Linicrypt: A Model for Practical Cryptography[*]

Brent Carmer[†] Mike Rosulek[†]

June 1, 2016

Abstract

A wide variety of objectively practical cryptographic schemes can be constructed using only symmetric-key operations and linear operations. To formally study this restricted class of cryptographic algorithms, we present a new model called *Linicrypt*. A Linicrypt program has access to a random oracle whose inputs and outputs are field elements, and otherwise manipulates data only via fixed linear combinations.

Our main technical result is that it is possible to decide *in polynomial time* whether two given Linicrypt programs induce computationally indistinguishable distributions (against arbitrary PPT adversaries, in the random oracle model).

We show also that indistinguishability of Linicrypt programs can be expressed as an existential formula, making the model amenable to *automated program synthesis.* In other words, it is possible to use a SAT/SMT solver to automatically generate Linicrypt programs satisfying a given security constraint. Interestingly, the properties of Linicrypt imply that this synthesis approach is both sound and complete. We demonstrate this approach by synthesizing Linicrypt constructions of garbled circuits.

1 Introduction

Throughout cryptography, we find many examples of objectively practical constructions that share common features. In particular, they treat blocks of bits as atomic units, and manipulate these units by calling a symmetric-key primitive or by interpreting them as elements in a field and applying *strictly linear* operations to them. Below are just some examples:

- Standard block cipher modes like CBC, OFB, PCBC for privacy, and LRW modes [LRW02] for tweakable block ciphers consist of calls to the underlying block cipher and XOR, the linear operation in $GF(2^n)$. (This ignores matters of padding/ciphertext stealing, where the input is not an exact multiple of field elements.)
- Constructions in other settings also consist of calls to an underlying symmetric primitive along with XOR operations: the Davies-Meyer construction & its variants [PGV94, BRS02] for collision-resistance; the Even-Mansour [EM93] and Feistel [LR86] constructions for PRPs; NMAC, HMAC [KBC97], and VMAC [KD07] for authenticity; Naor's commitment scheme [Nao91].
- Some constructions use $GF(2^n)$-linear transformations with (fixed) coefficients other than 1 (i.e., these constructions use multiplication by fixed field elements). These include: OCB mode [RBBK01] for authenticated encryption, CMC mode [HR03] for disk encryption, XE/XEX modes [Rog04] for tweakable block ciphers, PMAC [BR02] for authentication.

[*]Full version of a paper appearing in CRYPTO 2016.

[†]Oregon State University, {carmerb,rosulekm}@eecs.oregonstate.edu. Authors supported by NSF award 1149647.

- Signing algorithms for lightweight one-time signature schemes like those of Lamport [Lam79] and Winternitz [Win83] consist purely of calls to a one-way or [target] collision-resistant hash function. Variants like W-OTS+ [Hül13] incorporate XOR operations. Few-time signature schemes like HORS and variants [RR02, PWX04] also use only a random oracle. These simple signature schemes can be composed to give many-use signature schemes using Merkle trees [Mer90] and derivatives thereof [Gol87, NSW05, BHH$^+$15, PPB15, BDK$^+$07, BDH11, BGD$^+$06]. These extensions do not introduce any additional operations on the atomic field elements.

- Practical constructions of garbled circuits [NPS99, KS08, KMR14, ZRE15, GLNP15] simply use XOR and calls to an underlying hash function/KDF, while the construction of [PSSW09] uses polynomial interpolation (with fixed points of evaluation) over $GF(2^n)$, which is a linear operation.

1.1 Overview of Our Results

Inspired by the constructions above, we introduce a restricted model of computation called **Linicrypt**. Programs in the Linicrypt model have access to a random oracle (to model a symmetric-key primitive), whose inputs and outputs are elements of a field $\mathbb{F}$. The field $\mathbb{F}$ is public and its size should be exponential in the security parameter.

Beyond calling a random oracle, Linicrypt programs can manipulate field elements only by uniformly sampling them or by applying fixed linear combinations. More formally, a **(pure) Linicrypt program** is a fixed sequence of statements of the following form:

$v_i \xleftarrow{\$} \mathbb{F}$:	sample a value uniformly from $\mathbb{F}$.
$v_i := \sum_j c_j v_j$:	apply a linear combination to existing variables, using *fixed* coefficients.
$v_i := H(t \| v_{j_1} \| v_{j_2} \| \cdots \| v_{j_k})$:	call the random oracle on a set of existing variables, and optionally a string t which is fixed with the program (useful for domain separation).
output $(v_{j_1}, \ldots, v_{j_k})$:	output an ordered sequence of variables.

Linicrypt is expressive enough to capture cryptographic construction of interest, but still restrictive enough that it provides several key benefits:

1. It is tractable to reason about cryptographic properties of Linicrypt programs. Our **main technical result** is that it is possible to decide, *in polynomial time,* whether two Linicrypt programs induce indistinguishable output distributions (in the random oracle model, against *arbitrary* PPT adversaries).

   We also point out that unforgeability properties (*e.g.*, given the output of a program $\mathcal{P}$, it is hard to predict an internal value v^*) can be easily transformed into indistinguishability properties, making many standard styles of security definition expressible (and efficiently decidable) in Linicrypt.

2. Unlike in other restricted models, Linicrypt programs manipulate data as atomic units. This makes it possible to prove fine-grained lower bounds *to the level of optimal constant factors* (*e.g.*, "this cryptographic task cannot be done in Linicrypt with keys smaller than 5λ bits"). Such lower bounds for Linicrypt hold in the random oracle model, and hence they also imply impossibility of a black-box construction from one-way functions.

3. The question of finding a Linicrypt program whose output is indistinguishable from some specification (*e.g.*, its output is pseudorandom) can be expressed as an existential formula. One can then use an SAT/SMT solver to find a witness — *i.e.*, automatically *synthesize* a secure Linicrypt construction. Additionally, if the formula is found to be unsatisfiable, it implies that no secure Linicrypt construction exists for the task — *i.e.*, this paradigm for program synthesis is both *sound* and *complete.*

In Section 2 we formally define Linicrypt, develop techniques to reason about its algorithms, and prove our main technical result. Later in Section 3 we give an example application of our approach to program synthesis. We show how to use an SMT solver to synthesize secure Linicrypt constructions of garbled circuits. Specifically, for a given boolean function $f : \{0,1\}^k \to \{0,1\}^\ell$ (*e.g.*, an adder, a multiplexer), we synthesize Linicrypt procedures to garble f (as an atomic unit) in a way that is compatible with the Free XOR optimization of [KS08].

1.2 Related Work & Inspiration

Minicrypt. Linicrypt is inspired in name by Impagliazzo's [Imp95] Minicrypt, which refers to a hypothetical world in which one-way functions exist but no "fancier" cryptography is possible. Minicrypt is formalized (as in [IR90]) by having a random oracle and allowing adversaries to be computationally unbounded (but with only polynomially many queries to the oracle). In this way, the random oracle becomes the only available source of computational cryptography.

The main distinction therefore between Linicrypt & Minicrypt is the additional constraint of linearity. This restriction allows Linicrypt lower bounds to resolve optimal constant factors, whereas optimal constant factors are not typically well-defined in Minicrypt. For example, imagine instantiating a secure Minicrypt scheme with security parameter λ/c; as a function of λ, the resulting construction would typically have constants reduced by a factor of c but still be secure.

Generic group model. Linicrypt has many similarities to the generic group model (GGM) of Shoup [Sho97]. In the GGM, adversaries are restricted to manipulating elements of a *cyclic group* in a black-box way using only the prescribed group operations. While the GGM was originally proposed as a heuristic model for *adversaries*, one can also use GGM *constructions* to prove lower bounds. Dodis *et al.* [DHT12] show that full-domain hashing from RSA cannot be proven secure using techniques that treat the RSA group as a generic multiplicative group. Papakonstantinou *et al.* [PRV12] show that identity-based encryption is impossible via a GGM construction (without a bilinear pairing).

GGM lower bounds can identify *optimal constant factors*, which is one of the goals of Linicrypt. A line of work by Abe *et al.* [AGHO11, AGOT14b, AGOT14a] considers the case of *structure-preserving* digital signatures. They prove (among other things) that 3 group elements are optimal for structure-preserving signatures implemented by GGM algorithms. More recently, synthesis has been effectively applied [BFF$^+$15] to generate novel and optimal structure-preserving schemes.

Despite these similarities, we point out some important technical differences:

(1) In the GGM, group elements are represented via a random encoding into bits, and adversaries are allowed to "look at" these encodings. This is slightly less restricting than our compartmentalized approach in which encodings don't play a part (and hence Linicrypt programs cannot perform equality tests). In that regard, our model is similar to the generic-group variant of Maurer [Mau05]. Since our goal is to place restrictions on constructions rather than adversaries, the distinction does not seem to be very significant.

(2) Linicrypt includes a random oracle, which has not yet been considered in GGM lower bound results to the best of our knowledge. The random oracle is indeed a source of technical complications in Linicrypt.

(3) Both Linicrypt and GGM allow only linear operations (*e.g.*, in the GGM, a value "in the exponent" can only be manipulated in linear ways). However, a Linicrypt program must apply linear operations with *fixed* (*i.e.*, known to the adversary) coefficients, while the GGM model allows constructions to choose random (secret) coefficients. This difference is what allows Diffie-Hellman-style constructions to be modeled in GGM but not in Linicrypt. Namely, a GGM algorithm can hide a random value "in the exponent" by performing the generic operation $g \mapsto g^x$, but the analogous operation in Linicrypt ($v \mapsto xv$) hides nothing since x would always be considered fixed.

Algebraic cryptography model. Applebaum *et al.* [AAB15] define a model for *arithmetic cryptography*, building on earlier work by Ishai *et al.* [IPS09]. Their model has some similarities to Linicrypt but also fundamental differences. Compared to Linicrypt, the arithmetic model allows for general field operations on its elements, not just linear combinations. More importantly, the defining feature of the arithmetic model is that the construction is *oblivious* to the underlying field/ring — the construction must work no matter what field/ring is used. In order to model cryptographic practice, Linicrypt allows the ring to be *specified* by the construction. Additionally, their model does not currently include random oracles, and hence it is only applicable to information-theoretic constructions or computational assumptions that can be obtained from the algebraic structure in a black-box way. The model is not equipped to consider standard assumptions like the existence of pseudorandom functions or collision-resistant hash functions.

Linear Garbling. In this work we study Linicrypt programs in the context of garbled circuit constructions. This is inspired in part by the lower bound of Zahur *et al.* [ZRE15]. They too observe that practical garbled circuit constructions consist of only linear operations and calls to a random oracle. They prove a lower bound, namely, that such "linear garbling schemes" require 2 field elements to garble a single AND gate.

In concurrent and independent work, Pastro *et al.* [MPs16] extend the model of linear garbling and characterize security in terms of linear-algebraic properties like span. They generalize the garbling scheme of [ZRE15] to natively support low-degree polynomials (not just AND-gates).

Later in Section 3 we go into more detail about the ZRE lower bound in the context of Linicrypt. For now, we simply point out the main differences between our work and the two above: (1) in this work we present a full theory of Linicrypt, not constrained only to garbled circuits; (2) the above models of linear garbling only consider "Linicrypt programs" that make non-adaptive calls to the random oracle, whereas our general Linicrypt model has no such restriction (arguably, the ability to reason about arbitrary oracle queries is the most important feature of Linicrypt). The difference is important specifically in the context of garbled circuits since, in most schemes, adaptive oracle queries result when composing several gates together in a larger circuit.

Synthesis of cryptographic constructions. Synthesis has been effectively used in the generic group model to discover batching schemes for signature verification [AGHP12] and optimal structure-preserving signatures [BFF$^+$15]. Both of these results synthesize constructions involving bilinear pairings.

Malozemoff *et al.* [MKG14] synthesized IND-CPA secure block cipher modes by expressing the main loop of a mode as a directed graph. They defined typing rules for the vertices of this graph and showed that if a valid assignment of types exists, then the resulting scheme is secure.

Using a SAT solver, they were able to check for valid type assignments for candidate modes and subsequently enumerate secure modes. In a followup work, Hoang *et al.* [HKM15] extended the synthesis to authenticated encryption modes built from tweakable block ciphers.

Prior work of Gagné *et al.* [GLLS09, GLLS11] developed techniques for automated proofs of security for (CPA-secure) block cipher modes. Akinyele *et al.* [AGH13] use an SMT solver to automate transformations of pairing-based signature schemes.

In all of the works involving block cipher modes [GLLS09, GLLS11, MKG14, HKM15] the techniques are developed for modes involving just XOR operations and [tweakable] block cipher calls. This corresponds to a natural special case of Linicrypt. We emphasize, however, that in these works the methods are sound but not complete.[1]

2 Linicrypt

2.1 Basic Model

A **pure Linicrypt program** over field $\mathbb{F}$ is a tuple $\mathcal{P} = (\mathsf{in}, \mathsf{out}, \mathsf{cmds})$, where: in is a nonnegative integer, out is an ordered sequence of indices from $\{1, \dots, |\mathsf{cmds}|\}$, and cmds is an ordered sequence of **Linicrypt commands**. The ith command in cmds must have one of the following forms:

- $(\textsc{inp}, j)$, where $1 \le j \le \mathsf{in}$ [retrieve a value from input]
- $(\textsc{samp})$ [sample an element of $\mathbb{F}$]
- $(\textsc{lin}, c_1, \dots, c_{i-1})$, where each $c_j \in \mathbb{F}$ [perform a linear combination of values]
- $(\textsc{hash}, t, j_1, \dots, j_k)$, where $t \in \{0,1\}^*$ and $j_1, \dots, j_k < i$ [call the random oracle on a set of variables, and additional (fixed) string t]

Intuitively, the program $\mathcal{P}$ takes as input a vector from $\mathbb{F}^{\mathsf{in}}$, then performs the operations specified by cmds. Each of the internal values of $\mathcal{P}$ is assigned to a variable $v[i]$. Finally, the program outputs the values whose indices are in the set out. More formally, we define the behavior of $\mathcal{P}$ as a process via:

```
$\underline{\mathcal{P}^H(\vec{x} \in \mathbb{F}^{\mathsf{in}})}$:
  for $i = 1$ to $|\mathsf{cmds}|$:
    if $\mathsf{cmds}[i] = (\textsc{inp}, j)$:                $v[i] := \vec{x}[j]$
    if $\mathsf{cmds}[i] = (\textsc{samp})$:                  $v[i] \xleftarrow{\$} \mathbb{F}$
    if $\mathsf{cmds}[i] = (\textsc{lin}, c_1, \dots, c_{i-1})$:  $v[i] := \sum c_j v[j]$
    if $\mathsf{cmds}[i] = (\textsc{hash}, t, j_1, \dots, j_k)$: $v[i] := H(t; v[j_1], \dots, v[j_k])$
  return $\big(v[j]\big)_{j \in \mathsf{out}}$
```

Note that H is an oracle with type $H : \{0,1\}^* \times \mathbb{F}^* \to \mathbb{F}$. In informal discussions, we often omit the first argument to H when it is an empty string.

[1] In [MKG14] the authors explicitly say, "we prevent a random value from both being output as ciphertext and input into a PRF ... This does not mean there do not exist secure schemes which have this property; however, our tool does not allow such schemes." In [GLLS09, GLLS11] the techniques involve a logic that uses only local invariants.

```
sign(sk, x)
    σ := sk
    for i = 1 to x:
        σ := H(σ)
    return σ
```

$\Longrightarrow$

```
sign(sk, x)
    if x = 1:          v_1 := H(sk)
                       return v_1
    else if x = 2:     v_1 := H(sk)
                       v_2 := H(v_1)
                       return v_2
    else if ···
```

Figure 1: The signing algorithm for one-time Winternitz signatures as a *mixed Linicrypt program.* Each inner box on the right-hand side is a *pure* Linicrypt programs, $\mathsf{sign}(\cdot, x)$, for fixed x.

2.2 Mixed Linicrypt Programs & Modelling Real-World Primitives

Most of the cryptographic primitives listed in the introduction cannot actually be implemented strictly as pure Linicrypt programs. For example, consider the one-time Winternitz signature of a single "digit" $x \in [m]$. The secret key $sk \leftarrow \mathbb{F}$ is chosen uniformly. The public key is then $pk := H^{(m)}(sk)$. To sign x, release $\sigma := H^{(x)}(sk)$. Then to verify, check $pk \overset{?}{=} H^{(m-x)}(\sigma)$.

The main operations in Winternitz are simply repeated calls to the hash/one-way function H, which are certainly allowed in Linicrypt. However, the signing algorithm *uses x in a non-linear way* — to choose how many Linicrypt commands to execute!

We therefore extend the scope of Linicrypt beyond *pure* Linicrypt programs. A **mixed Linicrypt program** is one in which we designate some inputs to be *non-linear* and the others to be linear. For instance, in the signing algorithm of Winternitz signatures there is a for-loop whose exit condition is non-linear in x.

We can associate any *mixed* Linicrypt program with a collection of *pure* Linicrypt programs. Think of any *mixed* Linicrypt program as a switch/case statement (based on its non-linear input) selecting which *pure* Linicrypt program to run. See Figure 2.2 for the example of Winternitz signatures. Each $\mathsf{sign}(\cdot, x)$ is a pure Linicrypt program. Since x is public in the security definition for signatures, we can express the security of the (mixed) signing algorithm in terms of the properties of each (pure) program $\mathsf{sign}(\cdot, x)$.

The way one decides to model some inputs as non-linear and other inputs as linear is highly *application-specific.* In general, it makes the most sense to let the length of non-linear inputs to be a *constant* c: First, the complexity of deciding security and synthesizing constructions grows exponentially with c. Second, this implies that all of the security properties are a result of the Linicrypt operations (the random oracle and linear operations over a field $\mathbb{F}$, whose size is exponential in the security parameter) and not the non-linear behavior. In other words, in a security game an adversary could guess with constant probability the non-linear input, leaving a residual *pure* Linicrypt program. So security is reduced to the security properties of the individual pure Linicrypt programs in the collection.

Throughout the rest of this section we develop a general theory of Linicrypt, and restrict our attention to *pure* Linicrypt programs. Later when discussing specific applications of Linicrypt to garbled circuits, we explicitly discuss *mixed* Linicrypt programs and non-linear inputs, etc.

2.3 Algebraic Representation

Let $\mathcal{P}$ be a (pure) Linicrypt program with notation as above. Say that $v[i]$ is a **derived** variable if $\mathsf{cmds}[i]$ is of the form $(\textsc{lin}, \cdots)$. Otherwise say that $v[i]$ is a **base** variable. That is, a base variable is the result of a command with one of $\textsc{samp}$, $\textsc{hash}$, or $\textsc{inp}$. Let base denote the number of base variables. The main idea behind manipulating Linicrypt programs in an algebraic way is to observe that all values of importance can be expressed as linear functions of the *base* variables.

In more detail, fix an ordering of the base variables and denote them by the vector $\boldsymbol{v}_{\mathsf{base}}$. Then for the ith command in cmds, define $\mathsf{row}(i)$ to be the vector in $\mathbb{F}^{\mathsf{base}}$ such that $v[i] = \mathsf{row}(i) \cdot \boldsymbol{v}_{\mathsf{base}}$, where the $\cdot$ denotes dot product of vectors. More formally:

$$\mathsf{row}(i) \overset{\mathsf{def}}{=} \begin{cases} [\overbrace{0\ 0 \cdots\ 0}^{j-1}\ 1\ 0 \cdots 0] & \text{if } v[i] \text{ is the } j\text{th base variable} \\ \sum_j c_j \mathsf{row}(j) & \text{if } \mathsf{cmds}[i] = (\textsc{lin}, c_1, \ldots, c_{i-1}) \end{cases}.$$

We create a matrix to represent the output of a Linicrypt program:

$$\mathcal{M} \overset{\mathsf{def}}{=} \begin{bmatrix} \text{—}\ \mathsf{row}(o_1)\ \text{—} \\ \vdots \\ \text{—}\ \mathsf{row}(o_k)\ \text{—} \end{bmatrix}, \qquad \text{where } \mathsf{out} = (o_1, \ldots, o_k).$$

$\mathcal{M}$ therefore characterizes the *direct* correlations among the program's output variables. Yet, it contains no information about how these variables may be *correlated via the random oracle!* So, our characterization of a Linicrypt program includes a set of **oracle constraints**. The idea behind an oracle constraint $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle$ is that if the random oracle is called on input $(t; \mathcal{Q} \times \boldsymbol{v}_{\mathsf{base}})$ then the response will be $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$.

$$\mathcal{C} \overset{\mathsf{def}}{=} \left\{ \left\langle t, \begin{bmatrix} \text{—}\ \mathsf{row}(j_1)\ \text{—} \\ \vdots \\ \text{—}\ \mathsf{row}(j_k)\ \text{—} \end{bmatrix}, \mathsf{row}(i) \right\rangle \;\middle|\; \mathsf{cmds}[i] = (\textsc{hash}, t, j_1, \ldots, j_k) \right\}$$

Without loss of generality, we can assume that no two constraints share $(t, \mathcal{Q})$ in common. Under that restriction, the set $\{\boldsymbol{a} \mid \langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}\}$ is a linearly independent set — *i.e.*, the results of distinct random oracle queries are linearly independent.

Finally, we define the **algebraic representation** of a Linicrypt program $\mathcal{P}$ to be $(\mathcal{M}, \mathcal{C})$. We refer to $\mathcal{M}$ as the **output matrix** and $\mathcal{C}$ as the set of **oracle constraints.**

To demonstrate the different ways of viewing a Linicrypt program, consider the following example, with $\mathsf{in} = 0$:

plain-language:	Linicrypt cmds:	var type:
$v_1 \leftarrow \mathbb{F}$	1: $(\textsc{samp})$	base
$v_2 \leftarrow \mathbb{F}$	2: $(\textsc{samp})$	base
$v_3 := v_1 - v_2$	3: $(\textsc{lin}, 1, -1)$	derived
$v_4 := H(\texttt{foo}, v_3, v_2)$	4: $(\textsc{hash}, \texttt{foo}, 3, 2)$	base
$v_5 := v_4 + v_1$	5: $(\textsc{lin}, 1, 0, 0, 1)$	derived
return (v_4, v_5)	// $\mathsf{out} = (4, 5)$	

matrix representation:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \end{bmatrix}$$

algebraic representation:

$$\mathcal{M} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}; \quad \mathcal{C} = \left\{ \left\langle \texttt{foo}, \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, [0\ 0\ 1] \right\rangle \right\}$$

There are three base variables. With v_4, v_5 being output variables, the output matrix $\mathcal{M}$ consists of $\mathsf{row}(4), \mathsf{row}(5)$. There is one HASH-command "$v_4 := H(\mathtt{foo}, v_3, v_2)$," leading to a single oracle constraint $\langle \mathtt{foo}, \begin{bmatrix} \mathsf{row}(3) \\ \mathsf{row}(2) \end{bmatrix}, \mathsf{row}(4) \rangle$.

In the rest of this paper, we specialize to input-less (*i.e.*, $\mathsf{in} = 0$) Linicrypt programs. Restricting our domain to input-less programs simplifies the definitions & proofs. This is justified by our main application to garbled circuits. In the security definition for garbled circuits, the adversary chooses an input x to the function, but since we model x as non-linear input, what is left over is a collection of security experiments, one for each x, each involving an input-less (pure) Linicrypt program.

We hereafter overload notation and write $\mathcal{P} = (\mathcal{M}, \mathcal{C})$. We claim that $(\mathcal{M}, \mathcal{C})$ completely characterizes the behavior of $\mathcal{P}$. In more detail, let $\mathcal{P}$ be an input-less Linicrypt program, let $\mathcal{A}$ be an oracle machine, and consider the following **canonical simulation** of $\mathcal{P}$.

```
$\underline{\mathcal{S}_{\mathcal{P}}^{\mathcal{A}}():}$
1. $\boldsymbol{v}_{\mathsf{base}} \xleftarrow{\$} \mathbb{F}^{\mathsf{base}}$
2. $\boldsymbol{v}_{\mathsf{out}} := \mathcal{M}\, \boldsymbol{v}_{\mathsf{base}}$
3. $cache$ := empty associative array
4. return $\mathcal{A}^H(\boldsymbol{v}_{\mathsf{out}})$, where $H$ implemented as below:

$\underline{H(t; \boldsymbol{q} \in \mathbb{F}^*):}$
// if the adversary found a collision among oracle constraints
5. if $\exists \langle t, \mathcal{Q}, \boldsymbol{a} \rangle, \langle t, \mathcal{Q}', \boldsymbol{a}' \rangle \in \mathcal{C}$ with $\boldsymbol{a} \neq \boldsymbol{a}'$ and $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}} = \mathcal{Q}'\boldsymbol{v}_{\mathsf{base}} = \boldsymbol{q}$:      (1)
6.    abort
// if there is an oracle constraint for the query $\boldsymbol{q}$
7. if $\exists \langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}$ with $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}} = \boldsymbol{q}$:
8.    return $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$
// honest simulation of a random oracle beyond this point
9. if $cache[t; \boldsymbol{q}]$ does not exist:
10.    $cache[t; \boldsymbol{q}] \xleftarrow{\$} \mathbb{F}$
11.return $cache[t; \boldsymbol{q}]$
```

The idea is to simply sample *all* of the base variables upfront, instead of deriving some of them via calls to the random oracle. But then to make the simulation of the random oracle consistent, we "patch" the random oracle so that when queried on $(t, \mathcal{Q}\boldsymbol{v}_{\mathsf{base}})$, the consistent result $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$ is simulated (lines 7-8). The simulation aborts when two oracle constraints are in conflict (lines 5-6).

Lemma 1 (Canonical simulation)**.** *Let $\mathcal{P}$ be an input-less (i.e., $\mathsf{in} = 0$) Linicrypt program that executes n HASH-commands. Then for all oracle machines $\mathcal{A}$:*

$$\Pr\left[\mathcal{S}_{\mathcal{P}}^{\mathcal{A}}() = 1\right] - \Pr_H\left[\mathcal{A}^H(\mathcal{P}^H()) = 1\right] \leq \frac{n(n+1)}{2|\mathbb{F}|}.$$

We emphasize that $\mathcal{A}$ here is an arbitrary program. It need not be linear, it may be computationally unbounded, and (at least for this lemma) it is even unrestricted in the number of oracle queries it makes.

Sketch. Conditioned on the simulation not aborting in line 6, the simulation is perfect. Essentially, each query to H answered in lines 7-8 is answered with a randomly chosen base variable (since each $\boldsymbol{a}$ is a canonical basis vector), exactly matching how queries are answered by an honest random oracle. Hence, the error in the simulation is the probability that the condition in line 5 is true. This happens if $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}} = \mathcal{Q}'\boldsymbol{v}_{\mathsf{base}}$ for some distinct constraints $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle, \langle t, \mathcal{Q}', \boldsymbol{a}'\rangle \in \mathcal{C}$. Since WLOG no two constraints share $(t, \mathcal{Q})$, we have that $\mathcal{Q} - \mathcal{Q}'$ is a nonzero matrix, and therefore that

$$\mathcal{Q}\boldsymbol{v}_{\mathsf{base}} = \mathcal{Q}'\boldsymbol{v}_{\mathsf{base}} \iff (\mathcal{Q} - \mathcal{Q}')\boldsymbol{v}_{\mathsf{base}} = 0 \iff \boldsymbol{v}_{\mathsf{base}} \in \mathsf{kernel}(\mathcal{Q} - \mathcal{Q}').$$

Note that $\mathsf{kernel}(\mathcal{Q}-\mathcal{Q}')$ is a proper subspace of $\mathbb{F}^{\mathsf{base}}$ with maximum dimension $(\mathsf{base}-1)$. Then, when $\boldsymbol{v}_{\mathsf{base}}$ is chosen uniformly from $\mathbb{F}^{\mathsf{base}}$, the probability that it is in a particular proper subspace is at most $|\mathbb{F}|^{\mathsf{base}-1}/|\mathbb{F}|^{\mathsf{base}} = 1/|\mathbb{F}|$. Recall that $\mathcal{P}$ executes n HASH-commands. Then there are $\binom{n}{2} = n(n+1)/2$ possible pairs of distinct oracle constraints. By the union bound, the probability that there exist some pair of oracle constraints with $\mathcal{Q}$ and $\mathcal{Q}'$ for which $\boldsymbol{v}_{\mathsf{base}} \in \mathsf{kernel}(\mathcal{Q} - \mathcal{Q}')$ is at most $n(n+1)/2|\mathbb{F}|$. $\square$

2.4 Basis Changes & Composition

The algebraic representation for Linicrypt programs turns out to be convenient, as we can reason about algebraic representations that are not directly derived from a Linicrypt program. In particular, the algebraic representation allows us to *compose* Linicrypt programs. To do so, we must align the base variables and oracle constraints with a *basis change*.

In particular, let $\mathcal{P} = (\mathcal{M}, \mathcal{C})$ be a Linicrypt program as above. Recall that the width of the vectors in $\mathcal{M}$ and $\mathcal{C}$ is base. Now let B be a $\mathsf{base} \times \mathsf{base}$ invertible matrix with entries in $\mathbb{F}$ and consider the Linicrypt representation $(\mathcal{M}B, \mathcal{C}B)$, where

$$\mathcal{C}B \overset{\mathsf{def}}{=} \{\langle t, \mathcal{Q}B, \boldsymbol{a}B\rangle \mid \langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}\}.$$

We refer to $(\mathcal{M}B, \mathcal{C}B)$ as a **basis change** of B applied to $(\mathcal{M}, \mathcal{C})$. Such a basis change has no effect on the output distribution of the Linicrypt program. More precisely:

Proposition 2. *Let $\mathcal{P} = (\mathcal{M}, \mathcal{C})$ be an input-less Linicrypt program, and let $\mathcal{P}' = (\mathcal{M}B, \mathcal{C}B)$ for some invertible matrix B. Then for all oracle machines $\mathcal{A}$, we have:*

$$\Pr\left[\mathcal{S}^{\mathcal{A}}_{\mathcal{P}}() = 1\right] = \Pr\left[\mathcal{S}^{\mathcal{A}}_{\mathcal{P}'}() = 1\right].$$

Proof. A basis change by B is equivalent to adding a statement "$\boldsymbol{v}_{\mathsf{base}} := B\boldsymbol{v}_{\mathsf{base}}$" between lines 1 & 2 in Equation 1. Since B is invertible, this additional statement has no effect on the distribution of $\boldsymbol{v}_{\mathsf{base}}$. $\square$

Composition. We can use the idea of a basis change to reason algebraically about the composition of two Linicrypt programs. Let $\mathcal{P} = (\mathcal{M}, \mathcal{C})$ be a Linicrypt program with no input and out outputs, and let $\mathcal{P}' = (\mathcal{M}', \mathcal{C}')$ be a Linicrypt program with out inputs, so that it makes sense to feed the output of $\mathcal{P}$ as input to $\mathcal{P}'$. Without loss of generality, we make the following assumptions:

- Both programs have the same number of base variables (so that $\mathcal{M}$, $\mathcal{M}'$ have the same number of columns and so on).
- The first out base variables of $\mathcal{P}'$ are identified with its input variables.
- The output matrix $\mathcal{M}$ of $\mathcal{P}$ is full rank. If this is not the case, then modify $\mathcal{P}$ to output a basis for the rowspace of $\mathcal{M}$; then modify $\mathcal{P}'$ to internally reconstruct the original output of $\mathcal{P}$ by taking linear combinations of the input.

Now suppose there is a basis change B with the following properties:

1. $\mathcal{M}B = [I \mid \mathbf{0}]$ where I is the $\mathsf{out} \times \mathsf{out}$ identity matrix.
2. For every $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}$, if there exists $\langle t, \mathcal{Q}B, \boldsymbol{a}'\rangle \in \mathcal{C}'$, then $\boldsymbol{a}' = \boldsymbol{a}B$.

Then $(\mathcal{M}', \mathcal{C}B \cup \mathcal{C}')$ is a valid algebraic representation for the composition $\mathcal{P}' \circ \mathcal{P}$.

Intuitively, the idea is that $\mathcal{P}$ and $\mathcal{P}'$ are expressed each in their own "native basis" which need not be related. The property required of B is that it translates the program $\mathcal{P}$ into the native basis of $\mathcal{P}'$. The first condition above is that the output of $\mathcal{P}$ (represented by the matrix $\mathcal{M}$) is mapped to the input of $\mathcal{P}'$. In the native basis of $\mathcal{P}'$, its inputs are represented by the rows of $[I \mid \mathbf{0}]$. The second condition is that B does not introduce any disagreeing oracle constraints between the two programs. Finally, the output of the composed program is the output of $\mathcal{P}'$ which is represented by $\mathcal{M}'$.

2.5 Indistinguishability vs. Unpredictability

When we consider Linicrypt programs that implement cryptographic primitives, the most fundamental question is: when do two Linicrypt programs induce indistinguishable distributions (in the random oracle model)?

Definition 3. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two input-less Linicrypt programs over $\mathbb{F}$. Let $\lambda = \log|\mathbb{F}|$ be the security parameter. We say that $\mathcal{P}_1$ and $\mathcal{P}_2$ are* ***indistinguishable****, and write $\mathcal{P}_1 \cong \mathcal{P}_2$, if for every (possibly computationally unbounded) oracle machine $\mathcal{A}$ that queries its oracle a polynomial (in λ) number of times, we have*

$$\Pr[\mathcal{A}^H(\mathcal{P}_1^H()) = 1] - \Pr[\mathcal{A}^H(\mathcal{P}_2^H()) = 1] \text{ is negligible in } \lambda.$$

The probabilities are over the choice of random oracle H and the coins of $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{A}$.

We point out that *indistinguishability can be used to reason about unforgeability properties* as well. Suppose $\mathcal{P}$ is a Linicrypt program that has some special internal variable v^*, and we wish to formalize the idea that "v^* is hard to predict (in the random oracle model) given the output of $\mathcal{P}$." Now define the following two related programs:

- $\mathcal{P}_1$ runs $\mathcal{P}$ and outputs whatever $\mathcal{P}$ outputs, along with an additional output $v_{\mathsf{extra}} = H(t^*; v^*)$, where t^* is a "tweak" that is not used in $\mathcal{P}$.
- $\mathcal{P}_2$ runs $\mathcal{P}$ and outputs whatever $\mathcal{P}$ outputs, along with an additional output $v_{\mathsf{extra}} \xleftarrow{\$} \mathbb{F}$.

Note that $\mathcal{P}_1$ and $\mathcal{P}_2$ are a Linicrypt programs if $\mathcal{P}$ is. Now observe that the following statements are equivalent:

1. Given the output of $\mathcal{P}$, the probability that an adversary (with access to the random oracle) outputs v^* is negligible.
2. Given the output of $\mathcal{P}$, the probability that an adversary queries the random oracle on $H(t^*; v^*)$ is negligible.
3. Given the output of $\mathcal{P}$, the value $H(t^*; v^*)$ is indistinguishable from uniform. This follows simply from the definition of the random oracle model, and the fact that $\mathcal{P}$ itself does not use any values of the form $H(t^*; \cdot)$.

4. $\mathcal{P}_1 \cong \mathcal{P}_2$.

Hence, standard unforgeability properties of a Linicrypt program can be expressed as the indistinguishability of two Linicrypt programs. From now on, we therefore focus on indistinguishability only. And indeed, our main characterization theorem will include reasoning like that above, regarding which oracle queries can be made by an adversary with non-negligible probability.

2.6 Normalization

We now describe a procedure for "normalizing" a Linicrypt program. Specifically, normalizing corresponds to removing "unnecessary" calls to the oracle. We illustrate the ideas with a brief example, below:

plain language:	Linicrypt cmds:	matrix representation:
$v_1 \xleftarrow{\$} \mathbb{F}$	1: (SAMP)	$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_5 \end{bmatrix}$
$v_2 := H(\texttt{foo}, v_1)$	2: $(\text{HASH}, \texttt{foo}, 1)$	
$v_3 := v_1 - v_2$	3: $(\text{LIN}, 1, -1)$	
$v_4 := H(\texttt{bar}, v_3)$	4: $(\text{HASH}, \texttt{bar}, 3)$	
$v_5 := H(\texttt{baz}, v_3)$	5: $(\text{HASH}, \texttt{baz}, 3)$	
output (v_3, v_5)		

This program has 3 oracle queries, two of which are "unnecessary" in some sense.

- It is instructive to consider what information the adversary can collect about the base variables $\boldsymbol{v}_{\mathsf{base}}$. From the output of $\mathcal{P}$, one obtains $v_3 = [1\ {-1}\ 0\ 0] \cdot \boldsymbol{v}_{\mathsf{base}}$ and $v_5 = [0\ 0\ 0\ 1] \cdot \boldsymbol{v}_{\mathsf{base}}$. Then one can call the oracle as $H(\texttt{bar}, v_3)$ to obtain $v_4 = [0\ 0\ 1\ 0] \cdot \boldsymbol{v}_{\mathsf{base}}$. However, it is hard to predict $v_1 = [1\ 0\ 0\ 0] \cdot \boldsymbol{v}_{\mathsf{base}}$ given just the output of $\mathcal{P}$. More specifically, $[1\ 0\ 0\ 0]$ is not in the span of $\{[1\ {-1}\ 0\ 0], [0\ 0\ 1\ 0], [0\ 0\ 0\ 1]\}$.

  In other words, the probability of an adversary querying H on v_1 is negligible, so we call this oracle query **unreachable**. Conditioned on the adversary not querying H on v_1, its output $v_2 = H(\texttt{foo}, v_1)$ looks uniformly random. Removing the corresponding oracle constraint therefore has negligible effect. Note that removing the oracle constraint corresponds to replacing "$v_2 := H(\texttt{foo}, v_1)$" with "$v_2 \xleftarrow{\$} \mathbb{F}$"; *i.e.*, changing $\mathsf{cmds}[2]$ from $(\text{HASH}, \texttt{foo}, 1)$ to (SAMP).

- Oracle query $H(\texttt{bar}, v_3)$ is reachable, since the output of $\mathcal{P}$ includes v_3. However, its result is v_4 which is not used anywhere else in the program. This can be seen by observing that all other row vectors in the algebraic representation have a zero in the position corresponding to v_4. Hence this oracle call can be replaced with "$v_4 \xleftarrow{\$} \mathbb{F}$" with no effect on the adversary. We call this query *useless.*

- Oracle query $H(\texttt{baz}, v_3)$ is similarly reachable, but it is *useful.* The result of this query is $H(\texttt{baz}, v_3) = v_5$ which is included in the output of $\mathcal{P}$ and hence visible to the adversary. It cannot be removed because an adversary could query $H(\texttt{baz}, v_3)$ and check that it matches v_5 from the output.

More generally, we normalize a Linicrypt program by computing which oracle queries / constraints are *reachable* and which are *useless* in the above sense.

To compute which oracle queries are reachable, we perform the following procedure until it reaches a fixed point: Given Linicrypt program $\mathcal{P} = (\mathcal{M}, \mathcal{C})$, mark the rows of $\mathcal{M}$ as *reachable.*

```
$\underline{\mathsf{normalize}(\mathcal{P} = (\mathcal{M}, \mathcal{C}))\text{:}}$
  $\mathsf{Reachable} := \mathsf{rows}(\mathcal{M})$
  $\mathcal{C}' := \emptyset$
  until $\mathcal{C}'$ reaches a fixed point:
    for each $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C} \setminus \mathcal{C}'$:
      if $\mathsf{rows}(\mathcal{Q}) \subseteq \mathsf{span}(\mathsf{Reachable})$:
        add $\boldsymbol{a}$ to $\mathsf{Reachable}$
        add $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ to $\mathcal{C}'$

  $\mathsf{Useless} := \emptyset$
  until $\mathsf{Useless}$ reaches a fixed point:
    $V$ := (multiset of) all row vectors in $\mathcal{M}$ and $\mathcal{C}' \setminus \mathsf{Useless}$
    for each $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}' \setminus \mathsf{Useless}$:
      if $\boldsymbol{a} \notin \mathsf{span}(V \setminus \{\boldsymbol{a}\})$:
        add $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ to $\mathsf{Useless}$

  $\mathcal{C}'' := \mathcal{C}' \setminus \mathsf{Useless}$

  return $(\mathcal{M}, \mathcal{C}'')$
```

Figure 2: Procedure to normalize a Linicrypt program. Since V is a multiset, we clarify that "$V \setminus \{\boldsymbol{a}\}$" means to decrease the multiplicity of $\boldsymbol{a}$ in multiset V by only one. So $V \setminus \{\boldsymbol{a}\}$ may yet include $\boldsymbol{a}$. One reason for $\boldsymbol{a}$ to have high multiplicity in V is if $\boldsymbol{a}$ appears both in an oracle constraint and as a row of $\mathcal{M}$.

Then, if any oracle constraint $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}$ has every row of $\mathcal{Q}$ in the span of reachable vectors, then mark $\boldsymbol{a}$ as *reachable*.

Instead of computing which queries are useful, it is more straight-forward to compute which queries are *useless*, one by one. Intuitively, a constraint $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ is *useless* if $\boldsymbol{a}$ is linearly independent of all other vectors appearing in $\mathcal{M}$ and $\mathcal{C}'$ (either as rows of $\mathcal{M}$ or rows of some $\mathcal{Q}'$ or as an $\boldsymbol{a}'$). After removing one useless constraint, other constraints might become useless. For instance, consider a Linicrypt program that outputs v but also internally computes $H(H(H(v)))$. Only the outermost call to H is initially useless. After it is removed, the "new" outermost call is marked useless, and so on, until a fixed point is reached.

The details of the normalize procedure are given in Figure 2. In Appendix A.1 we prove the following:

Lemma 4. *If $\mathcal{P}$ is an input-less Linicrypt program, then* $\mathsf{normalize}(\mathcal{P}) \cong \mathcal{P}$ *(Figure 2).*

2.7 Main Characterization

We can now present our main technical theorem about Linicrypt programs:

Theorem 5 (Linicrypt Characterization)**.** *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two input-less Linicrypt programs over $\mathbb{F}$. Then $\mathcal{P}_1 \cong \mathcal{P}_2$ if and only if* $\mathsf{normalize}(\mathcal{P}_1)$ *and* $\mathsf{normalize}(\mathcal{P}_2)$ *differ by a basis change.*

Proof sketch. The full proof is given in Appendix A.2. The nontrivial case is to show the $\Rightarrow$ direction. Without loss of generality assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ are normalized, and suppose they do not differ by a basis change. The idea is to first construct a "profile" for $\mathcal{P}_1$ and for $\mathcal{P}_2$. In the code of normalize, we compute the reachable subspace of a program; the *profile* simply refers to the order in which reachable oracle constraints are activated during this process.

We use the profile to construct a family of *canonical distinguishers* for $\mathcal{P}_1$. It processes oracle constraints in the order determined by the profile. It maintains the invariant that at all stages of the computation, if $\mathcal{R}$ is the set of currently reachable vectors, the distinguisher holds $\vec{r} = \mathcal{R} \times \boldsymbol{v}_{\mathsf{base}}$, where $\boldsymbol{v}_{\mathsf{base}}$ refers to the base variables in the canonical simulation of $\mathcal{P}_1$.

A side-effect of normalization is that all oracle constraints are reachable and useful. Because of this, the set of reachable vectors will eventually contain non-trivial linear relations — as a matrix, the set of reachable vectors has a nontrivial kernel. A canonical distinguisher chooses some element $\vec{z}$ from this kernel and tests whether $\vec{z}^\top \vec{r} = 0$. By construction, $\vec{z}^\top \vec{r} = \vec{z}^\top \mathcal{R} \boldsymbol{v}_{\mathsf{base}}$. Since $\vec{z} \in \ker(\mathcal{R})$, the distinguisher always outputs true in the presence of $\mathcal{P}_1$.

Now the challenge is to show that, for some choice of $\vec{z} \in \ker(\mathcal{R})$, the distinguisher outputs false with overwhelming probability in the presence of $\mathcal{P}_2$. To see why, we consider the first point at which the profiles of $\mathcal{P}_1$ and $\mathcal{P}_2$ disagree (if the profiles agree fully, then it is easy to obtain a basis change relating $\mathcal{P}_1$ to $\mathcal{P}_2$). The most nontrivial case is when $\mathcal{P}_1$ contains an oracle constraint that no basis change can bring into alignment with $\mathcal{P}_2$. This implies that when the distinguisher makes the query in the presence of $\mathcal{P}_2$, it will not trigger any oracle constraint and the result will be random and independent of everything else in the system. But because this oracle constraint was useful in $\mathcal{P}_1$, we can eventually choose a final kernel-test $\vec{z}$ that is "sensitive" to the result in the following way: While in $\mathcal{P}_1$, the kernel-test always results in zero, in $\mathcal{P}_2$ the kernel test will be independently random.

The actual proof is considerably more involved concerning the different cases for why the profiles of $\mathcal{P}_1$ and $\mathcal{P}_2$ disagree. $\square$

Also in Appendix A.2 we discuss how indistinguishability of two Linicrypt programs can be decided in polynomial time.

3 Synthesizing Linicrypt Garbled Circuits

In this section we describe how to express the security of garbled circuits in the language of Linicrypt, culminating in a method to leverage an SMT solver to automatically synthesize secure schemes. We assume some familiarity with the classical (textbook) Yao garbling scheme. Roughly speaking, each wire in the circuit is associated with two *labels* (bitstrings) W^0 and W^1, encoding FALSE and TRUE, respectively. The evaluator will learn exactly one of these two labels for each wire. Then, for each gate in the circuit, the evaluator uses the labels for the input wires, along with *garbled gate* information (classically, the garbled truth table), to compute the appropriate label on the output wire. We restrict our synthesis technique to the context of two basic garbled circuit techniques: *Free-XOR* and *Point-and-Permute*.

Free-XOR. In the Free-XOR garbling technique of Kolesnikov and Schneider [KS08], the garbler chooses a random Δ that is global, and arranges for $W^0 \oplus W^1 = \Delta$ on every wire. Hereafter, we typically write the FALSE label simply as W and the TRUE wire label as $W \oplus \Delta$; more generally, the wire label encoding b is $W \oplus b\Delta$.

Using Free-XOR, no ciphertexts are necessary to garble an XOR gate. For instance, let A and B be the FALSE input wirelabels. Set the FALSE output wirelabel to $C = A \oplus B$. Then when the evaluator holds wirelabels $A^* = A \oplus a\Delta$ and $B^* = B \oplus b\Delta$ (encoding a and b, respectively), she can compute $A^* \oplus B^* = A \oplus a\Delta \oplus B \oplus b\Delta = C \oplus (a \oplus b)\Delta$. That is, the result will be the wire label correctly encoding truth value $a \oplus b$. We note that no garbled gate information is required in the garbled circuit, nor must the evaluator perform any cryptographic operations to evaluate the gate — just an XOR of strings.

Free-XOR is ubiquitous in practical implementations of garbled circuits. For that reason (and because it conveniently reduces degrees of freedom over choice of wire labels), we restrict our attention to garbling schemes that are compatible with Free-XOR.

Point-and-permute and Non-Linearity The *point-and-permute* optimization of [BMR90] is used in all practical garbling schemes. The idea is to append to each wire label a random bit χ (which we call the **"color bit"**). The two labels on each wire have opposite (but random) color bits.

Now consider the naive/classical garbling of an AND gate, in which the garbler generates 4 ciphertexts. Because color bits are independent of truth values, the garbler can arrange the ciphertexts in order of the color bits of the input wire labels. The evaluator selects and decrypts the correct ciphertext indicated by the color bits of the input wire labels she holds. Importantly, this makes the color bits *non-linear inputs* with respect to Linicrypt! The color bits determine which linear combination the evaluator will apply.

Similarly, the garbler's behavior is non-linear in a complementary way. We refer to σ as the **"select bit"** such that the wire label encoding truth value v has color $\chi = v \oplus \sigma$. Equivalently, σ is the (random) color bit of the FALSE wire. We emphasize that σ is known only to the garbler, and χ is known only to the evaluator, effectively hiding the truth value v. In typical garbling schemes, the garbler's behavior depends non-linearly on σ but is otherwise within the Linicrypt model.

We treat garbling schemes as mixed Linicrypt programs, as in Section 2.2. Then, a mixed Linicrypt garbling scheme is a collection of pure Linicrypt garbling programs indexed by color bits and select bits.

Restricting to Linicrypt with xor as the linear operation. Technically speaking, a Linicrypt program is an infinite family of programs, one for each value of the security parameter. Unfortunately, we can only synthesize an object of finite size. Hence we restrict our focus to *single* Linicrypt programs that are compatible with an infinite family of fields / security parameters, in the following way.

Suppose a Linicrypt program uses field $GF(p)$ for prime p. Then that Linicrypt program is also compatible with field $GF(p^\lambda)$ for any λ, since $GF(p) \subseteq GF(p^\lambda)$ in a natural way. A very natural special case is $p = 2$, which corresponds to Linicrypt programs that use $GF(2^\lambda)$ and use only linear combinations with coefficients from $\{0, 1\}$ — in other words, Linicrypt programs that are restricted to using XOR as their only linear operation. Hereafter we restrict our attention to XOR-only Linicrypt programs.

3.1 Gate-garbling

A garbling scheme for an entire circuit is a non-trivially large object — much too large to synthesize using a SAT/SMT solver. We instead focus on techniques for *garbling individual gates* in a way that allows them to be securely composed with other gates and the Free-XOR technique to yield a garbling scheme for arbitrary circuits.

Notation. A wirelabel that carries the truth-value FALSE is always signified W, a wirelabel that carries TRUE is always $W \oplus \Delta$, and a wirelabel carrying unknown truth-value is always W^*. We collect wirelabels into vectors notated as follows: $\vec{W} = W_1, \ldots, W_n$. Operations over vectors are computed componentwise. For instance, $\vec{A} \oplus \vec{B} = A_1 \oplus B_1, \ldots, A_n \oplus B_n$. When $\Delta \in GF(2^\lambda)$ and x is a string of n bits, we write $x\Delta$ to mean the vector $x_1\Delta, \ldots, x_n\Delta$. For example, if $\vec{W} = W_1, \ldots, W_n$ are a vector of FALSE wire labels, then $\vec{W} \oplus x\Delta$ is a vector of wire labels encoding truth values x.

Syntax. Let $\tau : \{0,1\}^m \to \{0,1\}^n$ be the functionality of an m-ary boolean gate that we wish to garble. Let $\sigma = \sigma_1 \,||\, \ldots \,||\, \sigma_m$ be a string of select bits and $\chi = \chi_1 \,||\, \ldots \,||\, \chi_m$ be a string of color bits. Then, a **free-XOR compatible garbled gate** consists of algorithms:

$$\begin{aligned}&\mathsf{GateGb}(\sigma;\, A_1, \ldots, A_m, \Delta) \to (C_1, \ldots, C_n;\, G_1, \ldots, G_\ell)\\ &\mathsf{GateEv}(\chi;\, A_1^*, \ldots, A_m^*, G_1, \ldots, G_\ell) \to (C_1^*, \ldots, C_n^*)\end{aligned}$$

The semantics are as follows. GateGb takes m FALSE input wirelabels $\vec{A} = A_1, \ldots, A_m$, their select bits σ, and global constant Δ. It returns the n FALSE output wirelabels $\vec{C} = C_1, \ldots, C_m$, and garbled gate information $\vec{G} = G_1, \ldots, G_\ell$. The evaluator takes m input wirelabels with *unknown* truth values $\vec{A}^* = A_1^*, \ldots, A_m^*$, their color bits χ, and the garbled gate information $\vec{G}$. It returns output wirelabels with *unknown* truth values $\vec{C}^* = C_1^*, \ldots, C_n^*$.

We emphasize that when GateGb and GateEv are Linicrypt programs, all inputs and outputs besides σ and χ are field elements in $GF(2^\lambda)$.

Correctness. If a gate garbling scheme is correct, then the evaluator can always produce the correct output wirelabels according to τ. That is, when the evaluator holds wire labels encoding x on the input wires, the result of evaluating the gate is the wire labels encoding $\tau(x)$ on the output wires.

Definition 6. *A Free-XOR-compatible garbled gate (GateGb, GateEv) correctly computes gate functionality $\tau : \{0,1\}^m \to \{0,1\}^n$ if for all inputs $x \in \{0,1\}^m$, select bit strings $\sigma \in \{0,1\}^m$, and color bit string $\chi \in \{0,1\}^m$, with $x = \sigma \oplus \chi$, false input wirelabels $\vec{A} = A_1, \ldots, A_m$, global Free-XOR constant Δ, and gate information $(\vec{C}, \vec{G}) \leftarrow \mathsf{GateGb}(\sigma;\, \vec{A}, \Delta)$,*

$$\mathsf{GateEv}(\chi;\, \vec{A} \oplus x\Delta,\, \vec{G}) = \vec{C} \oplus \tau(x)\Delta$$

Security. We define security in terms of the evaluator's view in a typical garbling scenario: (here we explicitly write the random oracle used by GateGb and GateEv for clarity) We define $\mathsf{View}^H(\chi, x)$ to encapsulate the information the evaluator sees for this gate, when the visible color bits are χ and the actual gate inputs are x:

```
$\underline{\mathsf{View}^H(\chi, x)}$:
    $\Delta, A_1, \ldots, A_m \leftarrow \{0,1\}^\lambda$
    $(\vec{C}, \vec{G}) \leftarrow \mathsf{GateGb}^H(\chi \oplus x;\, \vec{A}, \Delta)$
    return $(\vec{A} \oplus x\Delta,\, \vec{G},\, \vec{C} \oplus \tau(x)\Delta)$
```

Importantly, if GateGb^H is a Linicrypt program and parameters χ and x are fixed, then $\mathsf{View}^H(\chi, x)$ is a input-less Linicrypt program. We can therefore apply the results of Section 2 to reason about the indistinguishability and unforgeability properties required of View^H. The fact that these properties can be expressed algebraically is the core of our synthesis technique.

We define the following security property for a Free-XOR compatible garbled gate scheme:

Definition 7. *A Free-XOR compatible garbled gate is **secure** if:*

1. for all χ, $x \in \{0,1\}^m$ and all polynomial-time oracle algorithms A,

$$\Pr[\, A^H(\mathsf{View}^H(\chi, x)) = \Delta \,] \text{ is negligible in } \lambda,$$

2. for all $\chi, x, x' \in \{0,1\}^m$, we have $\mathsf{View}^H(\chi, x) \cong \mathsf{View}^H(\chi, x')$.

In other words, the garbled gate should not leak Δ to the evaluator (this is important for arguing that such garbled gates compose to yield a garbling scheme for circuits), and the garbled gates should hide the truth value.

Composition. We now discuss how (free-XOR-compatible) gate-level garbling procedures can be combined to yield a circuit garbling scheme. The details are given in Figure 3. Roughly speaking, we follow the general approach of Free-XOR garbling, first choosing a global offset Δ. Recall that for each wire i we associate a wire label W_i encoding FALSE; $W_i \oplus \Delta$ will encode TRUE. These false wire labels are chosen uniformly for input wires. Thereafter, we process gates in topological order. Each gate-garbling operation determines the garbled-gate information $\vec{G}$ as well as the FALSE wire labels of the gate's output wires.

For each wire we choose a random select bit σ_i as described above. For each gate, the garbling scheme must provide a way for the evaluator to learn the correct color bits for the output wires. In many practical schemes, the random oracle calls used to evaluate the gate can serve double-duty and also be made to convey the color bits. However, in our case, we aim for complete generality so our scheme manually encrypts the color bits (the G' values in Figure 3). In more detail, if the evaluator has color bits χ on the input wires, then she should obtain color bits $\sigma^{(out)} \oplus \tau(\sigma^{(in)} \oplus \chi)$ for the output wires, where $\sigma^{(in)}$ and $\sigma^{(out)}$ are the select bits for the input/output wires of this gate, respectively. We use the wire labels encoding truth value $\sigma^{(in)} \oplus \chi^{(in)}$ as the key to a one-time encryption that encodes the output color bits.

We point out that these color-ciphertexts are of constant size — 2^m of them, each n bits long (*e.g.*, for a traditional boolean gate with fan-in 2, the cost is 4 bits). As mentioned above, in specific cases it may be possible to eliminate the extra random oracle calls used for these color-bit encryptions.

One subtlety we point out is that each call to a gate-level garbling scheme is restriced to a disjoint set of possible random oracle calls — the gth gate is instructed to use $H(g; \cdot)$ as its random oracle. This domain separation is crucially important in arguing that the gate-level security properties are inherited by the circuit-level garbling scheme.

Lemma 8. *Let $\mathbb{B}$ be a set of boolean functions. Suppose for each $\tau \in \mathbb{B}$, $(\mathsf{GateGb}_\tau, \mathsf{GateEv}_\tau)$ is a correct and secure free-XOR-compatible gate garbling scheme for gate functionality τ (according to Definitions 6 & 7).*

Then the garbling scheme in Figure 3 satisfies the prv, aut, and obv security definitions of [BHR12] in the random oracle model, for circuits expressed in terms of $\mathbb{B}$-gates.

Proof sketch. We sketch here the proof of prv-security; that is, if $f(x) = f(x')$ then (F, X, d) collectively hide whether they were generated with $X = \mathsf{En}(e, x)$ or $X = \mathsf{En}(e, x')$. We show a sequence of hybrids, beginning with an interaction in which (F, X, d) are generated with $X = \mathsf{En}(e, x)$.

We then perform a conceptual shift. Observe that Gb is written in terms of what the garbler sees/knows. The only "persistent" values maintained throughout the main loop are the FALSE wire labels W_i and select bits σ_i. We rearrange Gb to instead be in terms of what the evaluator sees: the "visible" wire labels W^* and their color bits χ_i. We achieve this change by using x to compute

```
$\underline{\mathsf{Gb}^H(1^\lambda, f)}$:
  $\Delta \leftarrow \{0,1\}^\lambda$
  for each wire $i$ of $f$:
    $\sigma_i \leftarrow \{0,1\}$
  for each input wire $i$ of $f$:
    $W_i \leftarrow \mathbb{F}$
    $e[i,0] := (W_i, \sigma_i)$;    $e[i,1] := (W_i \oplus \Delta, \overline{\sigma_i})$
  for each gate $g$ in $f$, in topological order:
    let $g$ have input wires $i_1, \ldots, i_m$, output wires $j_1, \ldots, j_n$, functionality $\tau$
    $\vec{W}^{(in)} := (W_{i_1}, \ldots, W_{i_m})$
    $\sigma^{(in)} := \sigma_{i_1} \| \cdots \| \sigma_{i_m}$;    $\sigma^{(out)} := \sigma_{j_1} \| \cdots \| \sigma_{j_n}$
    $(\vec{W}^{(out)}; \vec{G}) \leftarrow \mathsf{GateGb}_\tau^{H(g,\cdot)}(\sigma^{(in)}; \vec{W}^{(in)}, \Delta)$
    $(W_{j_1}, \ldots, W_{j_n}) := \vec{W}^{(out)}$
    for $\chi$ in $\{0,1\}^m$:
      $v := \sigma^{(in)} \oplus \chi$
      $G'_\chi := H(\mathtt{color} \| g \| \chi; \vec{W}^{(in)} \oplus v\Delta) \oplus (\sigma^{(out)} \oplus \tau(v))$
    $F[g] := (\vec{G}; G'_{0^m}, \ldots, G'_{1^m})$
  for each output wire $i$ of $f$:
    $d[i,0] := H(\mathtt{out} \| i; W_i)$;    $d[i,1] := H(\mathtt{out} \| i; W_i \oplus \Delta)$
  return $F, e, d$

$\underline{\mathsf{En}(e, x)}$:
  for $i = 1$ to $|x|$:
    $X_i = e[i, x_i]$
  return $X$

$\underline{\mathsf{De}(d, Y)}$:
  for $i = 1$ to $|Y|$:
    if $Y_i = d[i,0]$ then $y_i = 0$
    elsif $Y_i = d[i,1]$ then $y_i = 1$
    else return $\perp$
  return $y$

$\underline{\mathsf{Ev}^H(F, X)}$:
  for each input wire $i$ of $f$:
    $(W_i^*, \chi_i) := X_i$
  for each gate $g$ in $f$, in topological order:
    let $g$ have input wires $i_1, \ldots, i_m$, output wires $j_1, \ldots, j_n$, functionality $\tau$
    $\chi^{(in)} := \chi_{i_1} \| \cdots \| \chi_{i_m}$
    $(\vec{G}; G'_{0^m}, \ldots, G'_{1^m}) := F[g]$
    $(W_{j_1}^*, \ldots, W_{j_n}^*) \leftarrow \mathsf{GateEv}_\tau^{H(g,\cdot)}(\chi^{(in)}; W_{i_1}^*, \ldots, W_{i_m}^*, \vec{G})$
    $\chi_{j_1} \| \cdots \| \chi_{j_n} := H(\mathtt{color} \| g \| \chi^{(in)}; W_{i_1}^*, \ldots, W_{i_m}^*) \oplus G'_{\chi^{(in)}}$
  for each output wire $i$ of $f$:
    $Y_i := H(\mathtt{out} \| i; W_i^*)$
  return $Y$
```

Figure 3: Gate-level garbling composed into a circuit garbling scheme.

the truth value v_i on each wire i. Then we replace all references to $W_i^{v_i}$ with W_i^*; references to $W_i^{\overline{v_i}}$ with $W_i^* \oplus \Delta$; references to σ_i with $\chi_i \oplus v_i$. The adversary's view in this modified hybrid is unchanged.

We now observe that (apart from the encryptions of the color bits), each main loop is a Linicrypt program that takes the previously-computed visible wire labels, along with Δ, and computes the next garbled gate and output wire labels. In fact, such a computation is essentially the one expressed in $\mathsf{View}(\chi, v)$, defined above, but with an additional GateEv computation to compute the visible output wire label.

The security of the GateGb components (Definition 6) says that $\mathsf{View}(\chi; v)$ and $\mathsf{View}(\chi; v')$ are indistinguishable. But this statement only applies when Δ is a *local variable* to these views, whereas in the garbling scheme Δ is shared among all gates. So first we must argue that this shared state is not a problem.

More formally, we prove a general composition lemma (Lemmas 9 & 10)which essentially says that if several programs *individually* satisfy Definition 6, and they use guaranteed disjoint calls to the random oracle, then their composition also satisfies Definition 6. It is in this composition lemma that we use the fact that the output of each View also hides Δ. We ensure disjointness of oracle queries by using random oracle $H(g; \cdot)$ when garbling gate g.

Although the calls to the random oracle for encrypting the color bits are not strictly Linicrypt, it is not hard to see that they also satisfy the properties required of this general composition lemma (and use distinct oracle calls). Collectively the entire output given to the adversary's view hides the truth values v_i which are used to select which View to run. The only other place where the v_i truth values are used is in the computation of the garbled decoding information d. And in this case, v_i are required only for the output wire labels, which are the same when garbling either x or x'. Hence, we can replace x with x' with negligible effect on the adversary's view, and the proof is complete.

The proofs of the other security properties obv & aut follows from the above argument, using standard modifications. □

3.2 Synthesis Approach

One of our motivating goals for Linicrypt is the ability to synthesize secure cryptographic constructions. We do precisely that for free-XOR-compatible gate garbling schemes.

We have written a synthesis tool, **Linisynth** which takes as input the desired parameters of a garbled gate construction. These parameters include:

- The gate functionality $\tau : \{0,1\}^m \to \{0,1\}^n$
- The arity of the random oracle $\mathsf{arity} \in \mathbb{N}$ (*e.g.*, whether the oracle is called with 1 or 2 field elements, etc.)
- The number of random oracle queries made by GateGb and GateEv: $\mathsf{calls}_{\mathsf{gb}}, \mathsf{calls}_{\mathsf{ev}} \in \mathbb{N}$
- The size (in field elements) of the garbled gate information $\mathsf{size} \in \mathbb{N}$
- Whether adaptive queries to the oracle are allowed $\mathsf{adaptive} \in \{0,1\}$ (see below).

Given such parameters, Linisynth constructs an appropriate SMT formula encoding the required security properties, invokes an SMT solver, and finally interprets the witness (if any) as a human-readable garbled gate construction.

In this section, we describe an overview of Linisynth's functionality and our results.

High-level outline. Gate garbling schemes as defined in Definitions 6 and 7 are meant to be nonlinear in their use of inputs σ and χ. Hence, to synthesize a complete gate-garbling scheme, we must actually synthesize a *collection* of $\mathsf{GateGb}(\sigma;\cdots)$ and $\mathsf{GateEv}(\chi;\cdots)$ — one for each choice of σ and χ — each of which is a *pure* Linicrypt program.

We now describe roughly how the gate-garbling search problem is expressed as an existential SAT/SMT formula. Recall that pure Linicrypt programs can be represented algebraically as an output matrix $\mathcal{M}$ and a set of oracle constraints $\mathcal{C}$. When restricted to Free-XOR compatible garbling, the entries in these matrices are single bits. These bits comprise the existentially quantified variables of our SMT formula.

Not every bit in the oracle constraints $\mathcal{C}$ has to be an unconstrained variable. Specifically, if the Linicrypt program in question has k input variables, then we identify these with the first k base variables. This means that the first oracle query made by the program can be a linear combination *only* of these first k base variables. For the corresponding oracle constraint $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle$, this means that each row of $\mathcal{Q}$ must end in a certain number of zeroes — say, i zeroes. Then we can associate the output of this oracle query with the $(k+1)$th base variable, fixing $\boldsymbol{a}$ to be $[\underbrace{0\ \cdots\ 0}_{k}\ 1\ 0\ \cdots\ 0]$. Then the next oracle query can be a linear combination of only the first $k+1$ variables, and so on. Overall, many of the existential variables comprising the oracle constraints can be fixed in this way. Furthermore, we can seamlessly enforce non-adaptive oracle queries by forcing all constraints $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle$ to have $\mathcal{Q}$ depending only on the input variables, and not on further base variables. This is what is referred to by the $\mathsf{adaptive}$ parameter.

We then express the requirements of Definitions 6 & 7 as clauses over the variables that comprise the programs themselves. The formula is satisfiable **if and only if** a secure gate-garbling scheme exists with the given parameters.

Correctness. Correctness (Definition 6) is defined in terms of a particular composition of GateGb with GateEv. Hence we can apply the ideas of Section 2.4 to reason about their composition.

Intuitively, we must first identify a basis change B that maps $\mathsf{View}(\chi, x)$ to the "reference frame" of GateEv. Recall that View generates the input wire labels, garbled gate information, and output wire labels. Let $\mathcal{M}_{\chi,x}$ denote the output matrix of $\mathsf{View}(\chi, x)$. We split this matrix into a top and bottom: $\mathcal{M}^{top}_{\chi,x}, \mathcal{M}^{bot}_{\chi,x}$, where the top matrix corresponds to the input wire labels for x along with garbled gate information, while the bottom matrix corresponds to the output wire labels for $\tau(x)$.

Following Section 2.4, the appropriate basis change B must map $\mathcal{M}^{top}_{\chi,x}$ to a matrix of the form $[I \mid 0]$, which represents the input base variables of $\mathsf{Ev}(\chi, \cdot)$. The basis change must also bring all oracle constraints between the two programs into alignment.

We assume that every oracle query made by GateEv is also made by GateGb. This is without loss of generality if we assume that GateEv is "minimal", since such oracle queries can be removed with no effect (if not, it is easy to see that correctness or security is violated). Hence, we check that for every oracle constraint in GateEv, the basis change brings one of the constraints of GateGb into agreement.

Having identified the correct basis change, we simply check that the output matrix of GateEv equals the output matrix $\mathcal{M}^{bot}_{\chi,x}$ (under the basis change). In other words, the wire labels that GateEv outputs always coincide with the "correct" wire labels specified by View.

We also must ensure that B is invertible. To do so we simply guess its inverse B^{-1} and check that $B \times B^{-1}$ is the identity matrix. We point out that multiplication of boolean matrices is straight-forward to express in an SMT formula.

Putting it all together, the clause is as follows. Recall that the input $x = \sigma \oplus \chi$. We use $(\mathcal{M}_{\chi,x}, \mathcal{C}_{\chi,x})$ to refer to the algebraic representation of $\mathsf{View}(\chi, x)$, and use $(\mathcal{M}_{\mathsf{GateEv},\chi}, \mathcal{C}_{\mathsf{GateEv},\chi})$ to denote the algebraic representation of $\mathsf{GateEv}(\chi, \cdot)$.

$$\begin{aligned}
&\forall \sigma, \chi \in \{0,1\}^m : \\
&\quad \exists B, B^{-1} : B \times B^{-1} = I \\
&\qquad\qquad \wedge \left[\forall \langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}_{\mathsf{GateEv},\chi} : \langle t, \mathcal{Q} \times B^{-1}, \boldsymbol{a} \times B^{-1}\rangle \in \mathcal{C}_{\chi,x}\right] \\
&\qquad\qquad \wedge \mathcal{M}_{\mathsf{GateEv},\chi} = \mathcal{M}^{bot}_{\chi,x} \times B
\end{aligned}$$

We point out that the universal quantifiers are over a constant number of terms (2^{2m} choices of (σ, χ) and $\mathsf{calls}_{\mathsf{ev}}$ constraints) and are explicitly expanded in the formula we pass to the SMT solver. Likewise, the test for $\langle t, \mathcal{Q} \times B^{-1}, \boldsymbol{a} \times B^{-1}\rangle \in \mathcal{C}_{\chi,x}$ is expressed as a logical-OR of $\mathsf{calls}_{\mathsf{gb}}$ equality checks.

Security. We now describe how to express Definition 7 as a SAT formula/clause. Recall that this definition has two conditions: the first is that Δ is *unreachable* given the output of $\mathsf{View}(\chi, x)$. The second is that $\mathsf{View}(\chi, x)$ is *indistinguishable* from $\mathsf{View}(\chi, x')$ for all inputs $x, x' \in \{0,1\}^m$. These are rather complicated properties to check, so we address them individually.

Security, condition 1. Our goal is to express the condition that $\mathsf{row}(\Delta)$ is not reachable (in the sense of Figure 2). In an ideal world where the SAT solver could discover the linear subspace $\mathcal{R}$ of reachable vectors, it could simply test whether this subspace includes $\mathsf{row}(\Delta)$. However, to do this directly is inefficient, so we employ a trick.

Our main idea is to consider a special kind of basis change B. Under B, the reachable space consists of all vectors ending ending in d zeroes — membership in this subspace is extremely easy to test. Then the SAT formula will work by guessing the basis change B and then:

- Checking that B maps the reachable subspace to this easily recognizable form,
- Checking that $\mathsf{row}(\Delta \times B)$ is *not* in the reachable space.

To check that the basis change B indeed maps the reachable subspace to a space of this special form, we observe that the reachable space is *characterized by* the following properties:

- Every row of the output matrix $\mathcal{M}$ is contained in the reachable space
- For every oracle constraint $\langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}$, if every row of $\mathcal{Q}$ is in the reachable space, then so is $\boldsymbol{a}$.

For the reachable space after the basis change, the membership condition is simply that the vector ends in the correct number of zeroes.

We note that from the input parameters, we can compute the dimension of the reachable space (and from that derive the required number of trailing zeroes in the vectors) as $d = m + \mathsf{calls}_{\mathsf{ev}} + \mathsf{size}$, where m is the number of inputs, $\mathsf{calls}_{\mathsf{ev}}$ is the number of oracle queries allowed the evaluator, and size is the size of the garbled gate information. This assumes that each oracle query of GateEv increases the dimension of the reachable space — an assumption that is without loss of generality for “minimal” schemes since oracle queries not of this kind are superfluous.

Putting everything together, the formula is as follows. We use $\mathsf{row}(\Delta)$ to refer to the appropriate vector in $\mathsf{View}(\chi, x)$:

$$\begin{aligned}
&\forall \sigma, \chi \in \{0,1\}^m : \\
&\qquad \exists B, B^{-1} : B \times B^{-1} = I \\
&\qquad\qquad \wedge \neg \mathsf{RightZeroes}(\mathsf{row}(\Delta) \times B) \wedge \mathsf{RightZeroes}(\mathcal{M}_{\chi,x} \times B) \\
&\qquad\qquad \wedge \big[\forall \langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}_{\chi,x} : \mathsf{RightZeroes}(\mathcal{Q} \times B) \Rightarrow \mathsf{RightZeroes}(\boldsymbol{a} \times B)\big]
\end{aligned}$$

Here $\mathsf{RightZeroes}$ simply means that the argument vector/matrix has the appropriate number of zeroes in its rightmost columns. The universal quantifiers are over a constant number of terms (2^{2m} choices of (σ, χ) and $\mathsf{calls}_{\mathsf{gb}}$ constraints) and are explicitly expanded in the formula we pass to the SMT solver.

We also emphasize that *the same basis B can be used* for both the correctness and security properties! Intuitively, the correctness condition puts a constraint on how B maps the *reachable subspace* of $\mathsf{View}(\chi, x)$. Namely, it must map (part of) the reachable space to a matrix of the form $[I \mid \vec{0}]$ to facilitate composition. On the other hand, the security constraint constrains how it maps the *unreachable* vectors in $\mathsf{View}(\chi, x)$. In particular, unreachable vectors cannot end in a certain number of trailing zeroes. As these two conditions are not mutually exclusive, the same basis B can be used for both conditions.

Security, condition 2. The second condition of Definition 7 is that $\mathsf{View}(\chi, x)$ and $\mathsf{View}(\chi, x')$ are indistinguishable. These are inputless Linicrypt programs, so from Theorem 5 it suffices to show that they differ by a basis change after normalization (unreachable and useless oracle queries removed).

Here we make several assumptions about the scheme that are without loss of generality for "minimal" schemes.

- First, the garbled input and garbled gate produced by $\mathsf{View}(\chi, x)$ — i.e., the values associated with output matrix $\mathcal{M}^{top}_{\chi,x}$ — are linearly dependent.

  If these values have nontrivial linear dependencies, then the likely case is that the redundant information can simply be removed, so that $\mathsf{GateEv}(\chi, \cdot)$ can reconstruct it itself. The only reason this would fail is if the rowspace of these values differs between some $\mathsf{View}(\chi, x)$ and $\mathsf{View}(\chi, x')$. But in that case, security is trivially broken.

- While for correctness we assume that all oracle queries made by GateEv were also made by GateGb, here we assume a complementary property. That is, any *other* queries made by GateGb are either unreachable or useless in $\mathsf{View}(\chi, x)$. Again this is without loss of generality. Fix some σ. Then if a query in $\mathsf{GateGb}(\sigma, \cdot)$ is unreachable, useless, or absent for *all* $\mathsf{View}(\chi, \chi \oplus \sigma)$, then the query can be safely removed from $\mathsf{GateGb}(\sigma, \cdot)$. On the other hand, if a query is reachable and useful for some $\mathsf{View}(\chi, x = \chi \oplus \sigma)$ but not for some other $\mathsf{View}(\chi', x' = \chi' \oplus \sigma)$, then the evaluator could easily distinguish between x and $x' \neq x$.

Hence it suffices to check whether all oracle constraints in $\mathsf{View}(\chi, x)$ are either in agreement with a constraint from GateEv (these are the reachable ones) or are unreachable.

Again, we can use the same basis change as in the above clauses, because it allows the reachability condition to be checked easily. In detail, after quantifying over σ, χ, B as in the other clauses, we check:

$$\begin{aligned}
\forall \langle t, \mathcal{Q}, \boldsymbol{a}\rangle \in \mathcal{C}_{\chi,x} : &\langle t, \mathcal{Q} \times B, \boldsymbol{a} \times B\rangle \in \mathcal{C}_{\mathsf{GateEv},\chi} \\
&\vee \neg \mathsf{RightZeroes}(\mathcal{Q} \times B)
\end{aligned}$$

As usual, the quantifications over constraints in $\mathcal{C}_{\chi,x}$ and $\mathcal{C}_{\mathsf{GateEv},\chi}$ are explicitly expanded.

3.3 Implementation Results

We implemented Linisynth using Python and the SMT solver Z3[2]. Linisynth extracts the resulting witness and prints it as a human-readable garbling scheme. We used Linisynth to successfully synthesize variants of known gate garbling schemes as well as some of our own creations (*i.e.*, garbled LT gates and garbled EQ gates). Linicrypt can also enumerate constructions that satisfy given parameters. We present a sampling of synthesized schemes in Appendix C.2. Our code is available at `https://github.com/osu-crypto/linisynth`.

Linisynth works as follows. For each value in the algebraic representation of GateGb and GateEv, it creates a boolean variable. After it has created all the variables, it makes a formula that constrains them in the following way. For each combination of σ and χ, the invertiblity, correctness, and security conditions from Section 3.2 hold (expressed as boolean formulas over the variables). This often results in rather large formulas (see Figure 4). Linisynth then hands the formula over to Z3. If Z3 finds a solution, it maps the satisfying assignment back to the garbling scheme and prints it.

Synthesis results. We rediscovered known constructions. For example, our tool was able to discover that XOR gates can be garbled for free. It also rediscovered many garbled AND-gate constructions that are equivalent to the half-gates construction of Zahur et al. [ZRE15] (costing 2 ciphertexts). We synthesized garbling schemes for a number of different gates (garbled $<$, garbled $=$, garbled MUX, etc), but they all had comparable performance to AND, explained below. A summary is presented in Figure 4, with examples of synthesized constructions given in Appendix C.2.

We were not able to synthesize a garbling scheme better than 2 ciphertexts per AND gate. We suspect that this may be a hard limit (if compatibility with free-XOR is required), in support of the half-gates lower-bound presented in [ZRE15]. We formalize that hypothesis here. First, note that $\mathbb{B} = \{ \text{AND}, \text{NOT}, \text{XOR} \}$ is a universal basis for boolean circuits. Then take any boolean gate τ and decompose it into some combination of AND, NOT, and XOR. Let $\mathsf{circ\text{-}min}_{\text{AND}}(\tau)$ be the minimum number of AND gates necessary to construct τ with basis $\mathbb{B}$. Our hypothesis is this: for all gates τ, the minimum number of ciphertexts to garble τ with full security and compatibility with free-XOR is $2 \times \mathsf{circ\text{-}min}_{\text{AND}}(\tau)$.

If we knew $\mathsf{circ\text{-}min}_{\text{AND}}(\tau)$, we could predict the number of ciphertexts needed to garble τ, and validate or invalidate our hypothesis. While we have no direct way to compute $\mathsf{circ\text{-}min}_{\text{AND}}(\tau)$, we can use a tool to approximate. Synudic, the program synthesis tool of Tiwari, Gascón, and Dutertre [TGD15] synthesizes straight-line programs by using any number of "dual interpretations" to constrain the choices of functions and arguments. We use Synudic to approximate the minimum number of AND gates in 2-BIT-LT.

Let 2-BIT-LT be a function that compares two two bit numbers. We sought to find out whether garbled 2-BIT-LT is more efficient than garbled AND. First, we used Synudic to approximate the minimum number of ANDs in 2-BIT-LT. Synudic can synthesize 2-BIT-LT using XOR, NOT, and 2 ANDs, but not 1 AND. Then, a garbling of 2-BIT-LT should take 4 ciphertexts. We asked Linisynth to find such a garbling. The results are that 2-BIT-LT is satisfiable with 4 ciphertexts but not with 3 ciphertexts, agreeing with our hypothesis.

Enumeration of solutions. Linisynth can also *enumerate* schemes. Let p be a formula generated according to Section 3.2 and let w be a satisfying assignment with $p(w) = 1$. When Linisynth gets

[2]https://github.com/Z3Prover/z3

w from the solver, it prints the corresponding scheme, sets $p := \neg w \wedge p$, and asks the solver to find a new solution. Since `pysmt` provides access to an active instance of Z3, we can use Z3's push/pop functionality to add an assertion without causing the solver to restart. This is very efficient. Each new scheme is found in a fraction of the time it takes to find the first one. Using enumeration, we found thousands of schemes equivalent to half-gates (with parameters $\mathsf{size} = 4, \mathsf{arity} = 1, \mathsf{calls}_{\mathsf{gb}} = 4$, $\mathsf{calls}_{\mathsf{ev}} = 2$, and $\mathsf{adaptive} = 0$). We made no attempt to reduce symmetries in the search space, so many of the constructions are very similar.

name	τ	size	arity	$\mathsf{calls}_{\mathsf{gb}}$	$\mathsf{calls}_{\mathsf{ev}}$	adaptive	vars	p-size	time	sat
free-xor	$\oplus : 2 \to 1$	0	1	0	0	0	224	5,102	1s	1
half-gate	$\wedge : 2 \to 1$	2	1	4	2	0	1,972	117,586	5s	1
half-gate-cheaper	$\wedge : 2 \to 1$	2	1	4	1	1	1,960	92,690	6.2h	0
half-gate-h2	$\wedge : 2 \to 1$	2	2	4	2	0	2,000	114,397	2h	0
one-third-gate	$\wedge : 2 \to 1$	1	1	4	2	1	4,104	716,454	74s	0
1-out-of-2-mux	$\textsc{mux} : 3 \to 1$	2	1	4	2	1	9,416	654,433	29s	1
2-bit-eq	$= : 4 \to 1$	2	1	4	2	1	44,144	3,497,286	6m	1
2-bit-eq-small	$= : 4 \to 1$	1	1	4	2	1	39,248	3,535,942	6m	0
2-bit-leq	$\leq : 4 \to 1$	1	1	2	1	1	23,296	1,155,686	77s	0
2-bit-lt	$< : 4 \to 1$	2	1	4	2	1	44,144	3,502,425	3.5h	0

Figure 4: Selection of our synthesis results on an Intel Xeon 3.4GHz processor with 16GB memory. Satisfiable schemes are listed in Appendix C.2. Notation: "$f : m \to n$" is shorthand for a function with m bits of input and n bits of output that performs the operation f on the input, "vars" and "p-size" refer to the number of variables and nodes in the security & correctness formula. "sat" refers to whether the formula was satisfiable.

Scalability issues. One limitation of our approach is that it *cannot* synthesize certain constructions used in current garbling schemes. We elaborate: Consider an AND gate with fan-in 3. To garble this gate using the half-gates construction, we write it as $a \wedge (b \wedge c)$ and use 2 ciphertexts per AND gate, a total of 4 ciphertexts.

Despite the fact that Linicrypt allows for nested calls to the oracle (which this garbling approach uses), our synthesis approach will *not* discover it. The reason is that in half-gates, the intermediate value $z = b \wedge c$ also has a color bit. So the subsequent processing of $a \wedge z$ depends on that additional color bit non-linearly (for both the evaulator and garbler). Using Linisynth, we were able to confirm that an additional color bit is in fact *necessary* to garble a 3-ary AND gate with 4 ciphertexts (and free-XOR compatibility).

To account for this, our approach must be modified. We augmented Linisynth to allow for extra "helper bits" σ', χ' beyond the color bits and select bits of the input wires. Then the correctness of the garbling scheme must be modified so that (roughly speaking) for every χ, x, there exists a 1-to-1 mapping of garbler to evaluator helper bits[3] such that for all garbler-helper bits σ', $\mathsf{GateGb}(\sigma \| \sigma', \cdot)$ and the appropriate $\mathsf{GateEv}(\chi \| \chi', \cdot)$ compose to give the correct answer. Unfortunately, this additional search space rendered the synthesis task unusable. When trying to synthesize the nested half-gates construction of a fan-in-3 AND gate, Z3 ran out of memory after weeks of computation.

[3]For example, the helper bits for the fan-in-3 AND gate satisfy $\chi' \oplus \sigma' = b \wedge c$, so the association between χ' and σ' is different for different χ, σ.

References

[AAB15] Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 143–151. ACM, 2015.

[AGH13] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 399–410. ACM Press, November 2013.

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Heidelberg, August 2011.

[AGHP12] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In Yu et al. [YDG12], pages 474–487.

[AGOT14a] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Mehdi Tibouchi. Structure-preserving signatures from type II pairings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 390–407. Springer, Heidelberg, August 2014.

[AGOT14b] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Mehdi Tibouchi. Unified, minimal and selectively randomizable structure-preserving signatures. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 688–712. Springer, Heidelberg, February 2014.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. Cryptology ePrint Archive, Report 2011/484, 2011. http://eprint.iacr.org/2011/484.

[BDK$^+$07] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 31–45. Springer, Heidelberg, June 2007.

[BFF$^+$15] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 355–376. Springer, Heidelberg, March / April 2015.

[BGD$^+$06] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS - an improved Merkle signature scheme. In Rana Barua and Tanja Lange, editors, *INDOCRYPT 2006*, volume 4329 of *LNCS*, pages 349–363. Springer, Heidelberg, December 2006.

[BHH$^+$15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, Heidelberg, April 2015.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Yu et al. [YDG12], pages 784–796.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 384–397. Springer, Heidelberg, April / May 2002.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Yung [Yun02], pages 320–335.

[DHT12] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 112–132. Springer, Heidelberg, March 2012.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993.

[GLLS09] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In Anupam Datta, editor, *Advances in Computer Science - ASIAN 2009*, volume 5913 of *LNCS*, pages 39–53. Springer, 2009.

[GLLS11] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In Joaquín García-Alfaro and Pascal Lafourcade, editors, *Foundations and Practice of Security*, volume 6888 of *LNCS*, pages 23–31. Springer, 2011.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Ray et al. [RLK15], pages 567–578.

[Gol87] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 104–110. Springer, Heidelberg, August 1987.

[HKM15] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In Ray et al. [RLK15], pages 84–95.

[HR03] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 482–499. Springer, Heidelberg, August 2003.

[Hül13] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT*, volume 7918 of *LNCS*, pages 173–188. Springer, 2013.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147. IEEE Computer Society, 1995.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, Heidelberg, August 1990.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: keyed-hashing for message authentication, 1997.

[KD07] Ted Krovetz and Wei Dai. VMAC: message authentication code using universal hashing. CFRG Working Group, 2007. http://www.fastcrypto.org/vmac/draft-krovetz-vmac-01.txt.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[LR86] Michael Luby and Charles Rackoff. How to construct pseudo-random permutations from pseudo-random functions (abstract). In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, page 447. Springer, Heidelberg, August 1986.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Yung [Yun02], pages 31–46.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference*, volume 3796 of *LNCS*, pages 1–12. Springer, 2005.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

[MKG14] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE 27th Computer Security Foundations Symposium, CSF*, pages 140–152. IEEE, 2014.

[MPs16] Tal Malkin, Valerio Pastro, and abhi shelat. An algebraic approach to garbling. Unpublished manuscript. Presented at Simons Institute workshop on securing computation: https://simons.berkeley.edu/talks/tal-malkin-2015-06-10, 2016.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, New York, NY, USA, 1999. ACM.

[NSW05] Dalit Naor, Amir Shenhav, and Avishai Wool. One-time signatures revisited: Have they become practical? Cryptology ePrint Archive, Report 2005/442, 2005. http://eprint.iacr.org/2005/442.

[PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, August 1994.

[PPB15] Geovandro C.C.F. Pereira, Cassius Puodzius, and Paulo S.L.M. Barreto. Shorter hash-based signatures. *Journal of Systems and Software*, 2015.

[PRV12] Periklis A. Papakonstantinou, Charles W. Rackoff, and Yevgeniy Vahlis. How powerful are the DDH hard groups? Cryptology ePrint Archive, Report 2012/653, 2012. http://eprint.iacr.org/2012/653.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

[PWX04] Josef Pieprzyk, Huaxiong Wang, and Chaoping Xing. Multiple-time signature schemes against adaptive chosen message attacks. In Mitsuru Matsui and Robert J. Zuccherato, editors, *SAC 2003*, volume 3006 of *LNCS*, pages 88–100. Springer, Heidelberg, August 2004.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM CCS 01*, pages 196–205. ACM Press, November 2001.

[RLK15] Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors. *ACM CCS 15*. ACM Press, October 2015.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, December 2004.

[RR02] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP 02*, volume 2384 of *LNCS*, pages 144–153. Springer, Heidelberg, July 2002.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[TGD15] Ashish Tiwari, Adria Gascón, and Bruno Dutertre. Program synthesis using dual interpretation. In *Automated Deduction-CADE-25*, pages 482–497. Springer, 2015.

[Win83] Robert S. Winternitz. Producing a one-way hash function from DES. In David Chaum, editor, *CRYPTO'83*, pages 203–207. Plenum Press, New York, USA, 1983.

[YDG12] Ting Yu, George Danezis, and Virgil D. Gligor, editors. *ACM CCS 12*. ACM Press, October 2012.

[Yun02] Moti Yung, editor. *CRYPTO 2002*, volume 2442 of *LNCS*. Springer, Heidelberg, August 2002.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

A Proofs of Linicrypt Technical Lemmas

A.1 Proof of Lemma 4

Proof. We break the proof into two steps. We let $\mathcal{P}'' = \mathsf{normalize}(\mathcal{P}) = (\mathcal{M}, \mathcal{C}'')$ and we also define $\mathcal{P}' = (\mathcal{M}, \mathcal{C}')$, where $\mathcal{C}'$ is the intermediate value defined in the program. Note that the only difference among $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{P}''$ is that $\mathcal{C}'' \subseteq \mathcal{C}' \subseteq \mathcal{C}$. We will show $\mathcal{P} \cong \mathcal{P}'$ and $\mathcal{P}' \cong \mathcal{P}''$ separately. More precisely, we will compare the *canonical simulations* (Equation 1) of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{P}''$.

($\mathcal{P} \cong \mathcal{P}'$) The logic within the $\mathsf{normalize}$ procedure uses only the concept of linear span; as such, the behavior of $\mathsf{normalize}$ is invariant under a basis change to $(\mathcal{M}, \mathcal{C})$. Since $\mathsf{span}(\mathsf{Reachable})$ is a linear subspace of $\mathbb{F}^{\mathsf{base}}$, consider a basis change under which $\mathsf{span}(\mathsf{Reachable}) = \{0\}^d \times \mathbb{F}^{\mathsf{base}-d}$. We now argue that the removal of oracle constraints in $\mathcal{C} \setminus \mathcal{C}'$ has no effect on an adversary.

Consider the following sequence of hybrid experiments: In hybrid $\#h$ we run the canonical simulation for $\mathcal{P} = (\mathcal{M}, \mathcal{C})$ but use $\mathcal{C}'$ in place of $\mathcal{C}$ in line 7 for the first h calls to the oracle (using $\mathcal{C}$ thereafter). Consider hybrid h at the moment when the adversary makes query number $h+1$ to its oracle. What information does the adversary have about $\boldsymbol{v}_{\mathsf{base}}$? The adversary has seen $\boldsymbol{v}_{\mathsf{out}} = \mathcal{M} \times \boldsymbol{v}_{\mathsf{base}}$ from line 4. Since $\mathsf{rows}(\mathcal{M}) \subseteq \mathsf{Reachable}$ by construction, $\mathcal{M}$ has zeroes in its first d columns so the expression does not depend on the first d base variables. The adversary is also given $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$ in line 8, but by the definition of the hybrids this line is reached on a constraint in $\mathcal{C}'$. Hence $\boldsymbol{a} \in \mathsf{span}(\mathsf{Reachable})$, and so again the expression does not depend on the first d base variables. Summarizing, at the time of query number $h+1$, the adversary's view of $\boldsymbol{v}_{\mathsf{base}}$ is *syntactically* independent of the first d base variables.

The adversary's advantage in distinguishing hybrid $\#h$ from $\#h+1$ is bounded by the probability that it can reach line 8 by triggering a constraint $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C} \setminus \mathcal{C}'$. For such a constraint, $\mathcal{Q}$ has a nonzero entry in its first d columns, so the adversary can query the oracle on $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}}$ with probability at most $1/|\mathbb{F}|$. By a union bound, the adversaries bias distinguishing hybrids $\#h$ and $\#h+1$ is at most $|\mathcal{C} \setminus \mathcal{C}'|/|\mathbb{F}|$.

If the adversary makes n queries to its oracle, then hybrid $\#0$ is the canonical simulation of $\mathcal{P}$ and hybrid $\#n$ is the canonical simulation of $\mathcal{P}'$. They can therefore be distinguished with bias at most $|\mathcal{C}|n/|\mathbb{F}|$.

($\mathcal{P}' \cong \mathcal{P}''$) Consider the first constraint $\langle t^*, \mathcal{Q}^*, \boldsymbol{a}^* \rangle$ that is added to $\mathsf{Useless}$. This happens when $\boldsymbol{a}^*$ is linearly independent of all other vectors in $\mathcal{M}$ and $\mathcal{C}'$. Consider a basis change under which $\boldsymbol{a}^* = (1, 0, \ldots, 0)$ and $V \setminus \{\boldsymbol{a}^*\} \subseteq \{0\} \times \mathbb{F}^{\mathsf{base}-1}$; this is possible since $\boldsymbol{a}^* \notin \mathsf{span}(V \setminus \{\boldsymbol{a}^*\})$.

Under this basis change, let us determine where the first base variable v_1 is used *syntactically* in the canonical simulation (of $\mathcal{P}'$, so that only constraints in $\mathcal{C}'$ are used). Since $\mathsf{rows}(\mathcal{M}) \subseteq V \setminus \{\boldsymbol{a}^*\}$, the first column of $\mathcal{M}$ is all zeroes and so v_1 is not used in line 4. Similarly, every expression $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}}$ or $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$ (for $\boldsymbol{a} \neq \boldsymbol{a}^*$) in the simulation does not depend on v_1. Indeed, the only place v_1 is used is when line 8 is reached because of the constraint $\langle t^*, \mathcal{Q}^*, \boldsymbol{a}^* \rangle$. In that case, the result is $\boldsymbol{a}^* \cdot \boldsymbol{v}_{\mathsf{base}} = v_1$.

Removing the constraint $\langle t^*, \mathcal{Q}^*, \boldsymbol{a}^* \rangle$ causes the corresponding oracle query to be answered by lines 9-11 instead of lines 7-8. But since v_1 is chosen uniformly in line 1, this is in fact the same behavior. Removing the constraint therefore has no effect on the adversary's view. We can repeatedly apply the same logic starting with $(\mathcal{M}, \mathcal{C}' \setminus \{\langle t^*, \mathcal{Q}^*, \boldsymbol{a}^* \rangle\})$ to complete the proof. $\square$

A.2 Proof of Theorem 5

Proof. ($\Leftarrow$) From the earlier discussion about basis changes, if $\mathsf{normalize}(\mathcal{P}_1)$ and $\mathsf{normalize}(\mathcal{P}_2)$ differ by a basis change, then surely $\mathsf{normalize}(\mathcal{P}_1) \cong \mathsf{normalize}(\mathcal{P}_2)$. Then $\mathcal{P}_1 \cong \mathcal{P}_2$ by Lemma 4.

($\Rightarrow$) Without loss of generality, assume both $\mathcal{P}_1$ and $\mathcal{P}_2$ are already normalized (*i.e.*, $\mathcal{P}_i = \mathsf{normalize}(\mathcal{P}_i)$). We will show that if no basis change relates $\mathcal{P}_1$ and $\mathcal{P}_2$, then we can construct a distinguisher (using the fact that $\mathcal{P}_1$ and $\mathcal{P}_2$ are normalized).

We first assume (without loss of generality) that both programs have the same number of base variables. If not, then unused base variables can be added with no effect. As a result, the algebraic representations of both programs include vectors of the same width.

Consider the oracle machine $\mathcal{A}$ below, along with a helper function $\mathsf{profile}$.

```
profile(P = (M, C)):
  $\mathcal{R} := \mathcal{M}$
  output $\mathcal{R}$
  $\mathsf{seen} := \emptyset$
  until $\mathcal{R}$ reaches a fixed point:
    for each $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C} \setminus \mathsf{seen}$:
      if $\mathsf{rows}(\mathcal{Q}) \subseteq \mathsf{rowspace}(\mathcal{R})$:
        append row $\boldsymbol{a}$ to $\mathcal{R}$
        output $t$, $\mathcal{Q}$, $\mathcal{R}$
        add $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ to $\mathsf{seen}$
```

```
// canonical distinguisher for $\mathcal{P}_1$
$\mathcal{A}^H(\boldsymbol{x})$:
  $(\mathcal{R}_0, t_1, \mathcal{Q}_1, \mathcal{R}_1, \ldots, t_n, \mathcal{Q}_n, \mathcal{R}_n)$
                    $:= \mathsf{profile}(\mathcal{P}_1)$
  $\boldsymbol{r}_0 := \boldsymbol{x}$
  for $i = 1$ to $n$:
    choose matrix $L_i$ s.t. $\mathcal{Q}_i = L_i \times \mathcal{R}_{i-1}$
    $\boldsymbol{r}_i := \boldsymbol{r}_{i-1} \| H(t_i; L_i \times \boldsymbol{r}_{i-1})$
  choose a vector $\boldsymbol{z}$ s.t. $\boldsymbol{z} \times \mathcal{R}_n = \boldsymbol{0}$
  return $\boldsymbol{z} \cdot \boldsymbol{r}_n \overset{?}{=} 0$
```

When this adversary is run in the canonical simulation of $\mathcal{P}_1$, it is easy to see that the invariant $\boldsymbol{r}_i = \mathcal{R}_i \times \boldsymbol{v}_{\mathsf{base}}$ holds with respect to the internal $\boldsymbol{v}_{\mathsf{base}}$ variable in the simulation. This holds no matter how the "choose" statements are instantiated. Hence, $\mathcal{A}$ outputs TRUE with overwhelming probability (the canonical simulation itself may abort with negligible probability).

We now show that (for suitable choice of L_i matrices and $\boldsymbol{z}$ vector) $\mathcal{A}$ outputs TRUE with only negligible probability when run in the canonical simulation of $\mathcal{P}_2$.

Note that the code of $\mathsf{profile}$ (and hence $\mathcal{A}$) may depend on the relative order of oracle constraints. Choose a reordering of constraints of $\mathcal{P}_1$ and $\mathcal{P}_2$ and a basis change to apply to $\mathcal{P}_2$ so that $\mathsf{profile}(\mathcal{P}_1)$ and $\mathsf{profile}(\mathcal{P}_2)$ agree in the longest possible prefix. We consider several different cases.

1. The profiles disagree already in $\mathcal{R}_0$. This can only happen if $\ker(\mathcal{M}_1) \neq \ker(\mathcal{M}_2)$ since otherwise a better choice of basis change to $\mathcal{P}_2$ could achieve $\mathcal{M}_1 = \mathcal{M}_2$ $(= \mathcal{R}_0)$. One of $\ker(\mathcal{M}_1), \ker(\mathcal{M}_2)$ must be nonempty; by symmetry suppose $\ker(\mathcal{M}_1)$ is nonempty. Then choose $\boldsymbol{z}$ of the form $[\boldsymbol{z}' \mid \boldsymbol{0}]$ where $\boldsymbol{z}' \in \ker(\mathcal{M}_1) \setminus \ker(\mathcal{M}_2)$. This $\boldsymbol{z}$ is a suitable choice in the penultimate line of $\mathcal{A}$.

But when $\mathcal{A}$ is executed in the canonical simulation of $\mathcal{P}_2$, its output is

$$0 \overset{?}{=} \boldsymbol{z} \cdot \boldsymbol{r}_n = \boldsymbol{z}' \cdot \boldsymbol{x} = \boldsymbol{z}'(\mathcal{M}_2 \boldsymbol{v}_{\mathsf{base}})$$

Since $\boldsymbol{z}'\mathcal{M}_2 \neq \boldsymbol{0}$ and $\boldsymbol{v}_{\mathsf{base}}$ is chosen uniformly, the result is FALSE with overwhelming probability.

2. The profiles first disagree on some $\mathcal{R}_i$ ($i > 0$). Because the profiles agree up until this point, $\mathcal{P}_1$ must have an oracle constraint $\langle t_i, \mathcal{Q}_i, \boldsymbol{a}^{(2)} \rangle$ and $\mathcal{P}_2$ must have a constraint $\langle t_i, \mathcal{Q}_i, \boldsymbol{a}^{(2)} \rangle$, with $\boldsymbol{a}^{(1)} \neq \boldsymbol{a}^{(2)}$.

   (a) Case: $\boldsymbol{a}^{(1)} \notin \mathsf{span}(\mathcal{R}_{i-1})$ and $\boldsymbol{a}^{(2)} \notin \mathsf{span}(\mathcal{R}_{i-1})$. Then a better choice of basis change could achieve $\boldsymbol{a}^{(1)} = \boldsymbol{a}^{(2)}$ without affecting the prior agreement of profiles up to $\mathcal{R}_{i-1}$. This contradicts the assumption that we are considering the longest possible agreement between profiles.

   (b) Case (exhaustive by symmetry): $\boldsymbol{a}^{(1)} \in \mathsf{span}(\mathcal{R}_{i-1})$. Let $\mathcal{R}_i^{(b)}$ denote $\mathcal{R}_{i-1}$ concatenated with row $\boldsymbol{a}^{(b)}$. Then $\ker(\mathcal{R}_i^{(0)}) \neq \ker(\mathcal{R}_i^{(1)})$ and $\ker(\mathcal{R}_i^{(0)})$ is nontrivial. Just as in case 1, we can choose a $\boldsymbol{z}$ of the form $[\boldsymbol{z}' \mid \boldsymbol{0}]$ where $\boldsymbol{z}' \in \ker(\mathcal{R}_i^{(0)}) \setminus \ker(\mathcal{R}_i^{(1)})$. When executed in the canonical simulation of $\mathcal{P}_2$, the adversary outputs FALSE with overwhelming probability.

3. The profiles first disagree on some t_i or $\mathcal{Q}_i$. Then (by symmetry) there is an oracle constraint $\langle t_i, \mathcal{Q}_i, \boldsymbol{a}_i \rangle$ in $\mathcal{P}_1$ but no constraint of the form $\langle t_i, \mathcal{Q}_i, \cdot \rangle$ in $\mathcal{P}_2$ (otherwise a reordering of constraints would have caused the profiles to agree for a longer prefix).

   Consider what happens when $\mathcal{A}$ is executed in the canonical simulation for $\mathcal{P}_2$. Even though $\mathcal{A}$ is "designed for" $\mathcal{P}_1$, the invariant $\boldsymbol{r}_j = \mathcal{R}_j \times \boldsymbol{v}_{\mathsf{base}}$ still holds for $j < i$ because of the profiles' agreement. In the ith time through the main loop, $\mathcal{A}$ calls the oracle on $H(t_i, \mathcal{Q}_i \times \boldsymbol{v}_{\mathsf{base}})$. With overwhelming probability this query does *not* match any constraint in $\mathcal{P}_2$, so the response is chosen independently of everything else in the system. Let ρ^* denote the response of this oracle query.

   (a) If $\boldsymbol{a}_i \in \mathsf{span}(\mathcal{R}_{i-1})$ then $\ker(\mathcal{R}_i)$ is nonzero. In fact, we can choose a vector $\boldsymbol{z}' \in \ker(\mathcal{R}_i)$ with last component nonzero, and then set $\boldsymbol{z} = [\boldsymbol{z}' \mid \boldsymbol{0}]$. This is a valid choice of $\boldsymbol{z}$ for $\mathcal{A}$. But when executed in the canonical simulation of $\mathcal{P}_2$, the expression $\boldsymbol{z} \cdot \boldsymbol{r}_n$ contains a term involving ρ^* which is independent of everything else in the system. Hence $\mathcal{A}$ returns FALSE with overwhelming probability.

   (b) Otherwise, $\boldsymbol{a}_i \notin \mathsf{span}(\mathcal{R}_{i-1})$. Since the profiles of $\mathcal{P}_1$ and $\mathcal{P}_2$ eventually disagree, let $\mathcal{R}_j$ and $\mathcal{Q}_j$ refer to the profiles of $\mathcal{P}_1$ for $j \geq i$.

   Since $\mathcal{P}_1$ is normalized, every constraint is *useful.* Hence, $\boldsymbol{a}_i$ is in the span of all of the $\mathcal{Q}_j$'s and $\mathcal{R}_j$'s in the profile of $\mathcal{P}_1$. In other words, there is eventually a $\mathcal{Q}_k$ such that we can write $\mathcal{Q}_k = L_k \times \mathcal{R}_{k-1}$ for some L_k that assigns nonzero coefficient to row $\boldsymbol{a}_i$ of $\mathcal{R}_{k-1}$. Consider the smallest such k for which this holds, and ensure that $\mathcal{A}$ uses this value of L_k when appropriate.

   When running $\mathcal{A}$ in the simulation of $\mathcal{P}_2$, we see that syntactically the value ρ^* is not used between the ith and $(k-1)$th oracle query. Then on the kth oracle query, $\mathcal{A}$ calls $H(t_k; L_k \times \boldsymbol{r}_{k-1})$. Since the expression $L_k \times \boldsymbol{r}_{k-1}$ involves the special value ρ^* which is independent of everything else so far, it is only with negligible probability that this oracle query triggers a constraint. Instead, with overwhelming probability the result is chosen uniformly, independent of everything else so far. We take the response to this oracle query to be the new "special position" ρ^* and proceed by induction. The important invariant is simply that we have a value which is chosen independent of everything else in the system. Eventually this case (3b) becomes impossible — at the very least, at the final (nth) oracle query; hence, we eventually hit case (3a).

4. The profiles agree completely. But this cannot happen, since it implies $\mathcal{P}_1$ and $\mathcal{P}_2$ (after basis change) are equal — their $\mathcal{M}$ matrices are identical and their sets of oracle constraints are identical. Note that all constraints will be considered because the programs are normalized: all oracle constraints are *reachable*.

□

On the computational complexity of determining indistinguishability of Linicrypt programs. We claim that it is possible to determine whether two Linicrypt programs $\mathcal{P}_1 = (\mathcal{M}_1, \mathcal{C}_1)$, $\mathcal{P}_2 = (\mathcal{M}_2, \mathcal{C}_2)$ are indistinguishable, taking only polynomial time (in the size of their algebraic representations).

The main idea is to first normalize both programs, then iteratively try to construct a basis change relating the programs, as follows. The process mirrors the execution of $\mathsf{profile}(\mathcal{P}_1)$. First, (using any linear-algebraic procedure) determine whether their output matrices $\mathcal{M}_1$ and $\mathcal{M}_2$ have the same rowspace/kernel. If so, obtain a (partial) change of basis relating $\mathcal{M}_1$ and $\mathcal{M}_2$; otherwise conclude that the programs are distinguishable.

Then for each $\langle t_i, \mathcal{Q}_i, \boldsymbol{a}_i\rangle \in \mathcal{C}_1$, check whether a corresponding constraint $\langle t_i, \mathcal{Q}_i, \boldsymbol{a}'_i\rangle \in \mathcal{C}_2$ (under the partial basis change computed so far). If not, then conclude the programs are distinguishable. If $\boldsymbol{a}_i = \boldsymbol{a}'_i$ then continue. If neither $\boldsymbol{a}_i, \boldsymbol{a}'_i \in \mathcal{R}_{i-1}$ (where $\mathcal{R}_{i-1}$ is the matrix from $\mathsf{profile}$), then extend the basis change to bring $\boldsymbol{a}_i$ and $\boldsymbol{a}'_i$ into agreement. Otherwise, conclude that the programs are distinguishable.

At the end of this process, either a suitable basis change will have been obtained, or else it will be clear that the programs are distinguishable.

B Composing Gate-Level Garbling into Circuit Garbling

B.1 Composing Linicrypt Programs with Joint State

The security properties of a single gate-garbling (Definition 7) consider the behavior of GateGb in isolation. In particular, the free-XOR offset value Δ is *local* to the View program that defines security. However, in our circuit garbling scheme the same Δ is used across all gates. An important part of the proof is to argue why this is safe.

Definition 7 includes two conditions; we first deal with the condition of not leaking Δ. Towards a general shorthand notation, suppose an input-less Linicrypt program $\mathcal{P}$ has an internal variable Δ. Then we say that Δ *is unreachable* from [the output of] $\mathcal{P}$ if $\Pr[A^H(\mathcal{P}^H()) = \Delta]$ is negligible for all adversaries A. As pointed out in Section 2.5, this property can be expressed in terms of indistinguishability of Linicrypt programs.

Recall that we consider oracles whose first argument is a string/tweak. If $\mathcal{P}$ is an oracle program, and T is a set of strings, then we write $\mathcal{P}^{H(T;\cdot)}$ to mean that all of $\mathcal{P}$'s oracle queries have first argument in T.

Lemma 9. *Let $\mathcal{P}_1^{H(T_1,\cdot)}$, $\mathcal{P}_2^{H(T_2,\cdot)}$ be pure Linicrypt programs, where the number of inputs to $\mathcal{P}_2$ is one more than the number of outputs of $\mathcal{P}_1$, and where $T_1 \cap T_2 = \emptyset$.*

Define the following input-less Linicrypt programs:

```
$\mathrm{exp}_1^{H(T_1;\cdot)}()$:
    $\Delta, \vec{U} \leftarrow \mathbb{F}$
    $\vec{V} \leftarrow \mathcal{P}_1^{H(T_1;\cdot)}(\Delta, \vec{U})$
    return $\vec{V}$
```

```
$\mathrm{exp}_2^{H(T_2;\cdot)}()$:
    $\Delta, \vec{V} \leftarrow \mathbb{F}$
    $\vec{W} \leftarrow \mathcal{P}_2^{H(T_2;\cdot)}(\Delta, \vec{V})$
    return $\vec{V}, \vec{W}$
```

```
$\mathrm{exp}_3^{H(T_1 \cup T_2;\cdot)}()$:
    $\Delta, \vec{U} \leftarrow \mathbb{F}$
    $\vec{V} \leftarrow \mathcal{P}_1^{H(T_1;\cdot)}(\Delta, \vec{U})$
    $\vec{W} \leftarrow \mathcal{P}_2^{H(T_2;\cdot)}(\Delta, \vec{V})$
    return $\vec{V}, \vec{W}$
```

Then if Δ is unreachable from exp_1 and unreachable from exp_2, then Δ is unreachable from exp_3. In other words, composing $\mathcal{P}_1$ with $\mathcal{P}_2$ using the same Δ preserves their properties of individually *not leaking Δ, as long as $\mathcal{P}_1$ and $\mathcal{P}_2$ use distinct oracle tweaks.*

Proof. We start by modifying $\mathcal{P}_1$ and $\mathcal{P}_2$ without loss of generality. For $i \in \{1, 2\}$, let $\mathcal{R}_i$ denote the reachable space (as defined in Lemma 4) given the output of exp_i. Now modify $\mathcal{P}_i$ so that its output matrix is a basis for all of $\mathcal{R}_i$. These modifications certainly do not increase what vectors are reachable given exp_i, so they maintain the fact that Δ is unreachable in exp_i. Furthermore, they do not compromise the ability to compose $\mathcal{P}_1$ with $\mathcal{P}_2$. The original output of $\mathcal{P}_1$ is contained in its reachable space, so $\mathcal{P}_2$ can be modified to reconstruct this original output via linear combinations to $\mathcal{P}_1$'s modified output.

Next, we transform $\mathcal{P}_1$ and $\mathcal{P}_2$ into a common basis. Fix an algebraic representation $(\mathcal{M}_3, \mathcal{C}_3)$ for exp_3. Then we observe exp_1 and exp_2 can be obtained from the following algebraic modifications to exp_3:

- Removing the last several rows from the output matrix exp_3 results in the program exp_1 — *i.e.*, only $\mathcal{P}_1$'s output $\vec{V}$ is given.

- Removing all of the oracle constraints in $\mathcal{C}_3$ corresponding to $\mathcal{P}_1$ effectively removes all oracle calls in $\mathcal{P}_1$ and makes its internal base variables all uniform. Since $\mathcal{P}_1$'s output matrix has full rank (by the modifications above, it is a basis for $\mathcal{R}_1$), this modification causes $\vec{V}$ to be uniform in exp_3. Furthermore $\vec{V}$ is linearly independent of Δ, since otherwise Δ is in the simple linear-algebraic span of exp_1's output, contradicting our assumption. Hence, this modification to exp_3 results in program exp_2.

In what follows we will consider algebraic representations for exp_1 and exp_2 obtained in this way from exp_3, so that all three of these programs are "in the same basis."

Now suppose for contradiction that Δ is reachable in exp_3. This means there is a sequence of Linicrypt operations, starting with the output of exp_3, and making oracle calls that were also made in exp_3, finally resulting in Δ with probability 1. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ each output a basis for their individual reachable spaces, it is not necessary to ever make an oracle query that is reachable in exp_1 or in exp_2 individually. So without loss of generality, we can assume that in the Linicrypt sequence that culminates in Δ, the first oracle query is one that is unreachable in both exp_1 and exp_2. Let $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ denote the corresponding oracle constraint. We consider the following cases:

- There is no first oracle query. In other words, Δ can be obtained without making any oracle queries. But if Δ's reachability does not depend on any oracle queries, Δ still remains reachable if we remove all of the oracle constraints corresponding to $\mathcal{P}_1$. But such a modification gives us exp_2 and contradicts the fact that Δ is unreachable in exp_2.

- $t \in T_2$, meaning that the first query "belongs to" $\mathcal{P}_2$. But since no oracle queries in $\mathcal{P}_1$ have been used so far, this implies that the same constraint $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ would have been also reachable in $\exp_2$. This contradicts the fact that the constraint is unreachable in $\exp_2$.

- $t \in T_1$, meaning that the first query is one belonging to $\mathcal{P}_1$. Let $\mathcal{R}_1^+$ denote the span of all base variables (reachable or not) used in $\mathcal{P}_1$. In particular $\mathsf{row}(\Delta)$ is in $\mathcal{R}_1^+$. Similarly, let $\mathcal{R}_2^+$ denote the span of all base variables used in $\mathcal{P}_2$. We claim that $\mathcal{R}_2^+ \cap \mathcal{R}_1^+ \subseteq \mathsf{span}(\mathcal{R}_1 \cup \{\mathsf{row}(\Delta)\})$. Intuitively, the only common values used in $\mathcal{P}_1$ and $\mathcal{P}_2$ are Δ and those values explicitly passed as output of $\mathcal{P}_1$ (via its output $\mathcal{R}_1$). Any other values used in $\mathcal{P}_2$ are the result of oracle queries, made *after* $\mathcal{P}_1$ is finished, and so they are linearly independent of Δ and any value used in $\mathcal{P}_1$.

  Now, back to the oracle constraint $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle$ in question. This constraint is unreachable in $\exp_1$, so $\mathcal{Q}$ contains a row $\boldsymbol{q}$ that is unreachable in $\exp_1$; *i.e.*, $\boldsymbol{q} \in \mathcal{R}_1^+ \setminus \mathcal{R}_1$. Yet $\boldsymbol{q}$ is reachable in $\exp_3$, and without making any previous oracle queries, so $\boldsymbol{q} \in \mathcal{R}_2^+$. The only vectors in $(\mathcal{R}_1^+ \setminus \mathcal{R}_1) \cap \mathcal{R}_2^+$ are vectors of the form $\boldsymbol{q} = \boldsymbol{q}' + c \cdot \mathsf{row}(\Delta)$ where $\boldsymbol{q}' \in \mathcal{R}_1$ and $c \neq 0$.

  Yet, if such a $\boldsymbol{q}$ is reachable, then so is $\mathsf{row}(\Delta) = c^{-1}(\boldsymbol{q} - \boldsymbol{q}')$, since $\boldsymbol{q}'$ is reachable ($\boldsymbol{q}' \in \mathcal{R}_1$). Hence, Δ is already reachable before the first oracle query is made. By the same reasoning as in the first case, Δ is reachable in $\exp_2$ — a contradiction.

  Intuitively, the additional output of $\mathcal{P}_2$ must have helped to make $\boldsymbol{q}$ reachable when it was not reachable in the absence of $\mathcal{P}_2$'s output. But the only way $\mathcal{P}_2$'s output can help is by leaking Δ since Δ is the only unreachable variable in $\mathcal{P}_1$ that is also available in $\mathcal{P}_2$.

In each case we obtain a contradiction to the assumption that Δ is reachable from $\exp_3$. □

Now we deal with the other condition in Definition 7. Suppose $\widetilde{\mathcal{P}}_1$ and $\widetilde{\mathcal{P}}_2$ are Linicrypt programs. For $i \in \{1,2,3\}$ define $\widetilde{\exp}_i$ to be identical to $\exp_i$ except that $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$ are used in place of $\mathcal{P}_1$ and $\mathcal{P}_2$.

Lemma 10. *If* $\exp_1 \cong \widetilde{\exp}_1$ *and* $\exp_2 \cong \widetilde{\exp}_2$*, and* Δ *is unreachable from all four of these programs, then* $\exp_3 \cong \widetilde{\exp}_3$*.*

Proof. In the previous proof, we showed that the reachable oracle queries in $\exp_3$ are exactly the union of reachable oracle queries in $\exp_1$ and $\exp_2$ individually — i.e., their composition introduces no new reachable oracle queries.

The same is true of the useful queries, since the oracle query responses are isolated to either $\mathcal{P}_1$ or $\mathcal{P}_2$, both syntactically and in terms of linear independence.

Altogether, this implies that normalizing $\exp_1$ and $\exp_2$ and then composing them gives the same result as first composing them and normalizing the resulting $\exp_3$. Without loss of generality, assume $\exp_1, \exp_2, \widetilde{\exp}_1, \widetilde{\exp}_2$ are normalized. Then $\exp_1$ and $\widetilde{\exp}_1$ differ only by a basis change, as do $\exp_2$ and $\widetilde{\exp}_2$. This implies that $\exp_3$ and $\widetilde{\exp}_3$ differ by a basis change (importantly, the basis changes can be applied to the $\mathcal{P}_1$ and $\mathcal{P}_2$ parts independently). Thus $\exp_3 \cong \widetilde{\exp}_3$, as desired. □

B.2 Security Proof for Circuit Garbling (Lemma 8)

Proof. We start with the proof of prv security. That is, the distribution of (F, X, d) — induced by $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f)$ — does not leak whether X was generated as $X \leftarrow \mathsf{En}(e, x)$ or $X \leftarrow \mathsf{En}(e, x')$, when $f(x) = f(x')$. We proceed in a sequence of hybrids:

Hybrid 0: This is the "real world" in which the adversary receives (F, X, d) generated honestly, illustrated by $\mathsf{Hyb0}$ in Figure 5.

```
$\underline{\mathsf{Hyb0}(1^\lambda, f, x)\text{:}}$
  $\Delta \leftarrow \{0,1\}^\lambda$
  for each input wire $i$ of $f$:
    $W_i \leftarrow \mathbb{F}$
    $\sigma_i \leftarrow \{0,1\}$
    $X[i] := (W_i \oplus x_i\Delta, \sigma_i \oplus x_i)$
  for each gate $g$ in $f$, in topological order:
    let $g$ have input wires $i_1,\dots,i_m$, output wires $j_1,\dots,j_n$, functionality $\tau$
    $\vec{W}^{(in)} := (W_{i_1},\dots,W_{i_m})$
    $\sigma^{(in)} := \sigma_{i_1}\|\cdots\|\sigma_{i_m}$
    $(\vec{W}^{(out)};\vec{G}) \leftarrow \mathsf{GateGb}_\tau^{H(g,\cdot)}(\sigma^{(in)};\vec{W}^{(in)},\Delta)$
    $(W_{j_1},\dots,W_{j_n}) = \vec{W}^{(out)}$
    $\sigma^{(out)} := \sigma_{j_1}\|\cdots\|\sigma_{j_n} \leftarrow \{0,1\}^n$
    for $\chi$ in $\{0,1\}^m$:
      $v := \sigma^{(in)} \oplus \chi$
      $G'_\chi := H(\mathtt{color}\|g\|\chi;\vec{W}^{(in)} \oplus v\Delta) \oplus (\sigma^{(out)} \oplus \tau(v))$
    $F[g] := (\vec{G}; G'_{0^m},\dots,G'_{1^m})$
  for each output wire $i$ of $f$:
    $d[i,0] := H(\mathtt{out}\|i;W_i)$
    $d[i,1] := H(\mathtt{out}\|i;W_i \oplus \Delta)$
  return $F, X, d$
```

Figure 5: Hybrid 0 for the security proof.

```
$\underline{\mathsf{Hyb1}(1^\lambda, f, x)\text{:}}$
  for each wire $i$:
    $v_i :=$ truth value on wire $i$ when circuit input is $x$
  for each input wire $i$ of $f$:
    $W_i^* \leftarrow \mathbb{F}$
    $\chi_i^* \leftarrow \{0,1\}$;
    $X[i] := (W_i^*, \chi_i^*)$
  $\Delta \leftarrow \{0,1\}^\lambda$
  for each gate $g$ in $f$, in topological order:
    let $g$ have input wires $i_1,\dots,i_m$, output wires $j_1,\dots,j_n$, functionality $\tau$
*   $\vec{W}^{(in)} := (W_{i_1}^* \oplus v_{i_1}\Delta,\dots,W_{i_m}^* \oplus v_{i_m}\Delta)$
*   $\chi^{(in)} := \chi_{i_1}^*\|\cdots\|\chi_{i_m}^*$;     $\sigma^{(in)} := \chi^{(in)} \oplus (v_{i_1}\|\cdots\|v_{i_m})$
*   $(\vec{W}^{(out)};\vec{G}) \leftarrow \mathsf{GateGb}_\tau^{H(g,\cdot)}(\sigma^{(in)};\vec{W}^{(in)},\Delta)$
*   $(W_{j_1}^*,\dots,W_{j_n}^*) = \vec{W}^{(out)} \oplus (v_{j_1}\|\cdots\|v_{j_n})\Delta$
    $\chi_{j_1}^*\|\cdots\|\chi_{j_n}^* \leftarrow \{0,1\}^n$
    for $\delta$ in $\{0,1\}^m$:
      $\tilde{\chi} := \chi^{(in)} \oplus \delta$
      $v := v_{i_1}\|\cdots\|v_{i_m}$
      $G'_{\tilde{\chi}} := H(\mathtt{color}\|g\|\tilde{\chi};(W_{i_1}^*,\dots,W_{i_m}^*) \oplus \delta\Delta) \oplus (\sigma^{(out)} \oplus \tau(\delta \oplus v))$
    $F[g] := (\vec{G}; G'_{0^m},\dots,G'_{1^m})$
  for each output wire $i$ of $f$:
    $d[i,v_i] := H(\mathtt{out}\|i;W_i^*)$
    $d[i,\overline{v_i}] := H(\mathtt{out}\|i;W_i^* \oplus \Delta)$
  return $F, X, d$
```

Figure 6: Hybrid 1 of the security proof.

Hybrid 1: Following the proof sketch, we conceptually shift the hybrid from the garbler's perspective to the evaluator's perspective. Instead of maintaining the false wire label W_i for each wire, and deriving the others by XORing with Δ, we maintain the wire label that will be visible to the evaluator (called W_i^*). This is done by computing for each wire i the truth value v_i that is carried on each wire, then conceptually replacing each reference to W_i with $W_i^* \oplus v_i\Delta$, and so on.

Similarly, we choose random χ_i for each wire, and then set $\sigma_i = v_i \oplus \chi_i$ (rather than choosing σ_i randomly first). The hybrid is given in detail in Figure 6.

Hybrid 2: Observe that in the previous hybrid, the starred lines can be thought of as a function that takes previous W_i^* visible wire labels and Δ as input, then computes outgoing wire labels W_j^* and garbled gate information. In particular, these lines are essentially $\mathsf{View}^{H(g,\cdot)}(\chi, v)$ from Definition 7. The entire main loop (apart from the encryptions of color bits) is a composition of View programs in the sense of Section B.1.

Even though Δ is shared jointly among all of these instances of View (unlike in the security definition for garbled gates), Lemmas 9 and 10 imply that we can freely change the v values used by these Views, with negligible effect on the adversary's view. Technically speaking, we must also account for the other uses of Δ in the hybrid — in particular, the encryptions of color bits. Although these calls to H return only one bit, it is possible to show that they have the analogous properties allowing us to switch their payloads. They use distinct oracle tweaks, they are pseudorandom when Δ is local to each call, and their output does not individually leak Δ.

Overall, we can replace the v_i values freely within the implicit calls to View and in the payloads of color-bit-encryptions. This leaves only the use of v_i values in the garbled decoding information d. But these correspond to only the output wires of the circuit. We will replace the v_i values with their corresponding values when using input x' to the circuit. In that case, the output wires will take on the same values for both x and x'. Overall, this establishes prv security.

In obv security, the value d is not given in the adversary's view, and the goal is to prove that (F, X) alone do not leak the value x that was used to generate $X \leftarrow \mathsf{En}(e, x)$. However, with d no longer being produced, the above argument establishes that we may freely change the value of x chosen to derive the v_i wire truth values.

In aut security, the goal is to show that given (F, X), no adversary can produce $\tilde{Y} \neq \mathsf{Ev}(F, X)$ such that $\mathsf{De}(d, \tilde{Y}) \neq \bot$. The only way this is possible is if $\tilde{Y}$ contains some $d[i, \overline{v}_i]$ value. But $d[i, \overline{v}_i]$ is chosen as the result of an *unreachable* oracle query $H(\mathsf{out}\|i; W_i^* \oplus \Delta)$, so it is only with negligible probability that the adversary can produce it. □

C Gate Garbling Scheme Synthesis

C.1 Calculating the Size of the Formula

We can determine the sizes of the matrices representing GateGb and GateEv using the parameters in the shortcut. We consider GateGb first. Let $(\mathcal{M}_{gb}, \mathcal{C}_{gb})$ be the algebraic representation of GateGb. We fix the order of the base variables: first are m FALSE input wirelabels, followed by the single global constant Δ, and finally $\mathsf{calls_{gb}}$ oracle responses. The width of $\mathcal{M}_{gb}$ is $\mathsf{width_{gb}} = m+1+\mathsf{calls_{gb}}$. The number of rows of $\mathcal{M}_{gb}$ is $\mathsf{size}+n$. Therefore a single $\mathcal{M}_{gb}$ contains $\mathsf{width_{gb}} \times (\mathsf{size}+n)$ variables.

C_{gb} is composed of $\mathsf{calls_{gb}}$ constraints of the form $\langle \mathcal{Q}, \boldsymbol{a} \rangle$. The widths of both $\mathcal{Q}$ and $\boldsymbol{a}$ are $\mathsf{width_{gb}}$ since each oracle query takes some linear combination of the base variables as input. The height of $\mathcal{Q}$ is the arity of the random oracle arity. Since the $\boldsymbol{a}$ of the ith oracle constraint will always be a constant vector $\boldsymbol{a}_i = \{0\}^{m+1+i}, 1, \{0\}^{\mathsf{calls_{gb}}-i-1}$, we never have to create new variables for $\boldsymbol{a}$. We also apply the optimization from Section 3.2. Therefore, the total number of variables in C_{gb} is $\mathsf{width}_{\mathsf{gb}} \times \mathsf{arity} \times \mathsf{calls}_{\mathsf{gb}} - \mathsf{calls}_{\mathsf{gb}}(\mathsf{calls}_{\mathsf{gb}} + 1)/2$.

Let $(\mathcal{M}_{ev}, \mathcal{C}_{ev})$ be the algebraic representation of GateEv. Again, we fix the order of the base vars: m input wirelabels (with truth value $\sigma \oplus \chi$), followed by size ciphertexts, and finally $\mathsf{calls}_{\mathsf{ev}}$ oracle responses. The width of $\mathcal{M}_{ev}$ is then $\mathsf{width}_{\mathsf{ev}} = m + \mathsf{size} + \mathsf{calls}_{\mathsf{ev}}$. The number of rows of $\mathcal{M}_{ev}$ is the number of outputs n. Then, a single $\mathcal{M}_{ev}$ contains $\mathsf{width}_{\mathsf{ev}} \times n$ variables. $\mathcal{C}_{ev}$ is created in the same way as $\mathcal{C}_{gb}$, so it contains $\mathsf{width}_{\mathsf{ev}} \times \mathsf{arity} \times \mathsf{calls}_{\mathsf{ev}}$ variables.

We now calculate the number of variables needed to synthesize the garbler using an adaptive oracle:

$$\begin{aligned}
\mathsf{width}_{\mathsf{gb}} &= m + 1 + \mathsf{calls}_{\mathsf{gb}} \\
|\mathcal{M}_{gb}| &= \mathsf{width}_{\mathsf{gb}}(\mathsf{size} + n) \\
|\mathcal{C}_{gb}| &= \mathsf{width}_{\mathsf{gb}} \times \mathsf{arity} \times \mathsf{calls}_{\mathsf{gb}} - \mathsf{calls}_{\mathsf{gb}}(\mathsf{calls}_{\mathsf{gb}} + 1)/2 \\
|\mathsf{GateGb}| &= 2^m(|\mathcal{M}_{gb}| + |\mathcal{C}_{gb}|)
\end{aligned}$$

And the evaluator:

$$\begin{aligned}
\mathsf{width}_{\mathsf{ev}} &= m + \mathsf{size} + \mathsf{calls}_{\mathsf{ev}} \\
|\mathcal{M}_{ev}| &= \mathsf{width}_{\mathsf{ev}} \times n \\
|\mathcal{C}_{ev}| &= \mathsf{width}_{\mathsf{ev}} \times \mathsf{arity} \times \mathsf{calls}_{\mathsf{ev}} - \mathsf{calls}_{\mathsf{ev}}(\mathsf{calls}_{\mathsf{ev}} + 1)/2 \\
|\mathsf{GateEv}| &= 2^m(|\mathcal{M}_{ev}| + |\mathcal{C}_{ev}|)
\end{aligned}$$

Additionally, we require 2^{2m} basis change matrices $\vec{B}$. Each basis change has size $\mathsf{width}_{\mathsf{gb}} \times \mathsf{width}_{\mathsf{gb}}$. This adds $|\vec{B}| = 2^{2m}\mathsf{width}_{\mathsf{gb}}^2$ variables to the formula.

C.2 Synthesized Garbling Schemes

We present instances of the satisfiable schemes from Figure 4. We use a shorthand here (and in our tool Linisynth) to condense the different versions of GateGb and GateEv. It works like this:

1. GateGb: When S is a set of indices, the notation "$[S]W$" refers to nonlinear behavior "if $\sigma \in S$ then W else 0^λ"

2. GateEv: When S is a set of indices, the notation "$[S]W$" refers to nonlinear behavior "if $\chi \in S$ then W else 0^λ"

Without further ado, here are some gate-garbling schemes, output directly from our Linisynth tool. We use $A, B, \ldots$ to denote input wire labels.

free-xor $\quad$ $\mathsf{size} = 0$ $\quad$ $\mathsf{calls}_{\mathsf{gb}} = 0$ $\quad$ $\mathsf{adaptive} = 0$

$\oplus : \{0,1\}^2 \to \{0,1\}$ $\quad$ $\mathsf{arity} = 1$ $\quad$ $\mathsf{calls}_{\mathsf{ev}} = 0$ $\quad$ $\mathsf{time} = 1\mathsf{s}$

```
$\mathsf{GateGb}^H(\sigma, A, B, \Delta)$ :
    return $A + B$

$\mathsf{GateEv}^H(\chi, A^*, B^*)$ :
    return $A^* + B^*$
```

half-gate size $= 2$ calls$_{\mathsf{gb}} = 4$ adaptive $= 0$

$\wedge : \{0,1\}^2 \to \{0,1\}$ arity $= 1$ calls$_{\mathsf{ev}} = 2$ time $=$ 5s

```
GateGb^H(σ, A, B, Δ) :
  $h_1 = H(A)$
  $h_2 = H(A + \Delta)$
  $h_3 = H(A + B)$
  $h_4 = H(A + B + \Delta)$
  $G_0 = [0,2]\Delta + h_3 + h_4$
  $G_1 = A + B + [0,2]\Delta + h_1 + h_2 + h_3 + h_4$
  $C_0 = B + [0]\Delta + [0,2]h_1 + [1,3]h_2 + [1,2]h_3 + [0,3]h_4$
  return $G_0, G_1, C_0$

GateEv^H(χ, A*, B*, G_0, G_1) :
  return $[1,3]A^* + [0,2]B^* +$
         $[0,1]G_0 + [1,3]G_1 +$
         $H(A^*) + H(A^* + B^*)$
```

mux size $= 2$ calls$_{\mathsf{gb}} = 4$ adaptive $= 1$

MUX : $\{0,1\}^3 \to \{0,1\}$ arity $= 1$ calls$_{\mathsf{ev}} = 2$ time $=$ 29s

```
GateGb^H(σ, A, B, C, Δ) :
  $h_0 = H(A + B + C)$
  $h_1 = H(A + B + C + \Delta)$
  $h_2 = H(A + B + h_0 + h_1)$
  $h_3 = H(A + B + \Delta + h_0 + h_1)$
  $G_0 = [0,3,4,7]\Delta + h_0 + h_1$
  $G_1 = A + B + C + [0,3,4,7]\Delta + h_0 + h_1 + h_2 + h_3$
  $C_0 = A + C + [0,3]\Delta + [1,2,4,7]h_0 + [0,3,5,6]h_1 + h_3$
  return $G_0, G_1, C_0$

GateEv^H(χ, A*, B*, G_0, G_1) :
  $h_0 = H(A^* + B^* + C^*)$
  $h_1 = H(A^* + B^* + G_0)$
  return $[0,3,4,7]A^* + [1,2,5,6]B^* + [0,3,4,7]C^* +$
         $[0,1,2,3]G_0 + [1,2,5,6]G_1 +$
         $h_0 + h_1$
```

eq size $= 2$ calls$_{\mathsf{gb}} = 4$ adaptive $= 1$

$\textsc{eq} : \{0,1\}^4 \to \{0,1\}$ arity $= 1$ calls$_{\mathsf{ev}} = 2$ time $=$ 6m

```
$\underline{\mathsf{GateGb}^H(\sigma, A, B, C, D, \Delta):}$
    $h_0 = H(A + C)$
    $h_1 = H(A + C + \Delta)$
    $h_2 = H(A + C + h_0 + h_1)$
    $h_3 = H(A + C + \Delta + h_0 + h_1)$
    $G_0 = A + C + [2,3,6,7,8,9,12,13]\Delta + h_0 + h_1 + h_2 + h_3$
    $G_1 = A + B + C + D + [0,2,5,7,8,10,13,15]\Delta + h_0 + h_1$
    $C_0 = [1,11,4,14]\Delta +$
            $[1,3,4,6,9,11,12,14]h_0 + [0,2,5,7,8,10,13,15]h_1 +$
            $[1,2,4,7,8,11,13,14]h_2 + [0,3,5,6,9,10,12,15]h_3$
    return $G_0, G_1, C_0$

$\underline{\mathsf{GateEv}^H(\chi, A^*, B^*, C^*, D^*, G_0, G_1):}$
    $h_0 = H(A^* + C^*)$
    $h_1 = H(B^* + D^* + G_1)$
    return $[0,2,5,7,8,10,13,15]A^* + [0,3,5,6,9,10,12,15]B^* +$
           $[0,2,5,7,8,10,13,15]C^* + [0,3,5,6,9,10,12,15]D^* +$
           $[2,3,6,7,8,9,12,13]G_0 \quad + [0,3,5,6,9,10,12,15]G_1 +$
           $h_0 + h_1$
```