Moving the Bar on Computationally Sound Exclusive-Or

Catherine Meadows(✉)

Naval Research Laboratory, Washington DC, USA
catherine.meadows@nrl.navy.mil

Abstract. Soundness of symbolic security with respect to computational security was originally investigated from the point of cryptographic protocol design. However, there has been an emerging interest in applying it to the automatic generation and verification of cryptographic algorithms. This creates a challenge, since it requires reasoning about low-level primitives like exclusive-or whose actual behavior may be inconsistent with any possible symbolic behavior.

In this paper we consider symbolic and computational soundness of cryptographic algorithms defined in terms of block ciphers and exclusive-or. We present a class of algorithms in which security is defined in terms of IND$-CPA-security, that is, security against an adaptive chosen plaintext adversary's distinguishing the output of the cryptosystem from a random string. We develop conditions for symbolic security and show that they imply computational security. As a result of this, we are able to identify a class of cryptosystems to which results such as Unruh's [25] on the impossibility of computationally sound exclusive-or do not apply, in the sense that symbolic security implies computational security against an adaptive adversary. We also show how our results apply to a practical class of cryptosystems: cryptographic modes of operation.

1 Introduction

Symbolic methods have been in use in cryptography for a long time, starting with Dolev and Yao's seminal paper [12], progressing on to powerful crypto protocol analysis tools, and most recently, to methods for the automated generation and analysis of cryptographic algorithms. Ideally, one would like these methods to be sound with respect to some computational model, so that symbolic security implies computational security. Although results in this area exist, it has been found difficult to extend them to certain low-level primitives such as group operations. To the best of my knowledge, it was Unruh [25] who first pointed out the difficulty, using low-level behavior of exclusive-or that can't be captured in a symbolic model.

Such behavior can have a direct effect on security proofs of cryptosystems. For example, many proofs of indistinguishability depend upon proving that the probability of a collision between random functions is negligible. But consider the counter-example below, which is similar, but not identical, to Unruh's.[1]

[1] I am grateful to an anonymous reviewer for this example.

E. Bertino et al. (Eds.): ESORICS 2021, LNCS 12973, pp. 275–295, 2021.
https://doi.org/10.1007/978-3-030-88428-4_14

Example 1. Let f be a random function from λ-length bitstrings to λ-length bitstrings (blocks, for short) computed by a challenger, and let $r_0, \ldots, r_m$ be randomly chosen bitstrings. Suppose that the challenger sends $f(r_0), r_1, \ldots, r_{\lambda+1}$ to the adversary, and the adversary returns a subset S of $\{r_1, \ldots r_{\lambda+1}\}$ to the challenger, who returns $f(r_0 \oplus \bigoplus_{r_i \in S} r_i)$, where $\bigoplus_{r_i \in S} r_i$ stands for the exclusive-or of all r_i such that $r_i \in S$. The adversary can never succeed in distinguishing the challenger's output from random in the symbolic model, since each term sent by the challenger is a different term standing for a different randomly generated bitstring. However, in the computational model the set of λ-bit strings is the vector space $\mathbb{Z}_2^\lambda$, and so no set of $m > \lambda$ bitstrings can be linearly independent modulo $\oplus$. Thus there is a subset S of $r_1, \ldots, r_m$ such that $\bigoplus_{r_i \in S} r_i = 0^\lambda$, which the adversary can find via Gaussian elimination. It can then send S to the challenger, which computes $f(r_0 \oplus \bigoplus_{r_i \in S} r_i) = f(r_0)$ and returns it to the adversary. In this way the adversary can force the first and last bitstrings sent by the challenger to be equal, and so its output is distinguishable from random.

One important feature of this counterexample is that the adversary is given all the information it needs in order to make its choice. If it was not shown any of the bitstrings before the final response from the encryptor, it would not know which subset would result in a collision with $f(r_0)$. Thus, for any subset it chose, the probability of a collision would be $2^{-\lambda}$. In some cases it may not be practical to deprive the adversary of adaptive choice altogether, but it may be possible to limit it so that its contribution to the adversary's advantage is negligible. For example, we can limit the adversary's choice of functions that it can compute. Consider the case in which the adversary is limited to choosing, instead of any subset of $r_1, \ldots, r_m$, one of a polynomial number $q(\lambda)$ of subsets of $r_1, \ldots, r_m$. In this case, the probability of a collision between $f(r_0)$ and $f(r_0 \oplus \bigoplus_{r_i \in S} r_i)$, for some S chosen by the adversary, would be bounded by $q(\lambda) \cdot 2^{-\lambda}$, a negligible function of λ. In this paper we will consider cryptosystems that impose a stronger condition on the adversary, in which not only adaptive choice, but any choice, is limited. For the purposes of this paper, it has the advantage that, not only it is simpler to reason about, but, as we shall see, there are non-trivial classes of cryptosystems that have this property. To the best of my knowledge this feature of a cryptosystem does not have a specific name; we shall call it *polynomially bounded execution choice* (PBEC). We will define this more formally later on.

Consider the following example: the cryptographic mode of operation Cipher Block Chaining.

Example 2 (Cipher Block Chaining (CBC)). Let E be a block cipher, and let $f = E_K$ denote encryption with E using a key K. The input to the algorithms is a sequence x_1 through x_n of plaintext blocks, and the output is a list of ciphertext blocks $C_0, \ldots, C_n$ returned only after all plaintext blocks are received:

1. $C_0 = r$, where r is a randomly generated block known as an *initialization vector* (IV), and;
2. $C_i = f(x_i \oplus C_{i-1})$ for $i > 0$.

Suppose that f is indistinguishable from a random function. Suppose also that the number of different terms $f(s)$ and r_i that the adversary can request the encryptor to compute is bounded by a polynomial in λ. Thus the maximum number of messages and the maximum length of a message must also be polynomially bounded, say by $p_1(\lambda)$ and $p_2(\lambda)$ respectively. It follows that the set of all terms that the adversary could possibly request the encryptor to compute is contained in the set $D_0 = \{r_i \mid 1 \leq i \leq p_1(\lambda)\} \cup D_1 = \{f(C_{i,j-1} \oplus x_{i,j}) \mid (1 \leq i \leq p_1) \wedge (1 \leq j \leq p_2(\lambda) - 1)\}$, where $C_{i,j}$ (respectively $x_{i,j}$) denotes the i'th ciphertext (respectively, plaintext) block in the j'th message. The cardinality of this set is $p_1(\lambda) \cdot p_2(\lambda)$, so the probability of a collision is bounded by $p_1(\lambda)^2 \cdot p_2(\lambda)^2 \cdot 2^{-\lambda}$, thus giving us polynomially bounded execution choice.

In the remainder of this paper we describe a class of cryptosystems using the symbols $\mathbf{f}$ for E_K as in Example 2, $\oplus$ for bitwise exclusive-or, a set $\mathbf{R}$ of bound variables r standing for random bitstrings, and a set of free variables $\mathbf{X}$ standing for blocks of adversarial input. A *symbolic history* of such a cryptosystem describes a sequence in which an adversary sends plaintext blocks to an encryptor and gets encrypted blocks in return. However, in a symbolic history the plaintext blocks are replaced by free variables, and the ciphertext blocks are replaced by symbolic terms that serve as recipes for computing the ciphertext blocks, with the free variables standing as placeholders for the plaintext. We show that such a cryptosystem satisfies IND\$-CPA-security (essentially, indistinguishability from random against an adaptive chosen plaintext adversary) if the following hold:

1. Nondegeneracy: There is no subset of terms sent in any symbolic history $\mathbf{H}$ whose elements $\oplus$-sum to zero.
2. Safety: There are no two subterms $\mathbf{f}(\mathbf{s})$ and $\mathbf{f}(\mathbf{t})$ of terms sent in any symbolic history $\mathbf{H}$ such that the normal form of $\mathbf{s} \oplus \mathbf{t}$ is $\mathbf{x} \oplus \mathbf{u}$ and $\mathbf{u}$ can be derived by $\oplus$-summing terms received by the adversary before it sends $\mathbf{x}$.
3. Polynomially bounded execution choice.

The rest of this paper is organized as follows. In Sect. 2 we give background and related work. We give symbolic preliminaries necessary for understanding the paper in Sect. 3. We describe our symbolic and computational models in Sect. 4 and give a precise statement of our main theorem sketched above. In Sect. 5 we prove the theorem. In Section 6 we describe some applications to cryptographic modes of operation. In Sect. 7 we conclude and discuss future work.

2 Background and Related Work

Probably the earliest work on proving computational soundness and completeness of symbolic methods for cryptographic proofs of security for protocols is that of Abadi and Rogaway in [2], against passive attackers. Shortly afterword, Backes et al. [4] and Micciancio and Warinschi [24] separately proved soundness and completeness of a symbolic model including a full Dolev-Yao adversary that interacts with principals over a network it completely controls. This work and the work that followed tended to avoid equational theories (e.g. $\mathbf{d}(\mathbf{e}(\mathbf{k}, \mathbf{m})) = \mathbf{m}$) as much as possible, relying instead on derivation rules (e.g. $\mathbf{k}, \mathbf{e}(\mathbf{d}, \mathbf{m}) \vdash \mathbf{m}$). These are equally expressive, as long as you can rely on the cryptographic mechanisms to rule out ill-formed terms such as $\mathbf{d}(\mathbf{k}, \mathbf{m})$. Moreover, since well-designed protocols generally outlaw such ill-formed terms, this is a reasonable restriction. Appropriate abstractions have even been found for more equationally rich algorithms such as Diffie-Hellman [9]. But it is harder to distinguish ill-formed terms for exclusive-or, since $\oplus$ is self-cancelling. Unruh's and others' examples give evidence that including the $\oplus$ theory in such systems may be problematic.

In spite of these issues, some progress has been made on computational soundness of symbolic models with equational theories. In [7] Baudet et al. develop computational implementations of equational theories for which soundness can be proved assuming a passive adversary, thus avoiding Unruh's counter-example. In [20] Kremer and Mazaré extend this to a computationally sound implementation of static equivalence (a symbolic analogue of indistinguishability) against an adversary that can adaptively request the encryptor to send members of a set of pre-defined message. We note, however, that Example 1 is secure according to their computational definition of static equivalence. This is not because of any fault in their computational realization. Rather, it is because static equivalence itself, which is expressed in terms of conditions on possible histories of message exchanges, does not fully capture the abilities of an adaptive adversary.

In the meantime, work has been ongoing on developing symbolic methods for reasoning about cryptoalgorithms that use exclusive-or and other Abelian group theories, much of it motivated by recent interest in the automatic generation and verification of cryptoalgorithms. In this approach, multiple instances of a particular class of algorithms (e.g. cryptographic modes of operation [17,22], garbled circuit schemes [11], collision-resistant hash functions [23], padding-based public key encryption algorithms [6]) are generated, usually automatically, and then checked for security and other desirable properties. Because a large number of candidate cryptosystems are generated, the verification techniques must be efficient. Symbolic verification of cryptosystems can often be easily automated, so symbolic verification techniques that can be proved sound and/or complete with respect to a computational model of security have become of more interest. Interestingly, many of the cryptosystems to which this automatic generation and verification approach have been applied involve exclusive-or or other types of group operations. However, these techniques tend to be applied to specific classes of algorithms, so general soundness and completeness results have not been as important. Indeed, we note that many of the cryptosystems have similar restrictions on the adversary; for example the modes of operation in [17,22] are PBEC, and the adversary in [11] is honest-but-curious.

We also discuss some related work on the symbolic design and analysis of cryptographic modes of operation. Gagné et al. [14] have developed a Hoare logic for proving semantic security of block cipher modes of encryption, and a program implementing the logic that can be used to automatically prove their security. However, their work concentrates on heuristically driven theorem proving techniques rather than on evaluating symbolic security conditions. We also note the work of Bard [5], who considers circumstances under which security for modes of encryption can be reduced to a collision-freeness property. Although [5] does not address symbolic security directly, our approach to deriving criteria for collision-freeness owes much to it. The work of Malezomoff et al. [22] and Hoang et al. [17] is probably the closest to that in this paper. They prove adaptive chosen plaintext security for cryptographic modes of operation based on deterministic block ciphers [22] and authenticated modes of encryption using block ciphers with tweaks [17] by defining a set of symbolic conditions checked on automatically generated modes using a messagewise schedule, in which the encryptor delays returning ciphertext until it has received all plaintext blocks, and they prove these conditions sufficient for security. The results in this paper extend the results of [22] to sufficient symbolic conditions for arbitrary schedules.

3 Symbolic Preliminaries

In this section we give definitions and results term in term algebras and unification used in this paper. Readers interested in a more in-depth discussion may find it in Baader and Nipkow's book *Term Rewriting and All That* [3].

A *signature* is a finite set of function symbols Σ of different arities. We write $\mathcal{T}_\Sigma(\mathcal{X})$ for the set of all terms constructed using function symbols from Σ and variables from a countable infinite set $\mathcal{X}$. $\mathcal{T}_\Sigma(\mathcal{X})$ is referred to as a *term algebra*. If $\mathbf{T} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$, we write $Sub(\mathbf{T})$ for the set of subterms of elements of $\mathbf{T}$. [2] Thus, if $\mathbf{T} = \{\mathbf{f}(\mathbf{g}(\mathbf{x}), \mathbf{a}), \mathbf{h}(\mathbf{f}(\mathbf{a}, \mathbf{b}))\}$, $Sub(\mathbf{T}) = \{\mathbf{f}(\mathbf{g}(\mathbf{x}), \mathbf{a}), \mathbf{g}(\mathbf{x}), \mathbf{a}, \mathbf{x}, \mathbf{h}(\mathbf{f}(\mathbf{a}, \mathbf{b})), \mathbf{f}(\mathbf{a}, \mathbf{b}), \mathbf{b}\}$. The set $\mathcal{X}$ is divided into a countable set of *names* bound by a quantifier ν and a countable set of free variables. We write $Var(\mathbf{t})$ (respectively $bv(\mathbf{t})$, $fv(\mathbf{t})$) for the set of variables (respectively, bound variables, free variables) present in a term $\mathbf{t}$. We say that a term is *ground* if it contains only function symbols and bound variables.

A Σ*-equation* is a pair $\mathbf{t} = \mathbf{t}'$. where $\mathbf{t}, \mathbf{t}' \in \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{X})$. A set E of Σ-equations induces a congruence relation $=_E$ on terms $\mathbf{t}, \mathbf{t}' \in \mathcal{T}_\Sigma(\mathcal{X})$, so that $\mathbf{t} =_E \mathbf{t}'$ if and only if $\mathbf{t}$ can be made equal to $\mathbf{t}'$ via applications of equations from E. An *equational theory* is a pair (Σ, E), where Σ is a signature and E a set of Σ-equations. We will refer to a term algebra $\mathcal{T}_\Sigma(\mathcal{X})$ together with an equational theory (Σ, E) as $(\mathcal{T}_\Sigma(\mathcal{X}), E)$. For example, suppose $\Sigma = \{\mathbf{d/2}, \mathbf{e/2}, \mathbf{k/0}, \mathbf{a/0}\}$, and $E = \{\mathbf{d}(\mathbf{x}, \mathbf{e}(\mathbf{x}, \mathbf{y})) = \mathbf{y}\}$. Then $\mathbf{e}(\mathbf{k}, \mathbf{d}(\mathbf{k}, \mathbf{e}(\mathbf{k}, \mathbf{a}))) =_{\mathbf{E}} \mathbf{e}(\mathbf{k}, \mathbf{a})$, by setting $\mathbf{x} = \mathbf{k}$ and $\mathbf{y} = \mathbf{a}$ in the equation $\mathbf{d}(\mathbf{x}, \mathbf{e}(\mathbf{x}, \mathbf{y})) = \mathbf{y}$.

A *substitution* σ is a mapping from free variables to $\mathcal{T}_\Sigma(\mathcal{X})$ that is the identity on all but a finite subset of the free variables known as the *domain* of σ. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. Application of σ to a term $\mathbf{t}$ is denoted by $\sigma\mathbf{t}$. The composition of two substitutions is $\sigma\theta\mathbf{t} = \sigma(\theta\mathbf{t})$. A substitution σ is an *E-unifier* of a system of equations $\mathbf{S} = \{\ldots, \mathbf{s_i} =^? \mathbf{t_i}. \ldots\}$ if

[2] We write symbolic terms in **bold** and computational terms in *italic* to make it easier to distinguish between the two.

$\sigma \mathbf{s_i} =_{\mathbf{E}} \sigma \mathbf{t_i}$ for every $\mathbf{s_i} =^? \mathbf{t_j} \in \mathbf{S}$. For example, consider $\mathbf{S} = \{\mathbf{w} =^? \mathbf{e^{-1}}(\mathbf{k}, \mathbf{z})\}$ over the algebra $(\mathcal{T}_\Sigma(\mathcal{X}), \{\mathbf{e^{-1}}(\mathbf{x}, \mathbf{e}(\mathbf{x}, \mathbf{y})) = \mathbf{y}\})$ described in the previous paragraph. Then the substitutions $\sigma_\mathbf{1} : \mathbf{w} \mapsto \mathbf{e^{-1}}(\mathbf{k}, \mathbf{z})$ and $\sigma_\mathbf{2} : \mathbf{z} \mapsto \mathbf{e}(\mathbf{k}, \mathbf{w})$ are both unifiers of $\mathbf{S}$ modulo $\mathbf{E}$. We will be interested in the algebra whose signature is the free unary symbol $\mathbf{f}$ plus an exclusive-or operator $\oplus$ and a null operator $\mathbf{0}$. We say that a term is $\mathbf{f}$-*rooted* if it is of the form $\mathbf{f}(\mathbf{s})$ for some term $\mathbf{s}$. The $\oplus$ operator is associative and commutative and satisfies $\mathbf{X} \oplus \mathbf{0} = \mathbf{0}$ and $\mathbf{X} \oplus \mathbf{X} = \mathbf{0}$. Equality of two terms modulo this theory is equivalent to equality under the theory $(R_\oplus \uplus AC)$ where AC is the associative and commutative rules for $\oplus$, and $R_\oplus$ is a set of *rewrite rules*, $\{\mathbf{X} \oplus \mathbf{0} \to \mathbf{X},\ \mathbf{X} \oplus \mathbf{X} \to\ \mathbf{0},\ \mathbf{X} \oplus (\mathbf{X} \oplus \mathbf{Y}) \to \mathbf{Y}\}$ oriented from left to right. A rewrite rule $\ell \to \mathbf{r}$ is applied to a term $\mathbf{t}$ by first finding a subterm $\mathbf{s}$ of $\mathbf{t}$ such that $\mathbf{s} = \sigma\ell$ modulo AC for some substitution σ, and then replacing $\mathbf{s}$ in $\mathbf{t}$ with $\sigma\ell$. Thus $\mathbf{0} \oplus \mathbf{a} \oplus \mathbf{b}$ can be reduced to $\mathbf{a} \oplus \mathbf{b}$ by noting that $\mathbf{0} \oplus \mathbf{a} \oplus \mathbf{b} = (\mathbf{0} \oplus \mathbf{a}) \oplus \mathbf{b} = \sigma\ell \oplus \mathbf{b}$ modulo AC, where ℓ is the left-hand side of $\mathbf{X} \oplus \mathbf{0} \to \mathbf{X}$, and $\sigma\mathbf{X} = \mathbf{a}$. In addition, every term $\mathbf{t}$ reduces after a finite number of steps to a *normal form* $\downarrow_\oplus \mathbf{t}$ to which no further rewrite rules can be applied, and this normal form is unique up to AC-equivalence. We refer to $(\mathbf{R}_\oplus \uplus \mathbf{AC})$ as the $\oplus$ theory for brevity, and use $\mathcal{T}_{(\Sigma_f, \oplus)}(\mathcal{X})$ to refer to the term algebra with signature $\{\mathbf{f}/\mathbf{1}, \oplus/\mathbf{2}, \mathbf{0}/\mathbf{0}\}$ and equational theory $\oplus$.

4 Symbolic and Computational Models

In this section we give a brief description of how we model the relationship between symbolic terms and computational functions. This is based on the abstract and concrete models of cryptosystems introduced by Baudet et al. in [7], with the main difference being that we allow free variables in the symbolic model to be replaced in the computational model by the output of Turing machines, instead of restricting ourselves to concrete computational representations of symbolic terms.

4.1 The Computational Model

We begin by giving a definition of IND\$-CPA security below. It uses the assumption that a block cipher is a member of a family of pseudorandom permutations index by a key K.

Definition 1. *Let $\mathcal{E}$ be a cryptosystem built using a keyed pseudorandom permutation from λ-length blocks to λ-length blocks. Let K be a key chosen uniformly at random. Suppose that a challenger, by flipping a coin, chooses either an encryption oracle that computes $\mathcal{E}_K$ on input from an adversary or a \$-oracle that returns a uniformly randomly chosen bitstring of the same length as the encryption oracle's response. The adversary is then allowed multiple queries to the oracle, and at the end of the game it outputs a bit. We say that $\mathcal{A}^{\mathcal{E}_K}(1^\lambda) = 1$ if the adversary outputs 1 after interacting with the encryption oracle, and $\mathcal{A}^\$(1^\lambda) = 1$*

if the adversary outputs 1 after interacting with the \$-oracle. We define the adversary's advantage *to be*

$$|Pr(K \xrightarrow{\$} Key; \mathcal{A}^{\mathcal{E}_K}(1^\lambda) = 1) - Pr(\mathcal{A}^{\$}(1^\lambda) = 1)|$$

We say that $\mathcal{E}$ is IND\$-CPA *secure if there is a negligible function τ of λ such that the advantage of any PPT adversary is bounded by $\tau(\lambda)$.*

4.2 Relationship Between Computational and Symbolic Models

We use the construction of Baudet et al. [7] as the basis for ours. Let λ be a security parameter. Each sequence of n ground terms $\mathbf{T} \subset \mathcal{T}_{(\mathbf{\Sigma},\oplus)}(\mathcal{X})$ determines a probability distribution T_λ over $\{0,1\}^{\lambda \cdot n}$ called the *computational realization* T_λ *of* $\mathbf{T}$. This is defined as follows: To compute the output of a bound variable $\mathbf{r}$, we choose a λ-length bitstring uniformly at random. If $\mathbf{r}$ occurs more than once in $\mathbf{T}$, it is replaced with the same random λ-length bitstring wherever it occurs. We also replace 0 by a bitstring of λ zeroes, $\oplus$ by bitwise exclusive-or on λ-length bitstrings, and f by a keyed pseudorandom permutation. Each time the recipe defined above is followed, it will produce an $n \cdot \lambda$-length bitstring sampled from the distribution T_λ, called an *output* of T and denoted by $[\![T]\!]_\lambda$. This is an abuse of notation, since T may have many possible outputs, but in general we will use $[\![T]\!]_\lambda$ to mean "the output just produced by T", so which output is meant should be clear. In addition, when we can do so without confusion, we will drop the λ.

We expand on [7] to consider the case where $\mathbf{T}$ contains free variables. Free variables play a special role: they are place holders for inputs to a term. In our case they will always stand for inputs from the adversary. In particular, the free variables $\mathbf{x}_1, \ldots \mathbf{x}_\ell$ appearing in $\mathbf{T}$ stand for inputs supplied to the computational realization T of $\mathbf{T}$ by $\theta_1, \ldots, \theta_\ell$ where each θ_i is a suite of $\theta_{i,\lambda}$ programs run by the adversary, whose input is the adversary's state, that supply λ-bit blocks used as input to T_λ. Note that we do not require that the θ_i themselves be computational realizations of elements of $\mathcal{T}_{(\Sigma,\oplus)}(\mathcal{X})$; they can be arbitrary probabilistic polynomial-time Turing machines. We call a map θ that maps a finite set of variables of $\mathcal{T}_{(\Sigma,\oplus)}(\mathcal{X})$ to such programs and is the identity on all other variables a *computational substitution*. If θ is a computational substitution, and $\mathbf{T}$ is a finite subset of $\mathcal{T}_{(\Sigma,\oplus)}(\mathcal{X})$, we define θT to be the substitution obtained by replacing each variable x used by the computational functions in T with θx. The composition of two computational substitutions is defined in the natural way.

If a θ is a computational substitution to t we define $[\![\theta t]\!]$ to be the output obtained from t by using $[\![\theta x]\!]$ wherever a free variable $\mathbf{x}$ appears in $\mathbf{t}$.

4.3 *MOO*$_\oplus$ Cryptosystems and Symbolic Histories

A *symbolic MOO*$_\oplus$ *history*[3] (symbolic history for short) gives a recipe for constructing a sequence of messages exchanged between an adversary and an encryp-

[3] The term comes from"mode of operation".

tor. The adversary's input to the encryptor is represented by free variables, where each free variable sent by the adversary is unique. The encryptor's output to an adversary is a sequence of $\mathcal{T}_{(\Sigma_f,\oplus)}(\mathcal{X})$ terms representing computations on input from the adversary, whose free variables are limited to variables previously received from the adversary in the history. $MOO_\oplus$ histories are analogous to the *frames* defined by Abadi and Fournet [1]; that is, they represent records of protocol executions.[4]

To give an example of a symbolic history, we consider the case, using cipher block chaining, in which an adversary interacts with an encryptor that allows it to encrypt two messages in parallel, timing its response so that the adversary receives the two IVs first, and the two first blocks of ciphertext right after sending the two first blocks of plaintext:

$$\nu\mathbf{r_1}.\nu\mathbf{r_2}[\mathbf{r_1},\mathbf{r_2}.\mathbf{x_1}.\mathbf{x_2}.\mathbf{f}(\mathbf{r_1}\oplus\mathbf{x_1}).\mathbf{f}(\mathbf{r_2}\oplus\mathbf{x_2})]$$

A $MOO_\oplus$ *program* specifies the possible interactions between the adversary and the encryptor. A $MOO_\oplus$ program is a program of bounded size that specifies 1) the encryptor's schedule for receiving input and sending output and 2) what terms the adversary can request the encryptor to evaluate and when.

Since the adversary in the IND\$-CPA game is polynomially bounded, there is a polynomial function p of the security parameter λ such that the number of blocks it can send and receive is bounded by $p(\lambda)$. This means that, for any symbolic history $\mathbf{H}$, a concrete adversary using security parameter λ can execute $\mathbf{H}$ as long as $length(\mathbf{H}) \leq p(\lambda)$. Thus, for every history $\mathbf{H}$, there is an infinite set $V_\mathbf{H}$ of values λ of the security parameter (namely, all (λ such that $length(\mathbf{H}) \leq p(\lambda)$) such that the adversary can interact with H_λ. Moreover, if $\mathbf{H}'$ extends $\mathbf{H}$, then $V_{\mathbf{H}'} \subseteq V_\mathbf{H}$. This makes it possible to allow arbitrarily long histories in the symbolic model.

5 $MOO_\oplus$ Games and Security Proofs

In this section we define a sequence of games and use them to prove our main result: that if a $MOO_\oplus$ -cryptosystem satisfies PBEC, then there are symbolic conditions that imply IND\$-CPA security.

There are four such games, described in Section Sect. 5.1: G_{crypt}, in which f is a pseudorandom permutation, G_{rperm}, where f is a random permutation of strings, G_{rstr}, where f is a random function from λ-length strings to λ-length strings, and G_{rsymb} in which f is a random function from symbolic terms to λ-length blocks. We also show that the output of G_{rsymb} is random as long as the cryptosystem is nondegenerate, and then show via game transformations that G_{rstr} and G_{rsymb} are identical up to *bad*, where the *bad* event is a collision between the input of any two f-rooted terms computed by the encryptor.

[4] We could indeed define histories as frames, but since in this work we have no need of the main feature of frames (that they are substitutions) we choose a simpler option.

In Sect. 5.2 we formally define the symbolic conditions, and show that, as long as the cryptosystem satisfies PBEC, then the probability of *bad* is negligible. We then use these results to derive security criteria for cryptographic modes of operation.

5.1 $MOO_{\oplus}$ Games G_{rstr} and G_{rsymb}

We begin by defining the games. We note that all four games are identical except for the method used to compute f. We will thus begin by describing a generic game G_{gen} which can be instantiated to any of the four games by choosing the appropriate method for computing f. Moreover, since the adversary's the advantage in distinguishing between G_{crypt}, G_{rperm}, and G_{rstr} can be estimated by assumption or known results from the literature, we will only describe in detail how f is computed in G_{rstr} and G_{rsymb}

G_{gen} proceeds in a series of steps. In each step the adversary sends the encryptor a sequence of symbolic terms $\mathbf{I}.\mathbf{O}$ and a set of λ-length bitstrings B, such that $\mathbf{I}$ is a (possibly empty) sequence of free variables $\mathbf{x_1}. \ \ldots \ .\mathbf{x_n}$, $\mathbf{O}$ is a sequence of terms, and $\mathbf{B}$ is a sequence of bitstrings $b_1 \ \ldots \ b_n$ such that $b_i = [\![\sigma x_i]\!]$, where σ is the substitution computed by the adversary. The encryptor checks whether the adversary is able to submit a request at this point, and if so, if this is a legal request to submit. If it is not, the game is aborted. If the request is valid, the encryptor returns the encrypted blocks specified by $\mathbf{O}$.

We now describe how the encryptors in G_{rstr} and G_{rsymb} work. In the following, we say "the encryptor" when both encryptors behave the same way. Otherwise we identify the encryptor as a G_{rstr} or a G_{rsymb} encryptor.

The encryptor maintains a symbolic history $\mathbf{H}$ describing its interaction with the adversary so far, as well as two databases that describe the output $[\![\sigma H]\!]$. The first database, DBI, stores the plaintext blocks sent by the adversary and the second, DBO, stores results of the random functions computed by the encryptor.

DBI consists of tuples of the form $[\mathbf{x}, [\![\sigma x]\!]]$, where $\mathbf{x}$ is a free variable, and $[\![\sigma x]\!]$ is the output of σx, where σ is the substitution computed by the adversary. DBO contains two types of tuples. The first are of the form $[\mathbf{r}, \mathit{outstr}]$, where $\mathbf{r}$ is a bound variable and outstr is $[\![r]\!]$. The second are of the form $[\mathit{instr}, \mathbf{F}, \mathit{outstr}]$, where $\mathbf{F}$ is a set of $\mathbf{f}$-rooted terms, and instr and outstr are λ-length bitstrings. The entries in this form of tuple are computed differently in G_{rstr} and G_{rsymb}. At the beginning, $\mathbf{H}$, DBO and DBI are all empty. They are extended by the encryptor as the protocol evolves.

We now describe the computations made by the encryptors. Suppose that an encryptor, whether in G_{rstr} or G_{rsymb}, receives an input $[\![\sigma I]\!]$ from the adversary and is required to return $[\![\sigma O]\!]$. It performs the following steps for each variable or $\mathbf{f}$-rooted term $\mathbf{t} = \mathbf{f}(\mathbf{s}) \in \mathit{Sub}(\mathbf{O})$ that has not been computed already, computing each bitstring $[\![\sigma s]\!]$ such that $\mathbf{s}$ is a proper subterm of $\mathbf{t}$ before computing $[\![\sigma t]\!]$.

1. If $\mathbf{I} = \mathbf{x_1}, \ldots, \mathbf{x_k}$, where the $\mathbf{x_i}$ are free variables, it stores $[\mathbf{x_i}, [\![\sigma x_i]\!]]$ in DBI.
2. For any bound variable $\mathbf{r} \in Sub(\mathbf{O})$ such that there is no tuple $[\mathbf{r}, str]$ in DBO, it chooses a λ-length bitstring str' uniformly at random and stores $[\mathbf{r}, str']$ in DBO.
3. For each $\mathbf{f}(\mathbf{t}) \in Sub(\mathbf{O})$ such that there is not already a tuple $[a, \mathbf{W}, b]$ in DBO with $\mathbf{f}(\mathbf{t}) \in \mathbf{W}$, the encryptor computes $instr$ using the outputs stored in DBI and DBO. That is, if

$$\mathbf{t} = (\bigoplus_{\mathbf{i=0}}^{\mathbf{m}} \oplus \alpha_{\mathbf{i}}\mathbf{x_i}) \oplus (\bigoplus_{\mathbf{i=1}}^{\mathbf{n}} \oplus \beta_{\mathbf{i}}\mathbf{r_i}) \oplus (\bigoplus_{\mathbf{i=1}}^{\mathbf{q}} \oplus \gamma_{\mathbf{i}}\mathbf{f}(\mathbf{s_i}))$$

   where the α_i, β_i, and γ_i are booleans, then for each $\alpha_{\mathbf{i}} \neq \mathbf{0}$ (respectively, $\beta_{\mathbf{i}} \neq \mathbf{0}$, $\gamma_{\mathbf{i}} \neq \mathbf{0}$) the encryptor finds the tuple $[\mathbf{x_i}, [\![\sigma x_i]\!]] \in DBI$ (respectively, $[\mathbf{r_i}, [\![r_i]\!]] \in DBO$, $[instr, \mathbf{W}, outstr] \in DBO$ such that $\mathbf{f}(\mathbf{s_i}) \in \mathbf{W}$), and computes the $\oplus$-sum of the final bitstrings of the tuples found. The result is bitstring $instr_0$ that will be used as the input to f.
4. If there is already a tuple $[instr, \mathbf{F}, outstr] \in DBO$ such that $\mathbf{f}(\mathbf{t}) \in \mathbf{F}$ then the encryptor takes no action. If not, G_{rstr} and G_{rsymb} differ as follows:
   (a) If there is already a tuple $[instr_1, \mathbf{F}, outstr_1] \in DBO$ such that $instr_0 = instr_1$, the G_{rstr} encryptor replaces it with $[instr_1, \mathbf{F} \cup \{\mathbf{f}(\mathbf{t})\}, outstr_1]$. Otherwise, it picks a random λ-length bitstring $outstr_0$ and stores the tuple $[instr_0, \{\mathbf{f}(\mathbf{t})\}, outstr_0]$ in DBO. $\sigma f(t)$ is defined to be $outstr_0$.
   (b) The G_{rsymb} encryptor picks a random λ-length bitstring $outstr_0$ and stores $[instr_0, \{\mathbf{f}(\mathbf{t})\}, outstr_0]$ in DBO. $\sigma f(t)$ is defined to be $outstr_0$.
5. The encryptor uses the output values stored in DBI and DBO to construct $[\![\sigma O]\!]$, and returns it to the adversary.

A pseudocode description of the top-level algorithm is given in Algorithm 1. Descriptions of the two implementations of f are given in Algorithms 2 and 3. Both of these make use of the subroutine given in Algorithm 4.

We note that the entire tuple $[instr, \mathbf{F}, outstr]$ is not necessary to compute f in either G_{rstr} or G_{rsymb}. G_{rstr} does not need the second entry in the tuple, and G_{rsymb} does not need the first. Their only purpose is to reduce the number of steps needed to transform the two games in a pair of identical-until-bad games. We make the real dependencies of these functions explicit in the following lemma.

Lemma 1. *Let f_{rstr} denote the function defined by $f_{rstr}(instr) = outstr$ if $[instr, \mathbf{F}, outstr] \in DBO$ in G_{rstr}, and let f_{rsymb} denote the function defined by $f_{rsymb}(\mathbf{f}(\mathbf{t})) = outstr$ if $[instr, \{\mathbf{f}(\mathbf{t})\}, outstr] \in DBO$ in G_{rsymb}. Then f_{rstr} is a random function from λ-bit strings to λ-bit strings, and f_{rsymb} is a random function from $\mathbf{f}$-rooted terms in $\mathbf{H}$ to λ-bit strings.*

Proof. This follows directly from the fact that, in the case of G_{rstr}, $outstr$ is chosen at uniformly at random when and only when a tuple $[instr, \mathbf{f}(\mathbf{t}), \_]$ needs to be added to DBO for a new string $instr$, and in the case of G_{rsymb}, $outstr$ is chosen uniformly at random when and only when a tuple $[instr, \{\mathbf{f}(\mathbf{t})\}, \_]$ is added to DBO for a new $\mathbf{f}$-rooted term $\mathbf{f}(\mathbf{t})$. $\square$

Algorithm 1. G_{gen}

```
1: ... Begin Setup ... $K \xleftarrow{\$} \{0,1\}^{\lambda}$ ... End Setup ...
2: Freevar, Bdvar $\subset$ Term
3: Integer: $i, n, \lambda$ ; String: $instr, outstr$ ; Term: $\mathbf{x}, \mathbf{t}, \mathbf{s}, \mathbf{u}, \mathbf{v}$ ; Set_of_Records:
   $DBI, DBO, a, a_0$; Boolean: $b, b', STOP$; Freevar: $\mathbf{x}, \mathbf{x_i}$, Bdvar: , $\mathbf{r}, \mathbf{r_i}$; Seq_of_Terms:
   $\mathbf{U_1}, \mathbf{U_2}, \mathbf{O}, \mathbf{O}', \mathbf{H}, \mathbf{W}$; Seq_of_Freevars: $\mathbf{I}$
4: $DBI, DBO \leftarrow \emptyset$; $\mathbf{H} \leftarrow []$; $i \leftarrow 0$; $b \leftarrow 0$ ; $STOP \leftarrow 0$
5: $a \leftarrow \emptyset$ {$a$ is the information the encryptor has sent to the adversary}
6: while $STOP \neq 1$ do
7:    $i \leftarrow i + 1$ ; $[\mathbf{I}.\mathbf{O}] \leftarrow \mathcal{A}(1^{\lambda}, a)$ {Adversary picks next tuple to execute}
      {Below, encryptor checks that adversary's request is valid}
8:    if $valid(\mathbf{H}, \mathbf{I}.\mathbf{O})$ then
9:       $\mathbf{H} \leftarrow \mathbf{H}.\mathbf{I}.\mathbf{O}$
10:      while $\mathbf{I} = \mathbf{x}.\mathbf{I}'$ do
11:         $outstr \leftarrow \mathcal{A}(1^{\lambda}, a)$ {Adversary sends plaintext block to encryptor}
12:         $DBI \leftarrow DBI \cup \{[\mathbf{x}, outstr]\}$ {Encryptor stores plaintext block in $DBI$}
13:         $\mathbf{I} \leftarrow \mathbf{I}'$
14:      end while
15:      $\mathbf{U_1} \leftarrow$ set of all bound variables in $sub(\mathbf{O})$ not yet computed
16:      $\mathbf{U_2} \leftarrow$ set of all $\mathbf{f}$-rooted terms in $sub(\mathbf{O})$ not yet computed
17:      while $\mathbf{U_1} \neq \emptyset$ do
18:         Choose $\mathbf{r} \in \mathbf{U_1}$ ; $outstr \xleftarrow{\$} \{0,1\}^{\lambda}$
19:         $DBO \leftarrow DBO \cup \{[0, \{\mathbf{r}\}, outstr]\}$
20:         $\mathbf{U_1} \leftarrow \mathbf{U_1} \setminus \{\mathbf{r}\}$
21:      end while
22:      while $\mathbf{U_2} \neq \emptyset$ do
23:         Choose a minimal element $\mathbf{f}(\mathbf{s})$ of $\mathbf{U_2}$ in the subterm partial order {We say
            $\mathbf{u} < \mathbf{v}$ in the subterm partial order if $\mathbf{u}$ is a proper subterm of $\mathbf{v}$.}
24:         $DBO \leftarrow$ Compute-f($\mathbf{f}(\mathbf{s}), \mathbf{DBO}$)
25:         $\mathbf{U_2} \leftarrow \mathbf{U_2} \setminus \{\mathbf{f}(\mathbf{s})\}$
26:      end while
27:      while $\mathbf{O} = \mathbf{t}.\mathbf{O}'$ do
28:         $a_0 \leftarrow Construct(\mathbf{t}, DBI \cup DBO)$
29:         $a \leftarrow a.a_0$ ; $\mathbf{O} \leftarrow \mathbf{O}'$
30:      end while
31:   end if
32:   $STOP \leftarrow \mathcal{A}(1^{\lambda}, a)$
33: end while
34: $b \leftarrow \mathcal{A}(1^{\lambda}, a)$
35: return $b$
```

Example 3. We can now use the G_{rstr} encryptor to model an attack on CBC when the blockwise schedule is used. In the blockwise schedule, each ciphertext block is sent by the encryptor as soon as it is able to compute it.

1. The adversary sends the encryptor a request for the initialization vector $\mathbf{r_0}$. The encryptor sets $\mathbf{H}$ equal to $\mathbf{r_0}$. The encryptor computes a random bitstring $outstr_0$, stores $[\{\mathbf{r_0}\}, outstr_0]$ in DBO, and sends $outstr_0$ to the adversary.

Algorithm 2. Subroutine Compute-f(f(s), DBI, DBO) (for G_{rstr})

```
1: if $\exists[instr, \mathbf{F} \cup \{\mathbf{f(s)}\}, outstr] \in DBO$ then
2:     $DBO \leftarrow DBO$
3: else
4:     $instr \leftarrow construct(\mathbf{s}, DBI, DBO)$
5:     if $\exists \mathbf{F}, outstr_0$ s.t. $[instr, \mathbf{F}, outstr_0] \in DBO$ then
6:         $DBO \leftarrow (DBO \setminus \{[instr, \mathbf{F}, outstr_0]\}) \cup \{[instr, \{\mathbf{f(s)}\} \cup \mathbf{F}, outstr_0]\}$
7:     else
8:         $outstr_1 \xleftarrow{\$} \{0,1\}^\lambda$ ; $DBO \leftarrow DBO \cup \{[instr, \}\{\mathbf{f(s)}\}, outstr_1]\}$
9:     end if
10: end if
11: return $(DBO)$
```

Algorithm 3. Subroutine Compute-f(**f(s)**, DBI, DBO) (for G_{rsymb})

```
1: if $\exists[instr, \{\mathbf{f(s)}\}, outstr] \in DBO$ then
2:     $DBO \leftarrow DBO$
3: else
4:     $instr \leftarrow construct(\mathbf{s}, DBI, DBO)$
5:     $outstr \xleftarrow{\$} \{0,1\}^\lambda$
6:     $DBO \leftarrow DBO \cup \{[instr, \{\mathbf{f(s)}\}, outstr]\}$
7: end if
8: return $(DBO)$
```

Algorithm 4. Subroutine Construct($\mathbf{s}, DBI \cup DBO$)

```
1: if s is a free variable x then
2:     Find $[\mathbf{x}, outstr] \in DBI]$ ; $c \leftarrow outstr$
3: else if s is a bound variable r then
4:     Find $[0, \mathbf{r}, outstr] \in DBO$ ; $c \leftarrow outstr$
5: else if s is an f-rooted term then
6:     Find $[instr, \mathbf{F}, outstr] \in DBO$ such that $\mathbf{s} \in \mathbf{F}$; $c \leftarrow outstr$
7: else if $\mathbf{s} = \bigoplus_{i=1}^k \oplus \mathbf{s_i}$, where each $\mathbf{s_i}$ is an f-rooted term or a free variable then
8:     $outstr \leftarrow construct(\mathbf{s_1}, DBO)$ ; $c \leftarrow outstr \oplus construct(\bigoplus_{i=2}^k \oplus \mathbf{s_i}, DBO)$
9: end if
10: return $c$
```

2. The adversary computes $[\![\sigma x_1]\!] = 0^\lambda$ and sends it to the encryptor, along with $\{\mathbf{x_1}\}.\{\mathbf{f}(\mathbf{x_1} \oplus \mathbf{r_0})\}$. The encryptor sets $\mathbf{H} = \mathbf{r_0}.\mathbf{x_1}.\mathbf{f}(\mathbf{x_1} \oplus \mathbf{r_0})$. It then stores $[\mathbf{x_1}, 0^\lambda]$ in DBI. Next it chooses a random bitstring $outstr_1$ and sets

$$[\![\sigma_1 f(x_1 \oplus r_0)]\!] = [\![f(0 \oplus r_0)]\!] = f(outstr_0) = outstr_1$$

and stores $[outstr_0, \{\mathbf{f}(\mathbf{x_1} \oplus \mathbf{r_0})\}, outstr_1]$ in DBO. It returns $[\![f(r_0)]\!] = outstr_1$ to the adversary.
3. The adversary computes $[\![\sigma x_2]\!] = [\![r_0\ ]\!] \oplus [\![\ f(r_0)]\!] = outstr_0 \oplus outstr_1$ and sends it to the encryptor, along with the sequence $\mathbf{x_2}.\mathbf{f}(\mathbf{x_2} \oplus \mathbf{f}(\mathbf{x_1} \oplus \mathbf{r_0}))$. $\mathbf{H}$ is updated by the encryptor as before. The encryptor computes the string

Algorithm 5. $\underline{G_{rstr}}$ and G_{rsymb} Algorithm $((\mathbf{f(s)}$, DBI, DBO)

```
1: if $\exists[instr, \mathbf{U} \cup \{\mathbf{f(s)}\}, outstr] \in DBO$ then
2:     $DBO \leftarrow DBO$
3: else
4:     $\mathbf{W} \leftarrow \emptyset$
5:     $instr \leftarrow construct(\mathbf{s}, DBI, DBO)$
6:     $outstr \xleftarrow{\$} \{0,1\}^{\lambda}$
7:     $outstr_0 \leftarrow outstr$
8:     if $\exists \mathbf{F}, outstr_1$ s.t. $[instr, \mathbf{F}, outstr_1] \in DBO$ then
9:         $bad \leftarrow 1$
10:        $\underline{outstr_0 \leftarrow outstr_1}$
11:        $\underline{\mathbf{W} \leftarrow \mathbf{F}}$
12:        $\underline{DBO \leftarrow (DBO \setminus \{[instr, \mathbf{F}, outstr_0]\})}$
13:    end if
14:    $DBO \leftarrow DBO \cup \{[instr, \mathbf{W} \cup \{\mathbf{f(s)}\}, outstr_0]\}$
15: end if
16: return $(DBO)$
```

$[\![\sigma(x_2 \oplus f(x_1 \oplus r_0))]\!] = [\![r_0]\!] \oplus [\![f(r_0)]\!] \oplus [\![f(r_0)]\!] = outstr_0 \oplus outstr_1 \oplus outstr_1 = outstr_0$. It finds the tuple $[outstr_0, \{\mathbf{f(x_1 \oplus r_0)}\}, outstr_1]$ in DBO, which it replaces with $[outstr_0, \{\mathbf{f(x_1 \oplus r_0)}, \mathbf{f(x_2 \oplus f(x_1 \oplus r_0))}\}, outstr_1]$, forcing a collision between the inputs to the two $\mathbf{f}$-rooted terms. It then sends $outstr_1$ to the adversary.

We note that such an attack is *not* possible when the G_{rsymb} encryptor is used. In that case the strings returned by the encryptor will be independently randomly generated, since the symbolic terms $\mathbf{r_0}, \mathbf{f(x_1 \oplus r_0)}$ and $\mathbf{f(x_2 \oplus f(x_1 \oplus r_0))}$ are all different. We now give the result that will be needed to prove our main theorem.

Lemma 2. *Algorithms 2 and 3 are equivalent to the identical-until-bad algorithms given in Algorithm 5.*

Proof. We first prove the result for Algorithm 3. Since lines 7 through 11 in Algorithm 5 result in no change once the underlined code is removed, we remove them. Once that is done, $\mathbf{W}$ is always empty, so we can remove line 4 and remove $\mathbf{W}$ from line 12. This gives us Algorithm 3.

For Algorithm 2, we open up a new "else" clause in Algorithm 5 and move statement 6 inside it. This is equivalent to Algorithm 5 because the statement is not used in the first clause. We then note that we can move line 14 inside both clauses of the "if" statement without changing the results. The rest is eliminating unnecessary assignments and variables, giving us Algorithm 2. □

By the fundamental lemma of game-playing [8], the adversary's advantage in distinguishing between the use of the f_{rstr} and f_{rsymb} functions in constructing $\mathbf{H}$ is bounded by the probability that *bad* is set to 1 in either of them. This only happens when $[\![\sigma u]\!] = [\![\sigma v]\!]$, where $\mathbf{f(u)}$ and $\mathbf{f(v)}$ are subterms of terms returned by the encryptor, so we will concentrate on estimating the probability of this occurring when the f_{rsymb} function is used.

5.2 Conditions Implying IND\$-CPA Security

In this section we define PBEC $MOO_{\oplus}$ cryptosystems. We also define two symbolic conditions that, together with PBEC, imply IND\$-CPA and use the games described in Sect. 5.1 to show that is the case. We first give some notation:

Definition 2. *Let* $\mathbf{H}$ *be a symbolic history,* $\mathbf{t}$ *a term, and* $\mathbf{x}$ *a free variable.*

1. *We say that* $\mathbf{H} \vdash_{\oplus} \mathbf{t}$ *if* $\mathbf{t}$ *can be derived by* $\oplus$*-summing a subset of* $\mathbf{H}$.
2. *We define* $\mathbf{H}[\mathbf{x}]$ *to be the sequence of terms exchanged between the adversary and the encryptor before the adversary sends* $\mathbf{x}$*, i.e.* $\mathbf{H} = \mathbf{H}[\mathbf{x}].\mathbf{x}.\mathbf{H}'$*, where* $\mathbf{x} \notin sub(\mathbf{H}[\mathbf{x}])$.
3. *We say that* $\mathbf{x} >_{\mathbf{H}} \mathbf{u}$ *if* $\mathbf{H}[\mathbf{x}] \vdash_{\oplus} \mathbf{u}$

Next, we define the two conditions.

Definition 3. *Let* $\mathbf{H}$ *be a symbolic history.*

1. *If* $\mathbf{f}(\mathbf{s})$ *and* $\mathbf{f}(\mathbf{t})$ *are terms in* $sub(\mathbf{H})$*, we say that* $\{\mathbf{f}(\mathbf{s}), \mathbf{f}(\mathbf{t})\}$ *is an* unsafe pair *if there is a free variable* $\mathbf{x}$ *such that* $\downarrow_{\oplus}(\mathbf{s} \oplus \mathbf{t}) =_{AC} \mathbf{x} \oplus \mathbf{u}$ *for some term* $\mathbf{u}$ *such that* $\mathbf{x} >_{\mathbf{H}} \mathbf{u}$*, and that* $\{\mathbf{f}(\mathbf{s}), \mathbf{f}(\mathbf{t})\}$ *is a safe pair otherwise. We say that a symbolic history is safe if it contains no unsafe pairs, and that it is safe otherwise. Finally, we say that a* $MOO_{\oplus}$ *program is safe if it admits only safe histories, and that it is unsafe otherwise.*
2. *We say that a symbolic history* $\mathbf{H}$ *is* degenerate *if there is a subset of* $\mathbf{H}$ *that* $\oplus$*-sums to 0. If a* $MOO_{\oplus}$ *program* $\mathcal{C}$ *admits a degenerate history, we say it is a degenerate program. If not, we say it is non-degenerate.*

Proposition 1. *1. Let* $\{\mathbf{f}(\mathbf{s}), \mathbf{f}(\mathbf{t})\}$ *be an unsafe pair in a symbolic history* $\mathbf{H}$*. Then there is an adversary that can compute a substitution* σ *such that* $[\![\sigma f(s)]\!] = [\![\sigma f(t)]\!]$ *with probability 1.*
2. No degenerate $MOO_{\oplus}$ *program is* IND\$-CPA*-secure.*

Proof. 1. By hypothesis, $\downarrow_{\oplus}(\mathbf{s} \oplus \mathbf{t}) = \mathbf{x} \oplus \mathbf{u}$, where $\mathbf{x} >_{\mathbf{H}} \mathbf{u}$. Since $[\![\sigma u]\!]$ has thus already been computed by the time $[\![\sigma x]\!]$ is computed, the adversary can thus choose σx such that $[\![\sigma x]\!] = [\![\sigma u]\!]$.
2. If there is a subset $\mathbf{T}$ of the terms returned by the encryptor in a symbolic history, such that $\sum_{\mathbf{t} \in \mathbf{T}} \oplus \mathbf{t} =_{\oplus} \mathbf{0}$, then for any substitution σ computed by the adversary, $\sum_{\mathbf{t} \in \mathbf{T}} \oplus [\![\sigma \mathbf{t}]\!] = \mathbf{0}$ as well. □

Definition 4. *Let* $\mathcal{C}$ *be a* $MOO_{\oplus}$ *program. Let* $\mathbb{H}_{\mathcal{C}}(n)$ *denote the set of all* $\mathcal{C}$ *symbolic histories of length* $\leq n$*. We say that* $\mathcal{C}$ *satisfies* polynomially bounded execution choice *(PBEC) if there is a polynomial* $p(n)$ *such that the number of different* f*-rooted subterms of* $\mathbb{H}_{\mathcal{C}}(n)$ *is* $\leq p(n)$ *for all* n.

Theorem 1. *Nondegenerate, safe, PBEC, $MOO_\oplus$ programs are* IND\$-CPA *secure.*

Proof. Let $\mathcal{C}$ be a $MOO_\oplus$ cryptosystem. The proof is by game transformation, starting with G_{crypt} (f is a block cipher), followed by G_{perm} (f is a random permutation), G_{rstr}, and then by G_{rsymb}. WLOG we assume that the adversary's computational bounds are the same for all games.

Advantage in Distinguishing G_{crypt} from G_{perm} and G_{perm} from G_{rstr}: By assumption, the adversary's advantage in distinguishing the block cipher from a random permutation is bounded by some negligible quantity ϵ_λ. In addition, it is known [8] that the adversary's advantage in distinguishing G_{perm} from G_{rstr} bounded by $q(\lambda) \cdot (q(\lambda) - 1) \cdot 2^{-\lambda-1}$, where $q(\lambda)$ is the maximum number of times the function f may be computed when the security parameter is λ. The maximum number of blocks exchanged between the adversary and the encryptor is bounded by a polynomial function ℓ of λ, and, $\mathcal{C}$ is PBEC, we have $q(\lambda)$ is bounded by $w(\ell(\lambda))$ where w is a polynomial. Thus $q(\lambda)$ is polynomially bounded, and hence so is $q(\lambda) \cdot (q(\lambda) - 1) \cdot 2^{-\lambda-1}$.

Probability of a Collision Between Two $\mathbf{f}$*-rooted terms in G_{rsymb}*: Safety implies that for any pair of terms $\mathbf{f}(\mathbf{s})$ and $\mathbf{f}(\mathbf{t})$, the G_{rsymb} adversary knows nothing about the potential value $[\![\sigma(s \oplus t)]\!]$ at the time it is computing σ on the free variables of $\mathbf{s}$ and $\mathbf{t}$, so $P([\![\sigma(s \oplus t)]\!] = 0)$ for any given $\mathbf{f}(\mathbf{s})$ and $\mathbf{f}(\mathbf{t})$ is $2^{-\lambda}$.

Advantage in distinguishing between G_{rstr} and G_{rsymb} : Suppose that the encryptor has already computed a set $\mathbf{A}$ of $\mathbf{f}$-rooted terms. Let $\mathbf{B}$ be the set of $\mathbf{f}$-rooted terms the adversary can potentially request to be evaluated. By hypothesis $\mid \mathbf{A} \mid \leq q(\lambda)$ and $\mid \mathbf{B} \mid \leq \ell(\lambda)$, where both are polynomially bounded. The number of possible new collisions offered by $\mathbf{B}$ is $\mid \mathbf{A} \mid \cdot \mid \mathbf{B} \mid + 1/2 \mid \mathbf{B} \mid \cdot (\mid \mathbf{B} \mid - 1)$, which is bounded by $q(\lambda) \cdot w(\ell(\lambda)) + \frac{1}{2} \cdot w(\ell(\lambda)) \cdot (w(\ell(\lambda)) - 1)$. Since the adversary has at most $\ell(\lambda)$ opportunities to request evaluated terms from the encryptor, the probability of two input strings being equal is bounded by

$$(\ell(\lambda) \cdot w(\ell(\lambda)) \cdot (q(\lambda) + \frac{1}{2} \cdot (w(\ell(\lambda)) - 1))) \cdot 2^{-\lambda}$$

Summing up the Results: Finally, we recall from Proposition 1 that in the non-degenerate case the output of the G_{rsymb} encryptor is random, so the adversary's advantage in distinguishing the G_{rsymb} encryptor from random is zero. Summing the remaining advantages in the sequences of games, we obtain an advantage of

$$\epsilon(\lambda) + q(\lambda) \cdot (q(\lambda) - 1) \cdot 2^{-\lambda-1} + (\ell(\lambda) \cdot w(\ell(\lambda)) \cdot (q(\lambda) + \frac{1}{2} \cdot (w(\ell(\lambda)) - 1))) \cdot 2^{-\lambda}$$

□

We now finish up with a result on cryptographic modes of operation. Although to the best of my knowledge there is no definition that precisely sets out the properties that modes must have, some conditions that are satisfied by most modes in the literature are given below.

Definition 5. *A MOO$_\oplus$ program is* well-behaved *if it satisfies the following properties:*

1. *The i'th term returned by the encryptor in any message is the i'th iteration of a deterministic recursive function;*
2. *for any single message, ciphertext blocks are returned according to a fixed schedule;*
3. *symbolic histories of different encrypted messages may be interleaved in an arbitrary fashion, and;*
4. *the number of new* $\mathbf{f}$*-rooted terms in any term returned by the encryptor is bounded by a constant D.*

Corollary 1. *Any safe, non-degenerate well-behaved MOO$_\oplus$ program is is* IND\$-CPA *secure.*

Proof. By Theorem 1 it is enough to show the program is PBEC, i.e. that for any n the set of f-rooted terms in $\mathbb{H}_{\mathcal{C}}(n)$ is bounded by a polynomial function of λ.

We call a symbolic history *unary* if it is a history the encryption of a single message, or an initial subsequence of such histories. Consider any length n interleaving of unary symbolic histories. We note that the symbolic terms sent in any such interleaving are the same no matter how the interleaving is done. We also note that at most n unary histories may be interleaved (the bound being achieved when the resulting symbolic history consists of n initial terms), and the length of any symbolic history may be at most n (the bound being achieved when a unary history of length n is used). Thus the number of different f-rooted terms in $\mathbb{H}_{\mathcal{C}}(n)$ is bounded by $D \cdot n^2$. Since the adversary is PPT, n is bounded by a polynomial function of λ. □

We illustrate these results with an example.

Example 4. Consider Cipher Feedback (CFB) mode, in which $C_0 = r$, and $C_i = f(C_{i-1}) \oplus x_i$ for $i > 0$, where r is a random block. Clearly, CFB is well-behaved. We show that CFB encryption is secure under the *blockwise schedule*, in which ciphertext is sent to the adversary as soon as it is computed. We note that no set of ciphertext terms $\oplus$ sums to 0, so CFB is nondegenerate. We now let $\mathbf{f}(\mathbf{C_{i,j}})$ and $\mathbf{f}(\mathbf{C_{\ell,k}})$ be two different f-rooted terms appearing in a symbolic history $\mathbf{H}$, where $\mathbf{C_{i,j}}$ denotes the i'th ciphertext block in the encryption of message j. Without loss of generality we may assume that $\mathbf{f}(\mathbf{C_{i,j}})$ appears after $\mathbf{f}(\mathbf{C_{\ell,k}})$ in $\mathbf{H}$. We note that $\mathbf{C_{i,j}} = \mathbf{f}(\mathbf{C_{i-1,j}}) \oplus \mathbf{x_{i,j}}$, and $\mathbf{f}(\mathbf{C_{i-1,j}}) \notin sub(\mathbf{H}_{\mathbf{x_{i,j}}})$. Thus, $(\mathbf{f}(\mathbf{C_{i,j}}), f(C_{\ell,k})$ is a safe pair. It follows that CFB is SBN-secure, and hence by from Corollary 1 it is IND\$-CPA secure.

6 Using Our Results to Analyze Modes

In this section we show how the results of this paper can be applied to the analysis of cryptorgraphic modes of operation. We begin by stating and proving some lemmas that allow us decide security by examining properties of single $\mathbf{f}$ rooted terms appearing in a symbolic history instead of properties of pairs of such terms.

Lemma 3. *Suppose that a mode $\mathcal{C}$ admits a history $\mathbf{H}$ containing a term $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ where $\mathbf{x} >_{\mathbf{H}} \mathbf{t}$. Then $\mathcal{C}$ is unsafe. Moreover, if $\mathbf{H} \vdash_{\oplus} \mathbf{f}(\mathbf{x} \oplus \mathbf{t})$, $\mathcal{C}$ is not* IND$-CPA.

Proof. Suppose that the first condition of the lemma holds. Then consider an adversary that runs $\mathbf{H}$ followed by $\mathbf{H}'$, where $\mathbf{H}'$ is identical to $\mathbf{H}$ except for different (both bound and unbound) variable names. Let $\mathbf{f}(\mathbf{x}' \oplus \mathbf{t}')$ be the copy of $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ computed in $\mathbf{H}'$. Then $\mathbf{x}' >_{\mathbf{H}} \mathbf{x} \oplus \mathbf{t} \oplus \mathbf{t}'$. If $\mathbf{H} \vdash_{\oplus} \mathbf{f}(\mathbf{x} \oplus \mathbf{t})$, then the adversary is able to derive $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ and $\mathbf{f}(\mathbf{x}' \oplus \mathbf{t}')$ from $\mathbf{H}.\mathbf{H}'$ and is able to both compute the substitution $\sigma x = x' \oplus t \oplus t'$ and observe that $\sigma f(x \oplus t) = \sigma f(x' \oplus t')$, thus distinguishing the two blocks from random. □

Lemma 4. *Let $\mathcal{C}$ be a well-behaved $MOO_{\oplus}$ program, such that for any $\mathbf{f}(\mathbf{s})$ appearing as a subterm of a term in some symbolic history $\mathbf{H}$, $\mathbf{s}$ has at least one $\mathbf{f}$-rooted or bound variable summand that does not appear in any term sent by the encryptor prior to computing $\mathbf{s}$. Then $\mathcal{C}$ is safe. If it is also non-degenerate, then it is* IND$-CPA *secure.*

Proof. Suppose $\mathcal{C}$ is not safe. Let $\mathbf{H}$ be an unsafe history, and let $\mathbf{f}(\mathbf{s}), \mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ be the first unsafe pair in $\mathbf{H}$, where $\mathbf{x} >_{\mathbf{H}} \downarrow_{\oplus}(\mathbf{s} \oplus \mathbf{t})$. The proofs in the case in which $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ is computed after $\mathbf{f}(\mathbf{s})$ and $\mathbf{f}(\mathbf{s})$ is computed after $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ are similar, so we prove only the first case. If $\mathbf{f}(\mathbf{x} \oplus \mathbf{t})$ is computed after $\mathbf{f}(\mathbf{s})$, then $\mathbf{t}$ must contain a summand $\mathbf{f}(\mathbf{w})$ that does not appear in any term sent by the encryptor prior to computing $\mathbf{t}$. But $\mathbf{f}(\mathbf{w})$ is not a summand of $\mathbf{s}$, it must be a summand of $\mathbf{s} \oplus \mathbf{t}$, contradicting our hypothesis. Thus, since $\mathbf{x} \not>_{\mathbf{H}} \mathbf{f}(\mathbf{w})$, we have $\mathbf{x} \not>_{\mathbf{H}} \mathbf{s} \oplus \mathbf{t}$.

The conclusion that, if $\mathcal{C}$ is also non-degenerate, then it is IND$-CPA secure, follows directly from Theorem 1. □

For modes, security often depends on the schedule in which ciphertext is returned to the adversary. The most conservative is the messagewise schedule, in which ciphertext blocks are not returned until all plaintext blocks have been received. The most eager is the blockwise schedule, in which the ciphertext blocks are returned as soon as they are computed. The delay-by-one schedule lies between the two: the ciphertext block encrypting the i'th plaintext block is returned after the $i + 1$'st block is received.

We now apply Lemmas 3 and 4 to some modes discussed by Bard [5]. They are listed in Table 1. The first column give the name of the mode. The references give where the mode was first described (if known), and where an attack was first described, if relevant. The second column describes the mode. The third column describes the most conservative schedule (if any) under which the mode is insecure, and the fifth the most eager schedule under which it is secure.

Table 1. Different modes and their security properties

Mode	Description	Insecure	Why	Secure	Why
CBC [13,18]	$C_0 = r$ $C_i = f(x_i \oplus C_{i-1})$	C_0 sent before x_1	Lemma 3 $x_1 >_H r$	1-delay	Lemma 4 C_{i-1} sent after x_i
PCBC	$C_0 = r$ $C_i = f(x_i \oplus x_{i-1} \oplus C_{i-1})$	C_0 sent before x_1	Lemma 3 $x_1 >_H r$	1-delay	Lemma 4 C_{i-1} sent after x_i
CFB	$C_0 = r$ $C_i = f(C_{i-1}) \oplus x_i$	none		bw	Lemma 4 $f(C_{i-1})$ new in C_i
OFB	$E_0 = f(r), E_i = f(E_{i-1})$ $C_i = E_i \oplus x_i$	none		bw	Lemma 4 $f(C_{i-1})$ new in C_i
IGE [10,15]	$C_0 = r, P_0 = r'$ $P_i = x_i$ $C_i = f(P_i \oplus C_{i-1}) \oplus P_{i-1}$	bw P_0 secret	Lemma 3 $x_i >_H C_{i-1}$	1-delay	Lemma 4 x_i sent after C_{i-1}
S-ABC [19]	$C_0 = r, E_0 = r'$ $E_i = x_i \oplus f(E_{i-1})$ $C_i = f(E_i \oplus C_{i-1}) \oplus E_{i-1}$	none		bw	Lemma 4 $f(E_i \oplus C_{i-1})$ new in C_i
H-IACBC [16,18]	$C_0 = E_0 = f(r), E_i = f(x_i \oplus (E_{i-1})$ $C_i = E_i \oplus S_i$ $S_i \oplus S_j =_{\oplus} S'_i \oplus S'_j$ any 2 mess. streams	C_0 sent before x_1	Lemma 3 plus identities on S_i	1-delay	Lemma 4 C_{i-1} sent after x_i

7 Conclusion and Open Problems

We have identified symbolic conditions sufficient to guarantee IND\$-CPA security for a class of cryptosystems that use exclusive-or. We used these results to generate conditions applicable to cryptographic modes of operation. Our approach relies on the use of identical-until-*bad* games that allow us to pinpoint a property implying a cryptosystem's security. Even though this property (in this case, negligible probability of collisions) is not always preserved in the computational model, it is possible to determine further conditions that guarantee that it will be, avoiding known problems with the computational soundness of exclusive-or.

This work opens up several directions for future research.

Verification of Symbolic Criteria: The first is to develop algorithms for checking the symbolic criteria developed in this paper, and for generating cryptosystems that meet them. We have already begun work on developing verification algorithms and a tool for generating and checking cryptosystems (https://symcollab.

github.io/CryptoSolve/). In addition, we have begun to study the complexity of the problem. In particular, in [21]it is shown that degeneracy is undecidable by reducing it to the Post correspondence problem.

Enriching the Set of Cryptographic Primitives and Properties: Block ciphers and exclusive-or are not the only cryptographic primitives that can be reasoned about symbolically. Some others include hash functions, modular exponentiation, finite field and Abelian group operations, and bilinear pairings. Computational interpretations of the symbolic versions of many of these have already been covered by Baudet et al. in [7] for the case of a passive adversary, but may still need to be adapted for the case of an adaptive one. For properties, we would be interested not only in secrecy properties but properties such as authentication, collision-freeness and so forth.

Weaker Properties: Several of the properties we use in this paper are stronger than strictly necessary. In particular, polynomially bounded execution choice-limits the adversarial choices, but although $MOO_{\oplus}$ programs satisfy it, all that is really required is that the adversary's *adaptive* choice be polynomially limited. Indeed, this is the case for the cryptosystem used as an example by Kremer and Mazaré in [20]. The adversary first non-adaptively compromises a proper subset of the principals in a network, and then interacts with the remaining principals in a polynomially bounded execution choice fashion. Counting opportunities for adaptive execution choice is more complex than counting opportunities for execution choice, but identifying design choices such that make such counting easier may make the analysis more tractable.

In addition, the safety property defined in this paper is stronger than is strictly necessary: it is possible for a cryptosystem to have internal collisions that are not visible to the adversary. For example, consider the symbolic history $\nu\mathbf{r}.\mathbf{r}.\mathbf{x_1}.\mathbf{x_2}.\mathbf{f}(\mathbf{r} \oplus \mathbf{f}(\mathbf{x_1}) \oplus \mathbf{f}(\mathbf{x_2}))$.[5] The pair $(\mathbf{f}(\mathbf{x_1}), \mathbf{f}(\mathbf{x_2}))$, and the adversary can force a collision by setting $\mathbf{x_1} = \mathbf{x_2}$. However, in that case the ciphertext $\mathbf{r}.\mathbf{f}(\mathbf{r})$ is still indistinguishable from random. But the technique used to prove IND\$-CPA for safe cryptosystems does not extend easily to cryptosystems whose only unsafe pairs are invisible, because safeness or unsafeness of a pair may now depend on where it is located with respect to other $\mathbf{f}$-rooted terms in a symbolic history.

Design and Analysis of Protocols: Although the work we describe is applied to cryptosystems, they are cryptosystems that are themselves simple protocols, whose security depends on communication rules governing the schedule used by the encryptor to return ciphertext. Thus the question of whether or not these results can be extended to more complex protocols is more a question of to what degree than of whether it is possible at all. Thus it may be possible to augment previous work on computationally sound symbolic cryptographic protocol analysis, which has employed more of a top-down approach in which algorithms are modeled as equation-free abstractions, with a more bottom-up approach in which one starts with equational theories that are more tightly connected to the computational model. However, the question of scale is not a

[5] I am grateful to Christopher Lynch for this example.

trivial one, in particular, since, as Unruh has shown, providing too much adaptive choice to an adversary can break the link between symbolic and computational security. This question will need to be studied in more depth.

Acknowledgements. I am grateful to Jonathan Katz for helpful comments on earlier versions of this paper, as well as for the comments of many anonymous reviewers. I am also grateful the fruitful discussions I have had with fellow members of the Crypto-Solve group, including Wei Du, Serdar Erbatur, Hai Lin, Christopher Lynch, Andrew Marshall, Paliath Narendran, Danielle Radosta, Veena Ravishankar, Brandon Rozek, and Luis Rovera. Any problems that remain with this paper are of course my own responsibility. This work was funded by ONR Code 311.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL 2001, pp. 104–115. ACM (2001)
2. Abadi, M., Rogaway, P.: Reconciling two views of cryptography. In: van Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P.D., Ito, T. (eds.) TCS 2000. LNCS, vol. 1872, pp. 3–22. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44929-9_1
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
4. Backes, M., Pfitzmann, B.: A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 1–12. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24597-1_1
5. Bard, G.V.: Blockwise-adaptive chosen-plaintext attack and online modes of encryption. In: Galbraith, S.D. (ed.) Cryptography and Coding 2007. LNCS, vol. 4887, pp. 129–151. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77272-9_9
6. Barthe, G., et al.: Fully automated analysis of padding-based encryption in the computational model. In: ACM Conference on Computer and Communications Security (2013)
7. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_53
8. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_25
9. Bresson, E., Lakhnech, Y., Mazaré, L., Warinschi, B.: A generalization of DDH with applications to protocol analysis and computational soundness. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 482–499. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_27
10. Campbell, C.: Design and specification of cryptographic capabilities. IEEE Commun. Soc. Mag. **16**(6), 15–19 (1978)
11. Carmer, B., Rosulek, M.: Linicrypt: a model for practical cryptography. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 416–445. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_15

12. Dolev, D., Yao, A.C.-C.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–207 (1983)
13. Ehrsam, W.F., Meyer, C.H., Smith, J.L., Tuchman, W.L.: Message verification and transmission error detection by block chaining, US Patent 4,074,066, 14 February 1978
14. Gagné, M., Lafourcade, P., Lakhnech, Y., Safavi-Naini, R.: Automated proofs of block cipher modes of operation. J. Autom. Reason. **56**(1), 49–94 (2016). https://doi.org/10.1007/s10817-015-9341-5
15. Gligor, V.D., Donescu, P., Katz, J.: On message integrity in symmetric encryption. In: 1st NIST Workshop on AES Modes of Operation (2000)
16. Halevi, S.: An observation regarding Jutla's modes of operation. IACR Cryptol. ePrint Arch. 2001:15 (2001)
17. Hoang, V.T., Katz, J., Malozemoff, A.J.: Automated analysis and synthesis of authenticated encryption schemes. In: Proceedings of 22nd ACM CCS, pp. 84–95 (2015)
18. Joux, A., Martinet, G., Valette, F.: Blockwise-adaptive attackers revisiting the (in)security of some provably secure encryption modes: CBC, GEM, IACBC. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 17–30. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_2
19. Knudsen, L.R.: Block chaining modes of operation. In: Proceedings of Symmetric Key Block Cipher Modes of Operation Workshop (2000)
20. Kremer, S., Mazaré, L.: Adaptive soundness of static equivalence. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 610–625. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74835-9_40
21. Lin, H., et al.: Algorithmic problems in the symbolic approach to the verification of automatically synthesized cryptosystems. In: Konev, B., Reger, G. (eds.) FroCoS 2021. LNCS, vol. 12941, pp. 253–270. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-86205-3_14
22. Malozemoff, A.J., Katz, J., Green, M.D.: Automated analysis and synthesis of block-cipher modes of operation. In: 2014 IEEE 27th Computer Security Foundations Symposium (CSF), pp. 140–152. IEEE (2014)
23. McQuoid, I., Swope, T., Rosulek, M.: Characterizing collision and second-preimage resistance in Linicrypt. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 451–470. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_18
24. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_8
25. Unruh, D.: The impossibility of computationally sound XOR. IACR Cryptology ePrint Archive, 2010:389 (2010)