers of parikh. Ann. Pure Appl. Logic 96(1-3), 43–55 (1999)
10. Barthe, G., Grégoire, B., Béguelin, S.Z.: Formal certification of code-based cryptographic proofs. In: POPL, pp. 90–101 (2009)
11. Barthe, G., Grégoire, B., Heraud, S., Béguelin, S.Z.: Computer-Aided Security Proofs for the Working Cryptographer. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 71–90. Springer, Heidelberg (2011)
12. Pereira, O., Lynch, N., Liskov, M., Kaynar, D., Cheung, L., Segala, R., Canetti, R.: Analyzing Security Protocols Using Time-Bounded Task-PIOAs. Discrete Event Dynamic Systems 18, 111–159 (2008)
13. Ramanathan, A., Mitchell, J., Scedrov, A., Teague, V.: Probabilistic Bisimulation and Equivalence for Security Analysis of Network Protocols. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 468–483. Springer, Heidelberg (2004)
14. Barthe, G., Daubignard, M., Kapron, B., Lakhnech, Y., Laporte, V.: On the Equality of Probabilistic Terms. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16 2010. LNCS, vol. 6355, pp. 46–63. Springer, Heidelberg (2010)
15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM (1987)
17. Canetti, R., Gajek, S.: Universally composable symbolic analysis of diffie-hellman based key exchange. Cryptology ePrint Archive, Report 2010/303 (2010), http://eprint.iacr.org/
18. Jutla, C.S., Roy, A.: A completeness theorem for pseudo-linear functions with applications to UC security. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 17, p. 92 (2010)
19. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: ACM Conference on Computer and Communications Security, pp. 244–250 (1993)
20. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally Composable Password-Based Key Exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005)