Automated Analysis and Synthesis of Block-Cipher Modes of Operation*

Alex J. Malozemoff† Jonathan Katz† Matthew D. Green‡

Abstract

Block ciphers such as AES are deterministic, keyed functions that operate on small, fixed-size blocks. Block-cipher *modes of operation* define a mechanism for probabilistic encryption of arbitrary length messages using any underlying block cipher. A mode of operation can be proven secure (say, against chosen-plaintext attacks) based on the assumption that the underlying block cipher is a pseudorandom function. Such proofs are complex and error-prone, however, and must be done from scratch whenever a new mode of operation is developed.

We propose an *automated* approach for the security analysis of block-cipher modes of operation based on a "local" analysis of the steps carried out by the mode when handling a *single* message block. We model these steps as a directed, acyclic graph, with nodes corresponding to instructions and edges corresponding to intermediate values. We then introduce a set of *labels* and *constraints* on the edges, and prove a meta-theorem showing that any mode for which there exists a labeling of the edges satisfying these constraints is secure (against chosen-plaintext attacks). This allows us to reduce security of a given mode to a constraint-satisfaction problem, which in turn can be handled using an SMT solver. We couple our security-analysis tool with a routine that automatically generates viable modes; together, these allow us to synthesize hundreds of secure modes.

1 Introduction

Designing and proving security of cryptographic constructions can be difficult, time-consuming, and error-prone. Developing a protocol for some task meeting various requirements involves creativity and hard work; even when only small changes to an existing construction are involved, each change typically requires a cryptographic proof of security to be redone from scratch.

To simplify, streamline, and speed up this process, it would be hugely beneficial to *automate* (parts of) this design process. Being able to automate the analysis of cryptographic constructions would enable rapid examination of proposed candidates with less chance of error; coupling this with a systematic way of generating candidates that satisfy a given set of functional requirements would enable automated synthesis of secure constructions. Here we can draw inspiration from the field of *program synthesis* [MW80], which has recently witnessed several successes (see Srivastava's Ph.D. thesis [Sri10] for a recent survey).

The past few years have seen the first steps toward automated analysis and synthesis of cryptographic constructions; we refer the reader to Section 1.1 for a review of prior work. In this work we focus on automated analysis and synthesis of *block-cipher modes of operation*. A *block cipher* is a deterministic, keyed function F; with a key k fixed, the function $F_k : \{0,1\}^n \to \{0,1\}^n$ operates on small, n-bit blocks. (A prominent example is the standardized block cipher AES, which has a 128-bit block size.) Secure[1] encryption using a block cipher requires choosing some *mode of operation* to handle both probabilistic encryption (so that the same message, encrypted twice, does not yield the same ciphertext) as well as messages of arbitrary

*The proceedings version of this work appears in the *27th IEEE Computer Security Foundations Symposium, Vienna Austria, July 19–22, 2014*. This is the full version.

†Department of Computer Science, University of Maryland. **Email:** {amaloz,jkatz}@cs.umd.edu

‡Department of Computer Science, Johns Hopkins University. **Email:** mgreen@cs.jhu.edu

[1]Here we mean the default notion of CPA-security, a.k.a., security against chosen-plaintext attacks.

length. Several secure modes of operation were specified as part of the DES standard in the 1970s, and modes such as CTR and CBC are in widespread use today. More recently, various modes of operation with stronger security properties, or based on newer primitives such as tweakable block ciphers, have been proposed; see http://csrc.nist.gov/groups/ST/toolkit/BCM for some examples.

The core idea of our approach to the automated analysis of block-cipher modes of operation is to focus on the operations carried out by a given mode when handling a *single* message block. We model these steps as a directed, acyclic graph in which nodes correspond to atomic operations (such as XORing two values together, evaluating a pseudorandom function on some value, etc.) and edges correspond to intermediate values. We introduce a set of labels for the edges in such graphs, along with a set of constraints on how edges can be labeled. Our central result is a meta-theorem stating that if a given graph, corresponding to some mode, can be labeled while satisfying the constraints, then that mode is secure. We thus reduce the security of any mode of operation to a constraint-satisfaction problem, which in turn can be addressed by an SMT solver.

Our meta-theorem shows that our constraints are *sound*, in that any mode classified as secure is, indeed, secure; it is not *complete*, and so may falsely reject a secure mode. Nevertheless, we show that our constraints are permissive enough to capture all commonly used (secure) modes we are aware of, including CBC, CTR, OFB, CFB, and PCBC.

Based on the above meta-theorem, we implemented a model checker designed to evaluate candidate modes of operation. Our model checker takes as input a mode specified as a graph, and determines both whether the mode is *correct* (i.e., whether there exists a corresponding decryption algorithm that recovers the message from the ciphertext) and whether it is *secure*. We then use our model checker to synthesize secure modes of operation in the natural way. That is, we implement a program synthesizer that automatically generates candidate modes of operation for analysis, and then filters out those modes that are not identified as being secure by our model checker. Doing this in a naïve manner would be prohibitively slow due to the combinatorial explosion in the number of candidate modes. To deal with this, we use several pruning strategies that allow us to eliminate more than 99.99% of potential candidates from consideration before invoking our model checker. Using our approach we are able to synthesize hundreds of secure modes of operation. Our implementation could easily be integrated with a simple tool assigning scores to different modes (e.g., expressing a preference for minimizing calls to the block cipher or for avoiding integer addition) to synthesize secure modes with desired properties.

Although our approach currently only addresses security against chosen-plaintext attacks, we are hopeful that our ideas can be extended to handle stronger security definitions such as authenticated encryption. More generally, we believe our techniques could find application to the automated design and verification of other cryptographic primitives and protocols.

1.1 Prior Work

Several works have recently considered automated generation and/or analysis of cryptographic algorithms. Akinyele et al. [AGHP12, AGH13] developed tools for automatically generating batch-verification algorithms for digital-signature schemes [AGHP12], for converting schemes using symmetric bilinear groups into ones using asymmetric bilinear groups [AGH13], and for compiling signature schemes to schemes achieving stronger security properties [AGH13]. In all these cases, their tools do not directly analyze or verify security of new constructions; instead, they take as input schemes that are assumed to be secure, and then attempt to apply a fixed set of transformations that have been proven to preserve (or amplify) security. Closer to our results are those of Barthe et al. [BCG+13], who propose a technique for the automated analysis of *public*-key encryption schemes based on trapdoor permutations and a small set of other instructions. They also use their analysis tool to synthesize secure schemes. However, the proof techniques and analysis methods are very different in the public-key setting as compared to the symmetric-key setting we consider here.

The work most similar to our own is that of Gagné et al. [GLLSN09, GLLSN12]. Building on earlier work [CDE+08], they propose using a compositional Hoare logic to analyze block-cipher modes of operation. They use an imperative language in which each operation updates a distribution over states of the program, and then use their Hoare logic to reason about the output distribution of modes expressed in this language.

Roughly, they show that if, after the execution of a mode, all variables are marked as indistinguishable from uniform, then the mode is secure. The required property can be proven for a given mode by applying their Hoare-logic rules to the entire sequence of operations used by the mode when encrypting some message. A drawback of their approach is that it can only reason about the encryption of messages of some pre-specified, fixed length, and their proofs provide no "inductive" guarantee about what happens when the mode is used to encrypt messages of other (arbitrary) lengths. Thus, to use their approach one would have to provide a separate proof for all possible message lengths one would expect to encounter in practice. Moreover, our "local" approach is fundamentally different from their "global" perspective. In contrast to Gagné et al., we analyze the operations carried out by a given mode when processing a *single* message block; our meta-theorem guarantees that if that (finite) set of operations has a valid labeling, then it corresponds to a mode of operation that is secure for encrypting messages of arbitrary (polynomial) length. Besides this conceptual difference, our approach also ensures that analysis of a given mode (for messages of unbounded length) can be reduced to the satisfiability of a *finite* set of constraints. This, in turn, can be easily solved (in practice) using an SMT solver. Our approach is thus particularly suited for the synthesis of *new* modes of operation.

In a similar vein, Courant, Ene, and Lakhnech [CEL07] use a *type system* (rather than a compositional Hoare logic) to analyze various symmetric-key encryption schemes, including modes of operation. However, this approach suffers some of the same drawbacks as the above work, in that state regarding specific values needs to be maintained across blocks, and thus it does not appear that this approach can be generalized to handle encrypting arbitrary length messages. In addition, their type system does not support the ability to increment a value (and thus cannot prove CTR mode secure), and the authors do not use their type system to synthesize modes, as is done in this work.

1.2 Outline

We introduce some background material in Section 2, and in Section 3 we discuss our approach to modeling block-cipher modes of operation. Section 4 contains our central meta-theorem and proof. Section 5 describes our implementation of the model checker and a program synthesizer that together can be used to automatically generate secure modes of encryption; the results of our experiments using those tools are described in Section 6. We conclude in Section 7.

2 Preliminaries

We use the standard definitions of private-key encryption schemes and block ciphers (i.e., pseudorandom permutations) [KL08]. We let n denote the block length of a block cipher F; thus, for a fixed key k, the function F_k defines a bijection on n-bit strings. Formally we treat n as a security parameter, and so "polynomial" (resp., "negligible") means "polynomial in n" (resp., "negligible in n"). A message/ciphertext block is just an n-bit string.

We utilize a notion of security for private-key encryption called IND$-CPA [Rog04], which is stronger than indistinguishability against chosen-plaintext attacks in that it requires ciphertexts to be indistinguishable from uniform strings.

Definition 1. Let $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme in which the encryption of an ℓ-block message yields an $(\ell+1)$-block ciphertext, and let $\$(\cdot)$ be an oracle that, when queried on input $m \in \{0,1\}^{\ell \cdot n}$, returns a uniform string of length $(\ell+1) \cdot n$. We say Π is IND$-CPA secure if the following is negligible for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\left| \Pr_{k \leftarrow \{0,1\}^n} \left[\mathcal{A}^{\mathsf{Enc}_k(\cdot)}(1^n) = 1 \right] - \Pr \left[\mathcal{A}^{\$(\cdot)}(1^n) = 1 \right] \right|.$$

Since the issue will arise in our analysis, we remark that the decryption algorithm Dec is irrelevant as far as security is concerned (although it is, of course, critical for correctness).

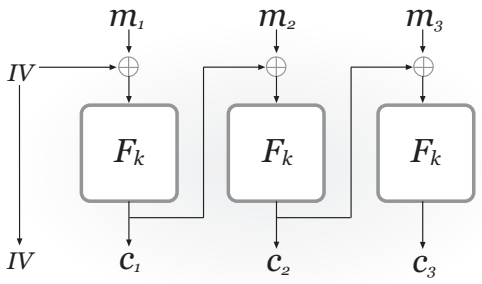

Figure 1: CBC mode.

3 Modeling Modes of Operation

A mode of operation refers to a mechanism for encrypting arbitrary length messages based on any underlying block cipher. For simplicity, we assume all messages being encrypted have length a multiple of the block length. (This is without loss of generality, as messages can be padded unambiguously to such a length using known techniques.) For our purposes in this and the next section, a mode is defined by a pair of efficient algorithms Init and Block that each expect oracle access to a function $F_k : \{0,1\}^n \to \{0,1\}^n$. These algorithms have the following functionality:

- Init: This is a probabilistic algorithm that, on input 1^n, outputs an initial ciphertext block $c_0 \in \{0,1\}^n$ and state information $z_0 \in \{0,1\}^n$.
- Block: This is a deterministic algorithm that takes as input a message block $m \in \{0,1\}^n$ and state information $z \in \{0,1\}^n$, and outputs a ciphertext block $c \in \{0,1\}^n$ and updated state information $z' \in \{0,1\}^n$.

(The above could be generalized so that, e.g., Block outputs two ciphertext blocks for each message block processed, or both Init and Block output two-block state. While this would require modifications to our constraints, our approach would be able to handle such generalizations.)

With some block cipher F fixed, a mode of operation (Init, Block) induces an encryption algorithm Enc. Specifically, if $\mathbf{m} = m_1 \| \cdots \| m_\ell$ is an ℓ-block message, then $\mathsf{Enc}_k(\mathbf{m})$ does the following:

1. Compute $(c_0, z_0) \leftarrow \mathsf{Init}(1^n)$, where Init is given oracle access to F_k.
2. For $i = 1, \ldots, \ell$, set $(c_i, z_i) := \mathsf{Block}(m_i, z_{i-1})$, where Block is given oracle access to F_k.
3. Output the ciphertext $c_0 \| c_1 \| \cdots \| c_\ell$.

As an example, consider the CBC mode of operation in Figure 1. In CBC mode, the Init algorithm simply generates a uniform initialization vector (IV); the IV is both output as the initial ciphertext block and also passed along as state information. The Block algorithm XORs the incoming state information z with the current message block m, passes the result through a block cipher F_k, and uses the resulting value $F_k(m \oplus z)$ as both the next ciphertext block and the updated state.

In this paper, a mode (Init, Block) is *secure* if the induced encryption scheme is IND$-CPA secure whenever F is a secure block cipher. (Here, we do not concern ourselves with decryption since we are only interested in security. For a given mode to be meaningful, it will also need to possess a corresponding decryption algorithm. When we synthesize modes, we check for existence of a suitable decryption algorithm in addition to verifying security.)

3.1 Viewing Modes as Graphs

The Init and Block algorithms constituting a mode of encryption can be viewed as directed, acyclic graphs, where the nodes correspond to instructions (such as $\oplus$ or application of F_k), and the edges correspond to

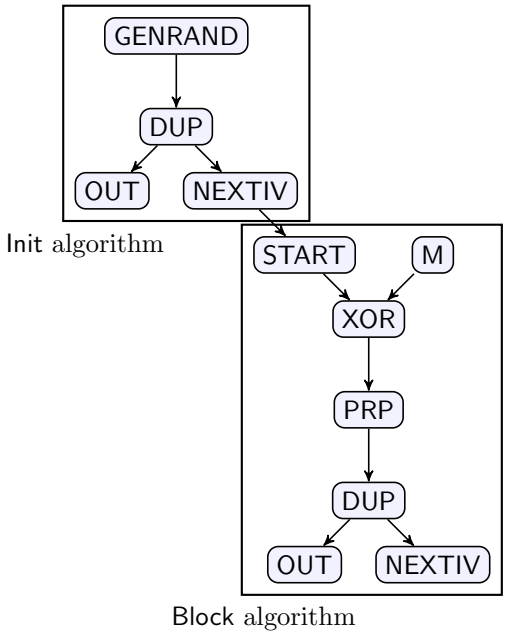

Figure 2: CBC mode expressed as a graph.

(intermediate) values, which are all n-bit strings. A node applies its associated operation on the value(s) contained on its ingoing edge(s), with the result(s) of the operation assigned to its outgoing edge(s). As an example, consider the graph version of CBC mode in Figure 2. The Init portion of the graph corresponds to the generation of a random IV (the GENRAND node) which is then duplicated (the DUP node) and output as part of the ciphertext (through the OUT node) as well as passed on to the Block algorithm (through the NEXTIV node). The Block portion of the graph contains a START node, which is how the state information is provided as input (this information comes from the NEXTIV node of either the Init algorithm or a previous invocation of Block), as well as an M node, which is how the current message block is provided as input. In the case of CBC mode, those input values are XORed and the result is then passed through F_k. That resulting value is duplicated, and again one copy is output as the next block of the ciphertext (via OUT) and another copy is passed on as state (via NEXTIV).

From now on, we view a mode of encryption as being defined by a pair of directed, acyclic graphs, corresponding to the Init and Block algorithms, whose nodes are labeled with instructions from the list below and which also satisfy certain constraints described next. The instructions we support include:

- DUP: Has in-degree one and out-degree two. Replicates the ingoing value on both its outgoing edges.
- GENRAND: Has in-degree zero and out-degree one. Generates a uniform value in $\{0,1\}^n$.
- M: Has in-degree zero and out-degree one. This corresponds to the current message block being provided as input. This instruction appears exactly once in the Block graph, and does not appear in the Init graph.
- PRF and PRP: Have in-degree one and out-degree one. These instructions both apply the block cipher to the value on the ingoing edge and assign the result to the outgoing edge; PRP is invertible but PRF need not be.

  With regard to security, we rely on the PRF/PRP switching lemma and treat both PRF and PRP instructions as pseudorandom functions (rather than permutations). For this reason, in the rest of this and the next section we only speak of a PRF instruction. We will later treat PRF and PRP differently with regard to *correctness*.

- XOR: Has in-degree two and out-degree one. XORs the values on its two ingoing edges and assigns the result to its outgoing edge.
- NEXTIV: Has in-degree one and out-degree zero. (The outgoing edge from NEXTIV in Figure 2 is conceptual.) The value on the ingoing edge is treated as the state that is output by Init or Block. This instruction appears exactly once in each of the Init and Block graphs.
- START: Has in-degree zero and out-degree one. (The ingoing edge to START in Figure 2 is conceptual.) The value on the outgoing edge is the state information provided as input to Block. This instruction appears exactly once in the Block graph, and does not appear in the Init graph.
- OUT: Has in-degree one and out-degree zero. The value on its ingoing edge represents the next ciphertext block that is output. This instruction appears exactly once in each of the Init and Block graphs.

The above provide a "basic" instruction set that we consider in the main body of this paper. In the appendix we additionally consider the following instruction:

- INC: Has in-degree one and out-degree one. Increments the value on its ingoing edge (modulo 2^n), and assigns the result to its outgoing edge.

The above instructions (including INC) are sufficient to model all the standard modes such as ECB, OFB, CFB, CBC, CTR, and PCBC.

3.2 Edge Labels and Constraints

In the previous section we defined what it means for a pair of directed, acyclic graphs to correspond to a *legal* mode. But not all legal modes are secure! In this section we introduce a set of edge labels, and also describe a set of constraints that a valid labeling must satisfy. Our meta-theorem states that if a legal mode of operation has a valid edge labeling, then that mode is secure.

We now give the details of the edge labeling for some (legal) mode of operation specified by directed, acyclic graphs $(\mathsf{Init}, \mathsf{Block})$. Let G denote the union of Init and Block, and let E denote the total number of edges in G. A *label* is a 3-tuple $(\mathsf{fam}, \mathsf{type}, \mathsf{flags})$ where

- $\mathsf{fam} \subseteq \{1, \ldots, E\}$ represents the set of *families* to which the edge belongs. Two edges e_1, e_2 with families $\mathsf{fam}_1, \mathsf{fam}_2$, respectively, are *related* if $\mathsf{fam}_1 \cap \mathsf{fam}_2 \neq \emptyset$.
- $\mathsf{type} \in \{\mathsf{R}, \perp\}$ represents a "type," where, intuitively, R denotes "random" and $\perp$ denotes "adversarially controlled." We impose the ordering $\perp < \mathsf{R}$.
- $\mathsf{flags} \in \{0,1\}^2$ is a bit-vector denoting whether the edge can be used as input to OUT or PRF, respectively. We index into this bit-vector by using the notation $\mathsf{flags}.OUT$ and $\mathsf{flags}.PRF$. We define the (partial) ordering $\mathsf{flags} \leq \mathsf{flags}'$ iff $\mathsf{flags}.OUT \leq \mathsf{flags}'.OUT$ and $\mathsf{flags}.PRF \leq \mathsf{flags}'.PRF$.

In trying to label G, we begin by assigning each edge in G a set of families using the following, deterministic procedure that first handles edges in Init before moving on to the edges in Block:

1. While there is an instruction node whose ingoing edges have all been assigned a set of families, assign its outgoing edge(s) a set of families using the following rules:
   - DUP: The outgoing edges are both assigned the same set of families as the ingoing edge.
   - PRF: Let int be the least unused value in $\{1, \ldots, E\}$. The outgoing edge is assigned the set of families $\{\mathsf{int}\}$.
   - XOR: Let the ingoing edges have sets of families fam and fam'. The outgoing edge is assigned the set of families $\mathsf{fam} \cup \mathsf{fam}'$.

2. Otherwise, we choose an instruction node with in-degree 0, with preference for the START node if its outgoing edge has not yet been labeled. The outgoing edge from this node is assigned $\{\mathsf{int}\}$, where int is the least unused value in $\{1, \ldots, E\}$.

Once each edge in G has been assigned a set of families, we then try to assign a type and flags to each edge in G subject to the following constraints:

- START: Say there is a NEXTIV node in G (whether in Init or in Block) whose ingoing edge has type type and flags flags. Then the outgoing edge from START has $\mathsf{type}' \leq \mathsf{type}$ and $\mathsf{flags}' \leq \mathsf{flags}$.
- GENRAND: The outgoing edge gets type R and $\mathsf{flags} = 11$.
- M: The outgoing edge gets type $\perp$ and $\mathsf{flags} = 00$.
- DUP: Say the input edge has type type and flags flags. The two output edges get types $\mathsf{type}_1 = \mathsf{type}_2 = \mathsf{type}$, and flags flags_1 and flags_2 such that the following hold: (1) $\mathsf{flags}_1 \,\&\, \mathsf{flags}_2 = 00$ and (2) $\mathsf{flags}_1 \,|\, \mathsf{flags}_2 = \mathsf{flags}$, where & and | refer to bit-wise AND and OR, respectively.
- PRF: We require that the ingoing edge to PRF has type R and $\mathsf{flags}.PRF = 1$. The outgoing edge gets type R and $\mathsf{flags} = 11$.
- XOR: We require that the ingoing edges to an XOR node are unrelated, and that at least one of them has type R. The outgoing edge gets type R. Say the input edges have flags flags_1 and flags_2, respectively. Then:
  1. If both ingoing edges have type R, the outgoing edge gets flags $\mathsf{flags}_1 \,|\, \mathsf{flags}_2$.
  2. If only one edge (say, the first) has type R, the outgoing edge gets flags flags_1.
- OUT: We require the ingoing edge to an OUT node to have type R and $\mathsf{flags}.OUT = 1$.

A labeling of G that satisfies all of the above constraints is called a *valid labeling*. One can verify that the graphs corresponding to each of CBC, CFB, and OFB modes have valid labelings,[2] whereas ECB mode (which is known to be insecure) does not. See Figure 3 for a valid labeling of CBC mode.

We now briefly describe some intuition behind these labels and constraints. We define two types for the edge labels: "random" (a.k.a., R) and "adversarially controlled" (a.k.a., $\perp$). For instructions like PRF and XOR, we want a guarantee that the output from these instructions is random. Thus, we do not, for example, allow adversarially controlled values as input into a PRF node, as the output is not "random" to an attacker who can query the PRF node on the same input multiple times.

There are only two instructions that introduce values: GENRAND produces a random value, and M produces an adversarially controlled value (as the adversary supplies the message to be encrypted). For random values, we want to enforce that they are used in ways that do not void their "randomness." This is where the flags bit-vector comes into play: by accounting for whether a random value can be input into an OUT node or a PRF node, we prevent a random value from both being output as ciphertext *and* input into a PRF node, which can lead to insecure schemes. (This does not mean there do not exist secure schemes which have this property; however, our tool does not allow such schemes.)

In the next section we prove that *any* mode that has a valid labeling is secure. Importantly, determining the existence of a valid labeling for a given mode requires reasoning over a finite graph G and a finite set of constraints, and we can therefore use an SMT solver to check whether a valid labeling is possible.

[2] If we include the INC instruction (and add constraints for this instruction as discussed in the appendix), then CTR mode has a valid labeling also.

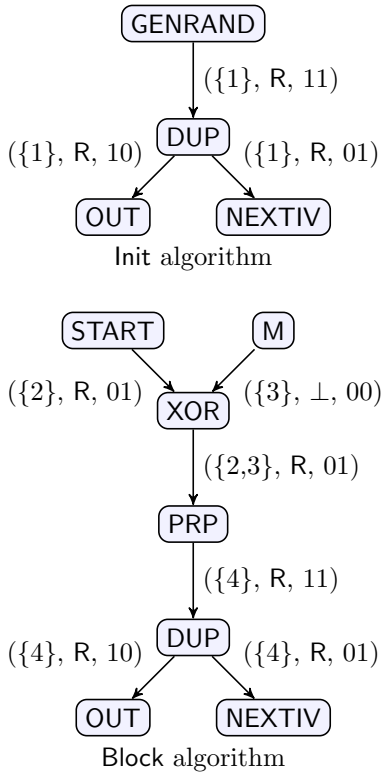

Figure 3: A valid labeling for CBC mode.

4 Meta-Theorem and Proof

Theorem 1. *Let* (Init, Block) *be a legal mode of operation. If it has a valid labeling, then it is secure. In other words, when the mode is instantiated with any secure block cipher, the resulting encryption scheme is IND$-CPA secure.*

Proof. Consider the encryption scheme that results from instantiating Init/Block with a secure block cipher F. The experiment in which a polynomial-time adversary interacts with this scheme proceeds as follows:

- A uniform key for the block cipher is chosen.
- The attacker may then (adaptively) specify messages to be encrypted. After an ℓ-block message $\mathbf{m} = m_1 \| \cdots \| m_\ell$ is specified, we imagine creating a connected graph G consisting of one copy of Init and ℓ copies of Block by simply adding an edge from a NEXTIV node of Init/Block to the START node of the subsequent copy of Block. To encrypt the message, we begin by assigning the ith message block m_i to be the value on the outgoing edge of the M-instruction in the ith copy of Block. We then iteratively assign values to the edges in G (based on the instruction at each node) until every edge is assigned a value. (The outgoing edge from a NEXTIV node or a START node is assigned the same value as the ingoing edge to that node.) This, in particular, assigns values to the ingoing edges to every OUT node, and so defines an $(\ell + 1)$-block ciphertext that is given to the attacker.

Using a standard hybrid argument, we may replace the block cipher (and choice of uniform key) with a function chosen uniformly from the space of all functions from n-bit inputs to n-bit outputs. This means that PRF instructions in the above process can now be handled as follows: If the value on the ingoing edge to a PRF node was previously input to a PRF node (whether in the course of encrypting the current message, or during encryption of some prior message), then assign to the outgoing edge the same output

value used previously. Otherwise, assign to the outgoing edge a uniform value in $\{0,1\}^n$. Once we make this substitution, the entire experiment becomes information-theoretic and we no longer need to be concerned with the running time of the attacker. (However, we continue to assume a polynomial upper bound on the number of messages the attacker can request to be encrypted as well as the block-length of each such message.) As the adversary is computationally unbounded we may, without loss of generality, treat the attacker as deterministic. The probability space of the attacker's interaction is then taken only over choice of the random function and any internal randomness of the mode (i.e., in GENRAND instructions). For each edge in all the graphs G (corresponding to all the messages whose encryption is requested by the attacker), the value assigned to that edge is a random variable. Let $\mathsf{val}(e)$ be the random variable corresponding to the value assigned to some edge e. If E is a set of edges, then $\mathsf{val}(E)$ denotes the multiset $\{\mathsf{val}(e) \mid e \in E\}$. We show that the joint distribution of the random variables corresponding to all the ingoing edges to OUT nodes—and hence the joint distribution of all ciphertext blocks returned to the attacker—is statistically close to uniform. This implies the scheme is IND$-CPA secure.

Fix some valid labeling of Init and Block. As discussed above, each time the attacker requests an encryption of some ℓ-block message we imagine creating a graph G consisting of one copy of Init and ℓ copies of Block. The valid labeling of Init and Block naturally extends to a labeling of this graph G; the edges in Init are labeled in the obvious way, and the corresponding edges in each copy of Block are assigned the same label. (The only special case is an edge from a NEXTIV node in one block to a START node in the next block, which we assign the same label as the ingoing edge to that NEXTIV node.)

Consider the step-by-step process by which edges in G are assigned values when a message $\mathbf{m}$ is encrypted. (Although we speak here of "assigning values," we will continue to treat these as random variables and so "assigning a value" is merely conceptual.) Edges are assigned values in topological order; i.e., the value of an edge is not assigned unless values have been assigned to its parents. We say an edge is *active* if it has been assigned a value but its children have not. (When we begin encrypting a message, the only active edges are the outgoing edges from all the M-nodes.) The only exceptions are ingoing edges to OUT nodes as well as the ingoing edge to the final NEXTIV node; these become active when they are assigned a value, and remain active until the relevant instruction is processed. (Processing an OUT instruction just means declaring the ingoing edge to that instruction as no longer being active, but instead being "output" as part of the ciphertext. Processing the final NEXTIV node means giving the computed ciphertext to the adversary.) For simplicity, we assume all edges in a given copy of Block are assigned a value before assigning a value to the outgoing edge of the NEXTIV node in that copy of Block.

When we are done encrypting a message we give the resulting ciphertext to the attacker, who then chooses the next message to be encrypted. We then create another graph and assign values as above. We stress that previous graphs are maintained: even though none of their edges are active, we may still refer to the values taken by edges in previous graphs.

At any step in the above process, define the following sets of edges:

$$\begin{aligned} PRF_a &= \text{edges of type R with } \mathsf{flags}.PRF = 1 \text{ that are active} \\ OUT_a &= \text{edges of type R with } \mathsf{flags}.OUT = 1 \text{ that are active} \\ OUT &= \text{edges output for the message currently being processed} \end{aligned}$$

These sets change as instructions are processed. We prove that with all but negligible probability the following invariants hold at the end of each 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.

Invariant 3 implies that when encryption of the given message is done, the resulting ciphertext is (statistically close to) uniform, which is what we wanted to show. The other two invariants are used as part of proving that Invariant 3 is maintained. (Intuitively, Invariant 1 enforces that the inputs into every PRF instruction are uniform, and thus distinct with high probability; this, in turn, means that all the outputs of the PRF instructions are uniform and independent. Invariant 2 enforces that values on ingoing edges to OUT instructions are uniform, and thus when we "output" these values the resulting ciphertext is also uniform.)

Clearly the above invariants all hold when beginning to encrypt some message. (At that point in time, PRF_a, OUT_a, and OUT are all empty.) Now, assume the invariants hold before executing some instruction; we show that, with high probability, they continue to hold after executing the instruction. We let PRF_a, OUT_a, and OUT denote the sets in question before the instruction, and let $\widehat{PRF_a}$, $\widehat{OUT_a}$, and $\widehat{OUT}$ denote the (possibly modified) sets following the instruction. (Observe that OUT only changes when processing an OUT instruction. Thus, Invariant 3 trivially holds for all other instructions, and we only consider it when we analyze the OUT instruction.)

- NEXTIV: The ingoing and outgoing edges have the same label and the same value; thus, the invariants trivially continue to hold.

- START: We consider Invariant 1; the argument is identical for Invariant 2.

  Let e, labeled $(\mathsf{fam}, \mathsf{type}, \mathsf{flags})$, be the ingoing edge to this instruction and let e', labeled $(\mathsf{fam}', \mathsf{type}', \mathsf{flags}')$, be the outgoing edge. The labeling constraints imply that $\mathsf{type}' \leq \mathsf{type}$ and $\mathsf{flags}' \leq \mathsf{flags}$. Note further that the only other active edges at the time this instruction is applied are outgoing edges from M nodes, none of which is related to either e or e', and none of which is in PRF_a or $\widehat{PRF_a}$. If $e' \notin \widehat{PRF_a}$ then Invariant 1 trivially holds after the instruction is applied. Otherwise, the fact that $\mathsf{type}' \leq \mathsf{type}$ and $\mathsf{flags}' \leq \mathsf{flags}$ means that $e \in PRF_a$. Since e and e' have the same value and Invariant 1 holds before the instruction is applied, the invariant continues to hold after the instruction is applied.

- DUP: We consider Invariant 1; the argument is identical for Invariant 2.

  Let e_0 be the ingoing edge to this instruction, and let e_1 and e_2 be the two outgoing edges. If $e_0 \notin PRF_a$, then $e_1, e_2 \notin \widehat{PRF_a}$; this means that $PRF_a = \widehat{PRF_a}$ and—since e_0, e_1, and e_2 all have the same families—the invariant continues to hold. If $e_0 \in PRF_a$, then exactly one of e_1 or e_2 is in $\widehat{PRF_a}$. Since the values of e_0, e_1, and e_2 are the same, and all these edges have the same families, the invariant again continues to hold.

- GENRAND: Because of the way outgoing edges of a GENRAND instruction are labeled, this instruction adds an edge to PRF_a and OUT_a. Since the value on this edge is uniform and independent of any previous values, the invariants continue to hold.

- PRF: Using Invariant 1, the value on the ingoing edge to the PRF instruction is uniform, even conditioned on the values of all previous ingoing edges to a PRF instruction. Therefore, with all but negligible probability, the value on this edge is distinct from all values previously used as input to a PRF instruction. When this is the case, the value on the outgoing edge is uniform and both invariants continue to hold just as for a GENRAND instruction.

- XOR: We consider Invariant 1; the argument is identical for Invariant 2.

  Let e_0 and e_0', labeled $(\mathsf{fam}_0, \mathsf{type}_0, \mathsf{flags}_0)$ and $(\mathsf{fam}_0', \mathsf{type}_0', \mathsf{flags}_0')$, respectively, be the two ingoing edges, and let e_1, labeled $(\mathsf{fam}_1, \mathsf{type}_1, \mathsf{flags}_1)$, be the outgoing edge. Note that e_0 and e_0' must be unrelated and at least one of e_0 and e_0' must have type R. Also, $\mathsf{fam}_1 = \mathsf{fam}_0 \cup \mathsf{fam}_0'$ and 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 $\widehat{PRF_a} = PRF_a$ and 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, $\mathsf{val}(e_1) = \mathsf{val}(e_0) \oplus \mathsf{val}(e_0')$ is uniform and the invariant continues to hold.

- OUT: Let e be the ingoing edge. This instruction reduces the number of active edges, so clearly Invariant 1 and Invariant 2 continue to hold.

  The labeling constraints ensure that $e \in OUT_a$. Also, note that $\widehat{OUT} = OUT \cup \{e\}$. Using Invariant 2, we have that $\mathsf{val}(e)$ is uniform even conditioned on $\mathsf{val}(OUT)$, and thus Invariant 3 continues to hold.

This completes the proof. □

It is possible to extract from our proof a bound on the *concrete* security of the mode of operation as a function of the number (and type) of instructions in Init/Block as well as the total number of message blocks encrypted.

5 Implementation

We have developed a prototype model checker for verifying whether a given mode has a valid labeling, as well as a synthesizer for generating modes. The code is written in OCaml, and is freely available at https://github.com/amaloz/modes-generator. The implementation includes use of the INC instruction, with the modified constraints as described in the appendix.

5.1 Model Checker

The core of our system is a model checker designed to evaluate a candidate encryption mode. Our model checker takes as input a description of the mode of operation (defined in terms of its Init and Block routines) and determines both whether the corresponding encryption algorithm is *correct* (i.e., whether there exists an efficient decryption algorithm) and whether it is *secure* (i.e., whether the mode is IND$-CPA-secure).

5.1.1 Correctness

We have developed an algorithm that takes as input a mode of operation, and determines whether decryption of that mode is possible (with knowledge of the key). This is useful for pruning out uninteresting modes. The algorithm works by examining the description of Block to see whether each message block can be recovered from the ciphertext. To do this, we proceed as follows:

- If the value on the outgoing edge of the M node in Block (i.e., the current message block) can be recovered from the ingoing edge to the OUT node (i.e., the current ciphertext block), the mode is decryptable. (This is not typical, but is true for, e.g., ECB mode.)
- Otherwise, we check Block to see whether the requisite message block can be recovered from both the current ciphertext block and the value on the outgoing edge of the START node (i.e., the state passed from one invocation of Init/Block to the next). If not, we reject the mode as being undecryptable.

  If so, then we need to verify that the state can be recovered from the previous ciphertext block(s). So:

  - We check the Init algorithm to see whether the value on the ingoing edge to the NEXTIV node can be recovered from the value on the ingoing edge to the OUT node. If not, we reject the mode as being undecryptable.
  - We check the Block algorithm to see whether the value on the ingoing edge to the NEXTIV node can be recovered from the value on the ingoing edge to the OUT node and the value on the outgoing edge of the START node.

```
1  (declare-const dup_l_type Int)
2  (declare-const dup_l_flag_out Bool)
3  (declare-const dup_l_flag_prf Bool)
4  (declare-const dup_r_type Int)
5  (declare-const dup_r_flag_out Bool)
6  (declare-const dup_r_flag_prf Bool)
7  (assert (= dup_l_type dup_r_type genrand_type))
8  (assert (= (and dup_l_flag_out dup_r_flag_out) false))
9  (assert (= (and dup_l_flag_prf dup_r_flag_prf) false))
10 (assert (= (or dup_l_flag_out dup_r_flag_out) genrand_flag_out))
11 (assert (= (or dup_l_flag_prf dup_r_flag_prf) genrand_flag_prf))
```

Figure 4: Example Z3 encoding of a DUP instruction.

Our algorithm is sound, and successfully identifies the standard modes as being decryptable.

To perform each of the above checks, we recursively progress through the graphs defining the Init/Block algorithms in the natural way. We begin, say, with the value on the ingoing edge to the OUT node known. If we know the value on any edge of a DUP node, then we can derive the value on all other edges incident on that node. If we know the values on any *two* edges incident on an XOR node, we can derive the value on the remaining edge. In this way we determine whether it is possible to derive the value on, say, the outgoing edge of the M node.

As an example, consider again the graph representation of CBC mode depicted in Figure 2. From the current ciphertext block (i.e., the value on the ingoing edge of the OUT node), we can recover the value on the ingoing edge to the DUP node and then, using the fact that the PRP node is invertible, derive the value on the outgoing edge of the XOR node. This is not enough, by itself, to recover the message block; however, if we are additionally given the state (i.e., the value on the outgoing edge of the START node), then we can recover the message block. We then need to verify that, in both Init and Block, the state (i.e., the ingoing edge to the NEXTIV node) can be derived from the ciphertext block.

5.1.2 Security

Theorem 1 states that if a mode has a valid labeling, it is a secure mode of operation. Thus, determining if a mode is secure reduces to a constraint-satisfaction problem in which the goal is to assign a valid label to all the edges in the Init/Block graphs while satisfying the constraints in Section 3.2. Viewing the problem in this way lets us use an SMT solver to derive a correct satisfying assignment if one exists. In our implementation, we use the Z3 Theorem Prover[3].

We check for a valid labeling in several phases. Because assigning families to edges is deterministic and straightforward, we do this first. After this we check whether two ingoing edges to any XOR node (if one exists in the graph) are related. If there are, a valid labeling cannot exist and we reject the mode as being (potentially) insecure. Otherwise, we proceed to a second phase in which we encode each constraint from Section 3.2 as an assertion in Z3. That is, we encode each edge of the graph as a variable, and each node as an assertion which forces the proper labels on the input and output variables of that node. We then run this through Z3, which tells us whether the expression is satisfiable or not given the asserted constraints.

Figure 4 presents an example encoding of a DUP node. Lines 1–6 define the labels for the outgoing edges. We encode the edge's type as an integer, treating $\perp$ as 0 and R as 2. (In the appendix we add a third type which we treat as 1.) Each bit in flags is treated as a Boolean value. In the referenced figure, `dup_l_type`, `dup_l_flag_out`, and `dup_l_flag_prf` denote type, flags.OUT, and flags.PRF, respectively, for the left outgoing edge of the DUP node, and likewise, `dup_r_type`, etc., denote the corresponding values for the right outgoing edge. In this example, the ingoing edge to the DUP node comes from GENRAND, whose labeling is denoted by `genrand_type`, etc.

[3]Available at https://z3.codeplex.com.

Lines 7–11 define the constraints. As an example, in Line 7 we "assert" that `dup_l_type`, `dup_r_type`, and `genrand_type` are all equal. This equates to the constraint on DUP as defined in Section 3.2. Likewise, Lines 8–11 enforce the correct behavior of the flags variable for each edge.

5.2 Program Synthesizer

We also developed a synthesizer for generating secure modes. Given a fixed size N (where by "size" here we mean the number of instructions in Block), one could naïvely iterate over all possible (valid) combinations of N instructions, feeding each resulting mode into our model checker to determine whether the mode is secure or not. Such an approach would be exceedingly slow, as the majority of instruction combinations result in trivially insecure (or undecryptable) schemes. Thus, we implement several simple but effective pruning mechanisms to remove incorrect or uninteresting modes from being analyzed. This pruning process eliminates more than 99.99% of the possible modes from being passed through the model checker, greatly improving the running time of our synthesizer.

We now describe some of our pruning techniques. One approach looks at the current instruction being added to the generated mode, and checks the previous program instruction(s) against a set of disallowed neighboring instructions. One such example is an XOR instruction applied to the two outgoing edges of a DUP instruction, which can never have a valid labeling. Another pruning strategy disallows two back-to-back PRP instructions; while this is not invalid per se, it only produces uninteresting modes (as no security is gained by the second PRP instruction).

6 Results

We ran our synthesizer to generate modes having Block algorithms containing up to 10 instructions[4]. We do not currently synthesize an Init algorithm, but instead used a fixed Init algorithm which generates a uniform IV and both outputs it and passes it on (as state) to the Block algorithm (cf. Figure 2). Our experiments include the INC instruction, which increments its input modulo 2^n, and the more complex constraints introduced in the appendix to handle this additional instruction.

Table 1 shows our results.[5] In the context of this table, a "valid" mode is defined as a mode whose acyclic graph representation is connected, contains at least one PRF or PRP node, contains exactly one M, START, OUT, and NEXTIV node, and contains no GENRAND nodes. A "decryptable" mode is a valid mode which passes our decryption check. Finally, a "secure" mode is one that also has a valid labeling. We also include a check for removing duplicate modes; that is, modes that have different graph layouts but are equivalent. (E.g., in one the left outgoing edge of a DUP node is output, and in the other the right outgoing edge of a DUP node is output.) Thus, the results in Table 1 represent *distinct* nodes.

We discovered 370 secure modes (out of 8836 valid modes). The modes containing seven instructions constitute the well-known CBC, OFB, and CFB modes. The modes containing eight instructions include CTR mode, as well as variants of CBC, OFB, and CFB mode with an additional PRF or PRP instruction introduced. We also synthesize PCBC mode.

In terms of performance, we found that we can synthesize secure modes of operation with ≤ 10 instructions in around 7 minutes on a standard laptop. However, we believe this can be greatly improved. Due to the fact that the Z3 OCaml bindings were not available at the time of this writing, we had to write our Z3 input to a file and then run a separate process to check the result. This additional process creation constitutes a large portion (nearly half) of the running time of our synthesizer. Porting our approach to use the Z3 OCaml bindings, once available, should greatly improve the overall running time of our tool.

[4] In our implementation we actually encode modes using a stack-based language which includes some additional instructions for moving items around the stack. When we say "up to 10 instructions," we include these extra instructions. However, the instruction counts in Table 1 do *not* include these extra instructions, and thus denote the actual sizes of the modes.

[5] This table differs slightly from the proceedings version due to bugs found in our code as well as modifications to the pruning techniques. The current table was generated from commit 3009664a7f9290b6733de6c8bd5116fb94aea079.

# Instructions	Valid	Decryptable	Secure
1–6	0	0	0
7	50	30	5
8	549	281	31
9	3130	1304	150
10	5107	1770	184
Total	8836	3383	370

Table 1: Number of valid, decryptable, and secure modes by number of instructions.

7 Conclusion and Future Work

We have introduced a method for reasoning about modes of operation using only "local" analysis of a single block of the given mode. We model modes as graphs and develop a labeling and constraint system on the edges of the graph, and show that if a mode can be correctly labeled then it is secure. Using this meta-theorem, we developed a model checker and synthesizer for both automatically verifying whether a mode is secure and automatically generating new modes. With these tools, we discovered 370 unique secure modes, many of which have never been studied before in the literature.

As future work, we plan to investigate whether it is possible to adapt our approach to the automated analysis and synthesis of authenticated encryption schemes and/or message authentication codes. We hope that the general framework presented in this paper can be applied to these primitives as well as other cryptographic tools.

In terms of extending the present work, it would be useful to couple our approach with automated generation of a proof of security in EasyCrypt [BGHB11]. One could also explore adding operations such as concatenation or field multiplication to our language.

Acknowledgments

The authors thank Michael Hicks and Jeff Foster for comments on an earlier draft of this work, and Martijn Stam for identifying issues with the results in Table 1 in the proceedings version.

Work of Alex J. Malozemoff was conducted with Government support awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. Work of Jonathan Katz was done for Exelis under contract number N00173-11-C-2045 to NRL. Work of Matthew D. Green was supported by the U.S. Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211.

References

[AGH13] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 399–410, Berlin, Germany, November 4–8, 2013. ACM Press.

[AGHP12] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 474–487, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BCG+13] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 1247–1260, Berlin, Germany, November 4–8, 2013. ACM Press.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Berlin, Germany.

[CDE+08] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 371–380, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press.

[CEL07] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In V. Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375, New Delhi, India, December 12–14, 2007. Springer, Berlin, Germany.

[GLLSN09] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In Anupam Datta, editor, *13th Asian Computing Science Conference*, pages 39–53. Springer, Berlin, Germany, 2009.

[GLLSN12] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Ali Miri, and Nadia Tawbi, editors, *5th International Symposium on Foundations and Practice of Security*, volume 7743 of *Lecture Notes in Computer Science*, pages 23–31. Springer, Berlin, Germany, 2012.

[KL08] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2008.

[MW80] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359, New Delhi, India, February 5–7, 2004. Springer, Berlin, Germany.

[Sri10] Saurabh Srivastava. *Satisfiability-Based Program Reasoning and Program Synthesis*. PhD thesis, University of Maryland, 2010.

Appendix

In this appendix, we modify the labels and constraints to support the INC instruction, and prove a version of Theorem 1 in this setting.

Edge labels. Recall we have a mode of encryption specified by directed, acyclic graphs Init and Block. Let G denote their union, and let E denote the total number of edges in G. Every edge is labeled with a 3-tuple $(\mathsf{fam}, \mathsf{type}, \mathsf{flags})$ where

- fam is as before.

- $\mathsf{type} \in \{\mathsf{R}, \mathsf{U}, \perp\}$ represents the "type" of the edge. Intuitively, R denotes "random," U denotes "unique," and $\perp$ denotes "adversarially controlled." We impose the ordering $\perp < \mathsf{U} < \mathsf{R}$.
- $\mathsf{flags} \in \{0,1\}^4$ is a bit-vector. The first three bits denote whether the edge can be used as input to OUT, PRF, or INC, respectively, and we index into this bit-vector using the notation $\mathsf{flags}.OUT$, $\mathsf{flags}.PRF$, and $\mathsf{flags}.INC$. In addition, we have a flag $\mathsf{flags}.INCd$ that, intuitively, indicates whether the given edge or one of its related edges ever had $\mathsf{flags}.INC = 1$ (either currently or in the past).

Constraints. In trying to label G, we begin by assigning each edge in G a set of families as in Section 3.2, with the following rule for INC nodes:

- INC: The outgoing edge is assigned the same set of families as the ingoing edge.

Once each edge in G has been assigned a set of families, we then try to assign a type and flags to each edge in G subject to the following constraints:

- START: Say there is a NEXTIV node in G (whether in Init or in Block) whose ingoing edge has type type and flags flags. Then the outgoing edge from START has $\mathsf{type}' \leq \mathsf{type}$, $\mathsf{flags}' \leq \mathsf{flags}$, and $\mathsf{flags}'.INCd = \mathsf{flags}.INCd$.
- GENRAND: The outgoing edge gets type R and $\mathsf{flags}.PRF = \mathsf{flags}.OUT = 1$. The value of $\mathsf{flags}.INC$ can be arbitrary, and we set $\mathsf{flags}.INCd = \mathsf{flags}.INC$.
- M: The outgoing edge gets $\mathsf{type} = \perp$ and $\mathsf{flags} = 0000$.
- DUP: Say the input edge has type type and flags flags. The two output edges get types $\mathsf{type}_1 = \mathsf{type}_2 = \mathsf{type}$ and flags flags_1 and flags_2 such that: (1) $\mathsf{flags}_1.OUT$ & $\mathsf{flags}_2.OUT = 0$, and (2) $\mathsf{flags}_1.OUT \mid \mathsf{flags}_2.OUT = \mathsf{flags}.OUT$ (the rules for $\mathsf{flags}.PRF$ and $\mathsf{flags}.INC$ are analogous), and (3) $\mathsf{flags}_1.INCd = \mathsf{flags}_2.INCd = \mathsf{flags}.INCd$.
- INC: We require that the ingoing edge to INC has $\mathsf{type} \in \{\mathsf{U}, \mathsf{R}\}$ and $\mathsf{flags}.INC = 1$. The outgoing edge gets $\mathsf{type}' = \mathsf{U}$, $\mathsf{flags}'.OUT = 0$, and $\mathsf{flags}'.INC = \mathsf{flags}'.PRF = \mathsf{flags}'.INCd = 1$.
- PRF: We require that the ingoing edge to PRF has $\mathsf{type} \in \{\mathsf{U}, \mathsf{R}\}$ and $\mathsf{flags}.PRF = 1$. The outgoing edge is treated just like the output of GENRAND.
- XOR: We require that the ingoing edges to an XOR node are unrelated, at least one of them has type R, and both have $\mathsf{flag}.INCd = 0$. The outgoing edge gets type R. Say the input edges have flags flags_1 and flags_2, respectively. Then:
  1. If both ingoing edges have type R, the outgoing edge gets flags flags' such that $\mathsf{flags}'.OUT = \mathsf{flags}_1.OUT \mid \mathsf{flags}_2.OUT$ (the rule for $\mathsf{flags}'.PRF$ is analogous), and $\mathsf{flags}'.INC = \mathsf{flags}'.INCd = 0$.
  2. If only one edge (say, the first) has type R, then $\mathsf{flags}'.OUT = \mathsf{flags}_1.OUT$ and $\mathsf{flags}'.PRF = \mathsf{flags}_1.PRF$ but $\mathsf{flags}'.INC = \mathsf{flags}'.INCd = 0$.
- OUT: We require the ingoing edge to an OUT node to have type R and $\mathsf{flags}.OUT = 1$.

One can verify that the following invariants hold for any labeling satisfying the above constraints:

- **Invariant 1:** If an edge has $\mathsf{flags}.INC = 1$, then it also has $\mathsf{flags}.INCd = 1$.
- **Invariant 2:** If edge e has $\mathsf{flags}.INCd = 1$ and edge e' has $\mathsf{flags}'.INCd = 0$, then e and e' are unrelated.

We rely on these when proving the following theorem.

Theorem 2. *Let* (Init, Block) *be a legal mode of operation specified using the instructions above (including INC). If the mode has a valid labeling, then it is a secure. In other words, when the mode is instantiated with any secure block cipher, the resulting encryption scheme is IND$-CPA secure.*

Proof. We assume the reader has read the proof of the analogous theorem in Section 4, and thus we jump straight to the details.

Fix some valid labeling of Init and Block. Each time the attacker requests encryption of some ℓ-block message we imagine creating a graph G consisting of one copy of Init and ℓ copies of Block. The valid labeling of Init and Block naturally extends to a labeling of this graph G; the edges in Init are labeled in the obvious way, and the corresponding edges in each copy of Block are assigned the same label. (The only special case is an edge from a NEXTIV node in one block to a START node in the next block, which we assign the same label as the ingoing edge to that NEXTIV node.)

Consider the step-by-step process by which edges in G are assigned values when a message $\mathbf{m}$ is encrypted. (Although we speak here of "assigning values," these are really random variables and so "assigning a value" is conceptual.) Edges are assigned values in topological order; i.e., the value of an edge is not assigned unless values have been assigned to its parents. We say an edge is *active* if it has been assigned a value but its children have not. (When we begin encrypting a message, the only active edges are the outgoing edges from all the M-nodes.) The only exceptions are ingoing edges to OUT nodes and the ingoing edge to the final NEXTIV node; these become active when they are assigned a value, and remain active until the relevant instruction is processed. (Processing an OUT instruction just means declaring the ingoing edge to that instruction as no longer being active, but instead being "output" as part of the ciphertext. Processing the final NEXTIV node means giving the computed ciphertext to the adversary.) For simplicity, we assume all edges in a given copy of Block are assigned a value before assigning a value to the outgoing edge of the NEXTIV node in that copy of Block.

When we are done encrypting a message we give the resulting ciphertext to the attacker, who then chooses the next message to be encrypted. We then create another graph and assign values as above. We stress that previous graphs are maintained: even though none of their edges are active, we may still refer to the values taken by edges in previous graphs.

At any step in the above process, define the following sets of edges in the graph:

$$\begin{array}{rcl} PRF & = & \text{edges with } \mathsf{type} \in \{\mathsf{U},\mathsf{R}\} \text{ and } \mathsf{flags}.PRF = 1 \text{ that are either active or} \\ & & \text{have entered a PRF node in the past} \\ PRF^* & = & \text{active edges with } \mathsf{type} = \mathsf{R},\ \mathsf{flags}.PRF = 1, \text{ and } \mathsf{flags}.INCd = 0 \\ INC & = & \text{active edges with } \mathsf{type} \in \{\mathsf{U},\mathsf{R}\} \text{ and } \mathsf{flags}.INC = 1 \\ OUT_a & = & \text{active edges with } \mathsf{type} = \mathsf{R} \text{ and } \mathsf{flags}.OUT = 1 \\ OUT & = & \text{edges output for the message currently being processed} \end{array}$$

Let T be a bound on the number of instructions in the graph G defined earlier. (Note that T must be polynomial, since the number of message blocks is assumed to be polynomial.) We prove that with all but negligible probability the following invariants hold at the end of each step:

- **Invariant 3:** For all $e, e' \in PRF$, $\mathsf{val}(e) \neq \mathsf{val}(e')$. (Formally, $\mathsf{val}(e)$ and $\mathsf{val}(e')$ are random variables, and what we mean here is that with all but negligible probability they take on different values.)
- **Invariant 4:** After executing the ith instruction, for all $e \in INC$ and $e' \in PRF$:

  $$\mathsf{val}(e) \notin \{\mathsf{val}(e') - T + i, \ldots, \mathsf{val}(e') - 1\}.$$

  Moreover, for all distinct $e, e' \in INC$:

  $$\mathsf{val}(e) \notin \{\mathsf{val}(e') - T + i, \ldots, \mathsf{val}(e')\}.$$

  (Formally, $\mathsf{val}(e)$ and $\mathsf{val}(e')$ are random variables, and what we mean formally in the first case—and analogously for the second case—is that with all but negligible probability the value x taken by $\mathsf{val}(e)$ is not in the set $\{x' - T + i, \ldots, x' - 1\}$, where x' is the value taken by $\mathsf{val}(e')$.)

- **Invariant 5:** For all $e \in PRF^*$ and any set E of active edges unrelated to e, $\mathsf{val}(e)$ is uniform even conditioned on $\mathsf{val}(E)$ and $\mathsf{val}(PRF \setminus \{e\})$.
- **Invariant 6:** (a) Random variables $\mathsf{val}(OUT)$ are jointly uniform, even conditioned on the values of all previous ciphertext blocks. Moreover, (b) for any $S \subseteq OUT_a$, random variables $\mathsf{val}(S)$ are jointly uniform, even conditioned on the values of all previous ciphertext blocks, $\mathsf{val}(OUT)$, and the values on all active edges unrelated to edges in S.

Note that Invariant 6(a) proves the theorem.

Clearly, the invariants hold at the beginning of the experiment. Now, assume the invariants hold before executing an instruction; we show that, with high probability, they continue to hold after executing the instruction. As in the proof of Theorem 1, we let "unhatted" variables denote the sets in question before executing the instruction, and let "hatted" variables denote the relevant sets following the instruction. We now consider the possible instructions:

- Begin processing the next message: Note there are no active edges before this instruction, and the active edges after this instruction are just the set of outgoing edges from M-nodes. $\widehat{PRF} = PRF$ and $\widehat{PRF}^* = \widehat{INC} = \widehat{OUT}_a = \widehat{OUT} = \emptyset$. Invariant 3 holds by the inductive assumption, and the remaining invariants trivially hold.
- NEXTIV: The ingoing and outgoing edges have the same label and value; thus, the invariants are not affected.
- START: Let e, with type type and flags flags, be the ingoing edge and e', with type type' and flags flags', be the outgoing edge. Edges e and e' have the same value but possibly different labels. However, the constraints ensure that $\mathsf{type}' \leq \mathsf{type}$, $\mathsf{flags}' \leq \mathsf{flags}$, and $\mathsf{flags}.INCd = \mathsf{flags}'.INCd$. Thus, $\mathsf{val}(\widehat{PRF}) \subseteq \mathsf{val}(PRF)$ and $\mathsf{val}(\widehat{INC}) \subseteq \mathsf{val}(INC)$, even when viewed as multisets, and thus Invariant 3 and Invariant 4 continue to hold. $OUT = \widehat{OUT}$, so Invariant 6(a) continues to hold.

  When processing the START instruction, the only active edges—besides e itself—are outgoing edges from M-nodes, which are not in OUT_a or PRF^* and do not have $\mathsf{flags}.INC = 1$. It is thus easy to verify the remaining invariants:

  - If $e' \notin \widehat{PRF^*}$, then $\widehat{PRF^*} = \emptyset$ and Invariant 5 is trivial; otherwise, $e \in PRF^*$ and by induction the invariant holds.
  - If $e' \notin \widehat{OUT}_a$, then $\widehat{OUT}_a = \emptyset$ and Invariant 6(b) is trivial; otherwise, $e \in OUT_a$ and by induction the invariant holds.
- DUP: Let e_0 be the ingoing edge and e_1, e_2 the outgoing edges.
  - Invariant 3: If $e_0 \notin PRF$, then $e_1, e_2 \notin \widehat{PRF}$. If $e_0 \in PRF$, then $e_0 \notin \widehat{PRF}$ and exactly one of e_1 or e_2 is in $\widehat{PRF}$. Since $\mathsf{val}(e_0) = \mathsf{val}(e_1) = \mathsf{val}(e_2)$, this means that $\mathsf{val}(PRF) = \mathsf{val}(\widehat{PRF})$ (even viewed as multisets), so Invariant 3 continues to hold.
  - Invariant 4: Arguing as above, $\mathsf{val}(\widehat{INC}) = \mathsf{val}(INC)$ even viewed as multisets. Since $\mathsf{val}(PRF) = \mathsf{val}(\widehat{PRF})$ also, the invariant continues to hold.
  - Invariant 5: Fix $\hat{e} \in \widehat{PRF^*}$ and a set $\widehat{E}$ of active edges unrelated to $\hat{e}$. There are two cases to consider:
    - $\hat{e} \in \{e_1, e_2\}$. Then $e_0 \in PRF^*$. Note that $\mathsf{val}(e_0) = \mathsf{val}(\hat{e})$. Since $\hat{e}$ and e_0 are related, all edges in $\widehat{E}$ are unrelated to e_0, and thus by induction $\mathsf{val}(e_0)$ is uniform even conditioned on $\mathsf{val}(\widehat{E})$ and $\mathsf{val}(PRF \setminus \{e_0\})$. Since $\mathsf{val}(e_0) = \mathsf{val}(\hat{e})$, the invariant continues to hold.
    - $\hat{e} \notin \{e_1, e_2\}$. In this case, $\hat{e} \in PRF^*$. If $\widehat{E} \cap \{e_1, e_2\} \neq \emptyset$, then let $\tilde{E}$ be the same as $\widehat{E}$ except with e_0 in place of e_1 and/or e_2; otherwise, let $\tilde{E} = \hat{E}$. Clearly, $\mathsf{val}(\tilde{E}) = \mathsf{val}(E)$ and the invariant continues to hold.

– Invariant 6: We have $OUT = \widehat{OUT}$ and so Invariant 6(a) is trivial. Note that $e_0 \in OUT_a$ iff exactly one of e_1 or e_2 is in $\widehat{OUT}_a$; thus, $\mathsf{val}(OUT_a) = \mathsf{val}(\widehat{OUT}_a)$, even when viewed as multisets. Since e_1 and e_2 are related, and e_0, e_1, and e_2 are all related to the same edges, Invariant 6(b) continues to hold as well.

- GENRAND: This instruction adds an edge to PRF and OUT_a, and possibly PRF^* and INC. Since the value of this edge is uniform and independent of any previous edges, and the edge is not related to any other edges, all the invariants continue to hold (with high probability).

- PRF: By Invariant 3, the value on the ingoing edge to the PRF instruction is distinct from all values previously used as input to a PRF instruction. Thus, the value on the outgoing edge is uniform, and all the invariants continue to hold (with high probability) just as in the case of a GENRAND instruction.

- INC: Denote the input edge by e_0 and the output edge by e_1. Note that $e_0 \in INC$, $e_0 \notin PRF^*$ (by Invariant 1) and $e_1 \in \widehat{INC} \cap \widehat{PRF}$, $e_1 \notin \widehat{PRF^*}$. Because Invariant 4 holds before this step, Invariant 3 and Invariant 4 hold after this step.

  – Invariant 5: Note first that $\widehat{PRF^*} = PRF^*$. Fix $e \in \widehat{PRF^*} = PRF^*$ and a set of active edges $\widehat{E}$ unrelated to e. Let E denote the active edges unrelated to e. The set $\widehat{E} \cup (\widehat{PRF} \setminus \{e\})$ is identical to $E \cup (PRF \setminus \{e\})$ except that e_1 is in the former (since $e_1 \in \widehat{PRF}$ but $e_1 \neq e$) and e_0 is in the latter (since $e_0 \in E$ by Invariant 2). Since e_1 is a deterministic function of e_0, the invariant continues to hold.

  – Invariant 6: Invariant 6(a) clearly continues to hold. Since $\widehat{OUT}_a \subseteq OUT_a$ and $\mathsf{val}(e_1)$ is a deterministic function of $\mathsf{val}(e_0)$, Invariant 6(b) also continues to hold.

- XOR: Let e_0, e_0' be the two ingoing edges, and e_1 the outgoing edge. We know that e_0, e_0' are unrelated, $\mathsf{flags}_0.INCd = \mathsf{flags}_0'.INCd = 0$, and at least one of e_0, e_0' (say, e_0) has $\mathsf{type}_0 = \mathsf{R}$.

  – Invariant 3: If $\mathsf{flags}_1.PRF = 0$ the invariant trivially continues to hold. Otherwise, there are two cases to consider:

    * $\mathsf{flags}_1.PRF = 1$ and $\mathsf{flags}_0.PRF = 1$: In this case, $e_0 \in PRF^*$. Let E be the set of active edges unrelated to e_0, and note $e_0' \in E$. Invariant 5 implies that $\mathsf{val}(e_0)$ is uniform even conditioned on $\mathsf{val}(e_0')$ and $\mathsf{val}(PRF \setminus \{e_0\})$. Thus $\mathsf{val}(e_1) = \mathsf{val}(e_0) \oplus \mathsf{val}(e_0')$ is uniform conditioned on $\mathsf{val}(\widehat{PRF} \setminus \{e_1\})$, and so Invariant 3 continues to hold with all but negligible probability.

    * $\mathsf{flags}_1.PRF = 1$ and $\mathsf{flags}_0.PRF = 0$: Here it must be the case that e_0' has type R and $\mathsf{flags}_0'.PRF = 1$. An argument as above, swapping e_0' and e_0, shows that Invariant 3 continues to hold except with negligible probability.

  – Invariant 4: Note that $\widehat{INC} = INC$. If $\mathsf{flags}_1.PRF = 0$, the invariant trivially continues to hold. Otherwise, $e_1 \in \widehat{PRF^*}$ and so at least one of e_0 or e_0' (say e_0) is in PRF^*. Invariant 2 says that every edge in INC is unrelated to e_0. Arguing as above (using Invariant 5) gives that $\mathsf{val}(e_1)$ is uniform even conditioned on $\mathsf{val}(\widehat{INC})$ and so Invariant 4 continues to hold.

  – Invariant 5: Fix $\hat{e} \in \widehat{PRF^*}$ and a set $\widehat{E}$ of active edges unrelated to $\hat{e}$. There are two cases to consider:

    * $\hat{e} = e_1$. As above, one can show that $\mathsf{val}(\hat{e}) = \mathsf{val}(e_1) = \mathsf{val}(e_0) \oplus \mathsf{val}(e_0')$ is uniform even conditioned on $\mathsf{val}(\widehat{E})$ and $\mathsf{val}(\widehat{PRF} \setminus \{\hat{e}\})$.

    * $\hat{e} \neq e_1$. Thus, $\hat{e} \in PRF^*$. If $e_1 \in \widehat{E}$, then let $\tilde{E}$ be the same as $\widehat{E}$ except with e_0 and e_0' in place of e_1; otherwise, let $\tilde{E} = \widehat{E}$. By induction, $\mathsf{val}(\hat{e})$ is uniform even conditioned on $\mathsf{val}(E)$ and $\mathsf{val}(PRF \setminus \{\hat{e}\})$, and thus we conclude that $\mathsf{val}(\hat{e})$ is also uniform even conditioned on $\mathsf{val}(\widehat{E})$ and $\mathsf{val}(\widehat{PRF} \setminus \{\hat{e}\})$.

  - Invariant 6: Invariant 6(a) clearly holds. As for Invariant 6(b), an argument as above shows the $\mathsf{val}(e_1)$ is uniform even conditioned on all the required values, and so this invariant continues to hold.

- OUT: This instruction reduces the number of active edges, so Invariant 3, Invariant 4, and Invariant 5 trivially continue to hold. Invariant 6 continues to hold because of how OUT_a is defined.

The above prove the theorem. □

Changelog

- Version 1.0 (September 30, 2014): First release. This is the same as the proceedings version except as follows:
  - Added discussion of work of Courant et al. [CEL07] in Section 1.1.
  - Updated Table 1 due to bug fixes in mode synthesis code.