Linicrypt: A Model for Practical Cryptography

Brent Carmer(✉) and Mike Rosulek

Oregon State University, Corvallis, USA
{carmerb,rosulekm}@eecs.oregonstate.edu

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 [34] 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 [13,47] for collision-resistance; the Even-Mansour [18] and Feistel [35] constructions for PRPs; NMAC, HMAC [31], and VMAC [32] for authenticity; Naor's commitment scheme [41].

Authors supported by NSF award 1149647.

M. Robshaw and J. Katz (Eds.): CRYPTO 2016, Part III, LNCS 9816, pp. 416–445, 2016.
DOI: 10.1007/978-3-662-53015-3_15

- 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 [50] for authenticated encryption, CMC mode [23] for disk encryption, XE/XEX modes [49] for tweakable block ciphers, PMAC [12] for authentication.
- Signing algorithms for lightweight one-time signature schemes like those of Lamport [33] and Winternitz [52] consist purely of calls to a one-way or [target] collision-resistant hash function. Variants like W-OTS+ [25] incorporate XOR operations. Few-time signature schemes like HORS and variants [45,48] also use only a random oracle. These simple signature schemes can be composed to give many-use signature schemes using Merkle trees [39] and derivatives thereof [11,14–16,21,40,44]. These extensions do not introduce any additional operations on the atomic field elements.
- Practical constructions of garbled circuits [22,29,30,42,53] simply use XOR and calls to an underlying hash function/KDF, while the construction of [46] 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 Sect. 2 we formally define Linicrypt, develop techniques to reason about its algorithms, and prove our main technical result. Later in Sect. 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 [30].

1.2 Related Work and Inspiration

Minicrypt. Linicrypt is inspired in name by Impagliazzo's [26] Minicrypt, which refers to a hypothetical world in which one-way functions exist but no "fancier" cryptography is possible. Minicrypt is formalized (as in [27]) 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 [51]. 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.* [17] 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.* [43] 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.* [1–3] 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 [7] 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 [38]. 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.* [6] define a model for *arithmetic cryptography*, building on earlier work by Ishai *et al.* [28]. 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.* [53]. 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.* [36] extend the model of linear garbling and characterize security in terms of linear-algebraic properties like span. They generalize the garbling scheme of [53] to natively support low-degree polynomials (not just AND-gates).

Later in Sect. 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 [5] and optimal structure-preserving signatures [7]. Both of these results synthesize constructions involving bilinear pairings.

Malozemoff *et al.* [37] 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.* [24] extended the synthesis to authenticated encryption modes built from tweakable block ciphers.

Prior work of Gagné *et al.* [19,20] developed techniques for automated proofs of security for (CPA-secure) block cipher modes. Akinyele *et al.* [4] use an SMT solver to automate transformations of pairing-based signature schemes.

In all of the works involving block cipher modes [19,20,24,37] 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] In [37] 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 [19,20] the techniques involve a logic that uses only local invariants.

$\{1, \ldots, |\mathsf{cmds}|\}$, 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 $1 \leq j \leq \mathsf{in}$ [retrieve a value from input]
- $(\textsc{samp})$ [sample an element of $\mathbb{F}$]
- $(\textsc{lin}, c_1, \ldots, c_{i-1})$, where each $c_j \in \mathbb{F}$ [perform a linear combination of values]
- $(\textsc{hash}, t, j_1, \ldots, j_k)$, where $t \in \{0,1\}^*$ and $j_1, \ldots, j_k < i$ [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:

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

2.2 Mixed Linicrypt Programs and 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 $pk \stackrel{?}{=} H^{(m-x)}(\sigma)$.

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 Fig. 1 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,

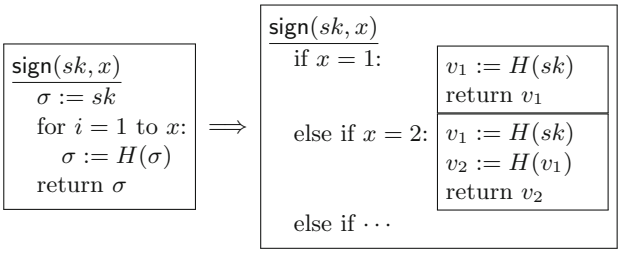

Fig. 1. The signing algorithm for one-time Winternitz signatures as a *mixed Linicrypt program*. Each inner box on the right-hand side is a *pure* Linicrypt programs, $\mathsf{sign}(\cdot, x)$, for fixed x.

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 SAMP, HASH, or 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:

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

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

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

$\mathcal{M}$ 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 $\boldsymbol{a} \cdot \boldsymbol{v}_{\mathsf{base}}$.

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

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:	matrix representation:
$v_1 \leftarrow \mathbb{F}$	1: (SAMP)	base	$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \end{bmatrix}$
$v_2 \leftarrow \mathbb{F}$	2: (SAMP)	base	
$v_3 := v_1 - v_2$	3: (LIN, 1, −1)	derived	
$v_4 := H(\mathtt{foo}, v_3, v_2)$	4: (HASH, foo, 3, 2)	base	
$v_5 := v_4 + v_1$	5: (LIN, 1, 0, 0, 1)	derived	
return (v_4, v_5)	// $\mathsf{out} = (4, 5)$		

algebraic representation:

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

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 $\mathcal{P}$.

$\underline{\mathcal{S}^{\mathcal{A}}_{\mathcal{P}}()}$:
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 $\mathcal{Q}\boldsymbol{v}_{\mathsf{base}} = \mathcal{Q}'\boldsymbol{v}_{\mathsf{base}} = \boldsymbol{q}$:
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 $cache[t; \boldsymbol{q}]$

(1)

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 $\mathcal{A}$:*

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

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.

Proof (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 $n(n+1)/2|\mathbb{F}|$.

2.4 Linear Transformations, Basis Changes and Composition

The algebraic representation for Linicrypt programs turns out to be convenient, as we can perform linear-algebraic manipulations to Linicrypt programs.

For instance, consider **applying a linear transformation** to a Linicrypt program. Let $\mathcal{P} = (\mathcal{M}, \mathcal{C})$ be a Linicrypt program. Recall that the width of the vectors in $\mathcal{M}$ and $\mathcal{C}$ is base. Now let B be a $\mathsf{base} \times \mathsf{base}$ matrix with entries in $\mathbb{F}$ and consider the Linicrypt representation $(\mathcal{M}B, \mathcal{C}B)$, where

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

When B is an invertible matrix, 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 and 2 in Eq. 1. Since B is invertible, this additional statement has no effect on the distribution of $\boldsymbol{v}_{\mathsf{base}}$.

Composition. We can use the idea of a linear transformation 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 algebraic representation of $\mathcal{P}'$ implicitly treats all of its input variables as linearly independent. So the case when $\mathcal{M}$ has full rank is easiest. To compose the programs, one simply applies a basis change to either program to align $\mathcal{P}$'s output variables ($\mathcal{M}$) and $\mathcal{P}'$'s input variables (expressed as $[I \mid \mathbf{0}]$, where I is the $\mathsf{out} \times \mathsf{out}$ identity matrix), and similarly align the oracle constraints of the programs. If such a basis change has been applied, then the composed program's output is characterized by $\mathcal{M}'$ and its oracle constraints are simply $\mathcal{C} \cup \mathcal{C}'$.

However, in general the output of $\mathcal{P}$ may have linear correlations, and this can have a serious effect on the behavior of $\mathcal{P}'$. Take for example the case where $\mathcal{P}'$ takes two input variables (v_1, v_2) and outputs $H(v_1) - H(v_2)$. Then the behavior of $\mathcal{P}'$ is qualitatively different when v_1 and v_2 are linearly independent vs. when they are correlated as $v_1 = v_2$, for instance.

In general, we consider applying a linear transformation to $\mathcal{P}'$ that "collapses" the appropriate base variables (they become associated with the same vector in the algebraic representation). Collapsing input base variables may result in the collapse of oracle queries that use these variables. In the example above, $H(v_1)$ and $H(v_2)$ are themselves base variables which are linearly independent in general; yet they collapse to the same base variable when $v_1 = v_2$.

Hence, to compose $\mathcal{P}$ with $\mathcal{P}'$ we consider a linear transformation Γ applied to $\mathcal{P}'$, with the following properties:

1. Γ aligns the input variables of $\mathcal{P}'$ (the first out base variables) with the output $\mathcal{M}$ of $\mathcal{P}$. That is, $\mathcal{M} = [I \mid \mathbf{0}] \times \Gamma$ where I is the $\mathsf{out} \times \mathsf{out}$ identity matrix.
2. Γ consistently aligns the oracle queries of $\mathcal{P}'$ to those in $\mathcal{P}$. That is, if $\langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}'\Gamma$, and $\langle t, \mathcal{Q}, \boldsymbol{a}' \rangle \in \mathcal{C}$, then $\boldsymbol{a} = \boldsymbol{a}'$.
3. Γ collapses appropriate oracle constraints in $\mathcal{P}'$: that is, if Γ causes (previously distinct) oracle constraints to now share the same t and $\mathcal{Q}$ components, then they must now also share the same $\boldsymbol{a}$ component. More formally, the constraints in $\mathcal{C}'\Gamma$ should all have distinct $t, \mathcal{Q}$ values. However, note that $\mathcal{C}'\Gamma$ may have fewer constraints than $\mathcal{C}'$ due to collapses induced by Γ.
4. Γ should only collapse base variables that are absolutely required by the above conditions. In other words, the rank of Γ should be as large as possible given the above constraints. Note that if $\mathcal{M}$ has full rank, then Γ will indeed be a basis change. However, in general Γ may not be a basis change — this is consistent with the fact that feeding linearly correlated values into $\mathcal{P}'$ may indeed fundamentally change its behavior. A basis change exactly preserves behavior.

Given such a transformation Γ, then $(\mathcal{M}'\Gamma, \mathcal{C} \cup \mathcal{C}'\Gamma)$ is an algebraic representation for the composition of programs $\mathcal{P}' \circ \mathcal{P}$.

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: (HASH, foo, 1)	
$v_3 := v_1 - v_2$	3: (LIN, 1, −1)	
$v_4 := H(\texttt{bar}, v_3)$	4: (HASH, bar, 3)	
$v_5 := H(\texttt{baz}, v_3)$	5: (HASH, 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 cmds[2] from (HASH, 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.* 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 $H(H(H(v)))$.

```
normalize(P = (M, C)):
  Reachable := rows(M)
  C' := ∅
  until C' reaches a fixed point:
    for each ⟨t, Q, a⟩ ∈ C \ C':
      if rows(Q) ⊆ span(Reachable):
        add a to Reachable
        add ⟨t, Q, a⟩ to C'

  Useless := ∅
  until Useless reaches a fixed point:
    V := (multiset of) all row vectors in M and C' \ Useless
    for each ⟨t, Q, a⟩ ∈ C' \ Useless:
      if a ∉ span(V \ {a}):
        add ⟨t, Q, a⟩ to Useless

  C'' := C' \ Useless

  return (M, C'')
```

Fig. 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 $\mathcal{M}$.

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 Fig. 2. In the full version we prove the following:

Lemma 4. *If $\mathcal{P}$ is an input-less Linicrypt program, then* $\mathsf{normalize}(\mathcal{P}) \cong \mathcal{P}$ *(Fig. 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 (Proof Sketch). 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 $\boldsymbol{r} = \mathcal{R} \times \boldsymbol{v}_{\mathsf{base}}$, 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 nontrivial linear relations — as a matrix, the set of reachable vectors has a nontrivial kernel. A canonical distinguisher chooses some element $\boldsymbol{z}$ from this kernel and tests whether $\boldsymbol{z}^\top \boldsymbol{r} = 0$. By construction, $\boldsymbol{z}^\top \boldsymbol{r} = \boldsymbol{z}^\top \mathcal{R} \boldsymbol{v}_{\mathsf{base}}$. Since $\boldsymbol{z} \in \ker(\mathcal{R})$, the distinguisher always outputs true in the presence of $\mathcal{P}_1$.

Now the challenge is to show that, for some choice of $\boldsymbol{z} \in \ker(\mathcal{R})$, 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 $\boldsymbol{z}$ 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.

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 [30], 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 wirelabel as $W \oplus \Delta$; more generally, the wirelabel 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 wirelabel 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 wirelabels), 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 [8] is used in all practical garbling schemes. The idea is to append to each wirelabel 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 wirelabels. The evaluator selects and decrypts the correct ciphertext indicated by the color bits of the input wirelabels 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 wirelabel 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 Sect. 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: $\boldsymbol{W} = W_1, \ldots, W_n$. Operations over vectors are computed componentwise. For instance, $\boldsymbol{A} \oplus \boldsymbol{B} = A_1 \oplus B_1, \ldots, A_n \oplus B_n$. 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 $\boldsymbol{W} = W_1, \ldots, W_n$ are a vector of FALSE wirelabels, then $\boldsymbol{W} \oplus x\Delta$ is a vector of wirelabels 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:

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

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

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 wirelabels encoding x on the input wires, the result of evaluating the gate is the wirelabels encoding $\tau(x)$ on the output wires.

Definition 6. *A Free-XOR-compatible garbled gate* $(\mathsf{GateGb}, \mathsf{GateEv})$ *correctly computes 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* $\boldsymbol{A} = A_1, \ldots, A_m$*, global Free-XOR constant* Δ*:*

$$(\boldsymbol{C}, \boldsymbol{G}) \leftarrow \mathsf{GateGb}(\sigma;\, \boldsymbol{A}, \Delta) \implies \mathsf{GateEv}(\chi;\, \boldsymbol{A} \oplus x\Delta, \boldsymbol{G}) = \boldsymbol{C} \oplus \tau(x)\Delta$$

Security. One important consideration is that in the free-XOR setting, the labels of different wires can have linear correlations. The gate should be secure even for such correlated input wirelabels.[2]

We define security in terms of the evaluator's view in a typical garbling scenario. Then we define $\mathsf{View}_R^H(\chi, x)$ to encapsulate the information the evaluator sees for this gate, when the visible color bits are χ, the logical gate inputs are x, and the input wirelabels have correlations described by an $m \times m$ matrix R.

$\mathsf{View}_R^H(\chi, x)$:
$\Delta, r_1, \ldots, r_m \leftarrow \{0,1\}^\lambda$
$\boldsymbol{A} = (A_1, \ldots, A_m) := R \times [r_1, \ldots, r_m]$
$(\boldsymbol{C}, \boldsymbol{G}) \leftarrow \mathsf{GateGb}^H(\chi \oplus x;\ \boldsymbol{A}, \Delta)$
return $(\boldsymbol{A} \oplus x\Delta,\ \boldsymbol{G},\ \boldsymbol{C} \oplus \tau(x)\Delta)$

We call R **non-degenerate** if no row of R is all-zeroes, as that would lead to a zero wirelabel (whose complementary wirelabel would immediately leak Δ). In particular, if $R = I$ then the wirelabels are independent.

Importantly, if GateGb^H is a Linicrypt program and parameters χ and x are fixed, then $\mathsf{View}_R^H(\chi, x)$ is a input-less Linicrypt program. We can therefore apply the results of Sect. 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, $x \in \{0,1\}^m$, *all non-degenerate* $R \in \{0,1\}^{m \times m}$, *and all polynomial-time oracle algorithms* A, *the probability* $\Pr[A^H(\mathsf{View}_R^H(\chi, x)) = \Delta]$ *is negligible in* λ,
2. *for all* $\chi, x, x' \in \{0,1\}^m$ *and all non-degenerate* $R \in \{0,1\}^{m \times m}$, *we have* $\mathsf{View}_R^H(\chi, x) \cong \mathsf{View}_R^H(\chi, x')$.

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. Furthermore, this should hold for all ways that the input wire labels could be correlated.

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 Fig. 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 wirelabel W_i encoding FALSE; $W_i \oplus \Delta$ will encode TRUE. These false wirelabels are chosen uniformly for input wires. Thereafter, we process gates in topological

[2] In fact, some natural garbled gate constructions are secure for independent input wirelabels but insecure when they are correlated, as illustrated strikingly in [9].

order. Each gate-garbling operation determines the garbled-gate information $\boldsymbol{G}$ as well as the FALSE wirelabels 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 Fig. 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 wirelabels 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 and 7).*

Then the garbling scheme in Fig. 3 satisfies the prv*,* aut*, and* obv *security definitions of* [10] *in the random oracle model, for circuits expressed in terms of $\mathbb{B}$-gates.*

Proof (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 $X = \mathsf{En}(e, x')$. The proofs of the other security properties obv & aut follow using standard modifications.

We show a sequence of hybrids, beginning with an interaction in which (F, X, d) are generated with $X = \mathsf{En}(e, x)$. In this initial hybrid, Gb is written in terms of what the garbler sees/knows. The only "persistent" values maintained throughout the main loop are the FALSE wirelabels W_i and select bits σ_i. We rearrange Gb to instead be in terms of what the evaluator sees: the "visible" wirelabels W^* and their color bits χ_i. We achieve this change by using x to compute 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.

After this change, each main loop is a Linicrypt program that takes the previously-computed visible wirelabels, along with Δ, and computes the next garbled gate and output wirelabels (we ignore the encryptions of color bits for now).

$\underline{\mathsf{Gb}^H(1^\lambda, f)}$:
  $\Delta \leftarrow \{0,1\}^\lambda$
  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 τ
    $\boldsymbol{W}^{(in)} := (W_{i_1}, \ldots, W_{i_m})$
    $\sigma^{(in)} := \sigma_{i_1} \| \cdots \| \sigma_{i_m}$; $\sigma^{(out)} := \sigma_{j_1} \| \cdots \| \sigma_{j_n}$
    $(\boldsymbol{W}^{(out)}; \boldsymbol{G}) \leftarrow \mathsf{GateGb}_\tau^{H(g,\cdot)}(\sigma^{(in)}; \boldsymbol{W}^{(in)}, \Delta)$
    $(W_{j_1}, \ldots, W_{j_n}) := \boldsymbol{W}^{(out)}$
    for χ in $\{0,1\}^m$:
      $v := \sigma^{(in)} \oplus \chi$
      $G'_\chi := H(\mathtt{color} \| g \| \chi; \boldsymbol{W}^{(in)} \oplus v\Delta) \oplus (\sigma^{(out)} \oplus \tau(v))$
    $F[g] := (\boldsymbol{G}; G'_{0^m}, \ldots, G'_{1^m})$
  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 τ
    $\chi^{(in)} := \chi_{i_1} \| \cdots \| \chi_{i_m}$
    $(\boldsymbol{G}; G'_{0^m}, \ldots, G'_{1^m}) := F[g]$
    $(W_{j_1}^*, \ldots, W_{j_n}^*) \leftarrow \mathsf{GateEv}_\tau^{H(g,\cdot)}(\chi^{(in)}; W_{i_1}^*, \ldots, W_{i_m}^*, \boldsymbol{G})$
    $\chi_{j_1} \| \cdots \| \chi_{j_n} := H(\mathtt{color} \| g \| \chi^{(in)}; W_{i_1}^*, \ldots, W_{i_m}^*) \oplus G'_{\chi^{(in)}}$
  for each output wire i of f:
    $Y_i := H(\mathtt{out} \| i; W_i^*)$
  return Y

Fig. 3. Gate-level garbling composed into a circuit garbling scheme.

In fact, such a computation is precisely $\mathsf{View}_R(\chi, v)$ defined above, for some appropriate R that describes the correlations among previous input wirelabels.

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. To do this, we prove a general composition lemma which shows 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.

We use similar reasoning to handle the color bits, since they are not strictly within the scope of Linicrypt (they use distinct oracle calls and do not leak Δ). 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 wirelabels, 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.

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 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.

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 and 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) can be expressed in terms of composing $\mathsf{View}_R(\chi, x)$ (which generates input wirelabels along with the garbled gate information) with $\mathsf{GateEv}(\chi, \cdot)$ in a particular way. We can apply the concepts of Sect. 2.4 to reason about their composition.

We make some simplifiying observations that lead us to synthesize only "minimal" gate garbling schemes:

- Correctness needs to hold only for independently distributed input wirelabels ($R = I$). In this setting, the wirelabel inputs to GateEv will have full rank.
- We can assume the garbled gate information has full rank. If any linear dependencies existed, then the same dependencies must exist in $\mathsf{GateGb}(\sigma, \cdot)$ for all σ, or else security is trivially violated (malicious evaluator can obtain information about σ by detecting a linear dependency among garbled gate info). Hence the correlations can be removed from *all* $\mathsf{GateGb}(\sigma, \cdot)$ and reconstructed if needed in *all* $\mathsf{GateGb}(\chi, \cdot)$. The result would be a smaller but equivalent & secure scheme.
- The *entire* input to GateEv (garbled gate information and input wirelabels *together*) has full rank. If there is a linear dependency between garbled gate information and input wirelabels, then the same dependency must exist regardless of σ, or else security will be trivially violated. Then again, the dependency could be removed from *all* $\mathsf{GateGb}(\sigma, \cdot)$ and reconstructed by *all* $\mathsf{GateGb}(\chi, \cdot)$, resulting in a smaller scheme.

We therefore consider a composition of $\mathsf{View}_R(\chi, x)$ and $\mathsf{GateEv}(\chi, \cdot)$ in which the input to GateEv is of full rank. This simplifies the task, since it now suffices to find a *basis change* to GateEv that aligns it with the corresponding output of $\mathsf{View}_R(\chi, x)$.

Let $\mathcal{M}_{R,\chi,x}$ denote the output matrix of $\mathsf{View}_R(\chi, x)$. We split this matrix into a top and bottom: $\mathcal{M}^{top}_{R,\chi,x}, \mathcal{M}^{bot}_{R,\chi,x}$, where the top matrix corresponds to the input wirelabels for x along with garbled gate information, while the bottom matrix corresponds to the output wirelabels for the result $\tau(x)$.

Following Sect. 2.4, we seek a basis change B such that $\mathcal{M}^{top}_{R,\chi,x} = [I \mid 0] \times B$, which represents the input base variables of $\mathsf{GateEv}(\chi, \cdot)$. 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 $\mathcal{M}^{bot}_{R,\chi,x}$ (under the basis change). In other words, the wirelabels that GateEv outputs always coincide with the "correct" wirelabels specified by View_R.

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 $x = \sigma \oplus \chi$, and that we have restricted $R = I$. We use $(\mathcal{M}_{R,\chi,x}, \mathcal{C}_{R,\chi,x})$ to refer to the algebraic representation of $\mathsf{View}_R(\chi, x)$, and use $(\mathcal{M}_{\mathsf{GateEv},\chi}, \mathcal{C}_{\mathsf{GateEv},\chi})$ to denote the algebraic representation of $\mathsf{GateEv}(\chi, \cdot)$.

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

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 $\langle t, \mathcal{Q} \times B, \boldsymbol{a} \times B\rangle \in \mathcal{C}_{R,\chi,x}$ is expressed as a logical-OR of $\mathsf{calls}_{\mathsf{gb}}$ equality checks.

Security, Condition 1. The first condition of Definition 7 is that $\mathsf{row}(\Delta)$ is unreachable (in the sense of Fig. 2). If 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 iteratively as in Fig. 2 is impractical in a SAT formula, so we employ a trick.

Our idea is to *guess* a basis change B that maps the reachable space to some canonical form that is easily testable by the SAT solver. In particular, consider a basis change B under which the reachable vectors are exactly those that have zero in their rightmost several positions. The SAT formula can easily check for such a condition. To check that our guess for B indeed maps the reachable subspace to the desired canonical 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 write $(\mathcal{M}_{R,\chi,x}, \mathcal{C}_{R,\chi,x})$ to denote the algebraic representation of $\mathsf{View}_R(\chi, x)$, which can be obtained in a systematic way from the algebraic representation of $\mathsf{GateGb}(\chi; \cdot)$ (which comprise the existentially quantified variables of the SAT formula). We use $\mathsf{row}(\Delta)$ to refer to the appropriate vector in this representation.

$$\begin{aligned}&\forall \sigma, \chi \in \{0,1\}^m, \textit{non-degenerate } R : \exists B, B^{-1} : \\ &B \times B^{-1} = I \wedge \neg \mathsf{RightZeroes}(\mathsf{row}(\Delta) \times B) \wedge \mathsf{RightZeroes}(\mathcal{M}_{R,\chi,x} \times B) \\ &\qquad \wedge \big[\forall \langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}_{R,\chi,x} : \mathsf{RightZeroes}(\mathcal{Q} \times B) \Rightarrow \mathsf{RightZeroes}(\boldsymbol{a} \times B)\big]\end{aligned} \tag{2}$$

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), 2^{m^2} choices of R, and $\mathsf{calls}_{\mathsf{gb}}$ constraints) and are explicitly expanded in the formula we pass to the SMT solver.

Security, Condition 2. The second condition of Definition 7 is that $\mathsf{View}_R(\chi, x)$ and $\mathsf{View}_R(\chi, x_0)$ are indistinguishable. Here we fix x_0 and show indistinguishability with respect to this fixed $\mathsf{View}_R(\chi, x_0)$. Since the programs involved are inputless Linicrypt programs, from Theorem 5 it suffices to show that they differ by a basis change after normalization (unreachable and useless oracle queries removed).

We make an assumption that all reachable oracle constraints in $\mathsf{View}_R(\chi, x)$ are in fact useful, and hence we can only synthesize gate-garbling schemes with this property. However, if a secure scheme has reachable and useless constraints in some $\mathsf{View}_R(\chi, x = \chi \oplus \sigma)$, then the same constraint must be also reachable and useless in all $\mathsf{View}_R(\chi, x' = \chi \oplus \sigma')$ by security. Hence it can be removed from every $\mathsf{GateGb}(\sigma; \cdot)$ resulting in an even less expensive yet equivalent and secure gate-garbling scheme.

To show that $\mathsf{View}_R(\chi, x)$ and $\mathsf{View}_R(\chi, x_0)$ are indistinguishable, we therefore only need to find a basis change aligning their output matrices and their *reachable* oracle constraints. Note that from the previous clause, the SAT solver has already obtained a basis B that maps the reachable subspace of $\mathsf{View}_R(\chi, x)$

to a canonical form (vectors ending in some number of zeroes). Hence we can easily check whether a given oracle constraint is reachable. Also note that B is not constrained in how it operates *within* the reachable subspace. Hence we can let this B basis serve double-duty and ask for it to also align the reachable subspace of $\mathsf{View}_R(\chi, x)$ to that of $\mathsf{View}_R(\chi, x_0)$.

In more detail, let $B_{R,\chi,x}$ be the basis matrix that is already quantified corresponding to $\mathsf{View}_R(\chi, x)$ from security condition 1. We want $\mathcal{M}_{R,\chi,x} \times B_{R,\chi,x}$ and $\mathcal{M}_{R,\chi,x_0} \times B_{R,\chi,x_0}$ to coincide, and we want $\mathcal{C}_{R,\chi,x} B_{R,\chi,x}$ and $\mathcal{C}_{R,\chi,x_0} B_{R,\chi,x_0}$ to coincide, but only for reachable constraints. Hence:

$$\mathcal{M}_{R,\chi,x} \times B_{R,\chi,x} = \mathcal{M}_{R,\chi,x_0} \times B_{R,\chi,x_0} \wedge$$
$$\Big[\forall \langle t, \mathcal{Q}, \boldsymbol{a} \rangle \in \mathcal{C}_{R,\chi,x} : \mathsf{RightZeroes}(\mathcal{Q} \times B_{R,\chi,x})$$
$$\Rightarrow \langle t, \mathcal{Q} \times B_{R,\chi,x} \times B^{-1}_{R,\chi,x_0}, \boldsymbol{a} \times B_{R,\chi,x} \times B^{-1}_{R,\chi,x_0} \rangle \in \mathcal{C}_{R,\chi,x_0}\Big]$$

Note that $\langle t, \mathcal{Q} \times B_{R,\chi,x} \times B^{-1}_{R,\chi,x_0}, \boldsymbol{a} \times B_{R,\chi,x} \times B^{-1}_{R,\chi,x_0} \rangle \in \mathcal{C}_{R,\chi,x_0}$ is equivalent to saying $\langle t, \mathcal{Q} B_{R,\chi,x}, \boldsymbol{a} B_{R,\chi,x} \rangle \in \mathcal{C}_{R,\chi,x_0} B_{R,\chi,x_0}$. Hence the bracketed expression captures the requirement that $\mathcal{C}_{R,\chi,x} B_{R,\chi,x}$ and $\mathcal{C}_{R,\chi,x_0} B_{R,\chi,x_0}$ coincide for reachable constraints.

As usual, the quantifications over constraints are expanded within the formula.

3.3 Implementation Results

We implemented Linisynth using Python and the SMT solver Z3[3]. 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. 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 Sect. 3.2 hold (expressed as boolean formulas over the variables). This often results in rather large formulas (see Fig. 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. [53] (costing 2 ciphertexts). An example of such a garbled AND-gate is given in Fig. 5. 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 Fig. 4.

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

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	$\text{MUX} : 3 \to 1$	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

Fig. 4. Selection of our synthesis results on an Intel Xeon 3.4 GHz processor with 16 GB memory. Satisfiable schemes are listed in the full version. 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.

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

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

$\underline{\mathsf{GateGb}^H(\sigma, A, B, \Delta) :}$
$h_1 = H(A)$
$h_2 = H(A + \Delta)$
$h_3 = H(A + B)$
$h_4 = H(A + B + \Delta)$
$G_0 = [0,2]\Delta + h_3 + h_4$
$G_1 = A + B + [0,2]\Delta + h_1 + h_2 + h_3 + h_4$
$C_0 = B + [0]\Delta + [0,2]h_1 + [1,3]h_2 + [1,2]h_3 + [0,3]h_4$
return G_0, G_1, C_0

$\underline{\mathsf{GateEv}^H(\chi, A^*, B^*, G_0, G_1) :}$
return $[1,3]A^* + [0,2]B^* +$
$\quad [0,1]G_0 + [1,3]G_1 +$
$\quad H(A^*) + H(A^* + B^*)$

Fig. 5. An example of one of our synthesized schemes. This scheme is an alternative to the half-gates AND gate of [53], with identical parameters (number of ciphertexts, and number of calls to H). The notation is as follows: GateGb: When S is a set of indices, "$[S]W$" refers to nonlinear behavior "if $\sigma \in S$ then W else 0^λ" GateEv: When S is a set of indices, "$[S]W$" refers to nonlinear behavior "if $\chi \in S$ then W else 0^λ"

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 [53]. 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 $2 \times \mathsf{circ\text{-}min}_{\text{AND}}(\tau)$. Verification of this hypothesis is left as future work.

Enumeration of Solutions. Linisynth can also *enumerate* schemes. Let p be a formula generated according to Sect. 3.2 and let w be a satisfying assignment with $p(w) = 1$. When Linisynth gets 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. 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 $\mathsf{adaptive} = 0$).

Acknowledgement. We thank Viet Tung Hoang for pointing out to us some subtleties that arise when wires have correlated labels.

References

1. Abe, M., Groth, J., Haralambiev, K., Ohkubo, M.: Optimal structure-preserving signatures in asymmetric bilinear groups. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 649–666. Springer, Heidelberg (2011)
2. Abe, M., et al.: Structure-preserving signatures from type II pairings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 390–407. Springer, Heidelberg (2014)
3. Abe, M., Groth, J., Ohkubo, M., Tibouchi, M.: Unified, minimal and selectively randomizable structure-preserving signatures. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 688–712. Springer, Heidelberg (2014)
4. Akinyele, J.A., Green, M., Hohenberger, S.: Using SMT solvers to automate design tasks for encryption and signature schemes. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 399–410. ACM Press, November 2013
5. Akinyele, J.A., Green, M., Hohenberger, S., Pagano, M.W.: Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 474–487. ACM Press, October 2012
6. Applebaum, B., Avron, J., Brzuska, C.: Arithmetic cryptography: extended abstract. In: Roughgarden, T. (ed.) Proceedings of the Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, 11–13 January 2015, pp. 143–151. ACM (2015)
7. Barthe, G., Fagerholm, E., Fiore, D., Scedrov, A., Schmidt, B., Tibouchi, M.: Strongly-optimal structure preserving signatures from type II pairings: synthesis and lower bounds. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 355–376. Springer, Heidelberg (2015)
8. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990
9. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy, pp. 478–492. IEEE Computer Society Press, May 2013
10. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In:Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACMPress, October 2012

11. Bernstein, D.J., et al.: SPHINCS: practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 368–397. Springer, Heidelberg (2015)
12. Black, J.A., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg (2002)
13. Black, J.A., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
14. Buchmann, J., Dahmen, E., Hülsing, A.: 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
15. Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle signatures with virtually unlimited signature capacity. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 31–45. Springer, Heidelberg (2007)
16. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS – an improved merkle signature scheme. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 349–363. Springer, Heidelberg (2006)
17. Dodis, Y., Haitner, I., Tentes, A.: On the instantiability of hash-and-sign RSA signatures. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 112–132. Springer, Heidelberg (2012)
18. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (1993)
19. Gagné, M., Lafourcade, P., Lakhnech, Y., Safavi-Naini, R.: Automated security proof for symmetric encryption modes. In: Datta, A. (ed.) ASIAN 2009. LNCS, vol. 5913, pp. 39–53. Springer, Heidelberg (2009)
20. Gagné, M., Lafourcade, P., Lakhnech, Y., Safavi-Naini, R.: Automated verification of block cipher modes of operation, an improved method. In: Garcia-Alfaro, J., Lafourcade, P. (eds.) FPS 2011. LNCS, vol. 6888, pp. 23–31. Springer, Heidelberg (2012)
21. Goldreich, O.: Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 104–110. Springer, Heidelberg (1987)
22. Gueron, S., Lindell, Y., Nof, A., Pinkas, B.: Fast garbling of circuits under standard assumptions. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 567–578. ACM Press, October 2015
23. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
24. Hoang, V.T., Katz, J., Malozemoff, A.J.: Automated analysis and synthesis of authenticated encryption schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 84–95. ACM Press, October 2015
25. Hülsing, A.: W-OTS+ – shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 173–188. Springer, Heidelberg (2013)
26. Impagliazzo, R.: A personal view of average-case complexity. In: Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, 19–22 June 1995, pp. 134–147. IEEE Computer Society (1995)
27. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 8–26. Springer, Heidelberg (1990)

28. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 294–314. Springer, Heidelberg (2009)
29. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014)
30. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
31. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication. In: IETF RFC 2104 (1997). https://www.ietf.org/rfc/rfc2104.txt
32. Krovetz, T., Dai, W.: VMAC: message authentication code using universal hashing. CFRG Working Group (2007). http://www.fastcrypto.org/vmac/draft-krovetz-vmac-01.txt
33. Lamport, L.: Constructing digital signatures from a one-way function. Technical report SRI-CSL-98, SRI International Computer Science Laboratory (1979)
34. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
35. Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 447–447. Springer, Heidelberg (1986)
36. Malkin, T., Pastro, V., Shelat, A.: An algebraic approach to garbling. Unpublished Manuscript (2016). Presented at Simons Institute workshop on securing computation: https://simons.berkeley.edu/talks/tal-malkin-2015-06-10
37. Malozemoff, A.J., Katz, J., Green, M.D.: Automated analysis and synthesis of block-cipher modes of operation. In: IEEE 27th Computer Security Foundations Symposium, CSF, pp. 140–152. IEEE (2014)
38. Maurer, U.M.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005)
39. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
40. Naor, D., Shenhav, A., Wool, A.: One-time signatures revisited: have they become practical? Cryptology ePrint Archive, Report 2005/442 (2005). http://eprint.iacr.org/2005/442
41. Naor, M.: Bit commitment using pseudorandomness. J. Cryptol. **4**(2), 151–158 (1991)
42. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139. ACM, New York (1999)
43. Papakonstantinou, P.A., Rackoff, C.W., Vahlis, Y.: How powerful are the DDH hard groups? Cryptology ePrint Archive, Report 2012/653 (2012). http://eprint.iacr.org/2012/653
44. Pereira, G.C., Puodzius, C., Barreto, P.S.: Shorter hash-based signatures. J. Syst. Softw. **116**, 95–100 (2016)
45. Pieprzyk, J., Wang, H., Xing, C.: Multiple-time signature schemes against adaptive chosen message attacks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 88–100. Springer, Heidelberg (2004)

46. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
47. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
48. Reyzin, L., Reyzin, N.: Better than BiBa: short one-time signatures with fast signing and verifying. In: Batten, L.M., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (2002)
49. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
50. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: O.C.B: a block-cipher mode of operation for efficient authenticated encryption. In: ACM CCS 2001, pp. 196–205.ACM Press, November 2001
51. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
52. Winternitz, R.S.: Producing a one-way hash function from DES. In: Chaum, D. (ed.) CRYPTO 1983, pp. 203–207. Plenum Press, New York (1983)
53. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015)