Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 92 (2010)

A Completeness Theorem for Pseudo-Linear Functions with Applications to UC Security

Charanjit S. Jutla
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
csjutla@us.ibm.com

Arnab Roy
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
arnabroy@us.ibm.com

Abstract

We consider multivariate pseudo-linear functions over finite fields of characteristic two. A pseudo-linear polynomial is a sum of guarded linear-terms, where a guarded linear-term is a product of one or more linear-guards and a single linear term, and each linear-guard is again a linear term but raised to the power q-1, where q is the field size. Pseudo-linear functions over GF(2^m) are given by pseudo-linear polynomials defined over GF(2).

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 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. 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. Proceeding further, we generalize the theorem to randomized pseudo-linear 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, and the decision procedure runs in computational time independent of m.

These theorems have implications for automatic proving of universally-composable security theorems for ideal and real functionalities composed of if-then-else programs with (uniform) random number generation and data objects from additive group of GF(2^m). The theorems imply that, within this language framework, there is a decision procedure to find out if a real functionality realizes an ideal functionality, and this procedure is in computational time independent of m (which is essentially the security parameter).

ISSN 1433-8092

1 Introduction

Before we define pseudo-linear functions, we mention that pseudo-linear functions originate as functions computed by 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, and closed under negation and conjunction.

So, consider a finite field $\mathbb{F}_q$, where $q = 2^m$. Then m-bit (bit-wise) exclusive-OR just corresponds to addition in this field. Further, an equality constraint of the form $l_1(\vec{x}) = l_2(\vec{x})$ can then be written as

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

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

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

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

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

A **pseudo-linear** multivariate polynomial defined over sub-field $\mathbb{F}_2$ is then a polynomial which is a sum of guarded linear-terms [Dij75]; a *guarded linear-term* is a polynomial which is the product of a linear (over $\mathbb{F}_2$) polynomial 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). In formal terms, the objects in consideration are in $\mathbb{F}_2[x_1, ..., x_n]/(x_1^q + x_1, ..., x_n^q + x_n)$. They are also further restricted by the fact that all expressions in the guards are linear instead of affine, but we will later see how to introduce constant additive terms from $\mathbb{F}_q$ (Appendix B).

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. This follows (by induction) from the observation that

$$(f_1(\vec{x})^{q-1} l_1(\vec{x}) + f_2(\vec{x}))^{q-1} = f_1(\vec{x})^{q-1}(l_1(\vec{x}) + f_2(\vec{x}))^{q-1} + (1 + f_1(\vec{x})^{q-1}) \cdot f_2(\vec{x})^{q-1}$$

where f_1 and f_2 are arbitrary pseudo-linear functions. The observation itself follows by considering the two cases where $f_1(\vec{x})$ is zero or not.

More importantly, the if-then-else programs mentioned above compute exactly the pseudo-linear functions. A more detailed description of such programs and how they relate to pseudo-linear functions can be found in Section 6.

Above, we saw how a multivariate polynomial $p(\vec{x})$ yields a function from $\mathbb{F}_q^n$ to $\mathbb{F}_q$. More generally, if we are given n polynomials $f_1(\vec{z})$ to $f_n(\vec{z})$ (where $\vec{z}$ are k formal variables), then $p(f_1(\vec{z}), ..., f_n(\vec{z}))$ yields a function from $\mathbb{F}_q^k$ to $\mathbb{F}_q$, which we say is a **pseudo-linear function** of $f_1, ..., f_n$.

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 result 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 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.

The primitives in this algebraic structure are motivated by cryptography, in particular the representation of security properties and cryptographic protocols in the the Universally Composable (UC) framework [Can01]. The core technique involved in the UC framework is to prove equivalence of the system in consideration - the real protocol - and an ideal functionality which naturally captures the security requirements. The proof is by demonstrating a *simulator* which has access to the ideal functionality and is able to produce results indistinguishable from an execution of the target function.

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 functions $f_1(\vec{z}, \vec{y}), f_2(\vec{z}, \vec{y}), ..., f_k(\vec{z}, \vec{y})$, where $\vec{y}$ are arguments which the simulator can supply. We then prove that if some f is a pseudo-linear function of $\vec{z}$, and can be computed by a sub-exponential (in m) length iterated composition of the given k functions, then it can be computed by a *probabilistic* iterated *pseudo-linear* composition of the given k functions, i.e. f_1, f_2, ..., f_k.

Proceeding further, we include random number generation as an additional primitive and extend the decision procedure to find if a *probabilistic poly time* simulator exists for the given set of *randomized* ideal functionalities and *randomized* target function.

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 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. There are additional issues involved, e.g. the real protocol may be given in a hybrid model [Can01], and the adversary may make iterative calls to the hybrid functionalities[1]. We discuss how our work is motivated by the UC framework and give an example in Appendix I. Although, there have been many pieces of work in formal methods for cryptographic protocols [AR00, CH06, MW04, DDMR07], this to the best of our knowledge is a novel approach to theorem proving of security protocols.

There is a technical restriction of a sub-exponential length iterated composition which is required to rule out deterministic brute force searches, which a computationally bounded simulator is not allowed anyway. Finally, we remark that our completeness results require sufficiently large fields (as a function of the number of variables in f), 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.

The difficulty in proving the completeness theorem stems from the fact that pseudo-linear polynomials can have individual degrees (i.e. of individual variables) exceeding q-1, and hence

[1]We do not deal with computational assumptions in this paper, and we expect that the hybrid functionalities themselves embody such assumptions (see e.g. [CG10])

it may be subject to reduction modulo $x^q = x$. Similar problems occur in local testing of low degree polynomials [JPRZ04, KR04], and we would like to point out that pseudo-linear functions are intimately related to Generalized Reed-Muller Codes [KLP68]. Thus, for example it is not immediately clear what constitutes a basis for pseudo-linear polynomials. We first show a basis for pseudo-linear polynomials, and then show a necessary and sufficient condition involving the basis for a pseudo-linear function of $\vec{x}$ to be a pseudo-linear function of other pseudo-linear functions of $\vec{x}$. A detailed example illustrating these issues and the proof idea can be found in Appendix H.1. We would also like to mention that the class of pseudo-linear functions do not form an ideal in $\mathbb{F}_q[\vec{x}]$, and hence the vast field of Gröbner basis is not applicable.

We remark that our theorem does not yet address stateful functions. Many important functionalities, e.g Random Oracle, Public Key Encryption etc. require stateful functionalities [Can01]. Moreover, these functionalities require arbitrarily large tables of state, although the entries in the table are of fixed size (i.e. depend only on m). This is important to note, as we show that with arbitrarily large sized entries in tables, the question of simulatability is undecidable (see Appendix J). However, we expect our positive results to extend to stateful functionalities, and also to functionalities with tables with limited capabilities, e.g. fixed sized entries. We also expect our results to extend to other groups and cryptographic constructs with appropriate axiomatization.

The rest of the paper is organized as follows. Section 2 describes and proves a basis for pseudo-linear functions. Section 3 proves an interpolation theorem for pseudo-linear function. Section 4 proves the Completeness theorem for pseudo-linear functions. Section 5 defines iterated composition of pseudo-linear functions and proves the Completeness theorem for iterated pseudo-linear functions. Section 6 relates the results in this paper to proof automation in the UC model.

2 A Basis for Pseudo-Linear Functions

In this section we fix a field $\mathbb{F}_q$ of size $q = 2^m$.

Let $\mathcal{L}$ stand for all linear expressions (including zero) in n variables, say $x_1, x_2, ..., x_n$ (the unordered collection will be referred to as X). We define the set of **elementary pseudo-linear (EPSELIN)** polynomials to be all polynomials of the form

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

where $p(\vec{x})$ is in $\mathcal{L}$, and J is any subset of $\mathcal{L}$ such that it 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 $\mathbb{F}_q$.

Generalizing (and specializing) the earlier definition of a guard, we will refer to expressions of the form

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

as **guards**.

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 then 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 J is set to zero. This for example, can be accomplished by considering a row-echelon normal form of J.
3. None of the variables in R occur in $p(\vec{x})$.

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 $p(\vec{x})$ 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 $l(\vec{x})$ in J *must have at least one variable from* R. Thus, $p(\vec{x})$ cannot be in J.

Finally, we define a REPSELIN-polynomial to be a **basic pseudo-linear** polynomial if the linear term $p(\vec{x})$ 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.

Next we argue that any pseudo-linear polynomial can be expressed as a (xor-) sum of basic pseudo-linear polynomials. For this, we just need to show that any pseudo-linear polynomial of the form

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

where J is a subset of $\mathcal{L}$, can be expressed as sum of EPSELIN-polynomials. This follows easily by induction on the size of J, and by noting that pseudo-linear polynomials with guards

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

such that $\mathcal{L} \setminus J$ includes a linearly dependent expression of J are identically zero.

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 amongst 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 $p(\vec{x})$. We refer to all linear combinations of $X \setminus R$ as $\mathcal{P}_i(X)$, including the linear term zero. Let $|\mathcal{P}_i(X)| = s_i + 1$. (Note, $(s_i + 1)$ is two to the power size of the subset of variables associated with g_i.) The linear terms in $\mathcal{P}_i(X)$ can be named $p_i^j(\vec{x})$, j ranging from 0 to s_i (not to be confused with exponent). W.l.o.g., zero will always be $p_i^0(\vec{x})$.

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

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

where T is a subset of $[1..t]$, and each $p_i^{j(\phi,i)}(\vec{x}) \in \mathcal{P}_i(X)$. 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 $p_i^0(\vec{x})$ is always taken to be zero. Thus the above representation of $\phi(\vec{x})$ 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 are the basic pseudo-linear polynomials a basis, a slightly more complicated characterization can be given for smaller fields.

Lemma 1 (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.*

Proof:

We have already shown above that any pseudo-linear polynomial can be represented as a sum of basic pseudo-linear polynomials, in fact defined by the map $j(\phi, \cdot)$ above. So, here we focus on showing that any pseudo-linear function $\phi(\vec{x})$ has a unique such representation.

So, for the sake of contradiction, suppose the everywhere zero function $\mathbf{0}$ has a non-zero representation, and let that be represented by the map $j(\mathbf{0}, i)$. Now consider any $\bar{i}$ where $j(\mathbf{0}, \bar{i}) \neq 0$, and lets call $p_i^{j(\mathbf{0},i)}(\vec{x})$ by just $p(x)$ ($\neq 0$). In other words, this representation of $\mathbf{0}$ has the term

$$g_{\bar{i}} \cdot p(\vec{x})$$

Let,

$$g_{\bar{i}} = \prod_{l \in \mathcal{L} \backslash J} l(\vec{x})^{q-1} \cdot \prod_{l \in J} (1 + l(\vec{x})^{q-1})$$

for some subspace $J \subseteq \mathcal{L}$. Let the rank of J be r. Let R be the lexicographically greatest set of r variables occuring in J which can be expressed in terms of smaller indexed variables when J is set to zero. Recall, by definition, none of the variables in R occur in $p(\vec{x})$.

Claim: With the set of equations J set to zero, we can solve for all linear expressions in $\mathcal{L} \backslash J$ to be non-zero, and hence also set $p(\vec{x})$ to non-zero.

This would first of all imply that all guards other than $g_i(\vec{x})$ evaluate to zero: if the guard $g_a(\vec{x})$ is given by subset $J_a \subseteq \mathcal{L}$ ($J_a \neq J$), then if $J \backslash J_a$ is non-empty, we get that $\mathcal{L} \backslash J_a$ has an $l(\vec{x})$ from J which makes $l(x)^{q-1}$ zero, and if $J_a \backslash J$ is non-empty, we get that J_a has an $l(\vec{x})$ from $\mathcal{L} \backslash J$ which makes $(1 + l(\vec{x})^{q-1})$ zero.

Further, the guard $g_{\bar{i}}(\vec{x})$ will be non-zero, and hence $g_{\bar{i}}(\vec{x})p(\vec{x})$ would be non-zero, and consequently the given representation $j(\mathbf{0}, i)$ leads to a non-zero function, a contradiction, which would prove the lemma.

Now, to prove the above claim, recall that J is closed under addition, and $p(\vec{x})$ is in $\mathcal{L} \backslash J$. Let r be the rank of J. Consider a basis $\bar{J}$ of a complementary subspace of J. If our underlying field is of size at least 2^{n-r}, we can set J to zero, and each $l_i(\vec{x})$ of $\bar{J}$ ($i \in [1..n-r]$) to e_i, where the e_i ($i \in [1..n-r]$) are linearly independent over $\mathbb{F}_2$. Thus, all linear expressions in $\mathcal{L} \backslash J$ evaluate to non-zero values, as any l in $\mathcal{L} \backslash J$ is a non-trivial linear combination of $\bar{J}$ plus an l' from J. □

Note on small fields. In smaller fields some of the basic pseudo-linear polynomials, which are non-trivial functions in large fields, turn out to be identically zero. Thus the basis is smaller, but more complicated to characterize.

Lemma 2 (Homomorphism) *For any pseudo-linear functions $\phi_1(\vec{x})$ and $\phi_2(\vec{x})$, 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. □

3 Interpolation Property for Pseudo-Linear Functions

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 $f(\vec{x})$ 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. $f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x})$ is pseudo-linear, it 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 $g_a(\vec{x})$ and $g_b(\vec{x})$ are F-**equivalent** if for every linear combination ϕ 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 3 *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 (2). 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 function $f(\vec{x})$, and any set F of pseudo-linear functions in X, we say that $f(\vec{x})$ 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 (3), 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 4 *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.*

Proof: Indeed, consider $\hat{T} = [1..t] \,/\, \cong_F$, where t is the number of guards, i.e. $|\mathcal{G}(X)|$. We pick the smallest elements from $[1..t]$ to represent each equivalence class in $\hat{T}$. Define a function $h(\vec{x})$ to be the following:

$$h(\vec{x}) = \sum_{u\in\hat{T}} \prod_{\phi\in\mathcal{L}(F):j(\phi,u)\neq 0} \phi(\vec{x})^{q-1} \cdot \prod_{\phi\in\mathcal{L}(F):j(\phi,u)=0} (1 + \phi(\vec{x})^{q-1}) \cdot \phi_u(\vec{x}) \qquad (1)$$

where for each u, ϕ_u is some function $\phi_\star$ satisfying the F-interpolatable property (i) above.

Now by definition, $h(\vec{x})$ is pseudo-linear in F. We now show that $h{=}f$, i.e. for all $\vec{x} \in (\mathbb{F}_q)^n$, $h(\vec{x}) = f(\vec{x})$. Fix any $\vec{x}^*$ in $(\mathbb{F}_q)^n$. Let $J \subseteq \mathcal{L}$, such that all linear functions in J evaluate to zero at $\vec{x}^*$, and all linear functions in $\mathcal{L}\backslash J$ evaluate to non-zero quantities at $\vec{x}^*$. Clearly, J is closed under addition, and hence J corresponds to a guard g_i. In other words, $g_i(\vec{x}^*) = 1$, and for all other $i' \in [1..t]$: $g_{i'}(\vec{x}^*) = 0$. Thus, $f(\vec{x}^*) = p_i^{j(f,i)}(\vec{x}^*)$, and similarly, for all $\phi \in \mathcal{L}(F)$, $\phi(\vec{x}^*) = p_i^{j(\phi,i)}(\vec{x}^*)$. By definition of i (i.e. g_i corresponding to J above, and hence $p_i^{j(\phi,i)} \in \mathcal{L}\backslash J$), it follows that $\phi(\vec{x}^*)$ is zero iff $j(\phi, i) = 0$.

Now, in equation (1), we show that the only u for which the "guards" evaluate to be non-zero (i.e. one), is the one corresponding to the equivalence class of i in $\hat{T}$ (say, u_i). In fact, for i (and its F-equivalent u_i) the "guards" indeed evaluate to 1. For all other i', if the "guards" evaluate to one, then by definition of F-equivalence, those i' are F-equivalent to i.

Thus, $h(\vec{x}^*) = \phi_{u_i}(\vec{x}^*)$, and since ϕ_{u_i} is pseudo-linear in X,

$$\phi_{u_i}(\vec{x}^*) = p_i^{j(\phi_{u_i},i)}(\vec{x}^*) = \sum_s c_s^{u_i} p_i^{j(f_s,i)}(\vec{x}^*) = \sum_s c_s^i p_i^{j(f_s,i)}(\vec{x}^*) = p_i^{j(\phi_i,i)}(\vec{x}^*).$$

Thus, $h(\vec{x}^*) = p_i^{j(f,i)}(\vec{x}^*)$, which is same as $f(\vec{x}^*)$. □

4 The Completeness Theorem for Pseudo-Linear Functions

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

Theorem 5 *Let $f_1, f_2, ..., f_k$ be k pseudo-linear functions in n variables X, over a field 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.*

Proof: We show that if f is not a pseudo-linear function of F, which by Lemma (4) means that it does not satisfy at least one of F-interpolatable properties (i) or (ii), then f is not a function of F.

Since $f(\vec{x})$ is a pseudo-linear polynomial 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. $f_1(\vec{x}), f_2(\vec{x})$,...., $f_k(\vec{x})$ is pseudo-linear, it can also be represented by $j(f_s, \cdot)$ ($s \in [1..k]$). Further, each linear combination of F is represented similarly.

So, first consider the case where f does not satisfy (i). In other words, for some $i \in [1..t]$, for no linear combination ϕ of F (including zero) is $j(f, i)$ equal to $j(\phi, i)$. Thus, by Lemma (2), $p_i^{j(f,i)}$ is linearly independent of all $p_i^{j(f_s,i)}$ ($s \in [1..k]$). Let $J \subseteq \mathcal{L}$ correspond to the guard g_i. Thus, $p_i^{j(f,i)}$ and all $p_i^{j(f_s,i)}$ (for $s \in [1..k]$) are linearly independent of J. Let r be the rank of J, and $k' \leq k$ be the rank of $p_i^{j(f_s,i)}$ collectively. Since $p_i^{j(f,i)}$ is linearly independent of all $p_i^{j(f_s,i)}$, we have that $r + k' + 1 \leq n$. Now, the subspace corresponding to J set to zero has dimension $n - r$, and hence has q^{n-r} points. However, we are interested in points where all expressions in $\mathcal{L}\backslash J$ evaluate to non-zero values, which would guarantee that $g_i = 1$, and all other guards are zero. Recall the

subspace $\mathcal{P}_i(X)$ generated by all variables not in the set R corresponding to guard g_i. Now, $\mathcal{L}\backslash J$ is a union of cosets of $(n-r)$ dimensional space $\mathcal{P}_i(X)$ shifted by subspace J. Consider a basis $\mathcal{B}$ for $\mathcal{P}_i(X)$, comprising of $p_i^{j(f,i)}$, a k'-ranked basis of $p_i^{j(f_s,i)}$, and $n-r-1-k'$ other linearly independent expressions $\mathcal{B}'$.

Assume the field $\mathbb{F}_q$ is of size at least 2^{n+1}, and hence has $n+1$ linearly independent (over $\mathbb{F}_2$) elements e_i. Thus, for every injective map setting $\mathcal{B}$ to these e_i, there is a distinct solution to J being zero, and all of $\mathcal{L}\backslash J$ evaluating to non-zero values. Thus, there are at least $\binom{n+1}{n-r}(n-r)!$ such points in $(\mathbb{F}_q)^n$.

So, we fix $p_i^{j(f_s,i)}$ to e_s ($s \in [1..k']$; assume w.l.o.g. that the first k' formed the basis), and similarly fix the $\mathcal{B}'$ expressions to $e_{k'+1}$ to e_{n-r-1}. This still leaves at least $(n+1-(n-r-1))$ choices for $p_i^{j(f,i)}$. Thus, we have the situation that there are two points in $(\mathbb{F}_q)^n$ where f evaluates to different values, whereas F has the same value, and hence f cannot be a function of F.

Now, consider the case where f does satisfy condition (i), but condition (ii) is violated. In other words, for each $i \in [1..t]$, $p_i^{j(f,i)}$ is same as some $p_i^{j(\phi_\star,i)}$, but there exist a and b in $[1..t]$ which are F-equivalent, but the $\phi_\star$'s linear representation coefficients c_s differ for a and b. Again, we will demonstrate two points where F evaluate to the same value, but f evaluates to different values.

Again, let's assume that the underlying field is large enough to have at least k' linearly independent (over $\mathbb{F}_2$) elements, say e_i.

Now we have two sets, J_a corresponding to guard g_a, and J_b, corresponding to guard g_b. However, there is an easy solution for setting J_a to zero, and setting $p_a^{j(f_s,a)}$ ($s \in [1..k']$) to e_s. Similarly, there is a solution for setting J_b to zero and setting $p_a^{j(f_s,a)}$ ($s \in [1..k']$) to e_s. Thus, in both cases all f_s ($s \in [1..k]$) evaluate to the same value, but f being a different linear combination in the two cases, evaluates to different values. □

5 Iterated Composition of Pseudo-Linear Functions

In this section, we consider pseudo-linear functions which can take arguments, modeling oracles which are pseudo-linear functions of secret values and arguments. Thus, for instance it may be required to find if there exists a simulator which given access to functionalities which are pseudo-linear functions of secret parameters X and arguments supplied by simulator/adversary, can compute a given a pseudo-linear 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 $f_i(\vec{x}, y)$, each pseudo-linear in $\vec{x}$ and y.

So, given a collection of k pseudo-linear 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 σ 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 σ being a function ϕ_s from F, and the second component an arbitrary function γ_s of $s-1$ arguments (over $\mathbb{F}_q$).

Given an iterated composition σ of F, one can associate a function f^σ of X with it as follows

by induction. For σ of length one, f^σ is just $\phi_1(\vec{x}, \gamma_1())$, recalling that $\phi_1 \in F$. For σ of length t,

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

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

Since, 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 function of the n variables. The collection of all functions that can be obtained by iterated composition of F will be referred to as **terms(F)**. If we restrict γ_s to be pseudo-linear functions of their $s-1$ arguments, we will refer to the iterated composition as **pseudo-linear iterated composition of** F, and the corresponding collection of functions associated with such sequences as pseudo-linear iterated terms or **pl-terms(F)**. Note that in this case γ_1 is just zero.

Note that an arbitrary program can only compute a function of the terms, whereas an arbitrary pseudo-linear program can only compute a pseudo-linear function of the pseudo-linear terms. We would *like to show* that if a function f of terms(F) is a pseudo-linear function of X, then it is a pseudo-linear function of pl-terms(F). However, as we demonstrate in Appendix H.2, this is not true in general, and a slight extension is required to the pl-terms, so as to enable probabilistic functions. An iterated composition will be called an **extended pseudo-linear iterated composition of** F if γ_s is either a pseudo-linear function or a constant function $c(\vec{x})$ evaluating to an element c in $\mathbb{F}_q$. For each such c, the corresponding collection of functions associated with such sequences will be called **extended-pl-terms(F, c)**.

We will also need to refine the definition of terms(F), by restricting to terms obtained within some T iterated compositions, for some positive integer T. Thus, $\mathbf{terms}_T(F)$ 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.

Theorem 6 (Main) *Let $f_1, f_2, ..., f_k$ be k pseudo-linear functions in n variables X and an additional variable y, over a field $\mathbb{F}_q$ such that $q > 2^{2n}$. Collectively, we will refer to these polynomials as $F(X, y)$. Let T be a positive integer less than $2^n (< \sqrt{q})$. Let f be another pseudo-linear function in X. Then, if f is a function of* $\mathbf{terms}_T(F(X, y))$*, then f can be defined as a pseudo-linear probabilistic function of* **extended-pl-terms**$(F(X, y), Seed)$*, where the probability is over Seed chosen uniformly from $\mathbb{F}_q$, and for each $\vec{x}$ the probability of this definition of f being correct is at least $1 - 1/\sqrt{q}$.*

In Appendices C, D, E, and F we extend these results to cover *randomized* pseudo-linear functions and consider *randomized* probabilistic simulators. These results lead us to apply our theorems to the Universally Composable framework as we will describe in the next section.

6 Proof automation in the Universally Composable model

The Universally Composable (UC) framework is a formal system for proving security of computational systems such as cryptographic protocols. The framework describes two probabilistic games: The *real world* that captures the protocol flows and the capabilities of an attacker, and the *ideal world* that captures what we think of as a secure system. The notion of security asserts that these

two worlds are essentially equivalent. A summary of this framework and cryptographic motivation for this paper with an example is given in Appendix I.

Formally, a proof of security in the UC model boils down to the following: as input, we are given two sets of algorithms:

1. Ideal Functionality: Set of algorithms $F = \{F_1, F_2, \cdots\}$
2. Real Protocol: 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 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 algorithm Z, the execution of A with calls to P is indistinguishable from the execution of S with calls to F.

We describe a language $L^{\$,\oplus,\mathsf{if}}$ in Table 1 for which we are able to develop a decision procedure to decide realizability of a given ideal functionality by a given real protocol.

(expressions)	AE	::=	$x_1 \mid x_2 \mid \cdots$	variables
	XE	::=	$AE \mid AE \oplus XE$	bitwise xor expression
	BE	::=	`true` $\mid (XE == XE) \mid BE \wedge BE \mid \neg BE$	boolean expression
(assignments)	a	::=	$x \leftarrow \$$	assign new random number
			$x := XE$	assign xor expression
(program)	π	::=	a;	single action
			πa;	sequence of actions
			`if` BE `then` π `else` π	conditional

Table 1: Programs in the Language $L^{\$,\oplus,\mathsf{if}}$.

Definition 7 ($L^{\$,\oplus,\mathsf{if}}$) *An Ideal Functionality F and a real protocol P are described in the language $L^{\$,\oplus,\mathsf{if}}$ if*

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

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

The **semantics** of this language is that $\vec{x}$ is a set of inputs passed by the environment at the outset of execution and $\vec{y}$ 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. While fairly constrained, we provide a cryptographic example and motivation for this language in Appendix I.

Theorem 8 (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.*

References

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, August 2000.

[BM93] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security*, pages 244–250, 1993.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CG10] Ran Canetti and Sebastian Gajek. Universally composable symbolic analysis of diffie-hellman based key exchange. Cryptology ePrint Archive, Report 2010/303, 2010. `http://eprint.iacr.org/`.

[CH06] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key-exchange). In *TCC*, 2006. Extended version at `http://eprint.iacr.org/2004/334`.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, 2005.

[DDMR07] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (pcl). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.

[Dij75] Edsger Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453–457, Winter 1975.

[JPRZ04] C. Jutla, A. Patthak, A. Rudra, and D. Zuckerman. Testing low-degree polynomials over prime fields. In *FOCS*, 2004.

[KLP68] T. Kasami, S. Lin, and W. W. Peterson. New Generalization of the Reed-Muller Codes Part I: Primitive Codes. *IEEE Transactions on Information Theory*, IT-14(2):189–199, March 1968.

[KR04] T. Kaufman and D. Ron. Testing polynomials over general fields. In *FOCS*, 2004.

[MW04] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

A Iterated Composition - Proof

Similar to Section 3, we first state an interpolatable property which is a sufficient condition for a pseudo-linear function of X to be a pseudo-linear function of pl-terms(F).

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 pseudo-linear 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 pseudo-linear functions in X and y, though for different y, they would evaluate to the same value. In other words, for an arbitrary guard $g_a(\vec{x})$, 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 pseudo-linear 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 $l(\vec{x}) \in \mathcal{P}_a(X)$ (including zero), *exactly one* of these (super-)guards has the property that $J'_{y|l(\vec{x})} = J$ (Note the subscript $y|l(\vec{x})$ means $l(\vec{x})$ is substituted for every occurrence of y in J'). This particular J' is given by

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

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

Now we define the pseudo-linear function which is the composition of f_s and h, i.e. $f_s \circ h$, where f_s is a pseudo-linear function in X and y, and h is a pseudo-linear function in X, by defining its components in the basis for pseudo-linear functions. For any guard $g_i(\vec{x})$ (of functions in X), let $g_I(\vec{x}, y)$ 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

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

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, $(f_s \circ h)(\vec{x}) = f_s(\vec{x}, h(\vec{x}))$.

Define **Compose**$(F(X, y), H(X))$, where $F(X, y)$ are a set of pseudo-linear functions in X, y and $H(X)$ is a set of pseudo-linear 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 pseudo-linear function f_s of X and y, we also need to define a pseudo-linear function (in X) called **degenerate(f_s)**, which for each guard $g_a(\vec{x})$, defines the corresponding p function using its degenerate super-guard. Thus,

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

where I is the degenerate super-guard of g_a.

Now, we are ready to define the iterated pseudo-linear 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 F_2), the above iteration reaches a fix-point at an i bounded by a function only of n. We will denote the fix-point by just $\mathcal{I}(F)$.

Now, we generalize the definitions of F-equivalence and F-interpolatable from Section 3. Two guards $g_a(\vec{x})$ and $g_b(\vec{x})$ are said to be F^*-equivalent if for every $\phi(\vec{x})$ in $\mathcal{I}(F)$, it is the case that $j(\phi, a) = 0$ iff $j(\phi, b) = 0$.

The definition of F^*-interpolatable property is same as the F-interpolatable property except that $\mathcal{L}(F)$ is replaced by $\mathcal{I}(F)$.

Instead of the closure $\mathcal{I}(F)$, it will also be useful to define the following set of functions

$$\mathcal{C} = \bigcup_i \text{Compose}(F, \mathcal{I}^i(F)) \cup \text{Compose}(F, \text{degenerate}(F)),$$

and it is easy to see that $\mathcal{I}(F)$ is just the linear closure of $\mathcal{C}$.

Lemma 9 *If f is a pseudo-linear function of n variables X over a field $\mathbb{F}_q$, and f satisfies the F^*-interpolatable property, for some set F of pseudo-linear polynomials in X, y, then f can be defined as a pseudo-linear probabilistic function of* **extended-pl-terms**$(F(X, y), Seed)$*, where the probability is over Seed chosen uniformly from $\mathbb{F}_q$, and for each $\vec{x}$ the probability of this definition of f being correct is at least $1 - 2^n/q$.*

Proof: The proof is similar to the proof of Lemma 4, but there is a small difference due to the probabilistic nature of this lemma.

Consider $\hat{T} = [1..t] \,/\, \cong_F$, where t is the number of guards, i.e. $|\mathcal{G}(X)|$. We pick the smallest element from $[1..t]$ to represent each equivalence class in $\hat{T}$. Define a function $h(\vec{x})$ to be the following:

$$h(\vec{x}) = \sum_{u \in \hat{T}} \prod_{\phi \in \mathcal{L}(F): j(\phi,u) \neq 0} \phi(\vec{x})^{q-1} \;\cdot \prod_{\phi \in \mathcal{L}(F): j(\phi,u) = 0} (1 + \phi(\vec{x})^{q-1}) \;\cdot\; \phi_u(\vec{x}) \tag{2}$$

where for each u, ϕ_u is some function $\phi_\star \in \mathcal{C}$ satisfying the F^*-interpolatable property (i).

Now by definition, $h(\vec{x})$ is pseudo-linear in $\mathcal{C}$. Now, the proof that $h{=}f$, i.e. for all $\vec{x} \in (\mathbb{F}_q)^n$, $h(\vec{x}) = f(\vec{x})$, is identical to the proof in Lemma 4. However, h is pseudo-linear only on $\mathcal{C}$, whereas we need to show a function pseudo-linear in **extended-pl-terms**$(F(X, y), \text{Seed})$.

Observe that for any f_s, exactly one of the following cases hold, for all y linearly independent of $\vec{x}$:

Case 1: $f_s(\vec{x}, y) = \text{degenerate}(f_s)(\vec{x})$

Case 2: $f_s(\vec{x}, y) = \text{degenerate}(f_s)(\vec{x}) + y$

Now define a function $\hat{h}(\vec{x}, c)$ to be same as h, except every occurrence of degenerate$(f_s)(\vec{x})$ is replaced by either of the following:

$$\begin{cases} f_s(\vec{x}, c) & \text{in Case 1} \\ f_s(\vec{x}, c) + c & \text{in Case 2} \end{cases}$$

(recall, f_s is a function of X and y, whereas degenerate(f_s) is a function of only X). Then, it is easy to see that $\hat{h}(\vec{x}, c)$ is in **extended-pl-terms**(F, c). We next show that for every $\vec{x}$, with c chosen uniformly from $\mathbb{F}_q$, probability that $\hat{h}(\vec{x}, c) = h(\vec{x})$ is at least $1 - 2^n/q$. For each $\vec{x}$, one and only one guard g_s is satisfied. the probability that for this guard, $c = l(\vec{x})$, for some $l(\vec{x}) \in \mathcal{P}_s(X)$ is at most $1/q$. Hence, by union bound, over all possible $l(\vec{x})$, the probability that c equals any $l(\vec{x})$ is at most $2^n/q$, as $|X| = n$. These are the only cases in which degenerate$(f_s)(\vec{x})$ may differ from $f_s(\vec{x}, c)$ or $f_s(\vec{x}, c) + c$, as the case may be. □

We now restate Theorem 6 below and prove it.

Theorem 10 (Main) *Let $f_1, f_2, ..., f_k$ be k pseudo-linear functions in n variables X and an additional variable y, over a field $\mathbb{F}_q$ such that $q > 2^{2n}$. Collectively, we will refer to these polynomials as $F(X, y)$. Let T be a positive integer less than $2^n (< \sqrt{q})$. Let f be another pseudo-linear function in X. Then, if f is a function of* **terms**$_T(F(X, y))$*, then f can be defined as a pseudo-linear probabilistic function of* **extended-pl-terms**$(F(X, y), Seed)$*, where the probability is over Seed chosen uniformly from $\mathbb{F}_q$, and for each $\vec{x}$ the probability of this definition of f being correct is at least $1 - 1/\sqrt{q}$.*

Proof:

For the sake of leading to a contradiction, suppose that f is not a pseudo-linear probabilistic function of **extended-pl-terms**$(F(X, y), Seed)$. Then by Lemma 9, f does not satisfy the F^*-interpolatable property. Hence, as in Theorem 5, we have two cases. Before we go into the analysis of the two cases, we recall a few relevant definitions, and state some useful properties.

Recall from Section 2, that $\mathcal{Q}(X)$ is the set of all basic pseudo-linear polynomials in variables X, and, $\mathcal{G}(X)$ is the set of all guards amongst these polynomials $\mathcal{Q}(X)$. Further, $|\mathcal{G}(X)| = t$. Also, recall for each guard g_s its corresponding set R from its REPSELIN representation, and the corresponding subspace $\mathcal{P}_s(X)$.

Also, recall the super-guards $g_I(\vec{x}, y)$ corresponding to guards $g_i(\vec{x})$. Thus, if J corresponded to g_i, then some J' such that $J \subseteq J' \subseteq \mathcal{L}(X, y)$, corresponds to super-guard g_I. Further, $(\mathcal{L}(X) \backslash J) \subseteq (\mathcal{L}(X, y) \backslash J')$. Hence, if some $y + l(\vec{x})$ is in J', we can w.l.o.g take as the corresponding R' (of g_I) to be $R \cup \{y\}$. Thus, in such cases $p_I(\vec{x}, y)$ are just linear expressions in $X \backslash R$. If on the other hand, for no $l(\vec{x})$ it is the case that $y + l(\vec{x})$ is in J', then $R' = R$, and $p_I(\vec{x}, y)$ will be a linear expression in $(X \backslash R) \cup \{y\}$.

Now we are ready to analyze the two cases.

Case 1: First, consider the case where f does not satisfy property (1) of F^*-interpolatable.

Then, it is the case that there exists an $s \in [1..t]$, such that for every linear polynomial ϕ in $\mathcal{I}(F)$, $j(f, s) \neq j(\phi, s)$. Thus, by Lemma 2, $p_s^{j(f,s)}$ is linearly independent of all $p_s^{j(\phi,s)}$, with $\phi \in \mathcal{C}$.

Let $J \subseteq \mathcal{L}$ correspond to the guard g_s. Thus, $p_s^{j(f,s)}$, as well as all $p_s^{j(\phi,s)}$ ($\phi \in \mathcal{C}$) are linearly independent of J (see the paragraph after definition of REPSELIN polynomials). Let r be the rank

of J, and k' be the rank of $p_s^{j(\phi,s)}$ collectively. Thus, $r + k' + 1 \leq n$. Consider a basis $\mathcal{B}$ of $\mathcal{P}_s(X)$ consisting of $p_s^{j(f,s)}$, a basis $\mathcal{B}''$ of $p_s^{j(\phi,s)}$, and another linearly independent set $\mathcal{B}'$ of expressions in $X \backslash R$ (of rank $n - r - k' - 1$).

Our aim is to demonstrate two different settings of X to values in $\mathbb{F}_q$, such that $f(\vec{x})$ has different values, while all of the $\mathbf{terms}_T(F(X, y))$ have the same value at the two settings. Now, fix a particular length T iterated composition σ of F. We will now show that each of $\gamma_t(f^{\sigma|1}(\vec{x}), f^{\sigma|2}(\vec{x}), ..., f^{\sigma|t-1}(\vec{x}))$, $t \in [1..T]$, as well as $f^{\sigma|t}(\vec{x})$ is a function of only $\mathcal{B}''$, and is independent of $p_s^{j(f,s)}$, and also independent of $\mathcal{B}'$ defined above. Thus in choosing the two different settings for X, we can first set the basis $\mathcal{B}''$ to some value, which will fix the $\gamma(...)$ values, and then we can set the $\mathcal{B}'$ and $p_s^{j(f,s)}$ to two different values, while assuring that all consistency requirements are met.

For the base case, $\gamma_1()$ is clearly not a function of $p_s^{j(f,s)}$, or $\mathcal{B}'$. Now, for the induction step, consider $f^{\sigma|t-1}(\vec{x})$. which is given by

$$\phi_{t-1}(\vec{x}, \gamma_{t-1}(f^{\sigma|1}(\vec{x}), f^{\sigma|2}(\vec{x}), ..., f^{\sigma|t-1}(\vec{x}))).$$

where ϕ_{t-1} is in F. Now, by induction the $\gamma_{t-1}(...)$ expression is not a function of $p_s^{j(f,s)}$ or $\mathcal{B}'$.

Now, it is possible that $\gamma_{t-1}(...)$ is equal to some $p_s^{j(\phi^*,s)}(\vec{x})$ ($\phi^* \in \mathcal{C}$), in which case $\phi_{t-1}(\vec{x}, \gamma_{t-1}(...))$ would just equal $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$, for I corresponding to the unique super-guard $g_I(\vec{x}, y)$ which is consistent with $y + p_s^{j(\phi^*,s)}(\vec{x}) = 0$. But, $p_I^{j(\phi_{t-1},I)}$ is either $p_s^{j(\phi^{**},s)}(\vec{x})$ ($\phi^{**} \in \mathcal{C}$) or such an expression plus y, by definition of $\mathcal{C}$ and definition of $p_I^{j(f_s \circ h,I)}(\vec{x}, y)$. In either case, it is a function of only $p_s^{j(\phi,s)}(\vec{x})$ ($\phi \in \mathcal{C}$) by induction.

If $\gamma_{t-1}(...)$ is not equal to any $p_s^{j(\phi^*,s)}(\vec{x})$ ($\phi^* \in \mathcal{C}$), we will show that we can choose $\vec{x}$ so as to assure that $\gamma_{t-1}(...)$ is not equal to any linear expression in $\mathcal{B}'$ (and $p_s^{j(f,s)}$) either, as $t < T < 2^n < \sqrt{q}$. In this case $\phi_{t-1}(\vec{x}, \gamma_{t-1}(...))$ returns $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$, where I corresponds to the degenerate super-guard of g_s given by $J' = J$. However, such $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$ is again either $p_s^{j(\phi^{**},s)}(\vec{x})$ ($\phi^{**} \in \mathcal{C}$) or such an expression plus y, since $\mathcal{C}$ includes Compose(F, degenerate(F)).

Now, we demonstrate the two different settings of X to values in $\mathbb{F}_q$. We first choose k' linearly independent (over $\mathbb{F}_2$) values in $\mathbb{F}_q$ and set the basis $\mathcal{B}''$ to these values, so that all expressions $p_s^{j(\phi,s)}$ ($\phi \in \mathcal{C}$) are non-zero. As explained above, the values $\gamma_t(...)$ are then fixed, and let this set of values along with zero be collectively called Γ. Next, we inductively assign values to the basis $\mathcal{B}'$(of size $n - r - k' - 1$) as follows. Let this basis be given by $l_1(\vec{x})$, ..., $l_{n-r-k'-1}(\vec{x})$. For $l_1(\vec{x})$, we pick any value in $\mathbb{F}_q$ which is not equal to any value in $\mathcal{L}(\mathcal{B}'') + \Gamma$, where the sum of two sets is defined naturally. For, the induction step, we choose for $l_i(\vec{x})$ a value in $\mathbb{F}_q$ which is not equal to any value in $\mathcal{L}(\mathcal{B}'' \cup \{l_1(\vec{x}), ..., l_{i-1}(\vec{x})\}) + \Gamma$.

Since $p_s^j(f, s)(\vec{x})$ is linearly independent of $\mathcal{B}''$ (as well as $\mathcal{B}'$), we choose a value for it which is not equal to any value in $\mathcal{L}(\mathcal{B}'' \cup \{l_1(\vec{x}), ..., l_{n-r-k'-1}(\vec{x})\}) + \Gamma$. Further we have at least two choices for it, given that $\mathbb{F}_q \geq 2 + 2^{n-1-r} \times (T + 1)$. This also proves our claim that Γ is never equal to any linear expression in $\mathcal{B}' \cup \{p_s^{j(f,s)}\}$. Further no linear combination of $\mathcal{B}'$ and $\mathcal{B}''$ will be zero. Then, we can choose the R variables corresponding to the guard g_s, which by definition of R are given in terms of the variables already chosen, so that the guard g_s is true in both cases.

Case 2: Now consider the case where condition (i) holds, but condition (ii) of the F^*-interpolatable property fails to hold for f. In other words, for each $i \in [1..t]$, $p_i^{j(f,i)}$ is same as some $p_i^{j(\phi_*,j)}$ ((for

$\phi_* \in \mathcal{C}$), but there exist a and b in $[1..t]$ which are F^*-equivalent, but the ϕ_*'s linear representation coefficients c_s differ for a and b. Again, we will demonstrate two points where $\text{terms}_T(F(X,y))$ evaluate to the same value, but f evaluates to different values.

We have two sets, J_a corresponding to guard g_a, and J_b, corresponding to guard g_b. Let $k' \leq n$ be the rank of $p_a^{j(\phi,a)}$ ($\phi \in \mathcal{C}$). Let r_a be the rank of J_a, and r_b be the rank of J_b. thus, $r_a + k' \leq n$, and $r_b + k' \leq n$. Let the R sets corresponding to guards g_a and g_b be called R_a and R_b respectively. Let $\mathcal{B}'_a$ be a basis for $X \backslash R_a$ excluding $\mathcal{L}(\mathcal{B}'')$, and similarly $\mathcal{B}'_b$ be a basis for $X \backslash R_b$ excluding $\mathcal{L}(\mathcal{B}'')$. We set the basis $\mathcal{B}''$ of $p_a^{j(\phi,a)}$ to linearly independent over GF2 values e_1 to $e_{k'}$. We set the basis of $p_b^{j(\phi,b)}$ also to the same values, recalling that the two bases, one for a and the other for b, can be chosen to have the same indices. Thus, all functions in $\mathcal{C}$ will have the same value when guards g_a or g_b are true. As in case 1, it follows that we can assure that by choosing $\mathcal{B}'_a$ and $\mathcal{B}'_b$ appropriately, each of $\gamma_t(f^{\sigma|1}(\vec{x}), f^{\sigma|2}(\vec{x}), ..., f^{\sigma|t-1}(\vec{x}))$, $t \in [1..T]$, as well as $f^{\sigma|t}(\vec{x})$ is only a function of $\mathcal{B}''$, and hence have the same values when guards g_a or g_b are true. However, since f has different linear combinations of $\mathcal{B}''$ at these two guards, it evaluates to different values. Further values for variables in R_a and R_b can be chosen so that guards g_a and g_b are indeed true. □

B Allowing a Few Constants

Let E be a set of linearly independent (over $\mathbb{F}_2$) elements of a field $\mathbb{F}_q$. Now, we *redefine* pseudo-linear polynomials where each linear term is as before defined over $\mathbb{F}_2$, but can in addition have an addend from E. The same rule also applies to all the linear terms in the guards. Then, we can prove the following theorem.

Theorem 11 *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 $q > 2^{n+|E|}$. 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.*

Proof is similar to that of Theorem 5, in that we treat E as formal independent variables, and then in the proof of Theorem 5, we set these formal variables to E where we set the k' basis elements of $p_i^{j(f_s,i)}$ to e_s.

A similar version holds for the iterated composition Theorem 6.

C 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 $\vec{x}$ and $\vec{r}$, where the variables $\vec{r}$ parametrize the family. Given such a f', a randomized pseudo-linear function f (in $\vec{x}$) is given by choosing $\vec{r}$ 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 $\vec{r}$ as formal variables, we can apply Lemma 1 to deduce that f' is expressible in terms of the basic pseudo-linear polynomials in $(\vec{x}, \vec{r})$. In particular,

$$f'(\vec{x}, \vec{r}) = \sum_{i \in T_{n+m}} g_i(\vec{x}, \vec{r}) \cdot p_i^{j(f',i)}(\vec{x}, \vec{r})$$

Consider a guard g_i in just the space of the input variables $\vec{x}$, with associated set J, i.e. $g_i = \prod_{l \in \mathcal{L} \setminus J} l(\vec{x})^{q-1} \cdot \prod_{l \in J}(1 + l(\vec{x})^{q-1})$. Consider the set of super-guards $\mathbb{I}_i$ which extend J to $\mathcal{L} \cup \mathcal{L}(\vec{r})$ and each super-guard $I \in \mathbb{I}_i$ corresponds to a different subspace $J_r \subseteq \mathcal{L}(\vec{r})$ 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 $f(\vec{x}, \vec{r})$, to signify the random variables over which the distribution is defined.

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

- The function $\hat{f}$ will only have non-zero p for guards involving only $\vec{x}$, i.e. p for guards involving $\hat{r}$ will be 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 $p_{I_0}^{j(f,I_0)}(\vec{x}, \vec{r}) = l_1(\vec{x}) + l_2(\vec{r})$. If l_2 is not identically 0, then set $p_i^{j(\hat{f},i)}(\vec{x}) = \hat{r}$, otherwise set $p_i^{j(\hat{f},i)}(\vec{x}) = l_1(\vec{x})$.

Lemma 12 *Let $\log q > 2(\rho + m)$, where ρ is the number of random variables and m is the number of input variables in f. The distribution $f(\vec{x})$ is statistically indistinguishable from $\hat{f}(\vec{x})$ with advantage $< 1/\sqrt{q}$.*

Proof: Since there are at most $2^{\rho+m}$ different linear expressions in $\vec{x}$ and $\vec{r}$, by union bound, with probability at least $1 - 2^{\rho+m}/q > 1 - 1/\sqrt{q}$, all linear expressions $l_1(\vec{x}) + l_2(\vec{r})$ are non-zero, when l_2 is not identically 0. In the rest of this proof, we therefore restrict ourselves to only on this overwhelming case.

Now consider any guard g_i in just $\vec{x}$. For any super-guard index $I \in \mathbb{I}_i - I_0$, at least one component $(1 + l(\vec{x}, \vec{r})^{q-1})$ in the super-guard will evaluate to 0, where $l(\vec{x}, \vec{r}) \in J$. Hence the guards corresponding to $(\mathbb{I}_i - I_0)$ all evaluate to 0.

For the index I_0, no component of J_{I_0} is identically zero since it is composed of $\vec{x}$ only. The components $(l_1(\vec{x})+l_2(\vec{r}))^{q-1}$, where l_2 is not identically zero, evaluate to 1, since these are non-zero by the earlier restriction. Hence they can be dropped off, and we are just left with the guard g_i over just the $\vec{x}$'s. Now, suppose $p_{I_0}^{j(f,I_0)}(\vec{x},\vec{r}) = l_1(\vec{x}) + l_2(\vec{r})$. If l_2 is identically zero, then $p_{I_0}^{j(f',I_0)}(\vec{x},\vec{r})$ is just a function of $\vec{x}$. If l_2 is not identically 0, then $p_{I_0}^{j(f',I_0)}(\vec{x},\vec{r})$ is just a uniformly distributed random number. We can set $p_i^{j(\hat{f},i)}(\vec{x}) = \hat{r}$. These random numbers $\hat{r}$ do not have to be different for every guard g_i, since for any fixed $\vec{x}$, exactly one of the guards is going to be equal to 1 and the rest of them will be 0. □

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 $\vec{x}$ only and just one random variable. Lemma 12 indicates that we can just focus on SRPL functions since any randomized pseudo-linear function is statistically close to an SRPL function.

Lemma 13 (Homomorphism) *For any SRPL functions $\phi_1(\vec{x})$ and $\phi_2(\vec{x})$, 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 $\vec{x}$, 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.

□

D Interpolation Property for SRPL Functions

Let $f_1, f_2, ..., f_k$ be k SRPL 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 SRPL function $f(\vec{x})$ in X, let its representation in terms of the basis be given by $j(f,\cdot)$. Note that there is also one special index j for the random variable r as well. Let this index be denoted by $j(f,\cdot) = \$$ uniformly.

Since each of the polynomials from F, i.e. $f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x})$ is SRPL, let it be represented by $j(f_s,\cdot)$ ($s \in [1..k]$). Further, each linear combination of F is represented similarly, with the rule that $r + \cdot$ is replaced by r in the sum (see lemma 13).

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

Lemma 14 *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 $\sum_{s\in S} p_b^{j(f_s,b)}$; also if for some subset $S \subseteq [1..k]$, the linear combination $\sum_{s\in S} p_a^{j(f_s,a)}$ is random, then so is $\sum_{s\in S} p_b^{j(f_s,b)}$.*

The Lemma follows by Lemma (13). 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 function $f(\vec{x})$, and any set F of pseudo-linear functions in X, we say that $f(\vec{x})$ has the **F-interpolatable** property if it satisfies the following two conditions:

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

(ii) For every $a, b \in [1..t]$ such that a and b are F-equivalent,
either $j(f,a) = j(f,b) = \$$, in which case we set $j(\phi_\star, i) = \$$;
or, the following holds: w.l.o.g. by Lemma (14), 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 15 *Let $X \in (\mathbb{F}_q)^n$. If f is an SRPL function in X, and f satisfies the F-interpolatable property, for some set F of SRPL functions in X, 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 $S^F(X)$.*

Proof: Indeed, consider $\hat{T} = [1..t] \,/\, \cong_F$, where t is the number of guards, i.e. $|\mathcal{G}(X)|$. We pick the smallest elements from $[1..t]$ to represent each equivalence class in $\hat{T}$. Define a random variable $h(\vec{x})$ to be the following:

$$h(\vec{x}) \stackrel{\text{R}}{\leftarrow} S^F(\vec{x}) \tag{3}$$

where for each u, ϕ_u is some function $\phi_\star$ satisfying the F-interpolatable property above and algorithm S^F as follows:

Algorithm $S^F(\vec{x})$
`for all` $u \in [1 \ldots t]/\cong_F$
  $guardfound := true$
  `for all` $\phi \in \mathcal{L}(F)$
    $\gamma_0 \stackrel{\text{R}}{\leftarrow} \phi(\vec{x});\ \gamma_1 \stackrel{\text{R}}{\leftarrow} \phi(\vec{x})$
    `if` $\gamma_0 \neq \gamma_1$ `then` $israndom := true$ `else` $israndom := false$
    `if` $\begin{pmatrix} & j(\phi,u) = \$ \text{ and } israndom := true \\ \text{or} & j(\phi,u) = 0 \text{ and } \gamma_0 = 0 \text{ and } israndom := false \\ \text{or} & j(\phi,u) \neq 0 \text{ and } \gamma_0 \neq 0 \text{ and } israndom := false \end{pmatrix}$
      `then continue for loop`
      `else` $guardfound := false$`; exit for loop`
  `if` $guardfound = true$ `then` $result \stackrel{\text{R}}{\leftarrow} \phi_u(\vec{x})$`; exit for loop`
    `else continue for loop`
`return` $result$

We now show that $h \approx f$, i.e. for all $\vec{x} \in (\mathbb{F}_q)^n$, $h(\vec{x}) \approx f(\vec{x})$. Fix any $\vec{x}^*$ in $(\mathbb{F}_q)^n$. Let $J \subseteq \mathcal{L}$, such that all linear functions in J evaluate to zero at $\vec{x}^*$, and all linear functions in $\mathcal{L} \backslash J$ evaluate to non-zero quantities at $\vec{x}^*$. Clearly, J is closed under addition, and hence J corresponds to a guard g_i. In other words, $g_i(\vec{x}^*) = 1$, and for all other $i' \in [1..t]$: $g_{i'}(\vec{x}^*) = 0$. Thus, $f(\vec{x}^*) = p_i^{j(f,i)}(\vec{x}^*)$, and similarly, for all $\phi \in \mathcal{L}(F)$, $\phi(\vec{x}^*) = p_i^{j(\phi,i)}(\vec{x}^*)$. By definition of i (i.e. g_i corresponding to J above, and hence $p_i^{j(\phi,i)} \in \mathcal{L} \backslash J$), it follows that $\phi(\vec{x}^*)$ is zero iff $j(\phi, i) = 0$, and $\phi(\vec{x}^*)$ is random iff $j(\phi, i) = \$$.

Now, in the algorithm S^F, we show that the only u for which the "guards" evaluate to be non-zero (i.e. one), is the one corresponding to the equivalence class of i in $\hat{T}$ (say, u_i). In fact, for i (and its F-equivalent u_i) the "guards" indeed evaluate to 1: the reasoning is same as in the proof of Lemma 4 for the non-random values; in the random case, when we call $\phi(\vec{x})$ twice, the responses should be different with probability $1 - 1/q$, hence $\gamma_0 \neq \gamma_1$ would evaluate to *true* - in any non-random case, $\gamma_0 \neq \gamma_1$ would evaluate to $false$ with probability one. For all other i', if the "guards" evaluate to one, then by definition of F-equivalence, those i' are F-equivalent to i.

Thus, $h(\vec{x}^*) \approx \phi_{u_i}(\vec{x}^*)$, and since ϕ_{u_i} is pseudo-linear in X,

$$\phi_{u_i}(\vec{x}^*) = p_i^{j(\phi_{u_i},i)}(\vec{x}^*) \approx \sum_s c_s^{u_i} p_i^{j(f_s,i)}(\vec{x}^*) = \sum_s c_s^i p_i^{j(f_s,i)}(\vec{x}^*) = p_i^{j(f,i)}(\vec{x}^*).$$

Thus, $h(\vec{x}^*) \approx p_i^{j(f,i)}(\vec{x}^*)$, which is same as $f(\vec{x}^*)$. □

E The Completeness Theorem for SRPL Functions

Theorem 16 *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, such that the distribution $f(X)$ is statistically indistinguishable from the distribution $S^F(X)$, then f is F-interpolatable.*

Proof: We show that if f does not satisfy at least one of F-interpolatable properties (i) or (ii), then f is not efficiently simulatable using F. Then from this theorem and Lemma 15, it will follow that deciding whether an efficient simulator exists is equivalent to checking F-interpolatability, which can be done in computational time independent of $\lg q$.

Since $f(\vec{x})$ is a pseudo-linear polynomial 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. $f_1(\vec{x}), f_2(\vec{x})$,...., $f_k(\vec{x})$ is pseudo-linear, it can also be represented by $j(f_s, \cdot)$ ($s \in [1..k]$). Further, each linear combination of F is represented similarly.

So, first consider the case where f does not satisfy (i). In other words, for some $i \in [1..t]$, $j(f, i)$ is neither \$, nor for any linear combination ϕ of F (including zero) is $j(f, i)$ equal to $j(\phi, i)$. Thus, by Lemma (2), $p_i^{j(f,i)}$ is linearly independent of all the non-random $p_i^{j(f_s,i)}$ (w.l.o.g let's say $s \in [1..\hat{k}]$). Let $J \subseteq \mathcal{L}$ correspond to the guard g_i. Thus, $p_i^{j(f,i)}$ and all $p_i^{j(f_s,i)}$ (for $s \in [1..\hat{k}]$) are linearly independent of J. Let r be the rank of J, and $k' \leq \hat{k}$ be the rank of $p_i^{j(f_s,i)}$ collectively. Since $p_i^{j(f,i)}$ is linearly independent of all $p_i^{j(f_s,i)}$, we have that $r + k' + 1 \leq n$. Now, the subspace corresponding to J set to zero has dimension $n - r$, and hence has q^{n-r} points. However, we are

interested in points where all expressions in $\mathcal{L}\backslash J$ evaluate to non-zero values, which would guarantee that $g_i = 1$, and all other guards are zero. Recall the subspace $\mathcal{P}_i(X)$ generated by all variables not in the set R corresponding to guard g_i. Now, $\mathcal{L}\backslash J$ is a union of cosets of $(n-r)$ dimensional space $\mathcal{P}_i(X)$ shifted by subspace J. Consider a basis $\mathcal{B}$ for $\mathcal{P}_i(X)$, comprising of $p_i^{j(f,i)}$, a k'-ranked basis of $p_i^{j(f_s,i)}$, and $n-r-1-k'$ other linearly independent expressions $\mathcal{B}'$.

Assume the field $\mathbb{F}_q$ is of size at least 2^{n+1}, and hence has $n+1$ linearly independent (over $\mathbb{F}_2$) elements e_i. Thus, for every injective map setting $\mathcal{B}$ to these e_i, there is a distinct solution to J being zero, and all of $\mathcal{L}\backslash J$ evaluating to non-zero values. Thus, there are at least $\binom{n+1}{n-r}(n-r)!$ such points in $(\mathbb{F}_q)^n$.

So, we fix $p_i^{j(f_s,i)}$ to e_s ($s \in [1..k']$; assume w.l.o.g. that the first k' formed the basis), and similarly fix the $\mathcal{B}'$ expressions to $e_{k'+1}$ to e_{n-r-1}. This still leaves at least $(n+1-(n-r-1))$ choices for $p_i^{j(f,i)}$. Thus, we have the situation that there are two points in $(\mathbb{F}_q)^n$ where f evaluates to different values, whereas F has the same value. When we extend this argument to distributions, we observe that the $\hat{k}$ functions which are non-random have a constant distribution (their value is fixed given just $\vec{x}$) and also f is a constant distribution. The functions in F other than these $\hat{k}$ functions are uniform distribution. Therefore, collectively the counter-example generated above are also *easily detectably different* distributions, and hence f cannot be a function of F.

Now, consider the case where f does satisfy condition (i), but condition (ii) is violated. In other words, for each $i \in [1..t]$, $p_i^{j(f,i)}$ is same as some $p_i^{j(\phi_\star,i)}$, but there exist a and b in $[1..t]$ which are F-equivalent, but one of the following two cases arise:

Case 1: $j(f,a) \neq \$$ and $j(f,b) \neq \$$, but $\phi_\star$'s linear representation coefficients c_s differ for a and b.

Again, we will demonstrate two points where F evaluate to the same value, but f evaluates to different values.

Again, let's assume that the underlying field is large enough to have at least k' linearly independent (over $\mathbb{F}_2$) elements, say e_i.

Now we have two sets, J_a corresponding to guard g_a, and J_b, corresponding to guard g_b. However, there is an easy solution for setting J_a to zero, and setting $p_a^{j(f_s,a)}$ ($s \in [1..k']$) to e_s. Similarly, there is a solution for setting J_b to zero and setting $p_a^{j(f_s,a)}$ ($s \in [1..k']$) to e_s. Thus, in both cases all f_s ($s \in [1..k]$) evaluate to the same value, but f being a different linear combination in the two cases, evaluates to different values.

Case 2: $j(f,a) = \$ \neq j(f,b)$.

In this case, we construct a counter-example in exactly the same manner as in Case 1. For guard g_a, we observe that f will follow a uniformly random distribution, whereas with guard g_b, f has a constant distribution. On the other hand the inputs have statistically indistinguishable distributions with the given counter-examples. Hence, f cannot be a function of F.

□

F 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 $f_i(\vec{x}, y)$, each SRPL in $\vec{x}$ 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 σ 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 σ being a function ϕ_s from F, and the second component an arbitrary *randomized* function γ_s of $s-1$ arguments (over $\mathbb{F}_q$).

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

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

where $\sigma_{|j}$ is the prefix of σ 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 **terms(F)**. If we restrict γ_s to be SRPL functions of their $s-1$ arguments, we will refer to the iterated composition as **SRPL iterated composition of** F, and the corresponding collection of functions associated with such sequences as SRPL iterated terms or **srpl-terms(F)**. Note that in this case γ_1 is just zero.

Note that an arbitrary randomized program can only compute a randomized function of the terms, whereas an arbitrary SRPL program can only compute an SRPL function of the SRPL terms. An iterated composition will be called an **extended SRPL iterated composition of** F if γ_s is either an SRPL function or a constant function $c(\vec{x})$ evaluating to an element c in $\mathbb{F}_q$. For each such c, the corresponding collection of functions associated with such sequences will be called **extended-srpl-terms(F, c)**.

We will also need to refine the definition of terms(F), by restricting to terms obtained within some T iterated compositions, for some positive integer T. Thus, $\mathbf{terms}_T(F)$ 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.

Similar to Appendix D, we first state an interpolatable property which is a sufficient condition for an SRPL function of X to be an SRPL function of srpl-terms(F).

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 $g_a(\vec{x})$, 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 $l(\vec{x}) \in \mathcal{P}_a(X)$ (including zero), *exactly one* of these (super-)guards has the property that $J'_{y|l(\vec{x})} = J$ (Note the subscript $y|l(\vec{x})$ means $l(\vec{x})$ is substituted for every occurrence of y in J'). This particular J' is given by

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

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

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 $g_i(\vec{x})$ (of functions in X), let $g_I(\vec{x}, y)$ 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

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

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, $(f_s \circ h)(\vec{x}) = f_s(\vec{x}, h(\vec{x}))$.

Define $\mathbf{Compose}(F(X, y), H(X))$, 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 $g_a(\vec{x})$, defines the corresponding p function using its degenerate super-guard. Thus,

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

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 F_2), the above iteration reaches a fix-point at an i bounded by a function only of n. We will denote the fix-point by just $\mathcal{I}(F)$.

Now, we generalize the definitions of F-equivalence and F-interpolatable from Section 3. Two guards $g_a(\vec{x})$ and $g_b(\vec{x})$ are said to be F^*-equivalent if for every $\phi(\vec{x})$ in $\mathcal{I}(F)$, it is the case that $j(\phi, a) = 0$ iff $j(\phi, b) = 0$ and $j(\phi, a) = \$$ iff $j(\phi, b) = \$$.

The definition of F^*-interpolatable property is same as the F-interpolatable property except that $\mathcal{L}(F)$ is replaced by $\mathcal{I}(F)$.

Instead of the closure $\mathcal{I}(F)$, it will also be useful to define the following set of functions

$$\mathcal{C} = \bigcup_i \text{Compose}(F, \mathcal{I}^i(F)) \cup \text{Compose}(F, \text{degenerate}(F)),$$

and it is easy to see that $\mathcal{I}(F)$ is just the linear closure of $\mathcal{C}$.

Lemma 17 *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 $2^n/q$.*

Proof: The proof is similar to the proof of Lemma 4, but there is a small difference due to the probabilistic nature of this lemma.

Consider $\hat{T} = [1..t] \,/\, \cong_F$, where t is the number of guards, i.e. $|\mathcal{G}(X)|$. We pick the smallest element from $[1..t]$ to represent each equivalence class in $\hat{T}$. Define a function $h(\vec{x})$ to be the following:

$$h(\vec{x}) \stackrel{\text{R}}{\leftarrow} S^F(\vec{x}) \tag{4}$$

where S^F is as defined below. For each u, ϕ_u is some function $\phi_\star \in \mathcal{I}(F)$ satisfying the F^*-interpolatable property (i).

Algorithm $S^F(\vec{x})$

```
for all u ∈ [1...t]/ ≅_F
  guardfound := true
  for all φ ∈ I(F)
    γ0 ←R φ(x); γ1 ←R φ(x)
    if γ0 ≠ γ1 then israndom := true else israndom := false
    if ( j(φ,u) = $ and israndom := true
       or j(φ,u) = 0 and γ0 = 0 and israndom := false
       or j(φ,u) ≠ 0 and γ0 ≠ 0 and israndom := false )
      then continue for loop
      else guardfound := false; exit for loop
  if guardfound = true then result ←R φ_u(x); exit for loop
    else continue for loop
return result
```

Now, the proof that $h{=}f$, i.e. for all $\vec{x} \in (\mathbb{F}_q)^n$, $h(\vec{x}) = f(\vec{x})$, is identical to the proof in Lemma 4.

For the efficiency argument, we now show a simulator with calls to **extended-pl-terms**$(F(X,y), \text{Seed})$. Observe that for any f_s, exactly one of the following cases hold, for all y linearly independent of $\vec{x}$:

Case 1: $f_s(\vec{x}, y) = \text{degenerate}(f_s)(\vec{x})$

Case 2: $f_s(\vec{x}, y) = \text{degenerate}(f_s)(\vec{x}) + y$

Now define a function $\hat{h}(\vec{x}, c)$ to be same as h, except every occurrence of degenerate$(f_s)(\vec{x})$ is replaced by either of the following:

$$\begin{cases} f_s(\vec{x}, c) & \text{in Case 1} \\ f_s(\vec{x}, c) + c & \text{in Case 2} \end{cases}$$

(recall, f_s is a function of X and y, whereas degenerate(f_s) is a function of only X). Then, it is easy to see that $\hat{h}(\vec{x}, c)$ is in **extended-pl-terms**(F, c). We next show that for every $\vec{x}$, with c chosen uniformly from $\mathbb{F}_q$, probability that $\hat{h}(\vec{x}, c) = h(\vec{x})$ is at least $1 - 2^n/q$. For each $\vec{x}$, one and only one guard g_s is satisfied. the probability that for this guard, $c = l(\vec{x})$, for some $l(\vec{x}) \in \mathcal{P}_s(X)$ is at most $1/q$. Hence, by union bound, over all possible $l(\vec{x})$, the probability that c equals any $l(\vec{x})$ is at most $2^n/q$, as $|X| = n$. These are the only cases in which degenerate$(f_s)(\vec{x})$ may differ from $f_s(\vec{x}, c)$ or $f_s(\vec{x}, c) + c$, as the case may be. □

Theorem 18 (Main) *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 $S^{\textbf{terms}_T(F(X,y))}$, such that the distribution $f(X)$ is statistically indistinguishable from the distribution $S^{\textbf{terms}_T(F(X,y))}(X)$, then f is F^*-interpolatable.*

Proof: We show that if f does not satisfy at least one of F^*-interpolatable properties (i) or (ii), then f is not efficiently simulatable using F. Then from this theorem and Lemma 17, it will follow that deciding whether an efficient simulator exists is equivalent to checking F^*-interpolatability, which can be done in computational time independent of $\lg q$.

As in Theorem 5, we have two cases. In both cases we will show the non-existence of a probabilistic poly-time simulator (i.e. with for each $\vec{x}$, the probability of the definition being correct must be more than, say a very liberal, $q^{1/4}$). Before we go into the analysis of the two cases, we recall a few relevant definitions, and state some useful properties.

Recall from Section 2, that $\mathcal{Q}(X)$ is the set of all basic SRPL polynomials in variables X, and, $\mathcal{G}(X)$ is the set of all guards amongst these polynomials $\mathcal{Q}(X)$. Further, $|\mathcal{G}(X)| = t$. Also, recall for each guard g_s its corresponding set R from its REPSELIN representation, and the corresponding subspace $\mathcal{P}_s(X)$.

Also, recall the super-guards $g_I(\vec{x}, y)$ corresponding to guards $g_i(\vec{x})$. Thus, if J corresponded to g_i, then some J' such that $J \subseteq J' \subseteq \mathcal{L}(X, y)$, corresponds to super-guard g_I. Further, $(\mathcal{L}(X) \backslash J) \subseteq (\mathcal{L}(X, y) \backslash J')$. Hence, if some $y + l(\vec{x})$ is in J', we can w.l.o.g take as the corresponding R' (of g_I) to be $R \cup \{y\}$. Thus, in such cases $p_I(\vec{x}, y)$ are just linear expressions in $X \backslash R$. If on the other hand,

for no $l(\vec{x})$ it is the case that $y + l(\vec{x})$ is in J', then $R' = R$, and $p_I(\vec{x}, y)$ will be a linear expression in $(X \backslash R) \cup \{y\}$.

Now we are ready to analyze the two cases.

Case 1: First, consider the case where f does not satisfy property (1) of F^*-interpolatable.

Then, it is the case that there exists an $s \in [1..t]$, such that for every linear polynomial ϕ in $\mathcal{I}(F)$, $j(f,s) \neq j(\phi,s)$, and further it is the case that $j(f,s)$ is not \$. Thus, by Lemma 2, $p_s^{j(f,s)}$ is linearly independent of all non-random $p_s^{j(\phi,s)}$, with $\phi \in \mathcal{C}$.

Let $J \subseteq \mathcal{L}$ correspond to the guard g_s. Thus, $p_s^{j(f,s)}$, as well as all $p_s^{j(\phi,s)}$ ($\phi \in \mathcal{C}$) are linearly independent of J (see the paragraph after definition of REPSELIN polynomials). Let r be the rank of J, and k' be the rank of $p_s^{j(\phi,s)}$ collectively. Thus, $r + k' + 1 \leq n$. Consider a basis $\mathcal{B}$ of $\mathcal{P}_s(X)$ consisting of $p_s^{j(f,s)}$, a basis $\mathcal{B}''$ of $p_s^{j(\phi,s)}$, and another linearly independent set $\mathcal{B}'$ of expressions in $X \backslash R$ (of rank $n - r - k' - 1$).

Note that since $j(f,s)$ is not random, it is a deterministic function of $\vec{x}$. Also, each term in $\mathbf{terms}_T(F(X,y))$ is an SRPL function, and hence is either a deterministic function of $\vec{x}$, or a random variable r. Thus, our aim is to demonstrate two different settings of X to values in $\mathbb{F}_q$, such that $f(\vec{x})$ has different values, while all of $\mathbf{terms}_T(F(X,y))$ have the same value (or distribution r) at the two settings. Now, fix a particular length T iterated composition σ of F. We will now show that each of $\gamma_t(f^{\sigma|1}(\vec{x}), f^{\sigma|2}(\vec{x}), ..., f^{\sigma|t-1}(\vec{x}))$, $t \in [1..T]$, as well as $f^{\sigma|t}(\vec{x})$ is a function of only $\mathcal{B}''$, and is independent of $p_s^{j(f,s)}$, and also independent of $\mathcal{B}'$ defined above. Thus in choosing the two different settings for X, we can first set the basis $\mathcal{B}''$ to some value, which will fix the $\gamma(...)$ distribution, and then we can set the $\mathcal{B}'$ and $p_s^{j(f,s)}$ to two different values, while assuring that all consistency requirements are met.

For the base case, $\gamma_1()$ is clearly not a function of $p_s^{j(f,s)}$, or $\mathcal{B}'$. Now, for the induction step, consider $f^{\sigma|t-1}(\vec{x})$. which is given by

$$\phi_{t-1}(\vec{x}, \gamma_{t-1}(f^{\sigma|1}(\vec{x}), f^{\sigma|2}(\vec{x}), ..., f^{\sigma|t-1}(\vec{x}))).$$

where ϕ_{t-1} is in F. Now, by induction the $\gamma_{t-1}(...)$ expression is not a function of $p_s^{j(f,s)}$ or $\mathcal{B}'$.

Now, it is possible that $\gamma_{t-1}(...)$ is equal to some $p_s^{j(\phi^*,s)}(\vec{x})$ ($\phi^* \in \mathcal{C}$), in which case $\phi_{t-1}(\vec{x}, \gamma_{t-1}(...))$ would just equal $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$, for I corresponding to the unique super-guard $g_I(\vec{x}, y)$ which is consistent with $y + p_s^{j(\phi^*,s)}(\vec{x}) = 0$. But, $p_I^{j(\phi_{t-1},I)}$ is either $p_s^{j(\phi^{**},s)}(\vec{x})$ ($\phi^{**} \in \mathcal{C}$) or such an expression plus y, by definition of $\mathcal{C}$ and definition of $p_I^{j(f_s \circ h,I)}(\vec{x}, y)$. In either case, it is a function of only $p_s^{j(\phi,s)}(\vec{x})$ ($\phi \in \mathcal{C}$) by induction.

If $\gamma_{t-1}(...)$ is not equal to any $p_s^{j(\phi^*,s)}(\vec{x})$ ($\phi^* \in \mathcal{C}$), we will *claim* that we can choose $\vec{x}$ so as to assure that $\gamma_{t-1}(...)$ is not equal to any linear expression in $\mathcal{B}'$ (and $p_s^{j(f,s)}$) either, with probability $> 1/q^{1/4}$ (the probability is over simulator's randomness and the randomness returned in the terms). In this case $\phi_{t-1}(\vec{x}, \gamma_{t-1}(...))$ returns $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$, where I corresponds to the degenerate super-guard of g_s given by $J' = J$. However, such $p_I^{j(\phi_{t-1},I)}(\vec{x}, y)$ is again either $p_s^{j(\phi^{**},s)}(\vec{x})$ ($\phi^{**} \in \mathcal{C}$) or such an expression plus y, since $\mathcal{C}$ includes Compose(F, degenerate(F)).

Now, we demonstrate the two different settings of X to values in $\mathbb{F}_q$. We first choose k' linearly independent (over $\mathbb{F}_2$) values in $\mathbb{F}_q$ and set the basis $\mathcal{B}''$ to these values, so that all expressions $p_s^{j(\phi,s)}$ ($\phi \in \mathcal{C}$) are non-zero.

Then, assuming the above claim holds, over T steps, the probability of some γ_t being a linear expression in $\mathcal{B}'$ is at most $T/q^{1/2}$. So, with probability $1 - T/q^{1/2}$, each γ_t is not equal to some linear expression in $\mathcal{B}'$, and hence the terms returned (i.e. ϕ_t) will be independent of $\mathcal{B}'$. Now, we show how to set $\mathcal{B}'$ and $p_s^{j(f,s)}$, so that the above claim holds. Let Γ be the set of values c (in the field) such that in some step (t in T), the probability of γ_{t-1} being c is more than $1/q^{1/2}$. (More formally, the proof should be done by maintaining an induction hypothesis about the claim, and building the set Γ inductively.) Note that $|\Gamma| < q^{1/2} \times (T+1)$.

Next, we inductively assign values to the basis $\mathcal{B}'$(of size $n - r - k' - 1$) as follows, so that the above claim holds. Let this basis be denoted by $l_1(\vec{x})$, ..., $l_{n-r-k'-1}(\vec{x})$. For $l_1(\vec{x})$, we pick any value in $\mathbb{F}_q$ which is not equal to any value in $\mathcal{L}(\mathcal{B}'') + \Gamma$, where the sum of two sets is defined naturally. For, the induction step, we choose for $l_\iota(\vec{x})$ a value in $\mathbb{F}_q$ which is not equal to any value in $\mathcal{L}(\mathcal{B}'' \cup \{l_1(\vec{x}), ..., l_{\iota-1}(\vec{x})\}) + \Gamma$.

Since $p_s^{j(f,s)}(\vec{x})$ is linearly independent of $\mathcal{B}''$ (as well as $\mathcal{B}'$), we choose a value for it which is not equal to any value in $\mathcal{L}(\mathcal{B}'' \cup \{l_1(\vec{x}), ..., l_{n-r-k'-1}(\vec{x})\}) + \Gamma$. Further we have at least two choices for it, given that $q \geq 2 + 2^{n-1-r} \times |\Gamma|$. Thus, no linear expression in $\mathcal{B}'$ or $p_s^{j(f,s)}$ is ever in Γ. Thus, f takes different values on these two settings os $\vec{x}$, whereas with probability $1 - T/q^{1/2} \geq 1 - 1/q^{1/4}$, all of $\mathbf{terms}_T(F(X, y))$ take the same value.

Case 2: Now consider the case where condition (i) holds, but condition (ii) of the F^*-interpolatable property fails to hold for f. This case is handled as in Theorem 16 but adapted with the analysis of Case 1 here. □

G Proof of Completeness of $L^{\$,\oplus,\mathsf{if}}$

We prove Theorem 8 in this section by starting off with the following lemma.

Lemma 19 *All the variables in an $L^{\$,\oplus,\mathsf{if}}$ program $f(\vec{z})$ are randomized pseudo-linear in $\vec{z}$.*

Proof: The proof is by structural induction on the grammar of expressions in $L^{\$,\oplus,\mathsf{if}}$. Since there are no loops in the language, we can assume wlog that no variable is assigned twice. In the base cases, the guards are derived according to the following rules ($[P]$ denotes the field polynomial corresponding to expression P) :

$$\begin{aligned}
&[x] && = x, \text{ for atom } x\\
&[XE_1 \oplus XE_2] && = [XE_1] + [XE_2]\\
\\
&[\texttt{true}] && = 1\\
&[XE_1 == XE_2] && = [XE_1 + XE_2]^{q-1}\\
&[BE_1 \wedge BE_2] && = [BE_1][BE_2]\\
&[\neg BE] && = 1 + [BE]
\end{aligned}$$

The conditional actions have an effect which can be viewed as follows for every relevant variable: `if` BE `then` $x := XE_1$ `else` $x := XE_2$. Then we will have $[x] = [BE][XE_1] + [\neg BE][XE_2]$, since $([BE], [\neg BE]) \in \{(0,1),(1,0)\}$.

It is easy to see that expressions constructed as above are pseudo-linear in the atoms. For the inductive case, xor-ing two pseudo-linear expressions again is a pseudo-linear expression. The only non-trivial case is the construction of conditional expressions from pseudo-linear expressions. We have to prove the following: *Any pseudo-linear polynomial raised to the power $q-1$ is a sum of* guard *expressions.* Given this the induction is straightforward.

To prove this recall that PLs can be expressed as sum of EPSELIN terms $\prod_{l\in\mathcal{L}/J} l(\vec{x})^{q-1} \cdot \prod_{l\in J}(1 + l(\vec{x})^{q-1}) \cdot p(\vec{x})$. Observe that the product of any two distinct EPSELIN guards $\prod_{l\in\mathcal{L}/J_1} l(\vec{x})^{q-1} \cdot \prod_{l\in J_1}(1 + l(\vec{x})^{q-1})$ and $\prod_{l\in\mathcal{L}/J_2} l(\vec{x})^{q-1} \cdot \prod_{l\in J_2}(1 + l(\vec{x})^{q-1})$ is 0.

Therefore, we can write down any pseudo-linear polynomial (in $\vec{x}$) as:

$$GE = (EPS_1 + EPS_2 + \cdots) = (G_1.L_1 + G_2.L_2 + \cdots),$$

where the EPS_i's are EPSELIN terms, the G_i's are guards and the L_i's are the corresponding linear expressions (after gathering all the linear terms with the same G_i together). Now, for any substitution of the atoms $\vec{x}$, at most one of the G_i's is equal to 1 and the rest of the G_i's are 0. This lets us write:

$$(G_1.L_1 + G_2.L_2 + \cdots)^{q-1} = \begin{cases} 0 & \text{if all the } G_i(\vec{x})\text{'s are 0}\\ L_1(\vec{x})^{q-1} & \text{if } G_1(\vec{x}) = 1\\ L_2(\vec{x})^{q-1} & \text{if } G_2(\vec{x}) = 1\\ \cdots & \end{cases}$$

Hence this is exactly equal to the polynomial $(G_1.L_1^{q-1} + G_2.L_2^{q-1} + \cdots)$ which is a sum of guard expressions in the atoms. □

We now proceed to the main proof.

Proof:[Theorem 8] By Lemma 19, all the functions in P and F compute randomized pseudo-linear functions in the inputs. By Lemma 12, with negligible error, we can assume that these are given as SRPL functions.

Now, by Theorem 18, 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 17, if f is F^*-interpolatable, then there exists a probabilistic polynomial time (in $\lg q$) simulator.

□

H Example

H.1 Pseudo-linear functions

We will consider some simple examples to get a flavor of the problem. Suppose we are given two input functions f_1 and f_2 defined as follows:

$$f_1(x_1, x_2) = x_1 + x_2$$
$$f_2(x_1, x_2) = \left\{ \begin{array}{ll} 0 & \text{if } x_1 = 0 \text{ or } x_2 = 0 \\ x_1 + x_2 & \text{otherwise} \end{array} \right\} = x_2^{q-1} x_1 + x_1^{q-1} x_2$$

We ask if it is possible to extract just x_1 given $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. That is, can we express $f(x_1, x_2) = x_1$, in terms of f_1 and f_2 alone? To do so, we construct the following truth table:

	x_1	x_2	$x_1 + x_2$	f_1	f_2	f
Row 1	0	0	0	0	0	0
Row 2	x_1	0	x_1	0	x_1	x_1
Row 3	0	x_2	x_2	0	x_2	0
Row 4	x_1	x_1	0	0	0	x_1
Row 5	x_1	x_2	$x_1 + x_2$	$x_1 + x_2$	$x_1 + x_2$	x_1

In the table above we list all linear combinations of the atoms, in this case just x_1, x_2 and $x_1 + x_2$. Each row corresponds to different combinations of cases where each linear combination can be zero or non-zero. Any non-zero entry under a column means that the particular linear combination is non-zero. Simplifications are performed when some of the linear expressions are zero - e.g. Row 4, where we write x_1 under the column x_2 since $x_1 + x_2 = 0 \Rightarrow x_2 = x_1$. It turns out that any pseudo-linear expression projects to a linear expression in any particular row - thus each such function can be given by a column of linear expressions, e.g. f_1, f_2 and f above. In this particular table for f_1, f_2 and f we can come up with several evidences that f is not a function of f_1, f_2. Consider Row 4: both f_1 and f_2 are 0, whereas f is x_1. Therefore, in accordance with the structure of the row, if we vary x_1, keeping it non-zero and $x_2 = x_1$, we get two pairs (x', x') and (x'', x'') with $x' \neq x''$ such that $f_1(x', x') = f_1(x'', x'') = 0$ and $f_2(x', x') = f_2(x'', x'') = 0$, but $f(x', x') = x' \neq x'' = f(x'', x'')$. Hence f cannot be a function of f_1, f_2. We can construct a counterexample using Row 5 as well: vary x_1 keeping $x_1 + x_2$ constant and keeping $x_1, x_2, x_1 + x_2$ all non-zero - e.g. in $\mathrm{GF}(2^3)$: $(x_1', x_2') = (001, 010)$ and $(x_1'', x_2'') = (101, 110)$. The common evidence in both rows is that f is not a linear combination of f_1, f_2.

However, this is not the only type of evidence. Consider the following $f'(x_1, x_2)$:

$$f'(x_1, x_2) = \left\{ \begin{array}{ll} x_1 & \text{if } x_2 = 0 \\ 0 & \text{otherwise} \end{array} \right\} = (1 + x_2^{q-1}) x_1$$

Now the table looks like:

	x_1	x_2	$x_1 + x_2$	f_1	f_2	f'
Row 1	0	0	0	0	0	0
Row 2	x_1	0	x_1	0	x_1	x_1
Row 3	0	x_2	x_2	0	x_2	0
Row 4	x_1	x_1	0	0	0	0
Row 5	x_1	x_2	$x_1 + x_2$	$x_1 + x_2$	$x_1 + x_2$	0

Now in each row, f' is a linear combination of f_1, f_2 (including the 0-combination). However, there is a problem with Rows 2 and 3. The problem surfaces when we try to write f' as a combination of f_1, f_2:

	x_1	x_2	$x_1 + x_2$	f_1	f_2	f'
Row 1	0	0	0	0	0	0
Row 2	x_1	0	x_1	0	$f_2\ (= x_1)$	$f_2\ (= x_1)$
Row 3	0	x_2	x_2	0	$f_2\ (= x_2)$	0
Row 4	x_1	x_1	0	0	0	0
Row 5	x_1	x_2	$x_1 + x_2$	$f_1\ (= x_1 + x_2)$	$f_1\ (= x_1 + x_2)$	0

The following pairs can be seen to be counter-examples in GF(2^2): $(x_1', x_2') = (01, 00), (x_1'', x_2'') = (00, 01)$. For these pairs we have: $f_1(x_1', x_2') = 00 = f_1(x_1'', x_2'')$, $f_2(x_1', x_2') = 01 = f_2(x_1'', x_2'')$, but $f'(x_1', x_2') = 01 \neq 00 = f'(x_1'', x_2'')$. This counter-example has been generated by looking at Rows 2 and 3: one of the technical challenges we solve is to systematically come up with counter-examples when arbitrary number of atoms and functions are involved.

Finally, consider the function f'':

$$f''(x_1, x_2) = \left\{ \begin{array}{ll} x_1 & \text{if } x_2 = 0 \\ x_2 & \text{if } x_1 = 0 \text{ and } x_2 \neq 0 \\ 0 & \text{otherwise} \end{array} \right\} = (1 + x_2^{q-1})x_1 + (1 + x_1^{q-1})x_2$$

Now the table looks like:

	x_1	x_2	$x_1 + x_2$	f_1	f_2	f''
Row 1	0	0	0	0	0	0
Row 2	x_1	0	x_1	0	x_1	x_1
Row 3	0	x_2	x_2	0	x_2	x_2
Row 4	x_1	x_1	0	0	0	0
Row 5	x_1	x_2	$x_1 + x_2$	$x_1 + x_2$	$x_1 + x_2$	0

Writing f'' as a combination of f_1, f_2:

	x_1	x_2	$x_1 + x_2$	f_1	f_2	f''
Row 1	0	0	0	0	0	0
Row 2	x_1	0	x_1	0	f_2 $(= x_1)$	f_2 $(= x_1)$
Row 3	0	x_2	x_2	0	f_2 $(= x_2)$	f_2 $(= x_2)$
Row 4	x_1	x_1	0	0	0	0
Row 5	x_1	x_2	$x_1 + x_2$	f_1 $(= x_1 + x_2)$	f_1 $(= x_1 + x_2)$	0

When we "collapse" the table to just the functions we have:

	f_1	f_2	$f_1 + f_2$	f''
Row 1	0	0	0	0
Row 2	0	f_2	f_2	f_2
Row 3	f_1	f_1	0	0

Now we claim that f'' is a function of f_1 and f_2 alone. In fact this can be verified easily:

$$f'' = f_1 + f_2$$

In this particular case we observe that f'' is pseudo-linear in f_1, f_2. We actually prove the general result that if the target function is a function of the input functions, then it is a pseudo-linear function of the input functions.

H.2 Iterated pseudo-linear functions

Consider the input function $f_1(x_1, y)$ and the target function $f(x_1)$ defined as follows:

$$f_1(x_1, y) = \left\{ \begin{array}{ll} x_1 & \text{if } y = x_1 \\ 0 & \text{otherwise} \end{array} \right\} = (1 + (x_1 + y)^{q-1})x_1$$
$$f(x_1) = x_1$$

It is easy to see that the *iterated compositions* of $f_1(x_1, y)$ is just the single function $\mathbf{0}$, which outputs 0 on any input. However, it is possible to compute $f(x_1)$ by calling $f_1(x_1, y)$ as the following algorithm demonstrates:

Algorithm Simulate_f$^{f_1}()$

```
repeat for all non-zero elements y in F_q
  t ← f_1(x_1, y)
  if (t ?= y)
    return t
  y ← next y
end repeat block
return 0
```

In this example, we observe that the complexity of the algorithm is $O(q)$.

Now consider the following input and target functions:

$$f_1'(x_1, y) = \left\{ \begin{array}{ll} 0 & \text{if } y = 0 \text{ or } y = x_1 \\ x_1 & \text{otherwise} \end{array} \right\} = y^{q-1}(x_1 + y)^{q-1}x_1$$

$$f'(x_1) = x_1$$

Here also the *iterated compositions* of $f_1'(x_1, y)$ is just the single function $\mathbf{0}$. Also, it is possible to compute $f(x_1)$ (with high probability) by calling $f_1'(x_1, y)$ as the following algorithm demonstrates:

Algorithm Simulate_f$^{f_1'}()$

```
choose y randomly from F_q
t ← f_1(x_1, y)
return t
```

In this example, we observe that the complexity of the algorithm is $O(1)$, but it works with probability $1 - O(1/q)$: the probability of y being different from 0 and x_1. For this particular example, it is also possible to come up with an efficient deterministic algorithm - but systematically coming up with efficient deterministic algorithms in all cases where it's possible, seems to be a hard problem. We do show how to systematically come up with randomized efficient algorithms in all the cases where it is possible to do so.

I A Cryptographic Application

We start by describing the Universally Composable (UC) framework in more detail. The UC framwork is a formal system for proving security of computational systems such as cryptographic protocols. The framework describes two probabilistic games: The *real world* that captures the protocol flows and the capabilities of an adversary, and the *ideal world* where dummy parties and the adversary interact with a trusted third party that characterizes the security requirements. The notion of security asserts that these two worlds are essentially equivalent. We now describe the real-world and the ideal world in more detail.

The real-world model. The players in the real-world model are all the entities of interest in the system (e.g., the nodes in a network, the processes in a software system, etc.), as well as *the adversary* A and *the environment* $\mathcal{Z}$. All these players are modeled as efficient, probabilistic, message-driven programs (formally, they are all interactive Turing machines).

The actions in this game should capture all the interfaces that the various participants can utilize in an actual deployment of this component in the real world. In particular, the capabilities of A should capture all the interfaces that a real-life attacker can utilize in an attack on the system. (For example, A can typically see and modify network traffic.) The environment $\mathcal{Z}$ is responsible for providing all the inputs to the players and getting all the outputs back from them. Also, $\mathcal{Z}$ is in general allowed to communicate with the adversary A. (This captures potential interactions where higher-level protocols may leak things to the adversary, etc.)

The ideal-world model. Security in the UC framework is specified via an "ideal functionality" (usually denoted $\mathcal{F}$), which is another interactive Turing Machine. The ideal-world model has the same environment as the real-world model, but we pretend that there is a completely trusted party (the functionality $\mathcal{F}$) that performs all the tasks that are required of the protocol. In the ideal world, participants just give their inputs to the functionality $\mathcal{F}$, which produces the correct outputs (based on the specification) and hands them back to the participants. $\mathcal{F}$ may interact with an adversary, but only to the extent that the intended security allows. (E.g., it can "leak" to the adversary things that should be publicly available, such as public keys.)

UC-Security. A protocol π **securely realizes** an ideal functionality $\mathcal{F}$ if for every probabilistic polynomial time (PPTM) adversary A in the real world, there exists a PPTM adversary A' in the ideal world, such that no environment $\mathcal{Z}$ can distinguish between interacting with A and π in the real world and interacting with A' and $\mathcal{F}$ in the ideal world.

I.1 UC Functionality for password-based key exchange

Password-based key exchange is an important security problem which has been studied extensively in cryptographic research [BM93], and which brings out the power of the UC framework particularly well. Canetti et al [CHK$^+$05] proposed an Ideal Functionality for password-based key exchange which is formally described in Figure 1.

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.

Functionality $\mathcal{F}_{\mathbf{pwKE}}$

The functionality $\mathcal{F}_{\text{pwKE}}$ is parameterized by a security parameter k. It interacts with an adversary S and a set of parties via the following queries:

Upon receiving a query $(\mathsf{NewSession}, sid, P_i, P_j, pw, role)$ **from party** P_i**:**
Send $(\mathsf{NewSession}, sid, P_i, P_j, role)$ to S. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record fresh.

Upon receiving a query $(\mathsf{TestPwd}, sid, P_i, pw')$ **from the adversary** S**:**
If there is a record of the form (P_i, P_j, pw) which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to S with "correct guess". If $pw \neq pw'$, mark the record interrupted and reply with "wrong guess".

Upon receiving a query $(\mathsf{NewKey}, sid, P_i, sk)$ **from** S**, where** $|sk| = k$**:**
If there is a record of the form (P_i, P_j, pw), and this is the first NewKey query for P_i, then:

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j, and (P_j, P_i, pw) was fresh at the time, then output (sid, sk') to P_i.
- In any other case, pick a new random key sk' of length k and send (sid, sk') to P_i.

Either way, mark the record (P_i, P_j, pw) as completed.

Figure 1: The password-based key-exchange functionality $\mathcal{F}_{\text{pwKE}}$

Our motivation in constructing the language $L^{\$,\oplus,\mathsf{if}}$ in this paper is driven by cryptographic definitions and protocols in the UC model. The functionality $\mathcal{F}_{\text{pwKE}}$ can be modeled in an extension of $L^{\$,\oplus,\mathsf{if}}$ as described in Figure 2. To simplify exposition we have not dealt with corruption and session ids in this description, however it is possible to describe those without further extension.

The extensions to $L^{\$,\oplus,\mathsf{if}}$ necessary to cover this description are: (i) variables which are persistent across function calls, and (ii) the functionality outputing to the environment. We do not consider these extensions in this paper, but to bring out the power of $L^{\$,\oplus,\mathsf{if}}$ itself, we we describe an abstract version of this functionality in $L^{\$,\oplus,\mathsf{if}}$ in Figure 3. This version is different from $\mathcal{F}_{\text{pwKE}}$, but captures some of its key combinatorial aspects. The main limitation of this description is that most of the information comes from the environment instead of the adversary.

The function f_{int1} capture the scenario where the messages from party 1 to 2 were modified; likewise for f_{int2}. The function f_{test} captures the fact that the adversary guesses the password correctly. The function f_{key} models the derivation of keys by both parties - but instead of outputting the keys to the environment, the difference ($\oplus$ for us) between the keys is returned.

The important aspects which are captured is that when the messages are tampered with, the difference in keys is uniformly random; when the password is guessed correctly by the adversary, the difference can be set to a value controlled by the adversary; when nothing is tampered with, the difference is zero.

Now we describe a protocol Π_{sPWKE^-} , in Figure 4 , which is a candidate to realize $\mathcal{F}_{\text{sPWKE}^-}$.

Functionality $\mathcal{F}_{\mathbf{pwKE^-}}$

$f_{newsession}(pw)$:

persistent variables r, sk_1, sk_2

$r \leftarrow \$;\ sk_1 \leftarrow \$;\ sk_2 \leftarrow \$$

$f_{testpwd1}(pw; w, k, ignore)$:

if $(ignore \neq 0)$ then

  if $(pw == w)$ then

    $sk_1 := k$

    $result :=$ "correct guess"

  else

    $sk_1 \leftarrow \$$

    $result :=$ "wrong guess"

else

  $sk_1 := r$

  $result :=$ "ignored"

return $result$

$f_{testpwd2}(pw; w, k, ignore)$:

if $(ignore \neq 0)$ then

  if $(pw == w)$ then

    $sk_2 := k$

    $result :=$ "correct guess"

  else

    $sk_2 \leftarrow \$$

    $result :=$ "wrong guess"

else

  $sk_2 := r$

  $result :=$ "ignored"

return $result$

$f_{newkey1}(pw)$:

output sk_1

$f_{newkey2}(pw)$:

output sk_2

Figure 2: The password-based key-exchange functionality $\mathcal{F}_{\text{pwKE}^-}$

Simplified Functionality $\mathcal{F}_{\mathbf{spwKE^-}}$

Let $\vec{x} = (pw, guess, c_1, c_1', c_2, c_2', \text{kdiff})$

$f_{int1}(\vec{x})$:

  return $c_1 \oplus c_1'$

$f_{int2}(\vec{x})$:

  return $c_2 \oplus c_2'$

$f_{test}(\vec{x})$:

  return $pw \oplus guess$

$f_{key}(\vec{x}; compromised, ignore)$:

```
if (ignore ≠ 0) then
  if (compromised ≠ 0) then
    skdiff := kdiff
  else
    skdiff ← $
else
  skdiff := 0
return skdiff
```

Figure 3: Simplified password-based key-exchange functionality $\mathcal{F}_{\text{SPWKE}^-}$

This is loosely based on protocols in the Ideal Cipher model [BPR00]. The idea is that when the password is guessed correctly, the difference can be set to a given value kdiff. Otherwise, there are two cases. When the messages in both directions are not tampered with, specifically $c_1 == c_1'$ and $c_2 == c_2'$, both parties end up with the key $r_1 \oplus r_2$ and hence the difference is 0. If either message is tampered with, the difference in keys ends up being uniformly random.

We prove that Π_{SPWKE^-} realizes $\mathcal{F}_{\text{SPWKE}^-}$ by demonstrating the following simulator. The constant c is any non-zero member of the underlying field.

Protocol $\Pi_{\mathbf{spwKE^-}}$

Let $\vec{x} = (pw, guess, c_1, c_1', c_2, c_2', \text{kdiff})$

$f_{spwke}(\vec{x})$:

$r_1 \leftarrow \$$
$r_2 \leftarrow \$$
`if` $(c_1 == c_1')$ `then`
  $r_1' := r_1$
`else`
  $r_1' \leftarrow \$$
`if` $(c_2 == c_2')$ `then`
  $r_2' := r_2$
`else`
  $r_2' \leftarrow \$$
$sk_1 := r_1' \oplus r_2$
$sk_2 := r_1 \oplus r_2'$
`if` $(guess == pwd)$ `then`
  skdiff := kdiff
`else`
  skdiff := $sk_1 \oplus sk_2$
`return` skdiff

Figure 4: The password-based key-exchange proctocol Π_{SPWKE^-}

Algorithm S

$comp1 := f_{int1}(\vec{x})$
$comp2 := f_{int2}(\vec{x})$
$testpwd := f_{test}(\vec{x})$
`if` $(testpwd \neq 0)$ `then`
  `if` $(comp1 == 0 \wedge comp2 == 0)$ `then`
    $kd := f_{key}(\vec{x}, c, 0)$
  `else`
    $kd := f_{key}(\vec{x}, 0, c)$
`else`
  $kd := f_{key}(\vec{x}, c, c)$
`return` kd

It is readily seen that the Simulator S is in fact an SRPL probabilistic function of **extended-pl-terms**$(\mathcal{F}_{\text{SPWKE}^-}, c)$, thus demonstrating Theorem 18.

J An Undecidable System

In this section we describe one language L_{tab} for which we are able to prove that no algorithmic procedure exists to decide equivalence. In particular, this arises when unbounded table lookup/storage operations are allowed (not even random access, but just storing and detecting whether some string is there in the table), even when there are no arithmetic operations, loops in subroutines or random number generation. Formally, we have the following theorem:

Theorem 20 (An Undecidable System) *Let L_{tab} be a language with input / outputs to the environment, send / receives to the adversary, conditional with equality checking of strings and table storage/lookup. We are given a real protocol $\mathcal{P}$ and a ideal functionality $\mathcal{F}$ with all subroutines described in L_{tab}. We show that it is impossible to algorithmically decide whether $\mathcal{P}$ realizes $\mathcal{F}$.*

We describe the language L_{tab} formally in Table 2. Note that there is no arithmetic, logical operation or nonce generation in the language. On the other hand, strings can be of arbitrary length.

J.1 Analysis

Theorem 21 *The language $Sim_{tab} = \{(\mathcal{P}, \mathcal{F}) \mid \mathcal{P}$ realizes $\mathcal{F}$, both $\mathcal{P}$ and $\mathcal{F}$ are described in $L_{tab}\}$ is undecidable .*

Proof: We reduce the problem $\{M \mid M$ is a Turing Machine which accepts the empty string $\epsilon\}$ to Sim_{tab}. Specifically, given the state transition representation of M, we construct a $(\mathcal{P}, \mathcal{F})$ instance whose membership in Sim_{tab} is equivalent to deciding whether M reaches an accepting state on input the empty string.

Construction of $\mathcal{P}$.

The real protocol $\mathcal{P}$ has just the following subroutine $\mathcal{P}_1$: [`receive` x; `output` "*success*";]

Construction of $\mathcal{F}$.

The functionality $\mathcal{F}$ models construction of "cells" corresponding to individual cells in the configuration of a Turing Machine computation. $\mathcal{F}$ consists of several small subroutines which perform operations like populating the table with the initial cells, copying non transiting cells to the next configuration, performing state transition at the TM header, detecting acceptance state and declaring success. We give formal descriptions of the subroutines in Appendix K. The informal description is as follows:

- $\mathcal{F}_{init}$: This is the subroutine which is called in the beginning. It stores some elements in the table corresponding to the empty string ϵ.
- $\mathcal{F}_{\rho}$: For each transition rule ρ, there is a corresponding subroutine which has the effect of producing the next configuration according to this transition rule.

(atomic terms)	u	::=	x	atomic term variable
			s	string
(terms)	t	::=	y	term variable
			u	atomic term
			$t.t$	pair of terms (bounded)
(table)	τ	::=	tab	table
(actions)	a	::=	`send` t	send a term t
			`receive` y	receive term into variable y
			`output` t	output t to the environment
			$y := t$	assign t to y
			$store(t, \tau)$	store t in τ
(booleans)	b	::=	$t = t$	equality check
			$lookup(t, \tau)$	true iff t is in τ
(program)	π	::=	a;	single action
			πa;	sequence of actions
			`if` (b) `then` $\{\pi\}$	conditional
(real protocol)	$\mathcal{P}$	::=	π	one program
(ideal functionality)	$\mathcal{F}$	::=	$\{\pi, \pi, \cdots, \pi\}$	set of programs

Table 2: L_{tab}: Language definition for the undecidable system

- $\mathcal{F}_{begin}$: Subroutine that constructs the beginning of the current configuration.
- $\mathcal{F}_{\omega}$: Subroutine that constructs the end of the current configuration.
- $\mathcal{F}_{persist}$: Subroutine that carries over bits from previous configuration to the current one, if unaffected by state transistion.
- $\mathcal{F}_{accept}$: Subroutine that transitions to an accepting state.
- $\mathcal{F}_{success}$: Subroutine that declares success upon completing an accepting configuration.

Simulation for an M that accepts ϵ.

The construction is such that there is a way to simulate transition from one configuration to the next. We show that if the Turing Machine accepts the empty string then there is a way for the simulator to simulate the actions of this Turing Machine faithfully, leading to successful calling of the subroutine $\mathcal{F}_{success}$ - we give an explicit procedure below:

▷ First $\mathcal{F}_{init}$ is called to store the empty string into the table.

▷ Let $CurrentConfiguration \leftarrow EmptyConfiguration$

▷ Repeat the following:

- Call $\mathcal{F}_{begin}$ to construct the first cell of $NextConfiguration$
- If current cell is in the neighborhood of the state cell, call $\mathcal{F}_{\rho}$ with the corresponding transition rule ρ. This subroutine also pushes the "end" cell, if the state cell goes past it. Record *AcceptanceStateReached* if new state is an accepting state.
- Otherwise call $\mathcal{F}_{persist}$ to just copy the bit from the corresponding cell in $CurrentConfiguration$ to $NextConfiguration$.
- If *AcceptanceStateReached* is true and "end" cell is reached, let set $CallSuccess$ to true.
- Otherwise, let $CurrentConfiguration \leftarrow NextConfiguration$

▷ Until $CallSuccess$ is true.

▷ Call $\mathcal{F}_{success}$ and halt.

Impossibility of simulation for an M that does not accept ϵ.

We show that it is impossible to register a configuration in the table, which does not follow from an already registered configuration. Thus if there is an accepting configuration in the table, then there must be a sequence of configurations in the table, beginning with the empty configuration and ending with accepting configuration, such that each configuration leads to the next by a single transition.

Every storage cell has the following structure:

```
Current Conf Id.  Current Cell Id.  Cell Element.  Next Cell Id.  Next Conf Id
```

Before any cell is stored in the table, the previous cell also has to be provided by the simulator. It is checked whether the immediate predecessor is already there or not: the corresponding Id's have to match up correctly - think of this as a 2-dimensional linked list, in which each configuration is a linked list and where `Next Cell Id` points to the next cell in the current configuration linked list and `Next Conf Id` points to the next configuration linked list. Each time a new configuration is being built, a new `Next Conf Id` has to be provided by the simulator. The functionality checks that it is new by first checking for presence and then storing it in the table as a type "confid" data. Similarly for new cells that go past the current tape length, there is checking and storing of a type "cellid" data. This combination of checks ensures the integrity of the 2-D linked list - in particular, there are no rogue pointers to elements in number of different configurations. Ensuring that the simulator provides the previous cell also "bootstraps" the cell records consistently - one cannot enter a new element if the previous element is not already there. Importantly, this ensures the following Lemma:

Lemma 22 *For a given `Current Conf Id` linked list, it is only possible to call a unique $\mathcal{F}_\rho$.*

This is because the simulator *has to* "bootstrap" up to the correct state cell following the pointers and only a unique transition works for a given state and cell after the state cell. Consequently, the following assertion holds:

Lemma 23 *If there is an "end" cell registered for a given `Current Conf Id`, the linked list of cells with this confid is the successor of some configuration in the table.*

This implies our original claim that if there is an accepting configuration in the table, then there must be a sequence of configurations in the table, beginning with the empty configuration and ending with accepting configuration, such that each configuration leads to the next by a single transition. This is a contradiction since no such sequence exists for the given M. Hence no simulator exists.

□

K Formal description of Ideal Functionality for the undecidable language

Init.

$\mathcal{F}_{init}$: [

receive $trigger$;

store “$conf0$”.“$cell0$”.“$begin$”.“$cell1$”.“$conf1$”, τ;

store “$conf0$”.“$cell1$”.“$qstart$”.“$cell2$”.“$conf1$”, τ;

store “$conf0$”.“$cell2$”.“ ”.“$cell3$”.“$conf1$”, τ;

store “$conf0$”.“$cell3$”.“end”.“$cell4$”.“$conf1$”, τ;

store “$cellid$”.“$cell0$”, τ;

store “$cellid$”.“$cell1$”, τ;

store “$cellid$”.“$cell2$”, τ;

store “$cellid$”.“$cell3$”, τ;

store “$confid$”.“$conf0$”, τ;

store “$confid$”.“$conf1$”, τ;

]

Transition Rule. For a transition rule $\rho : (q, 0) \to (r, 1, R)$, we have the subroutine:

$\mathcal{F}_\rho$: [

`receive` $cell_1, cell_2, cell_3, cell_4, cell_5$;
`receive` $cell'_1, cell'_2, cell'_3, cell'_4, cell'_5$;

`parse` $cell_1$ `as` $\sigma.\beta_1.\ b_1.\ \beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2.\ b_2.\ \beta_3.\sigma'$;
`parse` $cell_3$ `as` $\sigma.\beta_3.$ "q". $\beta_4.\sigma'$;
`parse` $cell_4$ `as` $\sigma.\beta_4.$ "$zero$". $\beta_5.\sigma'$;
`parse` $cell_5$ `as` $\sigma.\beta_5.\ b_4.\ \beta_6.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1.\ b_1.\ \beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2.\ b_2.\ \beta_3.\sigma''$;
`parse` $cell'_3$ `as` $\sigma'.\beta_3.$ "one". $\beta_4.\sigma''$;
`parse` $cell'_4$ `as` $\sigma'.\beta_4.$ "r". $\beta_5.\sigma''$;
`parse` $cell'_5$ `as` $\sigma'.\beta_5.\ b_4.\ \beta_6.\sigma''$;

`if` (none of $cell_1, cell_2, cell_3, cell_4, cell_5$ is in τ)
  `then output` "$failure$";
`if` ($cell'_1$ is not in τ)
  `then output` "$failure$";
`store` $cell'_2, cell'_3, cell'_4, cell'_5$ `in` τ;

]

For a transition rule $\rho : (q, 0) \rightarrow (r, 1, L)$, we have the subroutine:

$\mathcal{F}_\rho$: [

`receive` $cell_1, cell_2, cell_3, cell_4, cell_5$;
`receive` $cell'_1, cell'_2, cell'_3, cell'_4, cell'_5$;

`parse` $cell_1$ `as` $\sigma.\beta_1.\ b_1.\ \beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2.\ b_2.\ \beta_3.\sigma'$;
`parse` $cell_3$ `as` $\sigma.\beta_3.$ "q". $\beta_4.\sigma'$;
`parse` $cell_4$ `as` $\sigma.\beta_4.$ "$zero$". $\beta_5.\sigma'$;
`parse` $cell_5$ `as` $\sigma.\beta_5.\ b_4.\ \beta_6.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1.\ b_1.\ \beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2.$ "r". $\beta_3.\sigma''$;
`parse` $cell'_3$ `as` $\sigma'.\beta_3.\ b_2.\ \beta_4.\sigma''$;
`parse` $cell'_4$ `as` $\sigma'.\beta_4.$ "one". $\beta_5.\sigma''$;
`parse` $cell'_5$ `as` $\sigma'.\beta_5.\ b_4.\ \beta_6.\sigma''$;

`if` (none of $cell_1, cell_2, cell_3, cell_4, cell_5$ is in τ)
  `then output` "$failure$";
`if` ($cell'_1$ is not in τ)
  `then output` "$failure$";
`store` $cell'_2, cell'_3, cell'_4, cell'_5$ `in` τ;

]

Boundary Regions.

$\mathcal{F}_{begin}$: [

`receive` $cell_1, cell_2$;
`receive` $cell'_1, cell'_2$;

`parse` $cell_1$ `as` $\sigma.\beta_1$. "$begin$". $\beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2$. b. $\beta_3.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1$. "$begin$". $\beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2$. b. $\beta_3.\sigma''$;

`if` (none of $cell_1, cell_2$ is in τ)
  `then output` "$failure$";
`if` ("$confid$".σ'' is in τ)
  `then output` "$failure$";
`store` $cell'_1, cell'_2$ `in` τ;
`store` "$confid$".σ'' `in` τ;

]

For a transition rule of the form $\omega : (q, \_) \rightarrow (r, 0, R)$ at the end of the tape:

$\mathcal{F}_\omega$: [

receive $cell_1, cell_2, cell_3, cell_4, cell_5$;
receive $cell'_1, cell'_2, cell'_3, cell'_4, cell'_5, cell'_6$;

parse $cell_1$ as $\sigma.\beta_1.\ b_1.\ \beta_2.\sigma'$;
parse $cell_2$ as $\sigma.\beta_2.\ b_2.\ \beta_3.\sigma'$;
parse $cell_3$ as $\sigma.\beta_3.$ "q". $\beta_4.\sigma'$;
parse $cell_4$ as $\sigma.\beta_4.$ " ". $\beta_5.\sigma'$;
parse $cell_5$ as $\sigma.\beta_5.$ "end". $\beta_6.\sigma'$;

parse $cell'_1$ as $\sigma'.\beta_1.\ b_1.\ \beta_2.\sigma''$;
parse $cell'_2$ as $\sigma'.\beta_2.\ b_2.\ \beta_3.\sigma''$;
parse $cell'_3$ as $\sigma'.\beta_3.$ "$zero$". $\beta_4.\sigma''$;
parse $cell'_4$ as $\sigma'.\beta_4.$ "r". $\beta_5.\sigma''$;
parse $cell'_5$ as $\sigma'.\beta_5.$ " ". $\beta_6.\sigma''$;
parse $cell'_6$ as $\sigma'.\beta_6.$ "end". $\beta_7.\sigma''$;

if (none of $cell_1, cell_2, cell_3, cell_4, cell_5$ is in τ)
  then output "$failure$";
if ($cell'_1$ is not in τ)
  then output "$failure$";
if ("$cellid$".β_7 is in τ)
  then output "$failure$";
store $cell'_2, cell'_3, cell'_4, cell'_5, cell'_6$ in τ;

]

Persistence of other bits.

$\mathcal{F}_{persist}$: [

`receive` $cell_1, cell_2$;
`receive` $cell'_1, cell'_2$;

`parse` $cell_1$ `as` $\sigma.\beta_1.\ b_1.\ \beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2.\ b_2.\ \beta_3.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1.\ b_1.\ \beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2.\ b_2.\ \beta_3.\sigma''$;

`if` (none of $cell_1, cell_2$ is in τ)
  `then output` "$failure$";
`if` ($cell'_1$ is not in τ)
  `then output` "$failure$";
`store` $cell'_1$ `in` τ;

]

Detecting Machine Acceptance. For a transition rule $\rho : (q, 0) \rightarrow (q_{accept}, 1, R)$, we have the subroutine:

$\mathcal{F}_\rho$: [

`receive` $cell_1, cell_2, cell_3, cell_4, cell_5$;
`receive` $cell'_1, cell'_2, cell'_3, cell'_4, cell'_5$;

`parse` $cell_1$ `as` $\sigma.\beta_1.\ b_1.\ \beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2.\ b_2.\ \beta_3.\sigma'$;
`parse` $cell_3$ `as` $\sigma.\beta_3.$ "q". $\beta_4.\sigma'$;
`parse` $cell_4$ `as` $\sigma.\beta_4.$ "$zero$". $\beta_5.\sigma'$;
`parse` $cell_5$ `as` $\sigma.\beta_5.\ b_4.\ \beta_6.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1.\ b_1.\ \beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2.\ b_2.\ \beta_3.\sigma''$;
`parse` $cell'_3$ `as` $\sigma'.\beta_3.$ "one". $\beta_4.\sigma''$;
`parse` $cell'_4$ `as` $\sigma'.\beta_4.$ "$qaccept$". $\beta_5.\sigma''$;
`parse` $cell'_5$ `as` $\sigma'.\beta_5.\ b_4.\ \beta_6.\sigma''$;

`if` (none of $cell_1, cell_2, cell_3, cell_4, cell_5$ is in τ)
  `then output` "$failure$";
`if` ($cell'_1$ is not in τ)
  `then output` "$failure$";
`store` $cell'_2, cell'_3, cell'_4, cell'_5$ `in` τ;
`store` "$AcceptanceStateReached.''.\sigma'$ `in` τ;

]

Similarly for a transition rule $\rho : (q, 0) \rightarrow (r, 1, L)$.

Subroutine to check consistency at the end:

$\mathcal{F}_{success}$: [

`receive` $cell_1, cell_2$;
`receive` $cell'_1, cell'_2$;

`parse` $cell_1$ `as` $\sigma.\beta_1.$ " ". $\beta_2.\sigma'$;
`parse` $cell_2$ `as` $\sigma.\beta_2.$ "*end*". $\beta_3.\sigma'$;

`parse` $cell'_1$ `as` $\sigma'.\beta_1.$ " ". $\beta_2.\sigma''$;
`parse` $cell'_2$ `as` $\sigma'.\beta_2.$ "*end*". $\beta_3.\sigma''$;

`if` (none of $cell_1, cell_2$ is in τ)
  `then output` "$failure$";
`if` ($cell'_1$ is not in τ)
  `then output` "$failure$";

`if` ("$AcceptanceStateReached$".σ' is in τ)
  `then output` "$success$";

]

ECCC ISSN 1433-8092
http://eccc.hpi-web.de