A Complete Characterization of Security for Linicrypt Block Cipher Modes

Tommy Hollenberg, Mike Rosulek, Lawrence Roy
Oregon State University
{hollenbt,rosulekm,royl}@oregonstate.edu

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919676

***Abstract*—We give characterizations of IND$-CPA security for a large, natural class of encryption schemes. Specifically, we consider encryption algorithms that invoke a block cipher and otherwise perform linear operations (*e.g.*, XOR and multiplication by fixed field elements) on intermediate values. This class of algorithms corresponds to the Linicrypt model of Carmer & Rosulek (Crypto 2016). Our characterization for this class of encryption schemes is sound but not complete.**

We then focus on a smaller subclass of *block cipher modes*, which iterate over the blocks of the plaintext, repeatedly applying the same Linicrypt program. For these Linicrypt block cipher modes, we are able to give a sound and complete characterization of IND$-CPA security. Our characterization is linear-algebraic in nature and is easy to check for a candidate mode. Interestingly, we prove that a Linicrypt block cipher mode is secure if and only if it is secure against adversaries who choose all-zeroes plaintexts.

I. Introduction

Security against chosen-plaintext attacks (CPA) for symmetric-key encryption is one of the most fundamental security properties in cryptography. CPA security is a requirement about the behavior of an encryption scheme, in the presence of *adversarially chosen* inputs. As such, it is non-trivial in the general case to check whether a candidate scheme satisfies this property. In this work we examine alternative characterizations of CPA security that are more easily amenable to automated reasoning and even program synthesis (*i.e.*, transforming a specification into a program that meets the specification, in a programmatic way).

We cannot hope to apply automated analysis to *completely arbitrary* algorithms. Instead, we restrict our attention to a special class of algorithms. In this work we focus on block cipher modes. Most common block cipher modes in practice are built using a small set of simple operations: calls to the block cipher, XOR, and in some cases finite field multiplications. These simple operations fit well into the Linicrypt class of algorithms, introduced by Carmer and Rosulek [1]. A Linicrypt algorithm is one where all intermediate values are field elements and the only allowed operations are fixed linear combinations and calls to a random oracle. In our application of Linicrypt, we can replace the random oracle with a block cipher $F_K(\cdot)$ keyed with the encryption key. Linicrypt algorithms can be converted into an algebraic representation, and many security properties of the algorithm can be recast as linear-algebraic properties of this representation.

A. Related Work

a) Prior Work on Characterizing Block Cipher Modes: A series of works by Gagné et al [2], [3], [4] also consider automated methods for proving security of block cipher modes that consist of calls to the PRP, XOR, and increment operations. They develop an extensive Hoare logic for expressing CPA security of these modes. Their approach is sound but they do not claim that it is complete (*i.e.*, some modes may be secure for reasons other than those expressed in the Hoare logic).

Malozemoff, Katz, and Green [7] proposed an automated way to synthesize secure block cipher modes. In our work we point out some flaws with their approach, which we summarize directly below in Section I-B and present fully in Section VII.

Both their work and ours consider block cipher modes that consist of XORs and calls to the PRP. Their work further considers increment operations (as in CTR mode), whereas ours supports multiplication in $GF(2^n)$ by a fixed field element.[1] In this work we point out problems with the soundness of their characterization.

In a followup work, Hoang, Katz, and Malozemoff [5] extend this analysis to authenticated encryption modes constructed from a tweakable block cipher. In our work we focus only on CPA security, but to our knowledge the flaws in [7] are not inherited by [5]. Their approach is also deeply tied to the use of tweakable block ciphers as the underlying primitive, and it is not clear how to directly translate their approach to CPA-secure encryption from (plain) block ciphers.

Another closely related series of works *et al.* [9], [10], [6] develop a symbolic characterization of IND$-CPA for encryption schemes using a formal term algebra. In some respects, this model is more flexible than ours — it models encryption as a highly interactive process, with the adversary providing inputs and receiving outputs from a stateful encryption program at different times. In other respects, their model is weaker than ours, since it considers only XOR operations and not field multiplication by a constant. I.e., it would not be possible to model what we call *counter-like* block cipher modes in their model. It is undecidable in general to determine IND$-CPA security of an encryption scheme in their model; however, some special cases are decidable. One important distinction is that we relate the security of the encryption scheme as a whole (*i.e.*, an encryption algorithm which can support plaintexts of any length) to specific properties of the

[1] If a field element has high multiplicative order, then repeated multiplication by that element is very similar to an increment (addition mod 2^n).

DOI 10.1109/CSF54842.2022.00028

main round function (*i.e.*, an algorithm with inputs/outputs of fixed size). Due in part to the fact that we restrict to encryption schemes composed of a regular repeated loop, our main security characterization is decidable in polynomial time.

b) Prior Work on Security Properties for Linicrypt Programs: Linicrypt was introduced in [1]. That work showed an algebraic characterization of indistinguishability in the random oracle model. That is, given two Linicrypt programs (making calls to a random oracle that takes field elements as input/output), it is possible in polynomial time to determine whether the programs induce indistinguishable output distributions. Only input-less programs were considered in that work.

Followup work of McQuoid, Swope & Rosulek [8] considered collision resistance and second-preimage resistance for Linicrypt programs. They show an algebraic condition for whether a given deterministic Linicrypt program satisfies collision/second-preimage resistance.

One main difference between prior work and ours is that Linicrypt programs call a random function, and prior work considers attackers who also have access to that function (as a random oracle). In the context of CPA-security and our result, the random function is the keyed block cipher $F_K(\cdot)$, to which the attacker does not have direct access. In that sense, our scenario is simpler (although we do consider both encryption and decryption algorithm having access to $F_K(\cdot)$ for the purpose of determining whether an encryption scheme satisfies correctness). However, in our work the major challenges (not present in prior work on Linicrypt) are dealing with adversarially and *adaptively* chosen inputs and Linicrypt programs with input-dependent loops, as in a block cipher mode that loops over plaintext blocks.

B. Our Results

a) IND$-CPA for monolithic (fixed-input-length) Linicrypt programs: We first consider randomized encryption algorithms of the form $\mathsf{Enc}_K(m) = L^{F_K(\cdot)}(m)$ where L is a Linicrypt program and F is a PRP/PRF. We make no assumptions about L besides the fact that it is Linicrypt — *e.g.*, it may behave wildly differently for each plaintext block. Since a standard Linicrypt program has a fixed input size, this class of encryption algorithms supports only plaintexts of a fixed length (though potentially many blocks).

We give a linear-algebraic characterization of IND$-CPA security[2] for such encryption algorithms. The characterization is sound but not complete; however, it is complete for the special case of single-block plaintexts (a fact that we exploit below). The characterization can be checked in polynomial time in the size of the Linicrypt program's description.

This is the first work to consider indistinguishability of Linicrypt programs under adversarially-chosen inputs; prior work in [1] considered randomized Linicrypt programs that take no inputs.

b) IND$-CPA for Linicrypt block cipher modes: We define a Linicrypt block cipher mode to be an encryption algorithm that loops over the plaintext blocks, with a loop body that is a Linicrypt program. This main-loop-body takes as input the next plaintext block and a single chaining value, and it has oracle access to the keyed PRP $F_K(\cdot)$. See Section IV for a precise definition.

We give a sound and complete characterization of IND$-CPA security for such modes. In other words, given the description of the (Linicrypt) main-loop-body, we can decide whether that main-loop-body induces a secure encryption scheme supporting *variable-length* plaintexts.

This is the first work to consider any kind of looping construct for Linicrypt programs (standard Linicrypt programs are straight-line programs). Our characterization says (very roughly) that a Linicrypt mode is secure if and only if it is secure for a single block, and the chaining value has a non-repeating property. Both of these conditions can be expressed algebraically, and checked in polynomial time. Note that the first condition reduces to security of single-block Linicrypt encryption, which is both sound and complete.

Finally, we apply this characterization to exhaustively catalog all secure Linicrypt block cipher modes in which the main-loop-body makes just a single call to the block cipher.

c) Issues with prior work: Prior work by Malozemoff, Katz, and Green [7] studies IND$-CPA security of block cipher modes. They consider modes whose round function consists of calls to a block cipher (PRP or PRF), XOR, and increment (as in CTR mode), which are then expressed as a circuit (directed graph). They show a rule for labeling the edges of the graph, such that a mode is IND$-CPA if the graph of its round function has a valid labeling.

Unfortunately, we show that their labeling rules are unsound. We show a concrete block cipher mode which is profoundly insecure but whose circuit graph nevertheless admits a valid labeling. Essentially, their labeling rules do not correctly account for all possible correlations among intermediate values (induced by combinations of XORs and invocations of the PRP/PRF). Such linear correlations are fundamental to the Linicrypt model, however, and our characterization can easily show the insecurity of this counterexample.

The authors of [7] do not claim that their labeling rules are *complete*, although they provide no explicit counterexample. In the full version of this paper, we show such a counterexample that is IND$-CPA secure, but that has no valid labeling under their rules.

II. Preliminaries

A. Block Ciphers

In this work we consider encryption algorithms built from block ciphers. We exclusively consider block ciphers whose block size is the security parameter. In that case, it is well known by the PRF-PRP switching lemma that a secure block cipher (PRP) is also a secure PRF. This is the only property of the underlying block ciphers that we use.

Definition 1. *$F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ is a pseudo-random function if for all polynomial-time adversaries $\mathcal{A}$:*

$$\left| \Pr_K[\mathcal{A}^{F_K(\cdot)}(1^\kappa) = 1] - \Pr_R[\mathcal{A}^{R(\cdot)}(1^\kappa) = 1] \right| \text{ is negligible}$$

[2] In this work we exclusively consider randomized encryption, rather than nonce-based encryption.

where R is chosen uniformly from the set of functions $R : \{0,1\}^\kappa \to \{0,1\}^\kappa$.

B. IND$-CPA security

We exclusively consider security in the presence of chosen-plaintext attacks. Specifically, we consider the indistinguishability from random (IND$-CPA) goal.

Definition 2. *An encryption scheme* Enc *has* ***correctness*** *if there is an algorithm* Dec *so that* $\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m$ *with probability 1, for all m and all K.*

Definition 3. *An encryption scheme* Enc *is* ***IND$-CPA*** *secure if for all polynomial-time adversaries $\mathcal{A}$:*

$$\left|\Pr_K[\mathcal{A}^{\mathsf{Enc}_K(\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\$(\cdot)}(1^\kappa) = 1]\right| \text{ is negligible}$$

where $\$(\cdot)$ is a randomized function that on ℓ-block input returns an independent, uniformly chosen $(\ell+1)$-block output.

C. Linicrypt

The Linicrypt model was introduced in [1]. We now present a slight simplification of the Linicrypt model, which suffices for our purposes.

A Linicrypt program is a straight-line oracle program. All variables in the program take on values from some finite field $\mathbb{F}$. In this work we assume that the characteristic of $\mathbb{F}$ is polynomial in the security parameter, and we indicate the places where this assumption affects the result. The only valid instructions in a Linicrypt program are:

- Sample a value uniformly from $\mathbb{F}$.
- Call the oracle H on a previously-computed value. In this work, H always has type $H : \mathbb{F} \to \mathbb{F}$.
- Compute a linear combination of previously computed values, whose coefficients are fixed as part of the program.

Finally, a Linicrypt program outputs a list containing some of its intermediate values.

An example is given below, written in a more conventional way and in a more explicit way that names each intermediate value:

$$\boxed{\begin{array}{l}\underline{L^H(r,m)\text{:}}\\ \quad c := H(H(r)+r)+m\\ \quad s := H(H(r))\\ \quad \text{return } (c,s)\end{array}} \rightsquigarrow \boxed{\begin{array}{l}\underline{L^H(v_1,v_2)\text{:}}\\ \quad v_3 := H(v_1)\\ \quad v_4 := v_3 + v_1\\ \quad v_5 := H(v_4)\\ \quad v_6 := H(v_3)\\ \quad v_7 := v_5 + v_2\\ \quad \text{return } (v_6,v_7)\end{array}}$$

a) Algebraic representation: A Linicrypt program L can be represented in an algebraic way. First, we call an intermediate variable a **base variable** if it is either an input, the result of uniform sampling in $\mathbb{F}$, or the result of a call to H. Note that every intermediate value in the program is a linear combination of the base variables. In the example from above, the base variables are $(v_1, v_2, v_3, v_5, v_6)$, and all variables in the program are a linear combination of these base variables:

$$\begin{bmatrix}v_1\\v_2\\v_3\\v_4\\v_5\\v_6\\v_7\end{bmatrix} = \begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\1&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&1&0&1&0\end{bmatrix} \times \begin{bmatrix}v_1\\v_2\\v_3\\v_5\\v_6\end{bmatrix}$$

Let $\boldsymbol{v}_{\text{base}}$ denote the vector of base variables. For each variable (base or otherwise) v_i, we can associate a vector $\mathsf{row}(i)$ such that $v_i = \mathsf{row}(i) \cdot \boldsymbol{v}_{\text{base}}$. For each base variable v_j, we write $\boldsymbol{v}_j$ to mean the vector of base variables where v_j is one and every other base variable is zero. In this notation we can find what coefficient a particular derived variable v_i has on a base variable v_j: $\mathsf{row}(i) \cdot \boldsymbol{v}_j$.

The **input matrix** of a program is simply the matrix containing a row $\mathsf{row}(i)$ for each input base variable v_i. The **output matrix** of a program is simply the matrix containing a row $\mathsf{row}(i)$ for each output variable v_i.

To capture the relationships among the variables and the random oracle, we define a set of **oracle constraints** as follows. For an oracle query $v_i = H(v_j)$, the corresponding oracle constraint is written as "$\mathsf{row}(j) \mapsto \mathsf{row}(i)$". Note that if two oracle constraints have the same "left-hand side" ($\mathsf{row}(j)$ in the preceding example), this indicates that the two calls to H are always made on identical values.[3] In short, if $v_i = H(\cdots)$ and $v_k = H(\cdots)$ lead to oracle constraints with identical left-hand-sides, then the second oracle query can be removed and all references to v_k can be replaced with a corresponding reference to v_i.

The running example L has the following input matrix $M(L)$, output matrix $C(L)$ and set $\mathcal{O}(L)$ of oracle constraints:

$$M(L) = \begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\end{bmatrix} \quad \begin{array}{l}\text{“}v_1\text{”}\\ \text{“}v_2\text{”}\end{array}$$

$$C(L) = \begin{bmatrix}0&0&0&0&1\\0&1&0&1&0\end{bmatrix} \quad \begin{array}{l}\text{“}v_6\text{”}\\ \text{“}v_7\ (= v_2 + v_5)\text{”}\end{array}$$

$$\mathcal{O}(L) = \left\{\begin{array}{l}[1\ 0\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0\ 0],\\ [1\ 0\ 1\ 0\ 0] \mapsto [0\ 0\ 0\ 1\ 0],\\ [0\ 0\ 1\ 0\ 0] \mapsto [0\ 0\ 0\ 0\ 1]\end{array}\right\} \begin{array}{l}\text{“}v_3 := H(v_1)\text{”}\\ \text{“}v_5 := H(v_4)\text{”}\\ \text{“}v_6 := H(v_3)\text{”}\end{array}$$

Together, the input matrix $M(L)$ (often synonymous with the plaintext/“message”), the output matrix $C(L)$ (often “ciphertext”) and oracle constraints $\mathcal{O}(L)$ completely characterize the Linicrypt program.

One of the main techniques [1] used to prove theorems about Linicrypt programs is *canonical simulation*: one can think of the execution of an input-less Linicrypt program as first choosing uniform base variables $\boldsymbol{v}_{\text{base}}$, and then producing output $C(L)\boldsymbol{v}_{\text{base}}$. If later the oracle H is called on $\boldsymbol{q} \cdot \boldsymbol{v}_{\text{base}}$ and $\boldsymbol{q} \mapsto \boldsymbol{a}$ is an oracle constraint in $\mathcal{O}(L)$, then the output of H must be $\boldsymbol{a} \cdot \boldsymbol{v}_{\text{base}}$.

[3] If two left-hand-sides are different, then it is possible (but only with negligible probability for input-less programs) that they can correspond to calling H on the same values. This is explored further in [1].

b) Difference with [1]: The original Linicrypt model of [1] allows extra "nonce" values and any number of field elements to be given to H, but in our work H is a keyed block cipher with only one input and no additional nonce.

III. Secure Linicrypt Encryption

In this section we provide a characterization for IND$-CPA security of encryption algorithms described by Linicrypt programs. Linicrypt programs have no native looping construct, so this section considers encryption schemes that support plaintexts of only a single, fixed length. Later we build on the results in this section to characterize variable-length encryption schemes built from Linicrypt "round functions."

A. New Algebraic Properties

Prior characterizations in Linicrypt do not consider indistinguishability in the presence of adversarially chosen inputs. The main challenge of such a setting is that an adversary may cause two nominally unrelated oracle queries in a Linicrypt program to "collapse." As a concrete example, consider the Linicrypt program $(x, y) \mapsto H(x) \oplus H(H(y))$. In general, this program may make 3 distinct calls to H. However, an adversary may choose inputs such that $x = H(y)$, which causes some of the queries to H to be on repeated inputs, As a result, the adversary can force the output of this program to be zero.

In general, it is NP-hard to understand the collapsing potential of a Linicrypt program (Appendix A). In this section, we develop a new algebraic language for reasoning about these collapsing behaviors. Roughly speaking:

- The **zeroable subspace** $Z(L)$ of a Linicrypt program L represents the internal values of the program that the adversary can force to zero, by either setting inputs to zero, or by causing two ostensibly distinct calls to H to be made on the same value. There may be other ways to force some internal values to zero, but we do not consider them as part of this definition.
- The **repeatable subspace** $R(L)$ represents internal values that the adversary could cause to repeat between multiple calls to L. For example, if a program outputs $H(v)$, but v is in the zeroable space, then the adversary could force the program to output the same $H(0)$ on multiple invocations.
- For repeatable (in the above sense) internal values, we want to understand which of the program's inputs are vital in inducing repeats. This part of our definitions is the most subtle. Intuitively, an internal value q is **repeatable dependent on** an input variable v_i if q would cease to be repeatable if the adversary had no control over v_i (*i.e.*, if it was always sampled uniformly). That is, q is computed by combining repeatable things with either v_i or the results of oracle queries whose repeatability depends on v_i.
- We consider two different ways that an adversary can cause two oracle queries/constraints to "collapse" (and we call the pair of queries/constraints **collidable**): One way is to cause the difference between the oracle inputs to be zeroable (*i.e.*, cause the oracle inputs to be equal, using the rules of the zeroable space). Another way is to learn some "repeatable" (in the above sense) internal value from one call to the program, and then adaptively choose another input to "cancel out" that repeatable information to make oracle queries collapse. This situation is important, since we want to consider an adversary who makes repeated queries to the Linicrypt program.

Our definition of "collidable" is cautious: If some pair of queries is marked collidable, it is not guaranteed that an adversary can indeed consistently force these queries to collide (*e.g.*, in some cases an adversary can collapse one pair of queries or another pair of queries, but not both simultaneously). However, if an adversary can force two queries to collide, then that pair of queries must indeed be collidable.

These concepts are defined in terms of each other in a delicate way, but this apparent circularity is resolved by defining the subspaces $Z(L)$ and $R(L)$ to be the *smallest subspaces* satisfying the corresponding rules. Given a Linicrypt program L, these properties can be computed by applying the rules in the definition, iteratively, until a fixed point is reached. This process terminates in polynomial time (in the description of L) since in each iteration it considers only a polynomial number of pairs of oracle constraints.

Definition 4. *Given a Linicrypt program L, define its **zeroable subspace** $Z(L)$ to be the smallest linear subspace satisfying the following:*

1) *$Z(L)$ contains all rows of $M(L)$ (representing the input variables of L)*
2) *$\boldsymbol{a} - \boldsymbol{a}' \in Z(L)$ for all pairs of collidable oracle queries $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L)$.*

Definition 5. *Given a Linicrypt program L, define its **repeatable subspace** $R(L)$ to be the smallest linear subspace of the row vectors on the base variables satisfying the following rules.*

(i) $Z(L)$ is a subspace of $R(L)$.
(ii) If $\boldsymbol{q} \mapsto \boldsymbol{a} \in \mathcal{O}(L)$ and $\boldsymbol{q}$ is in $R(L)$, then $\boldsymbol{a}$ is also in $R(L)$.

Let $U(L)$ be a matrix whose rows form a basis for the repeatable subspace.

Definition 6. *Let v_i be an input variable. We say that a row vector $\boldsymbol{u}$ is **repeatable dependent on** v_i if for some j we can write $\boldsymbol{u} = c\boldsymbol{v}_j + \boldsymbol{u}'$, where:*

(i) $c \neq 0$, and
(ii) $\boldsymbol{u}' \in R(L)$, and
(iii) $\boldsymbol{u}' \cdot \boldsymbol{v}_j = 0$, i.e., the coefficient of v_j in $\boldsymbol{u}'$ is zero, and
(iv) either $i = j$ or there is an oracle query $\boldsymbol{q} \mapsto \boldsymbol{v}_j \in \mathcal{O}(L)$ such that $\boldsymbol{q}$ is repeatable dependent on v_i.

Definition 7. *The **collidable** oracle queries are defined to be the smallest equivalence relation containing all pairs of queries $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L)$ satisfying one of the following two conditions.*

(i) $\boldsymbol{q} - \boldsymbol{q}' \in Z(L)$, or
(ii) There exist distinct input variable $v_i \neq v_j$ such that $\boldsymbol{q} - \boldsymbol{q}'$ is repeatable dependent on v_i, and $(\boldsymbol{q} - \boldsymbol{q}') \cdot \boldsymbol{v}_j \neq 0$ (i.e., $\boldsymbol{q} - \boldsymbol{q}'$ contains a nonzero coefficient for v_j).

If L has only a single input variable then case (ii) of Definition 7 is never used, as there are no two distinct input variables. In this case $Z(L)$ completely captures what happens when the adversary provides the all-zero input to L. The only things in $Z(L)$ are input variables and the differences in outputs of identical queries, and all of these values go to zero in this case.

Lemma 8. *If $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L)$ are collidable, then $\boldsymbol{q} - \boldsymbol{q}' \in R(L)$.*

Proof: In case (i) of Definition 7, $\boldsymbol{q} - \boldsymbol{q}' \in Z(L) \subseteq R(L)$. In case (ii), $\boldsymbol{q} - \boldsymbol{q}'$ is repeatable dependent on some v_i, which implies that it is repeatable. Finally, $R(L)$ is a linear space so this still holds when taking the transitive closure (as collidability is defined as taking the transitive closure of pairs that satisfy (i) and (ii)). ∎

Lemma 9. *If $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L)$ are collidable, then $\boldsymbol{a} \in R(L) \iff \boldsymbol{a}' \in R(L)$.*

Proof: By Lemma 8, $\boldsymbol{q} - \boldsymbol{q}' \in R(L)$ for a colliding pair. Since $R(L)$ is a vector subspace, $\boldsymbol{q} \in R(L) \iff \boldsymbol{q}' \in R(L)$. By the definition of repeatability, $\boldsymbol{a} \in R(L)$ if and only if $\boldsymbol{q} \in R(L)$. ∎

B. Characterization

Our main result in this section is a sufficient condition for a collection of Linicrypt programs to be jointly indistinguishable from random.

Theorem 10. *If $L_1, L_2, \ldots, L_n$ is a collection of Linicrypt programs, and for each i the rows of $\begin{bmatrix} C(L_i) \\ U(L_i) \end{bmatrix}$ are linearly independent, then $L_1, \ldots, L_n$ are jointly IND$-CPA secure when instantiated with the same PRP F and key K.*

In other words, if $\ker \begin{bmatrix} C(L_i) \\ U(L_i) \end{bmatrix}^{\intercal} = \{0\}$ then, for all $\mathcal{A}$, the following advantage is negligible:

$$\left| \Pr_K[\mathcal{A}^{L_1^{F_K}(\cdot),\ldots,L_n^{F_K}(\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\$(\cdot),\ldots,\$(\cdot)}(1^\kappa) = 1] \right|$$

Proof: As initial step, we use the standard switching lemma to replace the PRP with a PRF, and then we can replace the PRF with a random function since the key is randomly generated and only used for the PRF. We now have each L_i connected to the same random oracle, and in this case the adversary does not have direct access to the random oracle.

For each L_i we construct a corresponding program H_i which does the following. On a given input, it first runs L_i on the same input and records the base variables $\boldsymbol{v}_{\text{base}}$ that were used. It then randomly samples a new set of base variables $\boldsymbol{v}'_{\text{base}}$ subject to the constraint that $U(L_i)\boldsymbol{v}'_{\text{base}} = U(L_i)\boldsymbol{v}_{\text{base}}$, and then outputs $C(L_i)\boldsymbol{v}'_{\text{base}}$.

We first claim that $H_1, \ldots, H_n$ give truly uniform random output. The rows of $\begin{bmatrix} C(L_i) \\ U(L_i) \end{bmatrix}$ are linearly independent by assumption, so for every $\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{u} \end{bmatrix}$, there is some $\boldsymbol{v}_{\text{base}}$ so that $\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} C(L_i) \\ U(L_i) \end{bmatrix} \boldsymbol{v}_{\text{base}}$. Furthermore, the number of solutions in $\boldsymbol{v}_{\text{base}}$ is the same for any particular $\boldsymbol{c}, \boldsymbol{u}$. This implies that for any $\boldsymbol{u}$, the distribution over $\boldsymbol{c}$ (which is the output of H_i) is uniform. This also implies that $\boldsymbol{c}$ tells the adversary nothing about $\boldsymbol{u}$.

Next, we use the h-coefficient technique to prove that the Linicrypt programs are indistinguishable from these H_i hybrids. In the standard method of h-coefficient technique, we classify a set of transcripts as *bad* and bound the probability of the hybrids generating a bad transcript. We then show that any good transcript has equal probability of being produced by the actual Linicrypt programs or by the hybrids. We define the transcript to consist of a list of pairs $(i, \boldsymbol{v}_{\text{base}})$, specifying that the ith Linicrypt program L_i or hybrid H_i was invoked by the adversary, and resulted in $\boldsymbol{v}_{\text{base}}$ as its choice of base variables. Note that both the choice of input/plaintext and output/ciphertext can be computed from such a transcript, as $M(L_i)\boldsymbol{v}_{\text{base}}$ and $C(L_i)\boldsymbol{v}_{\text{base}}$, respectively.

There are two ways that a transcript can be *bad*.

1) The term $(i, \boldsymbol{v}_{\text{base}})$ in the transcript reflects an unexpected collision inside L_i: there are two non-collidable oracle queries $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L_i)$ such that $\boldsymbol{q}\boldsymbol{v}_{\text{base}} = \boldsymbol{q}'\boldsymbol{v}_{\text{base}}$.
2) The term $(i, \boldsymbol{v}_{\text{base}})$ was preceded sometime earlier by $(j, \boldsymbol{v}'_{\text{base}})$ in the transcript, and they reflect an unexpected collision between L_i and L_j: there exist queries $\boldsymbol{q} \mapsto \boldsymbol{a} \in \mathcal{O}(L_i)$ and $\boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L_j)$ with $\boldsymbol{q}\boldsymbol{v}_{\text{base}} = \boldsymbol{q}'\boldsymbol{v}'_{\text{base}}$, but $\boldsymbol{a}$ is not in its repeatable subspace $R(L_i)$.

In the next Lemmas, we show that these two bad events happen with negligible probability. From this point forward, we consider only "good" transcripts and argue that such transcripts are assigned equal probability in both situations. In a good transcript, if the Linicrypt program L_i queries its oracle on a repeated value (within a single execution of L_i), those queries must correspond to $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L_i)$ where $\boldsymbol{a} - \boldsymbol{a}' \in Z(L_i) \subseteq R(L_i)$. If separate invocations of L_i and L_j both query their oracle on a common value (with L_j coming first), then the queries must correspond to $\boldsymbol{q} \mapsto \boldsymbol{a} \in \mathcal{O}(L_i)$ and $\boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}(L_j)$, where $\boldsymbol{a} \in R(L_j)$. In both cases when executing L_i the repeated query is equivalent to fixing $U(L_i)\boldsymbol{v}_{\text{base}}$ (since $U(L_i)$ is a basis for $R(L_i)$) to take a particular value, and everything else is sampled randomly. That is, in a good transcript, running L_i is equivalent to generating a uniformly random $\boldsymbol{v}_{\text{base}}$ subject to some constraint on $U(L_i)\boldsymbol{v}_{\text{base}}$. Both L_i and H_i generate $U(L_i)\boldsymbol{v}_{\text{base}}$ in the same way, and they both produce a uniformly random output subject to that constraint, so they each have equal probabilities of picking a particular $\boldsymbol{v}_{\text{base}}$. Applying this reasoning to every message, we find that when there is no bad event the transcript has equal probability of being generated by either $L_1, \ldots, L_n$ or $H_1, \ldots, H_n$. ∎

Lemma 11. *The first bad event (defined above) is negligibly likely, when responses are generated according to H_i.*

Proof: The bad event is that L_i queries the PRP/oracle on the same input multiple times within a single call. Assume without loss of generality that this is the first bad event.

Because the two oracle queries are not collidable, $\boldsymbol{q}-\boldsymbol{q}'$ is not in $Z(L_i)$. If $\boldsymbol{q}-\boldsymbol{q}'$ is not in $R(L_i)$ either, $\begin{bmatrix}\boldsymbol{q}-\boldsymbol{q}'\\ U(L_i)\end{bmatrix}$ is linearly independent so there are equally many possible values of $\boldsymbol{v}_{\text{base}}$ for each value of $(\boldsymbol{q}-\boldsymbol{q}')\boldsymbol{v}_{\text{base}}$, even if $U(L_i)\boldsymbol{v}_{\text{base}}$ is fixed. Then every value is equally likely, so there is probability $|\mathbb{F}|^{-1}$ of $(\boldsymbol{q}-\boldsymbol{q}')\boldsymbol{v}_{\text{base}}=0$.

Otherwise $\boldsymbol{q}-\boldsymbol{q}' \in R(L_i)$, so $(\boldsymbol{q}-\boldsymbol{q}')\boldsymbol{v}_{\text{base}}$ might not be freshly random. For case (ii) of Definition 7 to not have triggered, there must not be distinct input variables v_i, v_j such that $\boldsymbol{q}-\boldsymbol{q}'$ is repeatable dependent on v_i and $\boldsymbol{q}-\boldsymbol{q}'$ has a nonzero coefficient for v_j.

Assume that $\boldsymbol{q}-\boldsymbol{q}'$ has coefficient zero for all input variables v_i. In other words, it is a non-zeroable linear combination of oracle query results, with no adversarial input added, so without past unexpected collisions $(\boldsymbol{q}-\boldsymbol{q}')\boldsymbol{v}_{\text{base}}$ is uniformly random (though not necessarily fresh), and has probability $|\mathbb{F}|^{-1}$ of being 0.

Otherwise $c=(\boldsymbol{q}-\boldsymbol{q}')\cdot\boldsymbol{v}_i\neq 0$ for some input variable v_i, so it is repeatable dependent on v_i. However, it cannot be repeatable dependent on any other input v_j, or else case (ii) would have triggered. It being repeatable dependent on only v_j shows that it can be computed using only v_j. That is, if at any point in the computation of $(\boldsymbol{q}-\boldsymbol{q}')\boldsymbol{v}_{\text{base}}$, after all oracle queries on matching inputs have been merged, an input variable v_i was used, then that would show that oracle query is repeatable dependent on v_i. The query that used that one would be as well, and so on all the way back to $\boldsymbol{q}-\boldsymbol{q}'$ being repeatable dependent on v_j.

To cause the bad event, the adversary must set v_i to be $\boldsymbol{u}\boldsymbol{v}_{\text{base}}$ where $\boldsymbol{u}=\frac{1}{c}(\boldsymbol{q}-\boldsymbol{q}')-\boldsymbol{v}_i$. If $\boldsymbol{u}\boldsymbol{v}_{\text{base}}$ can be computed without using v_j, and so does not depend on any inputs at all, it would be in $R(L_j)$ for any past Linicrypt invocation to L_j, and from before we know that the hybrid leaks no information about the repeatable subspace, so the adversary cannot predict this value better than random.

Finally, if $\boldsymbol{u}$ is only repeatable dependent on v_i, then causing the bad event requires satisfying a circular chain of oracle constraints. That is, the value of v_j must be $\boldsymbol{u}\boldsymbol{v}_{\text{base}}$, and $\boldsymbol{u}$ has a nonzero coefficient for the output of some oracle query $\boldsymbol{q}_0 \mapsto \boldsymbol{a}_0$ that is repeatable dependent on v_j, and $\boldsymbol{q}_0$ has a nonzero coefficient for another query, etc., until $\boldsymbol{q}_n$ has a nonzero coefficient for v_j. It is impossible to satisfy such a circular constraint, even with oracle access to H. On each hash execution $H(x)$ (representing $\boldsymbol{q}_i \mapsto \boldsymbol{a}_i$), and each past hash $H(y)$ (representing $\boldsymbol{q}_{i-1} \mapsto \boldsymbol{a}_{i-1}$ for $i>0$, or otherwise $\boldsymbol{q}_n \mapsto \boldsymbol{a}_n$) that completes a possible cycle, the result $H(x)$ has exactly one possible value (out of $|\mathbb{F}|$) that will make y take the correct value to complete the cycle. ∎

Lemma 12. *The second bad event (defined above) is negligibly likely when responses are generated according to H_i.*

Proof: Since $\boldsymbol{q}\notin R(L_i)$, when the hybrid regenerates its base variables every possible value of $\boldsymbol{q}\boldsymbol{v}_{\text{base}}$ is equally likely, and at this point the invocation L_j had already happened so $\boldsymbol{q}'\boldsymbol{v}'_{\text{base}}$ was already fixed. Therefore, the probability of this bad event is $|\mathbb{F}|^{-1}$. ∎

C. A Useful Special Case

The previous theorem gives a sufficient but not necessary condition for IND$-CPA security. However, in a special case the condition is both necessary and sufficient:

Theorem 13. *If L is a **single-input** Linicrypt program that does not satisfy the condition of Theorem 10, then there is an attack against its CPA security that works with non-negligible probability. That is, there is a polynomial time adversary $\mathcal{A}$ such that*

$$\left|\Pr_K[\mathcal{A}^{L^{F_K}}(1^\kappa)=1]-\Pr[\mathcal{A}^{\$(\cdot)}(1^\kappa)=1]\right|\geq 1-|\mathbb{F}|^{-1}.$$

Proof: Make $\mathcal{A}$ ask for the encryption of the zero plaintext twice on L, and find the difference between the two outputs $\boldsymbol{c}-\boldsymbol{c}'=C(L)(\boldsymbol{v}_{\text{base}}-\boldsymbol{v}'_{\text{base}})$. Abort if $\boldsymbol{c}-\boldsymbol{c}'=0$. Note that if the probability of aborting is nonnegligible, then the scheme already fails to be CPA security, since the probability of this event is negligible in the case that $\boldsymbol{c}$ and $\boldsymbol{c}'$ are uniform. As we observed before, the zero plaintext will cause everything in $Z(L)$ to be zeroed, and so by construction $U(L)\boldsymbol{v}_{\text{base}}$ will take on the same values in both instances; i.e., $U(L)(\boldsymbol{v}_{\text{base}}-\boldsymbol{v}'_{\text{base}})$ will be zero. Since $\begin{bmatrix}C(L)\\ U(L)\end{bmatrix}$ does not have linearly independent rows, there exists a nonzero row vector $\boldsymbol{t}$ such that $\boldsymbol{t}\begin{bmatrix}C(L)\\ U(L)\end{bmatrix}=0$. Vector $\boldsymbol{t}$ must have a nonzero entry on at least one ciphertext value as $U(L)$ taken by itself has linearly independent rows (its rows are a basis for some subspace). Therefore, the adversary can check if $\boldsymbol{t}\begin{bmatrix}\boldsymbol{c}-\boldsymbol{c}'\\ 0\end{bmatrix}=0$, which happens with probability 1 for the Linicrypt encryption scheme but only probability $|\mathbb{F}|^{-1}$ for uniformly random ciphertext. ∎

Remark 14. *Although these theorems shows that determining IND$-CPA security for a single input Linicrypt program can be determined efficiently (in the description size of the program), our characterization is not complete for multiple input Linicrypt programs. In fact, we show in Appendix A that determining the security of multiple input Linicrypt programs is NP-hard.*

IV. Class of Block Cipher Modes

A block cipher mode promotes a simple block cipher (pseudorandom permutation) to a CPA-secure encryption scheme. In this work we consider a class of block cipher modes defined as follows:

- We assume that plaintexts are divided into whole **blocks** of length κ, where κ is also the input/output length of the PRP and the security parameter.[4] It is without loss of generality that we consider plaintexts whose length is an exact multiple of κ, because one can add support for arbitrary-length plaintexts in the usual way using any padding scheme. We do not consider more exotic approaches like ciphertext stealing in this work.

[4] In other words, we do not consider small-block PRPs.

- The encryption mode is characterized by a **round function** Rd, where each invocation of Rd processes a single plaintext block and produces a single ciphertext block.
- The entire encryption/decryption process uses the underlying PRP[5] without re-keying, giving Rd access only to the permutation induced by the PRP key. We only consider Rd functions that use the PRP in the forward direction (decryption may involve the reverse direction of the PRP).
- Data is passed between invocations of Rd in the form of a **chaining value**. We consider randomized encryption, where the initial chaining value (IV) is chosen uniformly and is included as part of the ciphertext. We exclusively consider schemes where the chaining value consists of a *single block.*
- Rd must be deterministic. Since the output of the round function has the same length as its input a nondeterministic round function cannot be decrypted.

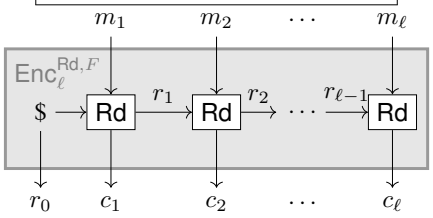
Formally, the round function has syntax:

$$(c, s) := \mathsf{Rd}^{\Pi}(r, m)$$

where r is the incoming chaining value, m is a plaintext block, c is a ciphertext block, s is the outgoing chaining value, and Π is a permutation on blocks. In the algebraic representation, pick the matrices $M(\mathsf{Rd})$, $C(\mathsf{Rd})$, and $S(\mathsf{Rd})$ so that $m = M(\mathsf{Rd})\boldsymbol{v}_{\text{base}}$, $c = C(\mathsf{Rd})\boldsymbol{v}_{\text{base}}$, and $s = S(\mathsf{Rd})\boldsymbol{v}_{\text{base}}$.

Given such a round function Rd and compatible PRP F, we define the corresponding randomized encryption algorithm $\mathsf{Enc}_\ell^{\mathsf{Rd},F}$ for messages of size ℓ as follows:

$\mathsf{Enc}_\ell^{\mathsf{Rd},F}(m_1 \| \cdots \| m_\ell)$:
$r_0 \leftarrow \{0,1\}^\kappa$
for $i = 1$ to ℓ:
$(c_i, r_i) := \mathsf{Rd}^{F_K(\cdot)}(r_{i-1}, m_i)$
return $r_0 \| c_1 \| \cdots \| c_\ell$



We will consider the class of such block cipher modes where Rd is a Linicrypt algorithm (see Section II-C). However, we make no restrictions on *adversaries* and therefore target the standard notion of CPA (IND$-CPA) security for randomized encryption.

V. Characterization of Linicrypt Modes

A. Security

Although our main focus is to consider (variable-length) encryption algorithms, it is helpful to conceptually restrict the encryption algorithm to ℓ blocks to form a single (plain) Linicrypt program which we denote $\mathsf{Enc}_\ell^{\mathsf{Rd}}$. We can apply the results of the previous section to the collection of all $\{\mathsf{Enc}_\ell^{\mathsf{Rd}}\}_\ell$ Linicrypt programs. The goal of this section is to express the security of this collection of Linicrypt programs in terms of Rd only. When analyzing the properties of Rd using the definitions in Section III, we will treat the input chaining value as a fresh, uniformly random value, rather than an input.

Note that combining Linicrypt programs together isn't as straightforward as concatenating their input matrix, output matrix, and base variables together. Each output chaining value must be substituted in for the following block's input chaining value. Also, some oracle queries from different blocks may end up being syntactically the same, necessitating a merger of base variables. Consequently, after merging ℓ instances of Rd together in $\mathsf{Enc}_\ell^{\mathsf{Rd}}$, we must iteratively find syntactically equal queries $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q} \mapsto \boldsymbol{a}' \in \mathcal{O}(\mathsf{Enc}_\ell^{\mathsf{Rd}})$, pick $\boldsymbol{a}$ to be the representative of the two and substitute it for $\boldsymbol{a}'$ whenever it appears, and finally remove $\boldsymbol{q} \mapsto \boldsymbol{a}'$ from $\mathcal{O}(\mathsf{Enc}_\ell^{\mathsf{Rd}})$.

Also note that queries from distinct blocks might be collidable. However, any pair of colliding queries in Rd will be collidable in $\mathsf{Enc}_\ell^{\mathsf{Rd}}$, and any repeatable query in Rd will be repeatable in $\mathsf{Enc}_\ell^{\mathsf{Rd}}$.

Definition 15. *Consider an equivalence class $\{v_i\}_i$ of base variables under the "collidable" equivalence relation. A row vector $\boldsymbol{q}$ **directly depends** with factor c on $\{v_i\}_i$ if $c = \sum_i \boldsymbol{q} \cdot \boldsymbol{v}_i \neq 0$ (i.e., c is $\boldsymbol{q}$'s total coefficient for base variables in the class).*

Definition 16. *A row vector $\boldsymbol{u}$ **depends** on a collidability-equivalence class of base variables $\{v_i\}_i$ if it directly depends on the class, or there is some other oracle constraint $\boldsymbol{q} \mapsto \boldsymbol{a}$ such that $\boldsymbol{q}$ directly depends on $\{v_i\}_i$ and $\boldsymbol{u}$ depends on $\boldsymbol{a}$'s equivalence class.*

We divide the security of Linicrypt modes into two properties. The first is

Definition 17. *A round function Rd is **secure** if $\mathsf{Enc}_1^{\mathsf{Rd}}$ is IND$-CPA, i.e. the mode is IND$-CPA for single block messages.*

This condition can be determined by applying Theorem 10 to the (plain) Linicrypt program $\mathsf{Enc}_1^{\mathsf{Rd}}$. By Theorem 13, the condition is a sound and complete characterization for CPA security of single-block encryptions.

Definition 18. Rd *is **counter-like** if $S(\mathsf{Rd})$ does not depend on the output of any oracle query $\boldsymbol{q} \mapsto \boldsymbol{a} \in \mathcal{O}(\mathsf{Rd})$ where $\boldsymbol{q}$ depends on the chaining value r.*

Definition 19. Rd *is **non-repeating** if Rd is either not counter-like or $S(\mathsf{Rd})$ depends directly with factor g on the chaining value r and g's order in $\mathbb{F}^*$ is superpolynomial.*

We will show later that if Rd is repeating then the adversary can cause the chaining value to repeat. Of course, if the chaining value repeats, then an encryption of all-zero blocks will have repeated ciphertext blocks.

A counter-like round function is meant to model modes like

[5] Although we only consider modes that use a single PRP, generalization to multiple PRPs and PRFs with multiple inputs is straightforward.

CTR where the chaining value is incremented by a fixed value each round. In our setting, the field has small characteristic, so incrementing with the chaining value via field addition would lead to repeated chaining values. Instead, the reader should think of small-characteristic-counter-mode that updates the chaining value as $r' = r \cdot g$ where g has high order.[6]

To show that each block of the ciphertext is uniformly random, we need to show that each ciphertext block incorporates randomness that is independent of all other blocks. I.e., we must assign to each ciphertext block a unique, "fresh" oracle query to be the reason that ciphertext block is pseudorandom. The difficult case is when the mode is not counter-like. The chaining value in these round functions depends on *new* oracle queries within that round. Hence, the chaining values "accumulate" more oracle queries with every block. So if an oracle query $\boldsymbol{q}$ made in round i collapses with one $\boldsymbol{q}'$ made in a later round $j > i$, then $\boldsymbol{q}'$ represents a function of its chaining value that uses fewer oracle calls than the corresponding function for $\boldsymbol{q}$. We can use this kind of reasoning to relate collapses between rounds with the ordering of oracle queries within a single round.

Definition 20. *Consider the set of oracle queries $\mathcal{O}_d(\mathsf{Rd}) \subseteq \mathcal{O}(\mathsf{Rd})$ that depend on the input chaining value. Let $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}'$ be oracle queries in $\mathcal{O}_d(\mathsf{Rd})$. Define $\boldsymbol{q}' \prec \boldsymbol{q}$ to hold if for some $i < j$, when encrypting a message of length j, the query corresponding to $\boldsymbol{q}$ in block i is collidable with the query corresponding to $\boldsymbol{q}'$ in block j in $\mathsf{Enc}_j^{\mathsf{Rd}}$.*

To show that the $\prec$ relation among oracle queries is acyclic, we need the following lemmas.

Lemma 21. *If the round function is non-repeating, then no chaining value is in $R(\mathsf{Enc}_\ell^{\mathsf{Rd}})$. In fact, no query in $\mathcal{O}(\mathsf{Enc}_\ell^{\mathsf{Rd}})$ that corresponds to a query in $\mathcal{O}_d(\mathsf{Rd})$ is repeatable.*

Proof: The first half of this statement is clear in the counter-like case, as every chaining value depends directly on the first, which is freshly random. The same holds for the first block when Rd is not counter-like.

The rest of the proof proceeds by induction. Any query in $\mathcal{O}_d(\mathsf{Rd})$ must either depend directly on the input chaining value, which is non-repeatable, or on another query in $\mathcal{O}_d(\mathsf{Rd})$, which is non-repeatable by the induction hypotheses. Therefore it must be non-repeatable as well.

Finally, when Rd is not counter-like we need to go from one block to the next. Assume all of the queries corresponding to $\mathcal{O}_d(\mathsf{Rd})$ in block i are non-repeatable. Then since the output chaining value depends directly on at least one of them, it is also non-repeatable. ∎

Lemma 22. *If the round function is non-repeating, the difference of two chaining value from distinct blocks is not in $R(\mathsf{Enc}_\ell^{\mathsf{Rd}})$.*

[6]Interestingly, a high characteristic field would require us to change the definition of non-repeating. A round function that updates the chaining value via $r' = r + H(0)$ is counter-like, depending directly on r with factor 1, which has *low* multiplicative order — yet, the adversary cannot force chaining values to repeat if the field has high characteristic.

Proof: First, consider the counter-like case, where the output chaining value must be $s = gr + h$ for some expression $h \in R(\mathsf{Rd})$. Substitute this equation into the encryption function to get an equation for non-adjacent chaining values.

$$r_j = g^{j-i} r_i + \sum_{k=i+1}^{j} g^{j-k} h_k$$

If $r_j - r_i \in R(\mathsf{Rd})$ then we get

$$(g^{j-i} - 1) r_i - \sum_{k=i+1}^{j} g^{j-k} h_k \in R(\mathsf{Rd}).$$

As $g^{j-i} \neq 1$ if g's order is greater than the message length, we would have $r_i \in R(\mathsf{Enc}_\ell^{\mathsf{Rd}})$, contradicting Lemma 21.

In the other case, Rd is not counter-like. Let $\mathcal{O}_{\text{chain}}$ be the set of oracle queries $\boldsymbol{q} \mapsto \boldsymbol{a}$ in $\mathcal{O}(\mathsf{Rd})$ where $\boldsymbol{q}$ depends on the input chaining value and the output chaining value depends on the collidable equivalence class of $\boldsymbol{a}$. Rd not being counter-like implies that $\mathcal{O}_{\text{chain}}$ is not empty. Let $i \leq j$ be blocks, and $\boldsymbol{q} \mapsto \boldsymbol{a}, \boldsymbol{q}' \mapsto \boldsymbol{a}' \in \mathcal{O}_{\text{chain}}$ be oracle queries, with either the blocks distinct, or the queries distinct with $\boldsymbol{q}'$ depending on $\boldsymbol{a}$ We prove by induction that the query corresponding to $\boldsymbol{q}$ in block i cannot collide with the query corresponding to $\boldsymbol{q}'$ in block j.

Assume the opposite, that there are colliding oracle queries $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \mathcal{O}(\mathsf{Enc}_j^{\mathsf{Rd}})$ with $\boldsymbol{q}_1$ corresponding to $\boldsymbol{q}$ in block i and $\boldsymbol{q}_2$ corresponding to $\boldsymbol{q}'$ in block j. Then by Lemma 8, $\boldsymbol{q}_2 - \boldsymbol{q}_1 \in R(\mathsf{Enc}_j^{\mathsf{Rd}})$. However, their inputs in $\mathcal{O}_{\text{chain}}$ are non-repeatable by Lemma 21. $\boldsymbol{q}'$ depends directly either on the input chaining value, or on $\boldsymbol{a}''$ for another query $\boldsymbol{q}'' \mapsto \boldsymbol{a}'' \in \mathcal{O}_{\text{chain}}$, where $\boldsymbol{q}''$ depends on (or is) $\boldsymbol{q}$ if $i = j$. In the latter case, then by induction the query $\boldsymbol{q}_3$ corresponding to $\boldsymbol{q}''$ in block j cannot cannot collide with anything in block i that $\boldsymbol{q}$ depends on, so $\boldsymbol{q}_2 - \boldsymbol{q}_1$ cannot be repeatable. In the former case, then use the output chaining value of the previous block instead of $\boldsymbol{a}''$, which must depend directly on a query $\boldsymbol{q}''$ in $\mathcal{O}_{\text{chain}}$, and a similar argument shows that this case as impossible as well.

Our result then follows from this induction. There must be at least one query in $\mathcal{O}_{\text{chain}}$ that the output chaining value depends on directly. The corresponding query in block i cannot is not repeatable by Lemma 21 and cannot collide with the corresponding query in block j. Collision is the only way for the difference of two non-repeatable oracle queries outputs to be repeatable. ∎

Theorem 23. *If the round function is non-repeating, $\prec$ defines a strict partial order on $\mathcal{O}_d(\mathsf{Rd})$*

Proof: First, we establish that $\prec$ is transitive. If $\boldsymbol{q}' \prec \boldsymbol{q}$ and $\boldsymbol{q}'' \prec \boldsymbol{q}'$ then by considering a suitably long plaintext we can have a situation where $\boldsymbol{q}$ in round i, $\boldsymbol{q}'$ in round j, and $\boldsymbol{q}''$ in round k are all collidable, and $i < j < k$. Although this requires increasing the plaintext size, it only requires expansion by a factor linear in the length of the Linicrypt program to handle the longest chain required for the transitive closure of all $\prec$ relations.

To establish irreflexivity, first assume that irreflexivity holds for all queries in $\mathcal{O}_d(\mathsf{Rd})$ that come before $\boldsymbol{q} \mapsto \boldsymbol{a}$ in program

order, but (for the sake of contradiction) $\boldsymbol{q} \prec \boldsymbol{q}$. Then there must be oracle queries $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \mathcal{O}(\mathsf{Enc}_j^{\mathsf{Rd}})$ where q_1 is the query corresponding to $\boldsymbol{q}$ in block i and q_2 is the corresponding query in block j, and q_1 is collidable with q_2.

Let $\mathcal{O}'$ be the subset of $\mathcal{O}_d(\mathsf{Rd})$ consisting of those queries $\boldsymbol{q}' \mapsto \boldsymbol{a}'$ where $\boldsymbol{q}$ directly depends on the collidable equivalence class of $\boldsymbol{a}'$. Then $\mathcal{O}'$ forms a strict partial order, so either it is empty or there exists some maximal query $\boldsymbol{q}_{\max} \mapsto \boldsymbol{a}_{\max} \in \mathcal{O}'$ that is not less than any query. If it is empty, $\boldsymbol{q}$ is a linear combination of the chaining value and something in $R(\mathsf{Enc}_j^{\mathsf{Rd}})$. By Lemma 8, $\boldsymbol{q}_2 - \boldsymbol{q}_1 \in R(\mathsf{Enc}_j^{\mathsf{Rd}})$, and as all other terms in $\boldsymbol{q}$ are repeatable this implies that the difference of the two chaining values is repeatble, contradicting Lemma 22.

Otherwise, maximality implies that $\boldsymbol{q}_{\max}$ from block j cannot collide with any query in $\mathcal{O}'$ from any earlier block i, and so cannot collide with any query that $\boldsymbol{q}_1$ depends directly on. Then $\boldsymbol{q}_2 - \boldsymbol{q}_1$ cannot be repeatable as it depends directly on $\boldsymbol{a}_{\max}$, which is not repeatable since it is in $\mathcal{O}_d(\mathsf{Rd})$ by Lemma 21. This contradicts Lemma 8. ∎

Finally, we can prove our result using this order on $\mathcal{O}_d(\mathsf{Rd})$.

Theorem 24. *A Linicrypt block cipher mode is IND$-CPA secure if and only if its round function is both secure and non-repeating.*

Proof: We first prove the if direction. Let $\mathcal{O}_{\text{out}}$ be the subset of $\mathcal{O}_d(\mathsf{Rd})$ consisting of those queries $\boldsymbol{q}' \mapsto \boldsymbol{a}'$ where $C(\mathsf{Rd})$ directly depends on the collidable equivalence class of $\boldsymbol{a}'$. There must be at least one query in $\mathcal{O}_{\text{out}}$ for the round function to be secure, as otherwise the adversary could subtract out the term containing the input chaining value afterwards, since the initial chaining value (the IV) is public, leaving a repeatable value. Doing this to two consecutive zero messages will produce the same result, which distinguishes the ciphertexts from uniform random while only using single block messages. Therefore, since $\prec$ defines a strict partial order, there must exist some maximal query $\boldsymbol{q}_m \mapsto \boldsymbol{a}_m$ that is not less than any other query in $\mathcal{O}_{\text{out}}$.

We want to show that $\mathsf{Enc}_\ell^{\mathsf{Rd}}$ satisfies the condition of Theorem 10 to prove that the block cipher mode is IND$-CPA. To show that the rows of $\begin{bmatrix} U(\mathsf{Enc}_\ell^{\mathsf{Rd}}) \\ C(\mathsf{Enc}_\ell^{\mathsf{Rd}}) \end{bmatrix}$ are linearly independent, we find a pivot base variable in each row of $C(\mathsf{Enc}_\ell^{\mathsf{Rd}})$, where each pivot is required to be linearly independent from every previous row of $\begin{bmatrix} U(\mathsf{Enc}_\ell^{\mathsf{Rd}}) \\ C(\mathsf{Enc}_\ell^{\mathsf{Rd}}) \end{bmatrix}$. The first pivot is the initial chaining value. For the ith subsequent row, the pivot is $\boldsymbol{a}_m$ in round i. It is not repeatable because it is in $\mathcal{O}_d(\mathsf{Rd})$. Because $\boldsymbol{q}_m$ is maximal, neither it nor any other query in its collidable (in Rd) class can collide (in $\mathsf{Enc}_\ell^{\mathsf{Rd}}$) with any query with a nonzero entry in $C(\mathsf{Enc}_\ell^{\mathsf{Rd}})$ in any previous round, so $\boldsymbol{a}_m$ is linearly independent from all previous rows.

Next we show the converse. If the round function is not secure then there is an attack against the encryption mode that involves only single-block plaintexts.

If the round function fails to be non-repeating, then it must be counter-like with g zero or having polynomial order n. The adversary can then send the all-zeros message of length cn, where $c = \text{char}(\mathbb{F})$ is the characteristic of the field. As with Lemma 22, $s = gr + h$ for $h \in R(\mathsf{Rd})$, and so

$$r_{cn} = g^{cn} r_0 + \sum_{k=1}^{cn} g^{cn-k} h_k = r_0 + \sum_{k=1}^{n} c g^{-k} h = r_0,$$

as h will be the same in each block since it is repeatable and the all zeroes message was sent each time, and because each block has only a single input so the all zeroes message causes all collidable queries within the block to collide. Similarly, if $g = 0$ then the second and third blocks of the message will have colliding input chaining values. The repeated chaining value then leads to a repeated ciphertext block, as the plaintext blocks were all the same, which distinguishes the output from random. ∎

Remark 25. *A often useful strategy when attacking the CPA security of an encryption scheme is to request an encryption of the all-zeroes plaintext and see what happens. Teachers of introductory cryptography classes may recall giving this advice to their students on many occasions. Notice that the adversaries constructed to show completeness in Theorem 24 always use the all-zeroes plaintext, which shows that this technique is enough for all Linicrypt block cipher modes.*

B. Correctness

Security considers what happens from the perspective of an adversary who does not have access to the oracle/PRP. In order to reason about correctness, we must now consider whether the plaintexts can be recovered by the decryption algorithm, which *does* have access to the oracle/PRP (and its inverse!). First, we reduce the problem to the single ciphertext case.

Theorem 26. *A Linicrypt block cipher mode satisfies correctness if $\mathsf{Enc}_1^{\mathsf{Rd}}$ does, i.e. if it does for one block plaintexts.*

Proof: We can decrypt a long ciphertext $(c_0, c_1, \ldots)$ inductively, as follows. The invariant is that we can compute not only the plaintext m_i for every i, but also the ith chaining value r_i. Note that the initial chaining value is contained in the ciphertext as c_0. Now, given r_i, we can decrypt the ciphertext (r_i, c_{i+1}) using the single-block decryption procedure, to obtain m_{i+1}. Then we can run Rd *in the forward direction* on r_i and m_{i+1}. Since Rd is deterministic, this will result in both c_{i+1} and the output chaining value r_{i+1}. ∎

Next, we need a way of efficiently deciding correctness for $\mathsf{Enc}_1^{\mathsf{Rd}}$. To do this, we use the notion of reachability from [1], adapting it slightly to handle decryption operations to invert the oracle queries.

Definition 27. *The **reachable values** of a Linicrypt program L are the smallest linear subspace $\mathcal{R}(L)$ such that*

(i) every row of $C(L)$ is in $\mathcal{R}(L)$, and

(ii) if $\boldsymbol{q} \mapsto \boldsymbol{a}$ is an oracle query and either $\boldsymbol{a}$ or $\boldsymbol{q}$ is in $\mathcal{R}(L)$, then the other is as well.

Theorem 28. *A Linicrypt block cipher mode satisfies correctness if and only if the plaintext $M(\mathsf{Enc}_1^{\mathsf{Rd}})$ is reachable in the Linicrypt program $\mathsf{Enc}_1^{\mathsf{Rd}}$.*

Proof: To construct a decryption algorithm, we show how to determine $\boldsymbol{u} \cdot \boldsymbol{v}_{\text{base}}$ for every $\boldsymbol{u} \in \mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$, including the plaintext. First, for every $\boldsymbol{u} \in \mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$ that owes its membership to case (a), we already have $\boldsymbol{u} \cdot \boldsymbol{v}_{\text{base}}$ as one of the components of the output ciphertext $C(\mathsf{Enc}_1^{\mathsf{Rd}})\boldsymbol{v}_{\text{base}}$. If for some oracle query $\boldsymbol{q} \mapsto \boldsymbol{a}$ we have $\boldsymbol{q}$ in $\mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$ then $\boldsymbol{q}\boldsymbol{v}_{\text{base}}$ is a linear combination of known values and so we can calculate it. Evaluating the block cipher on that value then gives us the corresponding $\boldsymbol{a}\boldsymbol{v}_{\text{base}}$. Similarly, we can use the block cipher in reverse to find $\boldsymbol{q}$ given a known $\boldsymbol{a}$. As $\mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$ is the smallest linear subspace generated by these operations every reachable value can be obtained by a finite number of these operations.

To show the converse we use the technique of canonical simulation. Canonical simulation requires a input-less Linicrypt program, and we choose to modify $\mathsf{Enc}_1^{\mathsf{Rd}}$ to generate a uniformly random plaintext to meet this requirement. A correct decryption algorithm would, a fortiori, work on a random plaintext.

As part of the proof of Lemma 4 from [1], we have that an efficient decryption algorithm given the output from a Linicrypt program and access to the oracle has negligible probability of guessing any value that is not reachable. To summarize their argument, we may perform a change of basis so that the reachable subspace $\mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$ is $\{0\}^d \times \mathbb{F}^{\text{base}-d}$, where base $- d$ is the dimension of the reachable subspace. Look at the first oracle query or inverse query to be made by the decryption algorithm that matches a query $\boldsymbol{q} \to \boldsymbol{a}$ made by the Linicrypt program, where $\boldsymbol{q}$ and $\boldsymbol{a}$ are not reachable. So far all information given to the decryption algorithm has been reachable as the IV and ciphertext are, and every previous oracle query either produced a random value as it didn't match one in the Linicrypt program or it matched a reachable one. Therefore, it is syntactically independent of the first d base variables, and the decryption algorithm will produce the same input to the oracle no matter what values these variables take. The query is not reachable so there must be at least one nonzero value in the first d entries of $\boldsymbol{q}$ and $\boldsymbol{a}$, and the decryption algorithm then has negligible probability of guessing either of them. ∎

VI. A Tour of the Secure Single-Query Modes

In the preceding sections, we established criteria for IND$-CPA security and correctness of Linicrypt block cipher modes. In this section we focus our attention on **single-query** modes, where the Rd function makes only a single call to the PRP. These are the most natural block cipher modes in practice. We apply the findings of the previous sections to this special class of Linicrypt modes to find all secure+correct modes.

In more detail, the objects we study are those Linicrypt Rd programs with

- two inputs: an input chaining value r and a message block m,
- two outputs: a ciphertext block c and an output chaining value s, and
- a single oracle query, $y := \Pi(x)$.

A generic template encompassing all candidate round functions is shown below, where $\mathsf{a}, \mathsf{b}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}, \mathsf{h}$, and i are fixed coefficients from $\mathbb{F}$.

$\mathsf{Rd}^{\Pi}(r, m)$:
$x := \mathsf{a}r + \mathsf{b}m$
$y := \Pi(x)$
$c := \mathsf{d}r + \mathsf{e}m + \mathsf{f}y$
$s := \mathsf{g}r + \mathsf{h}m + \mathsf{i}y$
return c, s

We now use the criteria for security and correctness developed in the preceding sections to determine all choices for these coefficients that produce secure, correct block cipher modes.

A. Security constraints

Application of Theorem 24 to this class of Linicrypt Rd functions yields the following result.

Lemma 29. *A single-query Linicrypt block cipher mode is IND$-CPA secure if and only if its Rd function satisfies:*

(i) $\mathsf{a} \neq 0$,
(ii) $\mathsf{f} \neq 0$, *and*
(iii) $\mathsf{i} \neq 0$, *or* $\mathsf{ord}(\mathsf{g})$ *in* $\mathbb{F}^{\times}$ *is superpolynomial in* κ.

Proof: Recall that the block cipher mode is secure if and only if its round function is secure and non-repeating. First we consider the "security" property which involves the mode's restriction to single-block messages. Let $L = \mathsf{Enc}_1^{\mathsf{Rd}}$. L receives m as input, generates r uniformly, and outputs $c_0 = r$ and $c_1 = \mathsf{Rd}^{\Pi}(r, m)$. The algebraic representation for $\mathsf{Enc}_1^{\mathsf{Rd}}$ is shown below.

$$M(L) = \begin{matrix} r & m & y \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}; \qquad \mathcal{O}(L) = \{\ [\mathsf{a}\ \mathsf{b}\ 0] \mapsto [0\ 0\ 1]\ \};$$

$$C(L) = \begin{matrix} \\ c_0 \\ c_1 \end{matrix} \begin{matrix} r & m & y \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ \mathsf{d} & \mathsf{e} & \mathsf{f} \end{bmatrix}.$$

The zeroable subspace $Z(L)$ is defined based on pairs of oracle queries. Since there is only one oracle query in this case, the zeroable subspace $Z(L)$ is simply $Z(L) = \text{rowspace}(M(L))$.

The repeatable subspace $R(L)$ is the subspace of vectors reachable from $Z(L)$ with access to Π, and $U(L)$ is defined to be a basis for $R(L)$. By Theorem 10 and Theorem 13, L is IND$-CPA secure if and only if the rows of $\begin{bmatrix} C(L) \\ U(L) \end{bmatrix}$ are linearly independent.

There are two cases to consider. First, consider $\mathsf{a} = 0$. Then the oracle query input $([\mathsf{a}\ \mathsf{b}\ 0])$ is in $Z(L)$, and so the oracle query output $[0\ 0\ 1]$ is included in the space $R(L)$. In this

case, the the basis $U(L)$ for $R(L)$ is $U(L) = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ and

$$\begin{bmatrix}C(L)\\ U(L)\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ \mathsf{d} & \mathsf{e} & \mathsf{f}\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

The rows of this matrix cannot be linearly independent, because there are more rows than columns, so the Rd function is not secure in this case. This result should make sense intuitively, as it describes a round function where the input to the block cipher is completely under the adversary's control (it doesn't depend on chaining value r).

For the other case, consider $\mathsf{a} \neq 0$. Then we can see that the oracle query input ([a b 0]) is not reachable from $Z(L)$, so we get $R(L) = Z(L)$ and therefore its basis is $U(L) = [0 \quad 1 \quad 0]$ and

$$\begin{bmatrix}C(L)\\ U(L)\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ \mathsf{d} & \mathsf{e} & \mathsf{f}\\ 0 & 1 & 0\end{bmatrix}$$

The rows of this matrix are linearly dependent if and only if $\mathsf{f} \neq 0$ (the ciphertext must depend on the output of the block cipher).

Thus, the induced single-block encryption scheme L is secure on a single-block message if and only if $\mathsf{a} \neq 0$ and $\mathsf{f} \neq 0$.

Next, we impose constraints necessary and sufficient for Rd to be non-repeating. Definition 19 says that there are two cases to consider based on whether or not Rd is counter-like.

The generic single-query Rd function we are considering is not counter-like, that is, s depends on the output of an oracle query that depends on r, if and only if $\mathsf{a} \neq 0$ (so that y depends on r) and $\mathsf{i} \neq 0$ (so that s depends on y).

If, on the other hand, $\mathsf{a} = 0$ or $\mathsf{i} = 0$, then Rd is counter-like, and the only source of randomness for all chaining values is the IV, s_0. In this case, for the chaining value s_j in some block j,

$$s_j = \mathsf{g}^j s_0 + \phi_j(m_1, ..., m_j),$$

where $\phi_j : \mathbb{F}^j \to \mathbb{F}$ is a linear map if $\mathsf{i} = 0$, but otherwise depends on Π. Therefore, if $\mathsf{a} = 0$ or $\mathsf{i} = 0$, Rd is non-repeating if and only if $\mathsf{g} \neq 0$ and has order superpolynomial in κ. For instance, we can choose g to be a primitive field element, i.e., $\langle \mathsf{g} \rangle = \mathbb{F}^\times$, so $\mathsf{ord}(\mathsf{g}) = |\mathbb{F}| - 1$.

Thus, Rd is non-repeating if and only if $\mathsf{a} \neq 0$ and $\mathsf{i} \neq 0$, or $\mathsf{g} \neq 0$ and $\mathsf{ord}(\mathsf{g})$ in $\mathbb{F}^\times$ is superpolynomial in κ.

Combining the constraints for single-block security and non-repeating chaining values yields those stated in the lemma. ■

B. Correctness constraints

Theorem 26 says that correctness of single-block encryption is sufficient for correctness of a Linicrypt block cipher mode, and Theorem 28 says that the single-block encryption program is correct if and only if the message is reachable from the ciphertext, i.e. if and only if $m \in \mathcal{R}(\mathsf{Enc}_1^{\mathsf{Rd}})$.

The ciphertext matrix and oracle constraints for $\mathsf{Enc}_1^{\mathsf{Rd}}$ are shown below.

$$C(\mathsf{Enc}_1^{\mathsf{Rd}}) = \begin{array}{c} \\ c_0 \\ c_1 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} r & m & y \end{array} \\ \begin{bmatrix}1 & 0 & 0\\ \mathsf{d} & \mathsf{e} & \mathsf{f}\end{bmatrix} \end{array}$$

$$\mathcal{O}(\mathsf{Enc}_1^{\mathsf{Rd}}) = \{\ [\mathsf{a}\ \mathsf{b}\ 0] \mapsto [0\ 0\ 1]\ \}$$

Proposition 30. *A single-query Linicrypt block cipher round function is correct if and only if* $(\mathsf{e} \neq 0) \oplus ((\mathsf{b} \neq 0) \wedge (\mathsf{f} \neq 0))$.

Proof: Assume that m is reachable from the ciphertext output.

Case 1: Suppose $\mathsf{e} = 0$. Then m is not a simple linear combination of c_0 and c_1, so reachability of m must use some oracle query.

Case 1a: $y = [0\ 0\ 1]$ is a linear combination of c_0 and c_1, so that x is reachable as $\Pi^{-1}(y) \to x$. This is true if and only if $\mathsf{f} \neq 0$. Then m is a linear combination of c_0, c_1, x. With $\mathsf{e} = 0$ and $\mathsf{f} \neq 0$ we must have m be a linear combination of c_0 and x alone, meaning that $\mathsf{b} \neq 0$.

Case 1b: $x = [\mathsf{a}\ \mathbf{b}\ 0]$ is a linear combination of c_0 and c_1, so that x is reachable as $\Pi(x) \to y$. This is true if and only if $\mathsf{f} = 0$. But with $\mathsf{e} = \mathsf{f} = 0$ we must now have $\mathsf{b} = 0$ and so m is not a linear combination of c_0, c_1, y. So m cannot be reachable in this case.

Case 2: Suppose $\mathsf{e} \neq 0$.

Case 2a: If $\mathsf{f} = 0$ then m is a simple linear combination of c_0 and c_1, and hence reachable (although the scheme is insecure in this case).

Case 2b: Otherwise $\mathsf{f} \neq 0$. With $\mathsf{e} \neq 0$, y will not be a linear combination of c_0, c_1, *i.e.*, decryption cannot make the $\Pi^{-1}(y) \to x$ query. Instead, $x = [\mathsf{a}\ \mathsf{b}\ 0]$ must be a linear combination of c_0 and c_1, making y reachable as $\Pi(x) \to y$. With $\mathsf{e}, \mathsf{f} \neq 0$ this is possible if and only if $\mathsf{b} = 0$. Subsequently, m must be a linear combination of c_0, c_1, y, but this is always the case for the constraints established so far. ■

Finally, we combine Theorem 29 and Theorem 30 to characterize all valid single-query Linicrypt block cipher modes.

Theorem 31. *A single-query Linicrypt* Rd *function is a correct, IND$-CPA secure block cipher mode if and only if*

(i) $\mathsf{a} \neq 0$,
(ii) $(\mathsf{b} \neq 0) \oplus (\mathsf{e} \neq 0)$,
(iii) $\mathsf{f} \neq 0$, *and*
(iv) $\mathsf{i} \neq 0$, *or* $\mathsf{ord}(\mathsf{g})$ *in* $\mathbb{F}^\times$ *is superpolynomial in* κ.

C. Graphical Summary of IND$-CPA Secure Single-Query Modes

We now summarize the results of Theorem 31 graphically. Recall that a single-query Rd function is parameterized by 8 coefficients. We partition the space of these coefficients into two main categories and four total subcategories. If we consider only whether the different coefficients are zero or non-zero, there are 24 different secure block cipher modes.

1) XOR-only Modes ($\mathsf{i} \neq 0$): When $\mathsf{i} \neq 0$, the coefficients $\mathsf{d}, \mathsf{g}, \mathsf{h}$ are unrestricted, and we further have a choice of which of b, e is zero and which is nonzero.

In this category, there is no restriction on g, which means it is possible (though not mandatory) to choose all coefficients from $\{0,1\}$. When a Linicrypt program over $GF(2^\lambda)$ has all coefficients from $\{0,1\}$, it corresponds to a program that consists only of XOR operations and calls to the oracle/PRP. Hence, we call this category of modes "XOR-only". When restricting coefficients to $\{0,1\}$, we see that the condition leads to a total of $2^4 = 16$ possible XOR-only modes.

Figure 1 shows all IND$-CPA secure Linicrypt modes in this category. The meaning of the lines in the figure is as follows:

- A solid line indicates multiplication by any *nonzero* coefficient.
- A dashed line indicates multiplication by *any* (even possibly zero) coefficient.

Of course, the absence of a line indicates the lack of any dependence between values.

The subfigure on the left corresponds to the case of $\mathsf{b} \neq 0$, $\mathsf{e} = 0$. If we set as many coefficients to zero as possible, and set the rest to one (*i.e.*, remove all dashed lines), we obtain **CBC mode**.

The subfigure on the right corresponds to the case of $\mathsf{b} = 0$, $\mathsf{e} \neq 0$. These modes do not require Π to be invertible, and consequently work with PRFs as well as PRPs. If we set as many coefficients to zero as possible, and set the rest to one (*i.e.*, remove all dashed lines), we obtain **OFB mode**.

2) Counter-like Modes ($\mathsf{i} = 0$): When $\mathsf{i} = 0$, we see that g must have high order, exactly one of b, e must be zero, while d, h are unrestricted.

We call this category of modes "counter-like" because, like CTR mode, the output chaining value does not depend on any calls to the PRP. Linicrypt does not allow incrementing as in standard CTR mode, but we can achieve an analogous effect by multiplying the previous chaining value by a (fixed) primitive field element g. One can think of this as incrementing the chaining value "in the exponent" of g.

Figure 2 summarizes IND$-CPA secure Linicrypt modes in this category. The subfigure on the left corresponds to the case $\mathsf{b} \neq 0, \mathsf{e} = 0$, in which the plaintext is incorporated into the PRP's input.

The subfigure on the right corresponds to the case $\mathsf{b} = 0, \mathsf{e} \neq 0$, in which the plaintext is combined with the PRP's output. These modes do not require Π to be invertible, and consequently work with PRFs instead of PRPs. Setting as many coefficients to zero as possible, choosing g to be a primitive element of $\mathbb{F}^\times$, and all other coefficients to be 1 leads to a multiplicative $GF(2^\lambda)$-analog of **CTR mode**. The jth ciphertext block has the form $c_j = F(k, r \cdot \mathsf{g}^{j-1}) \oplus m_j$ where r is the initialization vector.

VII. Problems with MKG14 Characterization

Now that we have presented our own characterization of secure Linicrypt block cipher modes, we briefly review the characterization of secure block cipher modes introduced by Malozemoff et al. in [7], which covers a similar class of programs. We show that their characterization is not sound.

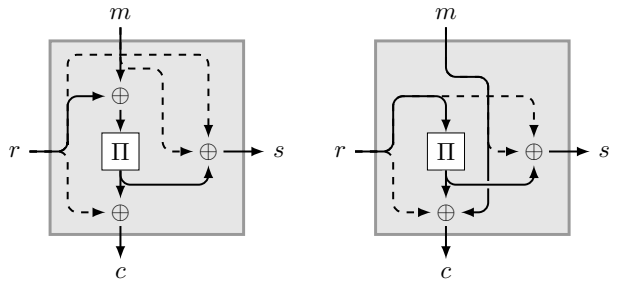


Fig. 1. All decryptable, IND$-CPA secure single-query **XOR-only** Linicrypt modes ($\mathsf{i} \neq 0$).

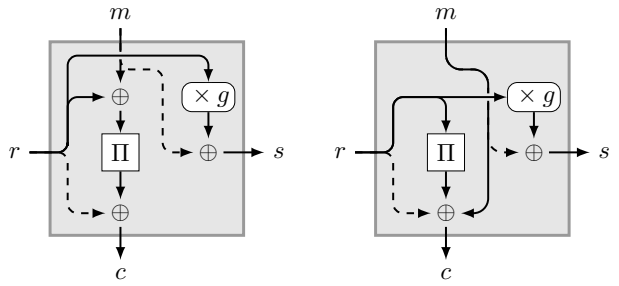


Fig. 2. All decryptable, IND$-CPA secure single-query **counter-like** Linicrypt modes ($\mathsf{i} = 0$).

A. Review of MKG14 Characterization

In [7], a block cipher mode consists of a probabilistic Init algorithm and a deterministic Block algorithm, both expecting oracle access to a keyed block cipher F_k:

$$(c_0, s_0) := \mathsf{Init}^{F_k}(1^n) \qquad (c, s) := \mathsf{Block}^{F_k}(r, m)$$

Init is a generalization of IV generation, where the initial chaining value $s_0 \in \{0,1\}^n$ and first ciphertext block $c_0 \in \{0,1\}^n$ need not be the same. Block is the round function. Each algorithm consists of a sequence of instructions that perform operations on inputs and intermediate values. Their basic instruction set includes taking the XOR of two values (the XOR instruction) and querying the block cipher F_k on a value (the PRF or PRP instruction). Additionally, random values can be generated in the Init algorithm (with the GENRAND instruction). A block cipher mode $(\mathsf{Init}, \mathsf{Block})$ is considered *secure* if the induced encryption scheme is IND$-CPA secure when F is a secure PRP.

The basic instruction set is a subset of the Linicrypt instruction set, so every MKG14 algorithm can be translated to an equivalent Linicrypt program. In the appendix of [7], they augment their characterization to include the increment (INC) instruction, which encompasses additional modes such as CTR mode. Increment is not a linear operation and, as such, is not allowed in Linicrypt (although similar functionality can be obtained by the linear operation $x \mapsto gx$ where $\mathsf{ord}(g)$ is superpolynomially large). We focus on their basic instruction set, where direct comparisons with our characterization are possible.

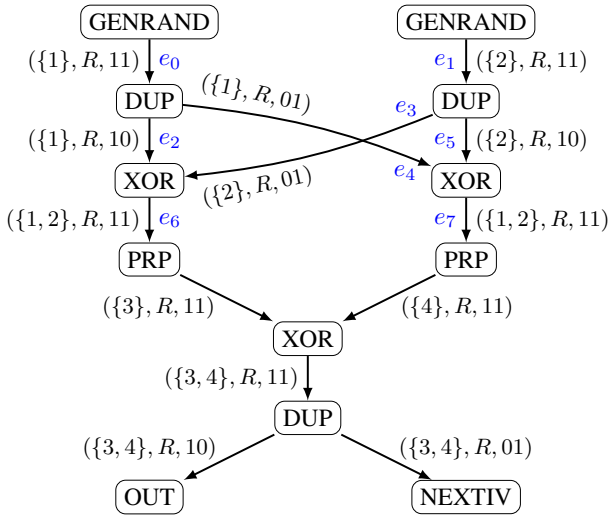


Fig. 3. An insecure Init algorithm (that always chooses $c_0 = s_0 = 0$) with a valid labeling under the MKG14 scheme.

The Init and Block algorithms are modeled by directed acyclic graphs whose nodes correspond to instructions and whose edges hold intermediate values. Since each edge is incident to exactly two nodes, values must be duplicated explicitly via a DUP instruction in order to be used more than once.

Let G be the union of the Init and Block graphs. The authors define an edge labeling scheme and claim that **if G has a valid labeling under this scheme, then** $(\mathsf{Init}, \mathsf{Block})$ **is a secure block cipher mode**. Note that this is not a biconditional statement. They do not claim completeness. We introduce relevant aspects of the labeling scheme in Appendix B-B. Please see [7] for the full details.

B. Counterexamples to Soundness and Completeness

In Figure 3 we show an Init algorithm which is insecure (always choosing an all-zeroes initialization vector), but which has a valid labeling under the MKG14 rules. Details of this counterexample are given in Appendix B-C. Roughly speaking, the labeling rules of MKG14 do not fully capture algebraic aspects of XOR expressions. In particular, it is possible to "fool" the labeling rules into finding two different expressions as jointly uniform, when they are actually identical expressions.

Although the authors of [7] do not claim that their characterization is complete, they do not provide any counterexample to its completeness. In the full versionwe give a concrete example of a secure block cipher mode which has no valid labeling under the MKG14 rules.

Acknowledgements

First two authors partially supported by NSF award 1617197. Third author supported by a DoE CSGF Fellowship. The authors are grateful to anonymous CSF reviewers for their helpful suggestions.

References

[1] B. Carmer and M. Rosulek. Linicrypt: A model for practical cryptography. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 416–445. Springer, Heidelberg, Aug. 2016.

[2] M. Gagné, P. Lafourcade, Y. Lakhnech, and R. Safavi-Naini. Automated security proof for symmetric encryption modes. In A. Datta, editor, *Advances in Computer Science - ASIAN 2009*, volume 5913 of *LNCS*, pages 39–53. Springer, 2009.

[3] M. Gagné, P. Lafourcade, Y. Lakhnech, and R. Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In J. García-Alfaro and P. Lafourcade, editors, *Foundations and Practice of Security*, volume 6888 of *LNCS*, pages 23–31. Springer, 2011.

[4] M. Gagné, P. Lafourcade, Y. Lakhnech, and R. Safavi-Naini. Automated proofs of block cipher modes of operation. *J. Autom. Reasoning*, 56(1):49–94, 2016.

[5] V. T. Hoang, J. Katz, and A. J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 84–95. ACM Press, Oct. 2015.

[6] H. Lin, C. Lynch, A. M. Marshall, C. A. Meadows, P. Narendran, V. Ravishankar, and B. Rozek. Algorithmic problems in the symbolic approach to the verification of automatically synthesized cryptosystems. In *International Symposium on Frontiers of Combining Systems*, pages 253–270. Springer, 2021.

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

[8] I. McQuoid, T. Swope, and M. Rosulek. Characterizing collision and second-preimage resistance in linicrypt. In D. Hofheinz and A. Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 451–470. Springer, Heidelberg, Dec. 2019.

[9] C. Meadows. Symbolic security criteria for blockwise adaptive secure modes of encryption. Cryptology ePrint Archive, Report 2017/1152, 2017. https://ia.cr/2017/1152.

[10] C. Meadows. Symbolic and computational reasoning about cryptographic modes of operation. Cryptology ePrint Archive, Report 2020/794, 2020. https://eprint.iacr.org/2020/794.

[11] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

Appendix A
NP-Hardness of Linicrypt Encryption Schemes

Theorem 32. *Determining IND$-CPA security of Linicrypt encryption schemes is NP-hard.*

Proof: We use reduction from one-in-three satisfiability [11], the problem of determining if there are assignment of the boolean variables $x_1, x_2, \ldots, x_n$ such that for every triplet (i, j, k) in the set S of clauses, exactly one of x_i, x_j, x_k is true. We create a Linicrypt program L that outputs one of its plaintext blocks if its input represents a satisfying assignment, but is uniformly otherwise. We design L to match the signature of the block cipher modes considered in this paper closely, though this increases its complexity. Specifically, there is only a single IV block sampled at the start, and the messages are all decryptable. The only difference is that the encryption is not structured as a function Rd that gets repeated for each block in the message.

We build a Linicrypt program L that encrypts $n + 2$ block messages. For the first $n + 1$ blocks, it is the multiplicative

analog of CTR mode (see Section VI-C).[7] The last message block is instead encrypted using GADGET, which we design to be random unless the adversary satisfies the clause.

$L(m_1, m_2, \ldots, m_{n+2})$:
  $r \leftarrow \$$
  for $i = 1$ to $n+1$:
    $c_i := \Pi(g^{i-1}r) + m_i$
  $c_{n+2} := m_{n+2}$
  for the ith clause $(x, y, z) \in S$
    c_{n+2} += GADGET$(\Pi(g^{i+n}r), \Pi(m_{n+1}), m_x, m_y, m_z)$
  return $r, c_1, \ldots, c_{n+2}$

Each clause i is given its own freshly random value $r_i = \Pi(g^{i+n}r)$. We use this random value to avoid overlaps between clauses, or between different calls to L. For each independently random r_i and each integer j, the subroutine $H_j(r, \cdot)$ defined below is an independent random function.

$H_j(r, x)$:
  return $\Pi(g^j r + x)$

First, we build a NAND gadget, enforcing that either $x = 0$ or $y = 0$. Otherwise, it will produce a uniformly random value.

$\text{NAND}_j(r, x, y)$:
  $a := H_j(r, x) - H_j(r, 0)$
  $b := H_{j+1}(r, y) - H_{j+1}(r, 0)$
  return $H_{j+2}(r, a) + H_{j+2}(r, b)$
    $- H_{j+2}(r, a + b) - H_{j+2}(r, 0)$

If $x = 0$ then $a = 0$, so $H_j(r, a)$ collides with $H_j(r, 0)$ and $H_j(r, a + b)$ collides with $H_j(r, b)$, so NAND outputs zero. Similarly, if $y = 0$ then $b = 0$, and so $H_j(r, b)$ collides with $H_j(r, 0)$ and $H_j(r, a+b)$ collides with $H_j(r, a)$, also making the output be zero.

In the other direction, for NAND each H_{j+2} must collide with another hash. This can happen in three ways. Two have already been covered in the $a = 0$ and $b = 0$ hypotheticals above, and in fact these collisions do imply that $a = 0$ or $b = 0$. In turn, that implies that $x = 0$ or $y = 0$, because if $a = 0$ then the two H_j hashes must collide. The final possible set of collisions is if $\text{char}(\mathbb{F}) = 2$ and $a = b$, which would also cause the output to be zero. But $a = b$ implies that $a = b = 0$, because otherwise there would need to be a collision between a H_j query and a H_{j+1} query, which has negligible probability.

Finally, we can construct the full gadget. First, we use NAND to enforce that at most one of x, y, z can be non-zero. Then, we require the remaining variable to be $p = \Pi(m_{n+1})$ by checking if the sum of the hashes is correct.

GADGET(r, p, x, y, z):
  $a := \text{NAND}_0(r, x, y) + \text{NAND}_3(r, y, z) + \text{NAND}_6(r, z, x)$
  $b := H_9(x) + H_9(y) + H_9(z)$
  $c := H_9(p) + H_9(0) + H_9(0)$
  return $a + b - c$

[7] For a construction that avoids using multiplication-by-a-constant, replace $\Pi(g^i r)$ with $\Pi(x) + x$, where x is the OFB chaining value $\Pi^{(i)}(r)$.

To stop GADGET from returning a fresh random value, the adversary needs to force $a = 0$ and $b = c$. The former holds if only if most one of x, y, z is nonzero, so assume without loss of generality that $y = z = 0$. Then $b = c$ holds if and only if $H_9(x) = H_9(p)$, or equivalently $x = p$. Therefore, can win only if, for each clause, one of the three variables is p (representing true) and the other two are 0 (representing false).

To find p, the adversary needs a pair $m_{n+1}, p = \Pi(m_{n+1})$. It can get them by encrypting the all-zeros plaintext, to get a uniformly random r, and $c_1 = \Pi(r)$. It can then use r as the m_{n+1} of its next message, and use $p = c_1$ to encode a satisfying assignment. Notice that if all clauses are satisfied then c_{n+2} will be m_{n+2}, breaking IND$-CPA security. Otherwise, if any clauses is not satisfied then c_{n+2} will be fresh randomness.

For decryptability, given $c_0, \ldots, c_{n+2}$, first use counter-like mode decryption to find $m_1, \ldots, m_{n+1}$. They can then be used to compute all calls to GADGET, and finally to subtract all of their results from c_{n+2} to get m_{n+2}.

■

APPENDIX B
MKG14 DETAILS

A. Design Limitations of MKG14

The treatment of the repeatable subspace in Theorem 10 shows that it is not necessary for security that all oracle query inputs be unique and unpredictable. However, the MKG14 model lacks a global transcript of block cipher queries. There is no equivalent to a Linicrypt program's oracle constraints. All node validity decisions are made locally based only on the labels of incident edges and the node instruction. Consequently, there is no mechanism to identify repeated queries and enforce relationships between their outputs. Due to these design limitations, they have no choice but to require all query inputs to be unique and assert that all query outputs are uniform and independent.

B. MKG14 Labeling Rules

The general strategy they employ to ensure block cipher inputs are unique is to allow each source of randomness (whether it be the input chaining value, a randomly generated value, or a block cipher output) to contribute randomness to at most one block cipher query, and necessitate that every block cipher query receive randomness from at least one source. A source of randomness can be algebraically involved in additional block cipher inputs as long as all of them past the first receive randomness from another source. We will show that this strategy is neither necessary nor sufficient to determine whether block cipher inputs are unique.

An edge *label* is defined as a tuple $(fam, type, flags)$, where

- $fam \subseteq \{1, ..., |E|\}$ is the set of "families" an edge belongs to. Two edges are *related* if their family sets intersect. Edge families are roughly equivalent to Linicrypt base variables. All inputs, generated random values, and block cipher outputs are assigned a unique family value.

- $type \in \{R, \perp\}$ specifies whether an edge is random (R) or adversarially controlled ($\perp$).
- $flags \in \{0,1\}^2$ is a bit vector where the second bit, $flags.PRF$, carries the "ability to contribute randomness" to a block cipher input that was mentioned above. The first bit, $flags.OUT$, indicates the similar capability to contribute randomness to a ciphertext output. Freshly generated random values (from GENRAND or PRF/PRP nodes) always get $flags = 11$. Adversarial inputs always get $flags = 00$.

Edges into PRF/PRP and OUT nodes are required to have $flags.PRF$ set and $flags.OUT$ set, respectively. These edges must also have type R, but this requirement is redundant, because there is no way to have any set $flags$ bits without also having type R.

When a value is duplicated via the DUP instruction, the $flags$ bits must be conserved, meaning the number of set $flags.PRF$ and $flags.OUT$ bits must be the same before and after processing the node. In particular,

$$flags_{out,1} \mid flags_{out,2} = flags_{in}$$
$$flags_{out,1} \,\&\, flags_{out,2} = 00.$$

This rule ensures that value duplication does not also duplicate the ability to contribute randomness.

The two XOR inputs must be unrelated, and at least one must have type R. The output has type R, and its family set and $flags$ vector are the set union and bitwise OR, respectively, of the inputs. The first rule prevents a random value from being XORed into an adversarially controlled value, giving it type R and $flags$ bits, and subsequently being XORed back out again. The second rule ensures that the output has the ability to contribute randomness if and only if this ability was possessed by at least one of two inputs, which are consumed by the instruction.

Formally, the goal of the labeling scheme is to maintain three invariants throughout execution of the encryption scheme. The authors call an edge *active* if it has been assigned a value but its children have not. They define PRF_a to be the active edges with $flags.PRF$ set, OUT_a to be the active edges with $flags.OUT$ set, and OUT to be the edges that have been output as ciphertext for the current plaintext block. They assert that a valid labeling guarantees the following three invariants hold with all but negligible probability at the end of each processing step:

> **Invariant 1:** For any $S \subseteq PRF_a$, the random variables $\mathsf{val}(S)$ are jointly uniform, even conditioned on the values on all active edges unrelated to edges in S and the values of all edges previously used as input to a PRF instruction.
> **Invariant 2:** For any $S \subseteq OUT_a$, the random variables $\mathsf{val}(S)$ are jointly uniform, even conditioned on the values of all previous ciphertext blocks (including those in OUT) and the values on all active edges unrelated to edges in S.
> **Invariant 3:** The random variables $\mathsf{val}(OUT)$ are jointly uniform, even conditioned on the values of all previous ciphertexts.

At the end of encryption, Invariant 3 gives IND$-CPA security. Invariant 2 upholds Invariant 3 by ensuring that the values we output will be uniform. Invariant 1 says that the inputs to PRF/PRP nodes are uniform and thus likely distinct, which ensures that the outputs are uniform and independent, upholding Invariant 2.

C. Counterexample to Soundness

The MKG14 scheme, through the $flags.PRF$ bit, allows each family to contribute its randomness to at most one block cipher query. However, if two families can "exchange" their $flags.PRF$ bits, the scheme allows two identical edges to both be block cipher inputs.

Conceptually, suppose family 1 contributes randomness to PRP 1 and family 2 contributes randomness to PRP 2. PRP 1 has received randomness from another family, so family 2 can be algebraically involved in the input to PRP 1, despite already contributing its randomness to another block cipher input. By symmetric reasoning, family 1 can be algebraically involved in the input to PRP 2. Thus, both PRP inputs are linear combinations of the same two families. Since MKG14 algorithms only allow XOR, this means the two PRP inputs are equal.

Proposition 33. *If, at any time during execution, there are two unrelated active edges, both with $flags.PRF = 1$, Invariant 1 can be broken within a valid algorithm under the MKG14 labeling scheme.*

Proof: Proof by valid subgraph construction, shown in Figure 4. The labels for e_0 and e_1 are valid by hypothesis. The edge triples (e_0, e_2, e_4) and (e_1, e_3, e_5) both satisfy the "conservation of flags" rule for DUP nodes. Both edge pairs (e_2, e_3) and (e_4, e_5) are unrelated and have at least one edge of type R, so the XOR nodes are valid. At the end of executing this subgraph, we have $PRF_a = \{e_6, e_7\}$ and $\mathsf{val}(e_6) = x \oplus y = \mathsf{val}(e_7)$. Therefore, the edges in PRF_a are not jointly uniform, so Invariant 1 is broken.

The x values for $flags.OUT$ on certain edges are "don't cares," indicating that the proposition holds either way. If both e_0 and e_1 have $flags.OUT = 1$, then this subgraph also breaks Invariant 2. However, this is not a critical point, because Invariant 2 can be broken, regardless, by sending both e_6 and e_7 through PRPs. ■

Theorem 33 alone is not enough to break soundness. We must first meet the precondition of two unrelated edges, both with $flags.PRF$ bits set, that are active simultaneously.

Block algorithms are required to be deterministic, so the active edges at the beginning of execution are only the input chaining value and the plaintext block. The plaintext block is adversarially chosen, so $flags = 00$. Then, the only potential source of $flags.PRF$ bits is the input state. The labeling rules ensure that $|PRF_a|$ is conserved by DUP nodes and can only be decreased (if not preserved) by XOR nodes. PRF/PRP nodes can increase the number of active $flags.OUT$ bits set by one, but have no effect on the number of active $flags.PRF$ bits. Therefore, $|PRF_a| \leq 1$ at all times during execution of a Block algorithm under the basic instruction set. However,

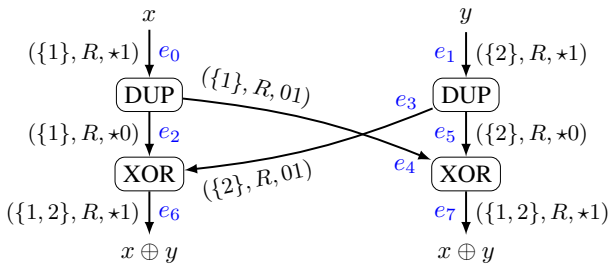


Fig. 4. A valid subgraph that, given two unrelated edges both with $flags.PRF = 1$, breaks Invariant 1. The $\star$ value in $flags.OUT$ on certain edges indicates "don't care" values. If both e_0 and e_1 have $flags.OUT = 1$, then Invariant 2 is also broken.

there is no scarcity of $flags$ bits in Init algorithms, where we can simply add GENRAND nodes. Thus, the precondition for Theorem 33 is achievable in a valid Init graph, so the soundness of the scheme can be broken.

One such Init algorithm, which contains Figure 4 as a subgraph, is shown in Figure 3. It has a valid labeling, but always outputs $c_0 = s_0 = 0$. The edges output by the two GENRAND nodes satisfy the condition for Theorem 33. With Invariant 1 broken, the two equal-valued edges, e_6 and e_7, are sent through PRP nodes to erase their family relationship, which enables a valid XOR node to combine the seemingly unrelated edges together, always producing zero.

When translated to a Linicrypt program, as shown below, there is only a single oracle constraint because the two query inputs coincide. It is obvious that the two PRP outputs are equal and thus cancel out when we XOR them together. When paired with any Block algorithm, the induced encryption scheme outputs zero as the first ciphertext block. Thus, the rows of $C(\mathsf{Enc}_\ell^{\mathsf{Init},\mathsf{Block}})$ are not linearly independent (without even considering the repeatable subspace), so no block cipher mode using this Init algorithm is IND$-CPA secure by Theorem 13.

```
Init(1^n):
r1 ← {0,1}^n
r2 ← {0,1}^n
x := Π(r1 ⊕ r2)
y := Π(r1 ⊕ r2)
c := x ⊕ y
return (c, c)
```

$\rightsquigarrow$

```
Init(1^n):
r1 ← {0,1}^n
r2 ← {0,1}^n
x := r1 ⊕ r2
y := Π(x)
c := y ⊕ y
return (c, c)
```

$\rightsquigarrow$

```
Init(1^n):
return (0^n, 0^n)
```

$$C(\mathsf{Init}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathcal{O}(\mathsf{Init}) = \{\ [1\ 1\ 0] \mapsto [0\ 0\ 1]\ \}$$

When and why did the MKG14 invariants fail? Consider the point in execution of the Init graph when we have duplicated both random values and processed one of the two XOR nodes. The active edges are, without loss of generality:

$$e_4 = (\{1\}, R, 01), \quad e_5 = (\{2\}, R, 10), \quad e_6 = (\{1, 2\}, R, 11)$$

Before processing the second XOR node in our Init graph, $PRF_a = \{e_4, e_6\}$. There is no choice of $S \subseteq PRF_a$ that allows consideration of all three active edges under Invariant 1, because $e_5 \notin PRF_a$ and e_6 is related to all other active edges. Thus, the linear dependence of $\mathsf{val}(e_6) = \mathsf{val}(e_4) \oplus \mathsf{val}(e_5)$ goes under the radar and does not violate Invariant 1. By a similar argument, Invariant 2 also holds.

However, once the XOR node is processed, all components of the dependency are present in $\widehat{PRF_a} = \{e_6, e_7\}$, with $\mathsf{val}(e_6) = \mathsf{val}(e_7)$, so Invariant 1 (and similarly, Invariant 2) is violated.

In their proof, the authors let PRF_a denote the set before the instruction, and $\widehat{PRF_a}$ denote the (possibly modified) set following the instruction. They then give the following argument for why Invariant 1 holds over XOR instructions, and claim that the argument is identical for Invariant 2:

> Let e_0 and e'_0, labeled $(fam_0, type_0, flags_0)$ and $(fam'_0, type'_0, flags'_0)$, respectively, be the two ingoing edges, and let e_1, labeled $(fam_1, type_1, flags_1)$, be the outgoing edge. Note that e_0 and e'_0 must be unrelated and at least one of the two must have type R. Also $fam_1 = fam_0 \cup fam'_0$, so anything related to e_0 or e'_0 is also related to e_1.
>
> If $e_1 \notin \widehat{PRF_a}$, then $e_0, e'_0 \notin PRF_a$. Thus $\widehat{PRF_a} = PRF_a$, so the invariant continues to hold. Otherwise, $e_1 \in \widehat{PRF_a}$ and at least one of e_0 or e'_0 (say e_0) is in PRF_a. By Invariant 1 we have that $\mathsf{val}(e_0)$ is uniform even conditioned on $\mathsf{val}(e'_0)$. Thus, $\mathsf{val}(e_0) \oplus \mathsf{val}(e'_0)$ is uniform and the invariant continues to hold.

The last sentence does not follow. In particular, Invariant 1 only guarantees that subsets of PRF_a are jointly uniform conditioned on ***unrelated*** active edges. Their argument ignores the possibility that there is another edge in PRF_a, not incident to this XOR node, that is related to both e_0 and e'_0.

In fairness to the authors, their program synthesis implementation only considered the trivial Init algorithm (which contains a single GENRAND node) and instead focused on synthesizing secure Block algorithms. This prioritization may have contributed to their oversight, because restricting the Init algorithm to a single GENRAND node restores soundness for the basic instruction set.

That being said, the soundness of the INC-extended labeling scheme, which was included in their synthesis results, can also be broken in the Block algorithm. To achieve their aim of modeling CTR mode, the INC instruction must set $flags.PRF$ on its output. Unfortunately, this allows, for instance, both r and $r+1$ to be sent through a PRF, producing two unrelated edges with $flags.PRF = 1$ that are active simultaneously, which satisfies the condition for Theorem 33. In the full version of this paper we present an INC-extended Block algorithm with a valid labeling that always outputs the plaintext as ciphertext. It should be mentioned that this Block algorithm contains more than the 10 node limit used in program synthesis, so the synthesis results published in [7] may not be affected by insecure modes of this nature.