Block Ciphers in Idealized Models: Automated Proofs and New Security Results

Miguel Ambrona
Midnight
Madrid, Spain
miguel.ambrona@iohk.io

Pooya Farshim
IOG
Zurich, Switzerland
Durham University
Durham, United Kingdom
pooya.farshim@gmail.com

Patrick Harasser
Cryptoplexity
Technische Universität Darmstadt
Darmstadt, Germany
patrick.harasser@tu-darmstadt.de

Abstract

We develop and implement AlgoROM, a tool to systematically analyze the security of a wide class of symmetric primitives in idealized models of computation. The schemes that we consider are those that can be expressed over an alphabet consisting of XOR and function symbols for hash functions, permutations, or block ciphers.

We implement our framework in OCaml and apply it to a number of prominent constructions, which include the Luby–Rackoff (LR), key-alternating Feistel (KAF), and iterated Even–Mansour (EM) ciphers, as well as substitution-permutation networks (SPN). The security models we consider are (S)PRP, and strengthenings thereof under related-key (RK), key-dependent message (KD), and more generally key-correlated (KC) attacks.

Using AlgoROM, we are able to reconfirm a number of classical and previously established security theorems, and in one case we identify a gap in a proof from the literature (Connolly et al., ToSC'19). However, most results that we prove with AlgoROM are new. In particular, we obtain new positive results for LR, KAF, EM, and SPN in the above models. Our results better reflect the configurations actually implemented in practice, as they use a single idealized primitive. In contrast to many existing tools, our automated proofs do not operate in symbolic models, but rather in the standard probabilistic model for cryptography.

Keywords

Automated Proofs, Idealized Models, Luby–Rackoff, Key-Alternating Feistel, Even–Mansour, Substitution-Permutation Network.

1 Introduction

Our community's understanding of cryptographic constructions has reached a level of maturity where large parts of the design and analysis of schemes can potentially be automated. Such automation, if possible, would address a number of shortcomings of the traditional pen-and-paper-based approach. First, security proofs can be error-prone or hard to carry out (or even verify) by hand. This, in turn, may lead authors to work towards weaker results, and introduce unnecessary assumptions to simplify analyses.[1] Second, small tweaks to a construction typically require redoing security proofs "from scratch," which of course is a tedious task. Furthermore, given the richness of design spaces, automation would allow researchers to focus their (creative) efforts on studying a wider class of schemes, leading, for instance, to improvements in efficiency. Automated tools can act as a sanity check and a first filter to assist designers in their search for good schemes. Finally, any future enhancements to a tool, such as the class of schemes covered, proof techniques that are automated, or formal verification of underlying components, translates into added trust in the analyzed schemes.

Mechanized design and analyses have appeared in the literature with various degrees of automation. Examples include the analysis of the hardness of group-based assumptions [3, 5, 6, 12, 14], symbolic analysis of cryptosystems [31, 59], automated discovery of attacks [63], and automated synthesis [13, 48, 57]. Most of these works, however, use symbolic methods, which require extra effort to show that they lead to actual computational security [1, 15]. Additionally, some of these symbolic models (e.g., standard XOR-theories) are known to be incompatible with computational soundness, at least for an unbounded number of symbolic operations [66]. We refer to Section 1.4 for further discussion of related works.

1.1 Our contributions

In this paper we continue the aforementioned lines of research. We develop and implement AlgoROM, an automated analysis tool for symmetric cryptographic schemes in a number of idealized models, including the random-oracle, random-permutation, and ideal-cipher models. The novelty of our work lies in *automating* game-playing proofs for a wide class of schemes: AlgoROM automatically determines the relevant bad events which one has to identify when applying the game-playing technique "by hand" (and allow transitioning to an appropriate target game), and then analyzes their probabilities. The latter is done via a subroutine called Collider, which is our core algorithmic contribution. Remarkably, the methodology we develop leads to *computational* security.

We then apply AlgoROM to a wide range of constructions, spanning hash functions and block ciphers, in a variety of security models. We reconfirm a number of existing results, and in one case find a flaw in prior work [30]. Most of our results, however, are new: Compared to the state of the art, we establish security both of more practical configurations that have not yet been studied, and with respect to stronger security models. As we discuss below, this is likely due to the overwhelming effort needed to carry out full proofs by hand. We refer to Section 6 for a presentation of selected security results that we obtain using our tool, and next give an overview of our methodology and algorithmic contributions.

[1] For example, a number of existing works study the security of permutation-based constructions with respect to independent round permutations, whereas the use of a single permutation more closely conforms to practice. The analyses of the latter are often more challenging, involving many more case distinctions.

1.2 Proof framework

Idealized models. Idealized models of computation are a standard methodology to render proofs of security tractable and assess the soundness of cryptographic schemes, especially those that are simple, efficient, and deployed in practice. They are widely used in provable security, ranging from block ciphers and hash functions to public-key encryption and more advanced primitives such as succinct non-interactive arguments of knowledge (SNARKs). Three such models that we consider in this work are the random-oracle (RO), random-permutation (RP), and ideal-cipher (IC) models. In the RO model all parties are given oracle access to a uniformly random function on a given domain and range, while in the RP model (resp., the IC model) this access is to a (keyed) random permutation and its (keyed) inverse. When proving schemes secure in these models, evaluations of some specific function (resp., permutation or block cipher) appearing in the scheme are replaced by invocations to the corresponding ideal object. Notable examples of constructions that have been studied in these models include the key-alternating Feistel (KAF) [53, 55] and iterated Even–Mansour (EM) ciphers [37], substitution-permutation networks (SPN) [28, 34], and compression functions [47, 62].

Schemes. The class of schemes that we consider fall under a language involving constants, variables, XOR-ing of expressions, applying a function symbol (and possibly its inverse) to an expression, and concatenation/projection of expressions. The schemes mentioned above, as well as many others, can all be expressed in this language. Our tool contains the necessary procedures to manipulate such expressions.

Security models. The security models that we consider are also those that can be expressed in the XOR-language above. For block ciphers we investigate the standard goals of (strong) PRP-security (which we call CPA and CCA for notational uniformity), where outputs are required to be indistinguishable from random when computed using a random secret key. We then consider CPA/CCA-security extended to related-key (RK) attacks, where an adversary can see evaluations on offsets of the secret key. We also study security on key-dependent (KD) inputs [46], where block cipher inputs can depend on the key via adversarially chosen offsets. Finally, we consider a combined RK and KD model, the so-called key-correlated (KC) model [23, 30].

The security notions above are standard and widely studied. CPA and CCA are of course basic security requirements. RK-security is a popular notion (especially under offsets), and is relevant both from a constructive perspective (e.g., for building tweakable ciphers, where it is assumed that a construction behaves independently on keys that are offsets of each other [54], or in practical protocols such as 3GPP) and from a cryptanalytic perspective (e.g., in the context of fault injection attacks [8, 20]). KD-security is relevant, for example, in the context of full disk encryption, where the encryption key (on HDD) itself gets encrypted. KC-security is a more recent (and stronger) combined notion; it has applications, for example, in MPC and the garbling of XOR gates via the free-XOR approach [9].

Security proofs. We now outline our approach to proving security of the constructions above in these security models. Recall that a forgetful random oracle (FRO) is an oracle that returns independent random outputs regardless of its inputs, thus not respecting functional consistency. FROs have been extensively used in provable security; we refer to Shoup's tutorial [65] for instructive examples. A core first step in our tool is to systematically replace a sufficient number of invocations of the underlying ideal primitives in a scheme with FROs, with the goal that its outputs are fully randomized and hence indistinguishable from those of its ideal counterpart. In more detail, the steps taken by AlgoROM are as follows.

Oracle replacement: AlgoROM will exhaustively search over all possible replacements of oracle gates with FROs, until one that randomizes the construction is found. This replacement is applied to all interfaces of the scheme, which for block ciphers are the encryption and decryption algorithms, and for hash functions the evaluation procedure. We emphasize that only oracle gates in the scheme may be replaced; direct oracles offered to the adversary are never changed. We let G_0 be the security game for the original scheme in the real world, and G_1 be the security game for the scheme with replaced oracle gates. Eventually, our goal is to bound the difference in winning games G_0 and G_1.

Randomization: We need to prove that the oracle outputs in game G_1 are uniform. For this, we formulate a simple game G_1' where the interfaces of the scheme return forgetful random outputs. (Note that this is now done for the entire interfaces, and not for single oracle gates.) To show that G_1 and G_1' are close, we apply a randomization detection procedure that expresses the randomness used by the FROs in terms of the randomness contained in the scheme outputs, thus showing that they are in one-to-one correspondence. This can be carried out via simple algebraic manipulations; see Section 2 for an example.[2]

Switching: In the notions that we consider, one side of the security game typically offers oracles implementing random functions or ciphers. In the latter case, we rely on the standard RP/RF switching lemma to argue that forgetful random outputs are close to those of an ideal cipher, as long as there are no repeated queries. This step is independent of the details of the construction; it is proven once and for all, and reused across different analyses.

Indistinguishability until bad: We next bound the difference between G_0 and G_1. To do so, we identify a set of bad events until which the two games are identical. We consider the oracles in the two games implemented via lazy sampling (to enable code-based analysis). That is, oracles are realized via tables and, when asked a query not yet in the domain of the table, they choose a correctly distributed reply and store it in their table before returning it. FROs work analogously, but do not store the sampled outputs. A difference in the construction outputs of G_0 and G_1 then arises when an oracle in game G_0 uses an entry in a table that is not stored by FRO in G_1. This translates to the bad event where the adversary queries a replaced oracle on an input which is also queried at a different point: If that happens, G_0 would return a consistent answer, whereas G_1 would answer independently of outputs generated so far. AlgoROM enumerates all such bad events corresponding to a repeated query to a replaced oracle. In more detail, AlgoROM computes the expressions for

[2] Looking into existing literature, a natural step here would be to use tools from static equivalence [27, 52]. This, however, introduces a symbolic step in the overall analysis, which we aim to avoid.

the queries corresponding to the two colliding queries, which are then equated to form the winning condition of a collision game. These collision games are represented in an appropriate format so they can be analyzed later. When the model offers encryption and decryption oracles, the order of the queries is important. Accordingly, AlgoROM needs to consider separate collision games for the cases where both queries are to encryption, the first is an encryption and the second a decryption query or vice versa, both are decryption queries, the first is an encryption and the second a primitive query, and so on. Further care is needed when dealing with invertible primitives: The bad event corresponds to an entry which would have been used in G_1 if the oracle had not been replaced, and this entry could arise as a result of a forward or a backward query.

Collision analysis: We rely on the fundamental lemma of code-based game-playing [18] and conclude by bounding the probability of the bad events above. We do this in G_1 (which has a simpler structure), where we can allow for an unbounded number of queries to the construction oracle(s), because their replies are random and can be simulated. (This is unlike symbolic tools, and the result of automating standard cryptographic proofs.) Note that the number of primitive queries is still bounded, as the winning condition involves a bounded number of primitive queries (the construction only makes a bounded number of queries), and replies to the remaining queries can be assumed random and independent of the collision game. This step is performed by our Collider procedure, which forms our main algorithmic contribution. We now take a closer look at this procedure.

1.3 Collision analysis

The types of collision games that arise in our analysis consist of a sequence of steps, in each of which the stateful adversary $\mathcal{A}^{\mathsf{F}}$ with oracle access to an ideal primitive F either outputs an element X_i, or receives an element C_j, all of which are in the XOR-language. The order in which the adversary and the game exchange these elements is encoded by a partial order σ. The adversary wins the game if the elements X_i and C_j satisfy a set of equalities and inequalities that may use F. These (in)equalities will always be in the XOR-language, since they arise from considering constructions that are themselves within this language.

Deciding whether or not such collision games can be won is performed via the Collider procedure of AlgoROM, and is one of our main contributions. Collider starts with a collision game and progressively refines it to one or more games until either a solution is found, or it is declared that no solution exists for any of the games. These refinements are performed through the application (in any order) of a number of rules, which we briefly discuss below.

Peeling: This rule identifies the set of queries X to ideal primitives in the current set of (in)equalities defining the collision game. The answer $\mathsf{F}(X)$ to query X to F is replaced by a fresh random variable C, and the dependency $X <_\sigma C$ is introduced to encode that query X must correspond to a game step that precedes the one where the adversary learns C. The rule is sound because outputs of F are unpredictable and independent, and thus if $\mathsf{F}(X)$ appears in the winning condition of the game, query X must come as part of one of the queries of the adversary.

Assignment: If an adversarially chosen variable X must satisfy an expression in terms of some variables, all of which appear prior to X, then this rule fixes X in terms of those variables.

Contradiction: If the equations defining a game contain a contradiction (encoded as $0 \neq 0$), then the game cannot be won.

Unpredictability: This rule uses the fact that if an unpredictable variable C must satisfy an expression in terms of some set of variables, all of which appear prior to C (i.e., C is "out of the adversary's control"), then the game is unwinnable, except with negligible probability.

Partitioning: For each pair of variables $V, W \in \{X_i, C_j\}$, this rule considers two sub-cases, one where they are the same (i.e., with unified variables $V \mapsto W$), and one where they are distinct (by adding a new inequality $V \neq W$).

Ordering: For two variables $V, W \in \{X_i, C_j\}$ that are are labeled as different and not already ordered, this rule considers two subcases consisting of ordering the variables in the two possible ways $V <_\sigma W$ and $W <_\sigma V$.

The formal semantics for the rules above can be found in Fig. 13, and a more detailed discussion in the proof of Lemma 5.2. For a concrete example showing how these rules are applied, see Section 5.

To prove termination of Collider, we associate a 4-tuple of positive integers to its state, and show that each rule results in at least one of the components strictly decreasing. Termination then follows from the fact that these state values must all be positive. For completeness (i.e., no solution is missed), notice that whenever Collider outputs false, the input game is indeed not solvable as either a contradiction has been reached or an unpredictable value is guessed (note that no solution is removed by the intermediate steps). Soundness (i.e., no new solutions are introduced) follows from the fact that the final solution returned by Collider corresponds to an adversary that can win the initial game: All free variables are instantiated randomly, and the remaining ones by enforcing the given equations. By the union bound, such an assignment also satisfies the inequalities involved with high probability.

Related keys and key-dependent inputs. To prove security in CCA models, AlgoROM checks if an encryption (resp., decryption) query is trivial in the sense that it was an output of decryption (resp., encryption), and if so does not consider it as constituting a valid pair of queries that can trigger a bad event. In the KD and KC security models, we allow the adversary to also pick key-dependent messages (and in the KC model also ciphertexts) of the form $\sum_{i=1}^{r} \alpha_i K_i \oplus \Delta$ for $\alpha_i \in \{0, 1\}$, where K_i are components of the key, and Δ is an adversarially chosen offset. For the RK and KC models we also allow the adversary to offset keys in the form $K \oplus \Delta$. Accordingly, collision games in these models are considered with respect to related keys and/or key-dependent inputs. AlgoROM considers each of the 2^r possibilities for the α_i, and carries out the analysis with respect to an arbitrary chosen Δ.

1.4 Related works

Malozemoff et al. [57] focus on automatic synthesis of IND$-CPA-secure encryption schemes. They view a synthesized construction as a graph, and show that if it has a "valid labeling," then the construction is IND$-CPA-secure. Their approach is to reduce security to a problem which is then fed to an SMT solver.

Hoang et al. [48] generate thousands of AE schemes with provable security guarantees, in particular new competitive variants of OCB and CCM.

Gagné et al. [42] use compositional Hoare logic to analyze encryption schemes. They show that if each operation in a scheme updating a distribution over states of the program remains indistinguishable from uniform, then the scheme is secure.

In the Linicrypt framework of Carmer and Rosulek [25], algorithms make calls to a random oracle and otherwise manipulate values via fixed linear operations. McQuoid et al. [58] build on this framework and study the collision-resistance and second-preimage resistance for Linicrypt programs, as long as they achieve domain separation on their random-oracle queries.

Meadows [59] considers the IND$-CPA security for symmetric schemes built out of modes of operation. This work identifies a syntactically checkable sufficient condition for IND$-CPA security.

Barthe et al. [11] introduce logics for proving CPA and CCA-security of padding-based public-key encryption schemes, and automated methods for finding attacks against these notions. They also develop ZooCrypt, a toolset implementing these methods.

In this work, we automate the standard game-hopping proof technique, and our results hold in cryptographic idealized models. We prove security of a wider set of cryptographic primitives under a variety of notions. In particular, we are able to prove new security results, in an automated way, about existing schemes.

1.5 Paper outline

To illustrate the steps taken by AlgoROM to prove security, we present in Section 2 the analysis of the CPA-security of a 3-round KAF. In Section 3 we recall the formal definition of our reference XOR-theory, the well-known class of constructions that fall under it, and the security models that we consider (which also fall within the XOR-theory). In Section 4 we provide the details of our proof framework, focusing on oracle replacement and generating (non-normal) collision games. Section 5 contains the details of our Collider procedure. We present our experimental results in Section 6, and conclude with future directions in Section 7.

2 A Concrete Example

To exemplify the steps of our algorithm, we present the CPA-security analysis of the 3-round KAF with distinct round functions F_1, F_2, and F_3, key K, and key schedule $[K, 0, K]$,[3] as performed by AlgoROM. For formal definitions of this construction and security notion, please see Section 3. On input $L \mid R$, the output of the encryption procedure is $C_1 \mid C_2$, where

$$C_1 = R \oplus \mathsf{F}_2(L \oplus \mathsf{F}_1(R \oplus K))\,, \quad \text{and}$$
$$C_2 = L \oplus \mathsf{F}_1(R \oplus K) \oplus \mathsf{F}_3\big(R \oplus K \oplus \mathsf{F}_2(L \oplus \mathsf{F}_1(R \oplus K))\big)\,.$$

The queries made to the round functions when computing $C_1 \mid C_2$ are

$$Q_{1\mathrm{e}}(K, L \mid R) = R \oplus K\,, \qquad Q_{2\mathrm{e}}(K, L \mid R) = L \oplus \mathsf{F}_1(R \oplus K)\,, \quad \text{and}$$
$$Q_{3\mathrm{e}}(K, L \mid R) = R \oplus K \oplus \mathsf{F}_2(L \oplus \mathsf{F}_1(R \oplus K))\,.$$

[3] This means that K, 0, and K are XOR-ed into the inputs of the first, second, and third round functions F_1, F_2, and F_3, respectively.

Game CG:
$K, \$_1, \$_2 \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $L_1 \mid R_1 \twoheadleftarrow \mathcal{A}^{\mathsf{F}_1,\mathsf{F}_2,\mathsf{F}_3}$
$L_2 \mid R_2 \twoheadleftarrow \mathcal{A}^{\mathsf{F}_1,\mathsf{F}_2,\mathsf{F}_3}(R_1 \oplus \$_2 \mid L_1 \oplus \mathsf{F}_1(R_1 \oplus K) \oplus \$_3)$
return $\big((Q_{2\mathrm{e}}(K, L_1 \mid R_1) = Q_{2\mathrm{e}}(K, L_2 \mid R_2))$
$\quad \wedge (L_1 \mid R_1 \neq L_2 \mid R_2)\big)$

Figure 1: The (2e, 2e) **collision game of 3-round KAF analysis.**

Replacements. There are 8 possible replacements of round functions with forgetful random oracles (FROs). These correspond to replacing F_i for all i in $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, or $\{1,2,3\}$. (The last case, for example, corresponds to replacing all F_i with FROs.) AlgoROM tries these one by one, and for each forms the associated randomization and collision games. It first checks randomization and, if successful, goes on to analyze the corresponding collision games. If randomization fails for a candidate replacement, that replacement is discarded and the next one is examined.

Randomization. To exemplify this step, we first consider the candidate replacement $\{3\}$, i.e., we let F_3 be replaced by an FRO. The output of encryption on $L \mid R$ is now the pair $C_1' \mid C_2'$, where

$$C_1' = R \oplus \mathsf{F}_2(L \oplus \mathsf{F}_1(R \oplus K))\,, \qquad C_2' = L \oplus \mathsf{F}_1(R \oplus K) \oplus \$_3\,,$$

and $\$_3$ is the output of the FRO in round 3. It is easy to see that this replacement does not result in full randomization of the outputs, because for fixed L, R, K, F_1, and F_2, the output part C_1' is completely determined. Therefore, AlgoROM will discard this replacement.

Now suppose that the next candidate replacement examined by AlgoROM is $\{2,3\}$, i.e., we let F_2 and F_3 be replaced by FROs. The output of encryption on $L \mid R$ is now the pair $C_1'' \mid C_2''$, where

$$C_1'' = R \oplus \$_2\,, \qquad C_2'' = L \oplus \mathsf{F}_1(R \oplus K) \oplus \$_3\,,$$

and $\$_2$ and $\$_3$ are the outputs of the FROs in rounds 2 and 3. Observe that the construction is now randomized: We can rewrite the equations above as

$$\$_2 = R \oplus C_1''\,, \qquad \$_3 = L \oplus \mathsf{F}_1(R \oplus K) \oplus C_2''\,,$$

and conclude that for fixed L, R, K, and F_1, the output ciphertexts and the randomness of FROs are in one-to-one correspondence. In other words, this replacement results in full randomization. AlgoROM will thus proceed with this replacement.

Collision games. Next, AlgoROM constructs the collision games for the replacement $\{2,3\}$. These correspond to collisions in queries

$$(2\mathrm{p}, 2\mathrm{e})\,, \quad (2\mathrm{e}, 2\mathrm{p})\,, \quad (3\mathrm{p}, 3\mathrm{e})\,, \quad (3\mathrm{e}, 3\mathrm{p})\,, \quad (2\mathrm{e}, 2\mathrm{e})\,, \quad (3\mathrm{e}, 3\mathrm{e})\,,$$

where 2p means a primitive query to F_2 and 2e means a query to F_2 as a result of an encryption query, and similarly for 3p and 3e.

As an example, the collision game for (2e, 2e) is given in Game CG of Fig. 1. The winning condition is:

$$L_1 \oplus \mathsf{F}_1(R_1 \oplus K) = L_2 \oplus \mathsf{F}_1(R_2 \oplus K) \quad \wedge \quad L_1 \mid R_1 \neq L_2 \mid R_2\,.$$

Furthermore, variables L_1, R_1, L_2, and R_2 must be chosen before K is known (we assume w.l.o.g. that all random variables are revealed to the adversary at the end of the game). This is denoted by $L_1, R_1, L_2, R_2 <_\sigma K$, where σ is a partial order on the game variables.

For the analysis of this collision game, we first *purify* the game by introducing auxiliary variables X_1 and X_2 as follows:

$$L_1 \oplus \mathsf{F}_1(X_1) = L_2 \oplus \mathsf{F}_1(X_2) \quad \wedge \quad L_1 \mid R_1 \neq L_2 \mid R_2 \\ \wedge \quad X_1 = R_1 \oplus K \quad \wedge \quad X_2 = R_2 \oplus K\,.$$

We could now perform *partitioning* and consider two different cases: (i) $X_1 = X_2$ and (ii) $X_1 \neq X_2$.

Case (i) is easy to rule out: The first equation simplifies to $L_1 = L_2$, and combining the last two we also get $R_1 \oplus K = R_2 \oplus K$, which implies $R_1 = R_2$. But this contradicts inequality $L_1 \mid R_1 \neq L_2 \mid R_2$.

Now consider case (ii), i.e., add inequality $X_1 \neq X_2$ to the system. This allows us to apply the *peeling* rule to the first equation. That is, we introduce fresh unpredictable variables Y_1 and Y_2 with $X_1 <_\sigma Y_1$ and $X_2 <_\sigma Y_2$, and modify the first equation to be $L_1 \oplus Y_1 = L_2 \oplus Y_2$. Equation $X_1 = R_1 \oplus K$ now gives $K <_\sigma X_1$. This is because we have $R_1 <_\sigma K$, and if $X_1 <_\sigma K$, we could apply the *unpredictability* rule: Sampling K after R_1 and X_1 have been chosen would make this equation unsatisfiable except with negligible probability. Therefore, we must have $L_1, L_2 <_\sigma K <_\sigma X_1 <_\sigma Y_1$. Analogously, we deduce from $X_2 = R_2 \oplus K$ that $L_1, L_2 <_\sigma K <_\sigma X_2 <_\sigma Y_2$. This allows us to apply the *unpredictability* rule to equation $L_1 \oplus Y_1 = L_2 \oplus Y_2$, either on variable Y_1 (if $Y_2 <_\sigma Y_1$) or on Y_2 (otherwise), deriving the final contradiction. After having found a contradiction in all case distinctions, we conclude that this collision game is unwinnable.

Rest of the analysis. Next, AlgoROM analyzes all other collision games in the same manner. All these games can be shown to be unwinnable (except with negligible probability), hence the construction is proven to be CPA-secure.

3 Preliminaries

Basic notation. We let $\mathbb{N} := \{0, 1, \dots\}$ be the set of natural numbers and $\{0,1\}^*$ be the set of finite-length bit strings. For $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$, and denote the empty string by ε. For any $X, Y \in \{0,1\}^*$, $X \mid Y$ denotes the concatenation of X and Y. Vectors are written in boldface, and unless stated otherwise or clear from the context, indexes of vectors and strings start from 1. The cardinality of a set $\mathcal{X}$ is denoted $|\mathcal{X}|$, and the length of a string $X \in \{0,1\}^*$ or a vector $\boldsymbol{V}$ by $|X|$ or $|\boldsymbol{V}|$. Given sets $\mathcal{D}$ and $\mathcal{R}$, the sets of functions from $\mathcal{D}$ to $\mathcal{R}$ and of permutations on $\mathcal{D}$ are denoted as $\mathrm{Func}(\mathcal{D}, \mathcal{R})$ and $\mathrm{Perm}(\mathcal{D})$; when $\mathcal{D} = \{0,1\}^d$ (resp., $\mathcal{R} = \{0,1\}^r$), we abbreviate $\mathcal{D}$ by d (resp., $\mathcal{R}$ by r). We write P^- for the inverse of a permutation P. Sampling from a random variable $\mathcal{X}$ is denoted $X \twoheadleftarrow \mathcal{X}$; when $\mathcal{X}$ is a finite set, this means sampling from the uniform distribution over $\mathcal{X}$. For a table T, we write $\mathsf{T}[x] \leftarrow y$ to mean that the x-th entry of T is set to y. We write $\mathrm{Dom}(\mathsf{T})$ for the set of values x such that $\mathsf{T}[x] \neq \perp$, and $\mathrm{Rng}(\mathsf{T})$ for the set of all $T[x]$ with $x \in \mathrm{Dom}(\mathsf{T})$. Given a function F, we let $\mathsf{F}^n := \mathsf{F} \circ \cdots \circ \mathsf{F}$ (n times) denote its n-th functional power.

Cryptographic games [18]. We use the code-based game-playing framework of Bellare and Rogaway. A game G is a random variable run by a challenger C and an adversary $\mathcal{A}$. The challenger initializes the game by setting up a challenge value, which is then handed over to $\mathcal{A}$, who is tasked with solving it. To model potential out-of-bound capabilities, C may offer $\mathcal{A}$ a set of oracles that help the adversary find a solution. The output of $\mathcal{A}$ is then passed back to C, who verifies the purported solution and returns a decision bit. We say that adversary $\mathcal{A}$ wins game G if C's final output is 1; in this case we write $\mathrm{G}^{\mathcal{A}} = 1$ or, more briefly, $\mathrm{G}^{\mathcal{A}}$.

Let G_1 and G_2 be two games, run with an adversary $\mathcal{A}$, whose descriptions are identical except for the consequent inside one if-branch, and let Bad be the event that the boolean condition in the if-statement is triggered. Then $|\Pr[\mathrm{G}_1^{\mathcal{A}}] - \Pr[\mathrm{G}_2^{\mathcal{A}}]| \leq \Pr[\mathrm{Bad}]$.

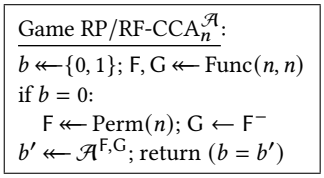

Figure 2: Definition of the RP/RF-CCA **game from the** RP/RF **switching lemma.**

The RP/RF switching lemma [18, 50]. The RP/RF-ATK advantage of an adversary $\mathcal{A}$ making at most q oracle queries is bounded by

$$\mathrm{Adv}_n^{\text{rp/rf-cpa}}(\mathcal{A}) \leq q^2/2^{n+1}\,, \qquad \mathrm{Adv}_n^{\text{rp/rf-cca}}(\mathcal{A}) \leq 3q^2/2^n\,,$$

where $\mathrm{Adv}_n^{\text{rp/rf-atk}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{RP/RF-CCA}_n^{\mathcal{A}}\right] - 1$, and the game RP/RF-CCA$_n^{\mathcal{A}}$ is given in Fig. 2 (top left). If ATK = CPA, we require that $\mathcal{A}$ never queries oracle G. If ATK = CCA, $\mathcal{A}$ never queries G on an output of F or vice versa.

XOR-theory. The XOR-algebra is a tuple $\boldsymbol{A} = (A, F, i, o)$, where $A = \{0,1\}^l$ for some $l \in \mathbb{N}$, and F is the set containing $\oplus\colon A^2 \to A$, projections $\pi_{i,n}\colon A^n \to A$ of all $n \in \mathbb{N}$ and all $1 \leq i \leq n$, and additional function symbols for the oracles offered by any given security game. Functions $i, o\colon F \to \mathbb{N}$ return the input and output arity of each function in F.

The algebra of first-order terms over the XOR-algebra $\boldsymbol{A}$ is a pair $\boldsymbol{T} := \boldsymbol{T}(\boldsymbol{A}) := (T, \alpha)$, where T is a set and $\alpha\colon T \to \mathbb{N}$ is a function with the following property: T is the smallest set U such that (1) if $t \in A \cup V$, where $V := \{x_i\}_{i \in \mathbb{N}}$, then $t \in U$ and $\alpha(t) = 1$; (2) if $t, t' \in U$, then $t \mid t' \in U$ and $\alpha(t \mid t') = \alpha(t) + \alpha(t')$; and (3) if $t \in U$ and $f \in F$ such that $\alpha(t) = i(f)$, then $f(t) \in U$ and $\alpha(f(t)) = o(f)$.

The XOR-theory is the algebra of first-order terms above, where we enforce on the collection of terms the set of equalities generated by the following equations: (1) for every $t_1, t_2, t_3 \in A \cup V$, $t_1 \oplus 0^l = t_1$, $t_1 \oplus t_1 = 0^l$, $t_1 \oplus t_2 = t_2 \oplus t_1$, and $t_1 \oplus (t_2 \oplus t_3) = (t_1 \oplus t_2) \oplus t_3$; (2) for every $t_1, t_2, t_3 \in T$, $t_1 \mid t_2 \mid t_3 := t_1 \mid (t_2 \mid t_3) = (t_1 \mid t_2) \mid t_3$; (3) for every $n \in \mathbb{N}$, $1 \leq i \leq n$, and $t_1, \dots, t_n \in A \cup V$, $\pi_{i,n}(t_1 \mid \cdots \mid t_n) = t_i$; (4) for every $t \in T$ with $\alpha(t) = q$, $\pi_{1,q}(t) \mid \cdots \mid \pi_{q,q}(t) = t$; (5) for every invertible function symbol $f \in F$ and every $x, k \in A \cup V$, $f^{-1}(k, f(k, x)) = x$ and $f(k, f^{-1}(k, x)) = x$.

3.1 Cryptographic schemes

We now recall hash functions and block ciphers, the main primitives we consider in this work. After fixing their syntax, we formalize some popular constructions of these primitives.

Proc. $\mathsf{I}_{\mathrm{ROM}(\boldsymbol{m},\boldsymbol{n})}$:	Proc. $\mathsf{I}_{\mathrm{RPM}(\boldsymbol{n})}$:	Proc. $\mathsf{I}_{\mathrm{ICM}(\boldsymbol{k},\boldsymbol{n})}$:
for $i = 1$ to $\lvert\boldsymbol{n}\rvert$:	for $i = 1$ to $\lvert\boldsymbol{n}\rvert$:	for $i = 1$ to $\lvert\boldsymbol{n}\rvert$:
$\quad \mathbf{F}_i \twoheadleftarrow \mathrm{Func}(\boldsymbol{m}_i, \boldsymbol{n}_i)$	$\quad \mathbf{P}_i \twoheadleftarrow \mathrm{Perm}(\boldsymbol{n}_i)$	$\quad \mathbf{E}_i \twoheadleftarrow \mathrm{Block}(\boldsymbol{k}_i, \boldsymbol{n}_i)$
return $\mathbf{F}$	return $(\mathbf{P}, \mathbf{P}^-)$	$\quad \mathbf{D}_i \leftarrow \mathbf{E}_i^-$
		return $(\mathbf{E}, \mathbf{D})$

Figure 3: Random variables returning the oracles for the corresponding idealized models. Here, $|\boldsymbol{k}| = |\boldsymbol{m}| = |\boldsymbol{n}|$.

Hash functions. Let $\mathrm{k}, \mathrm{m}, \mathrm{n} \in \mathbb{N}$ with $\mathrm{m}, \mathrm{n} \geq 1$. A hash function with key length k, input length m, and output length n is a function $\mathsf{H}\colon \{0,1\}^{\mathrm{k}} \times \{0,1\}^{\mathrm{m}} \to \{0,1\}^{\mathrm{n}}$. We denote by $\mathrm{Hash}(\mathrm{k}, \mathrm{m}, \mathrm{n})$ the set of all such hash functions.

Block ciphers. Let $\mathrm{k}, \mathrm{n} \in \mathbb{N}$ with $\mathrm{n} \geq 1$. A block cipher with key length k and block length n is a function $\mathsf{E}\colon \{0,1\}^{\mathrm{k}} \times \{0,1\}^{\mathrm{n}} \to \{0,1\}^{\mathrm{n}}$ such that $\mathsf{E}(K, \cdot)$ is a permutation on $\{0,1\}^{\mathrm{n}}$ for every $K \in \{0,1\}^{\mathrm{k}}$. We denote by $\mathrm{Block}(\mathrm{k}, \mathrm{n})$ the set of all such block ciphers. Every block cipher E uniquely defines a map $\mathsf{D} \in \mathrm{Block}(\mathrm{k}, \mathrm{n})$ via $\mathsf{E}(K, \mathsf{D}(K, M)) = \mathsf{D}(K, \mathsf{E}(K, M)) = M$ for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{\mathrm{n}}$, called inverse block cipher to E and denoted $\mathsf{E}^- := \mathsf{D}$.

Constructions. Often applications such as hash functions and block ciphers are not designed from the ground up, but built via a construction C from a vector of "simpler" primitives $\mathbf{Q}$ (which in turn can be, for instance, simpler hash functions, permutations, or block ciphers). In this paper, we are interested in constructions C that make black-box use of $\mathbf{Q}$; that is, C can only depend on the input-output behavior of the primitives in $\mathbf{Q}$, but not otherwise on their code. Thus, for the purpose of this work, a construction is a circuit C with Boolean and oracle gates of different types which, once the oracle gates are fixed, results in an instance of the application to be constructed. For a vector of primitives $\mathbf{Q}$, we write $\mathrm{C}^{\mathbf{Q}}$ for the circuit C when all oracle gates of type i are set to $\mathbf{Q}_i$, for all $i \in [|\mathbf{Q}|]$. If C is a construction of a block cipher, we denote by C^- the construction of the inverse block cipher.

Idealized models [17, 64]. A successful methodology to study many forms of security of a wide range of applications $\mathrm{C}^{\mathbf{Q}}$ is to carry out the analysis in a so-called idealized model of computation. In this setting, one treats (some of) the underlying primitives $\mathbf{Q}$ as random reference objects $\mathbf{O}$ that are meant to idealize the instantiations in $\mathbf{Q}$. In more detail, the challenger of the idealized security game first samples objects $\mathbf{O}$ according to a distribution specified by the model. He then runs the game describing the security notion, but replaces all evaluations of $\mathbf{Q}$ with calls to $\mathbf{O}$. He also offers $\mathbf{O}$ as a set of oracles to all parties playing the game, who are expected to interact with $\mathbf{O}$ in place of $\mathbf{Q}$.

For an idealized model MOD, we denote by $\mathsf{I}_{\mathrm{MOD}}$ the random variable returning the oracles for MOD. Examples of idealized models include the random-oracle model $\mathrm{ROM}(\boldsymbol{m}, \boldsymbol{n})$, the (two-sided) random-permutation model $\mathrm{RPM}(\boldsymbol{n})$, and the ideal-cipher model $\mathrm{ICM}(\boldsymbol{k}, \boldsymbol{n})$, whose initialization functions are presented in Fig. 3. A combination of these models (with interfaces offering oracles of different types), or models sampling from different distributions (where, e.g., related random objects are offered at distinct interfaces) are also possible.

$\underline{\mathrm{MD}^{\mathbf{F}}(K, M)}$:	$\underline{\mathrm{Sp}^{\mathbf{P},\mathbf{P}^-}(K, M)}$:
$D \leftarrow 0^{n-\mathrm{k}} \mid K$	$D \leftarrow 0^{r+c-\mathrm{k}} \mid K$
parse $M = M_1 \mid \cdots \mid M_t$	parse $M = M_1 \mid \cdots \mid M_t$
for $i = 1$ to t:	for $i = 1$ to t:
$\quad D \leftarrow \mathbf{OC}_i^{\mathbf{F}}(M_i \mid D)$	$\quad D \leftarrow \mathbf{OC}_i^{\mathbf{P},\mathbf{P}^-}(D \oplus (M_i \mid 0^c))$
return D	return $D[1..r]$

Figure 4: Left: Definition of the t-round Merkle–Damgård construction in $\mathrm{ROM}(\boldsymbol{m}', \boldsymbol{n})$. Here, $|M_i| = m$ for every $i \in [t]$. Right: Definition of the t-round sponge construction in $\mathrm{RPM}(\boldsymbol{n})$. Here, $|M_i| = r$ for every $i \in [t]$.

Oracle and key configurations. An oracle configuration (OC) is a vector $\mathbf{OC}$ of stateless deterministic circuits, each with access to ℓ oracles provided by the idealized model. A key configuration (KC) is a vector $\mathbf{KC}$ of stateless deterministic circuits, each taking as input the secret key and returning a round key. We write $\mathrm{C}[\mathbf{OC}]$ for a construction C whose queries are specified by an oracle configuration $\mathbf{OC}$, and also $\mathrm{C}^-[\mathbf{OC}^-]$ for the inverse construction. When we want to make the key configuration explicit, we also write $\mathrm{C}[\mathbf{OC}, \mathbf{KC}]$ and $\mathrm{C}^-[\mathbf{OC}^-, \mathbf{KC}^-]$, respectively.

OCs are meant to determine which oracles the construction queries in each round. Examples are $\mathbf{OC} := \mathbf{F}$, where a different oracle is called in each round, or $\mathbf{OC} := (\mathsf{F}) := (\mathsf{F}, \ldots, \mathsf{F})$, with the same oracle repeating throughout. Different patterns, more complex expressions (e.g., F^2 or $\mathbf{F}_2 \circ \mathbf{F}_1$) or nested constructions are also possible. Similarly, KCs specify how the round keys are computed from the "master" secret key. Cases of interest include pairwise different round keys, some random and some constant keys, or some keys being fixed offsets of other keys.

We only consider OCs and KCs that can be expressed in the XOR-language, and will specify their signature separately for every application we present.

We now recall a few classical constructions of hash functions and block ciphers that we will study in this work, beginning with hash functions. Throughout, let $\mathrm{k}, m, n, t \in \mathbb{N}$ with $m, n, t \geq 1$.

Merkle–Damgård [33, 60]. Assume $\mathrm{k} \leq n$, let $\boldsymbol{m'} := (m+n)_{i=1}^{\ell}$, $\boldsymbol{n} := (n)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Func}(m+n, n)^t$. The t-round Merkle–Damgård construction in the $\mathrm{ROM}(\boldsymbol{m'}, \boldsymbol{n})$ with oracle configuration $\mathbf{OC}$ is $\mathrm{MD}^{\mathbf{F}} := \mathrm{MD}_{\mathrm{k},m,n,t}[\mathbf{OC}]^{\mathbf{F}} \in \mathrm{Hash}(\mathrm{k}, tm, n)$ where, for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{tm}$, $\mathrm{MD}^{\mathbf{F}}(K, M)$ is defined in Fig. 4 (left).

Sponge [19]. Assume $\mathrm{k} \leq c$, let $\boldsymbol{n} := (r+c)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Perm}(r+c)^t$. The t-round sponge construction in the $\mathrm{RPM}(\boldsymbol{n})$ with oracle configuration $\mathbf{OC}$ is $\mathrm{Sp}^{\mathbf{P},\mathbf{P}^-} := \mathrm{Sp}_{\mathrm{k},r,c,t}[\mathbf{OC}]^{\mathbf{P},\mathbf{P}^-} \in \mathrm{Hash}(\mathrm{k}, tr, r)$ where, for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{tr}$, $\mathrm{Sp}^{\mathbf{P},\mathbf{P}^-}(K, M)$ is defined in Fig. 4 (right).

PGV [21, 62]. Assume $\mathrm{k} \leq n$, and fix $(\bar{K}, \bar{M}, \bar{O}) \in \{M, D, M \oplus D, 0^n\}^3$. The PGV compression function in the $\mathrm{ICM}(n, n)$ with assignment $(\bar{K}, \bar{M}, \bar{O})$ is the function $\mathrm{F}^{\mathsf{E},\mathsf{E}^-} := \mathrm{F}^{\mathsf{E},\mathsf{E}^-}_{n,\bar{K},\bar{M},\bar{O}} \in \mathrm{Func}(2n, n)$ defined via $\mathrm{F}^{\mathsf{E},\mathsf{E}^-}(M \mid D) := \mathsf{E}(\bar{K}, \bar{M}) \oplus \bar{O}$. Here, $M, D \in \{0,1\}^n$ denote the left and right halves of the input, respectively.

$\underline{\psi^{\mathbf{F}}(M)}$:
$C \leftarrow M$
for $i = 1$ to t:
  parse $C = L \mid R$
  $C \leftarrow R \mid \mathbf{OC}_i^{\mathbf{F}}(R) \oplus L$
return C

$\underline{\mathrm{SPN}^{\mathbf{P},\mathbf{P}^-}(K, M)}$:
$C \leftarrow M$
for $i = 1$ to t:
  $K_i \leftarrow \mathbf{KC}_i(K)$; $C \leftarrow \mathbf{A}_i(K_i, C)$
  parse $C = C_1 \mid \cdots \mid C_w$
  for $j = 1$ to w: $C_j \leftarrow \mathbf{OC}_{ij}^{\mathbf{P},\mathbf{P}^-}(C_j)$
  $C \leftarrow C_1 \mid \cdots \mid C_w$
$K_{t+1} \leftarrow \mathbf{KC}_{t+1}(K)$
$C \leftarrow \mathbf{A}_{t+1}(K_{t+1}, C)$
return C

$\underline{\mathrm{CC}^{\mathbf{E},\mathbf{E}^-}(K, M)}$:
$C \leftarrow M$; for $i = 1$ to t: $K_i \leftarrow \mathbf{KC}_i(K)$; $C \leftarrow \mathbf{OC}_i^{\mathbf{E},\mathbf{E}^-}(K_i, C)$
return C

Figure 5: Top left: Definition of the Feistel permutation in the $\mathrm{ROM}(\boldsymbol{n}, \boldsymbol{n})$**. Here,** L **and** R **denote the left and right halves of the input. Top right: Definition of the substitution-permutation cipher in the** $\mathrm{RPM}(\boldsymbol{n})$**. Here,** $|C_j| = n$ **for every** $j \in [w]$**. Bottom: Definition of the cascade cipher in the** $\mathrm{ICM}(\boldsymbol{k}, \boldsymbol{n})$**.**

The t-round PGV construction in the $\mathrm{ICM}(n, n)$ with assignment $(\bar{K}, \bar{M}, \bar{O})$ is the hash function $\mathrm{PGV}^{\mathsf{E},\mathsf{E}^-} := \mathrm{PGV}^{\mathsf{E},\mathsf{E}^-}_{\mathrm{k},n,\bar{K},\bar{M},\bar{O},t} := \mathrm{MD}^{\mathsf{F}^{\mathsf{E},\mathsf{E}^-}_{\bar{K},\bar{M},\bar{O}}}_{\mathrm{k},n,n,t} \in \mathrm{Hash}(\mathrm{k}, tn, n)$.

We next recall some prominent constructions of block ciphers. Throughout, fix $\mathrm{k}, k, n, w, \ell, t \in \mathbb{N}$ with $n, w, t \geq 1$.

Feistel permutation [41, 55]. Let $\boldsymbol{n} := (n)_{i=1}^{\ell}$, $\mathbf{OC} \in \mathrm{Func}(n, n)^t$. The t-round Feistel permutation in the $\mathrm{ROM}(\boldsymbol{n}, \boldsymbol{n})$ with configuration $\mathbf{OC}$ is $\psi^{\mathbf{F}} = \psi_{n,t}[\mathbf{OC}]^{\mathbf{F}} \in \mathrm{Perm}(2n)$ defined in Fig. 5 (top left).

Remark. Note that the "textbook" version of the Feistel permutation omits the swap in the final round. We chose to present the "uniform" variant since it is easier to describe, and swapping in the final round does not affect security.

Key-alternating Feistel (KAF) cipher [53]. Let $\boldsymbol{n} := (n)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Func}(n, n)^t$ and $\mathbf{KC} \in \mathrm{Func}(\mathrm{k}, n)^t$. The t-round key-alternating Feistel cipher in the $\mathrm{ROM}(\boldsymbol{n}, \boldsymbol{n})$ with configuration $(\mathbf{OC}, \mathbf{KC})$ is $\mathrm{KAF}^{\mathbf{F}} = \mathrm{KAF}_{\mathrm{k},n,t}[\mathbf{OC}, \mathbf{KC}]^{\mathbf{F}} \in \mathrm{Block}(\mathrm{k}, 2n)$ where $\mathrm{KAF}^{\mathbf{F}}(K, M) := \psi_{n,t}[\mathbf{OC}']^{\mathbf{F}'}(M)$ for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{2n}$. Here, $\mathbf{OC}'^{\mathbf{F}'}_i = \mathbf{F}'_i$ and $\mathbf{F}'_i(R) := \mathbf{OC}_i^{\mathbf{F}}(\mathbf{KC}_i(K) \oplus R)$ for every $R \in \{0,1\}^n$ and $i \in [t]$.

Luby–Rackoff (LR) cipher [55]. Let $\boldsymbol{m} := (k + n)_{i=1}^{\ell}$, $\boldsymbol{n} := (n)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Func}(k+n, n)^t$, $\mathbf{KC} \in \mathrm{Func}(\mathrm{k}, k)^t$. The t-round Luby–Rackoff cipher in the $\mathrm{ROM}(\boldsymbol{m}, \boldsymbol{n})$ with configuration $(\mathbf{OC}, \mathbf{KC})$ is the block cipher $\mathrm{LR}^{\mathbf{F}} = \mathrm{LR}_{\mathrm{k},k,n,t}[\mathbf{OC}, \mathbf{KC}]^{\mathbf{F}} \in \mathrm{Block}(\mathrm{k}, 2n)$, where $\mathrm{LR}^{\mathbf{F}}(K, M) := \psi_{n,t}[\mathbf{OC}']^{\mathbf{F}'}(M)$ for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{2n}$. Here, $\mathbf{OC}'^{\mathbf{F}'}_i = \mathbf{F}'_i$ and $\mathbf{F}'_i(R) := \mathbf{OC}_i^{\mathbf{F}}(\mathbf{KC}_i(K) \mid R)$ for every $R \in \{0,1\}^n$ and $i \in [t]$.

Substitution-permutation network (SPN) [41, 64]. Let $\boldsymbol{n} := (n)_{i=1}^{\ell}$, fix $\mathbf{OC} \in \mathrm{Perm}(n)^{t \times w}$, $\mathbf{KC} \in \mathrm{Func}(\mathrm{k}, k)^{t+1}$, $\mathbf{A} \in \mathrm{Block}(k, wn)^{t+1}$.

Game $\mathrm{CR}^{\mathcal{A}}_{\mathrm{C},\mathsf{I}}$:
$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$
$(M, M') \twoheadleftarrow \mathcal{A}^{\mathbf{O}}(K^*)$
return $\big(\mathrm{C}^{\mathbf{O}}(K^*, M) = \mathrm{C}^{\mathbf{O}}(K^*, M')\big)$

Game $\mathrm{PRF\text{-}CPA}^{\mathcal{A}}_{\mathrm{C}}$:
$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $b \twoheadleftarrow \{0,1\}$
$K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $\mathsf{F} \leftarrow \mathrm{C}^{\mathbf{O}}(K^*, \cdot)$
if $b = 0$: $\mathsf{F} \twoheadleftarrow \mathrm{Func}(\mathrm{m}, \mathrm{n})$
$b' \twoheadleftarrow \mathcal{A}^{\mathbf{O},\mathsf{F}}$; return $(b = b')$

Figure 6: Left: Game defining the CR**-security of** C**. Right: Game defining the** PRF-CPA**-security of** C**. In both figures,** C **is a construction of a hash function with key, input and output lengths** k, m **and** n **from oracles O.**

The t-round substitution-permutation network in the $\mathrm{RPM}(\boldsymbol{n})$ with diffusion layer $\mathbf{A}$ and configuration $(\mathbf{OC}, \mathbf{KC})$ is the block cipher $\mathrm{SPN}^{\mathbf{P},\mathbf{P}^-} = \mathrm{SPN}_{\mathrm{k},k,n,w,t}[\mathbf{OC}, \mathbf{KC}, \mathbf{A}]^{\mathbf{P},\mathbf{P}^-} \in \mathrm{Block}(\mathrm{k}, wn)$ where, for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^{wn}$, $\mathrm{SPN}^{\mathbf{P},\mathbf{P}^-}(K, M)$ is defined in Fig. 5 (top right).

Even–Mansour (EM) cipher [37]. Let $\boldsymbol{n} := (n)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Perm}(n)^t$ and $\mathbf{KC} \in \mathrm{Func}(\mathrm{k}, n)^{t+1}$. The t-round Even–Mansour cipher in the $\mathrm{RPM}(\boldsymbol{n})$ with configuration $(\mathbf{OC}, \mathbf{KC})$ is the block cipher $\mathrm{EM}^{\mathbf{P},\mathbf{P}^-} = \mathrm{EM}_{\mathrm{k},n,t}[\mathbf{OC}, \mathbf{KC}]^{\mathbf{P},\mathbf{P}^-} \in \mathrm{Block}(\mathrm{k}, n)$, where we let $\mathrm{EM}^{\mathbf{P},\mathbf{P}^-}(K, M) := \mathrm{SPN}_{\mathrm{k},n,n,1,t}[\mathbf{OC}, \mathbf{KC}, \mathbf{A}']^{\mathbf{P},\mathbf{P}^-}(K, M)$ for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^n$. Here, $\mathbf{A}'_i(K, C) := K \oplus C$ for every $K, C \in \{0,1\}^n$ and $i \in [t+1]$.

Cascade cipher [2, 18, 36]. Let $\boldsymbol{k} := (k)_{i=1}^{\ell}$, $\boldsymbol{n} := (n)_{i=1}^{\ell}$, and fix $\mathbf{OC} \in \mathrm{Block}(k, n)^t$ and $\mathbf{KC} \in \mathrm{Func}(\mathrm{k}, k)^t$. The t-round cascade cipher in the $\mathrm{ICM}(\boldsymbol{k}, \boldsymbol{n})$ with configuration $(\mathbf{OC}, \mathbf{KC})$ is the cipher $\mathrm{CC}^{\mathbf{E},\mathbf{E}^-} := \mathrm{CC}_{\mathrm{k},k,n,t}[\mathbf{OC}, \mathbf{KC}]^{\mathbf{E},\mathbf{E}^-} \in \mathrm{Block}(\mathrm{k}, n)$ where, for every $K \in \{0,1\}^{\mathrm{k}}$ and $M \in \{0,1\}^n$, $\mathrm{CC}^{\mathbf{E},\mathbf{E}^-}(K, M)$ is defined in Fig. 5 (bottom).

3.2 Security notions

Having introduced the constructions that we will later investigate, we now recall the security notions for these constructions that we target in this work, starting with those for hash functions. Since our analyses will be in appropriate idealized models of computation, we formulate these notions in that setting. Throughout, fix a construction $\mathrm{C}^{\mathbf{O}} \in \mathrm{Hash}(\mathrm{k}, \mathrm{m}, \mathrm{n})$ of a hash function with access to oracles $\mathbf{O}$ provided by a model MOD with initialization function $\mathsf{I} := \mathsf{I}_{\mathrm{MOD}}$, and let $\mathcal{A}$ be an adversary in the game formalizing the security notion.

Collision resistance [32]. The CR advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\mathrm{cr}}_{\mathrm{C},\mathsf{I}}(\mathcal{A}) := \Pr\big[\mathrm{CR}^{\mathcal{A}}_{\mathrm{C},\mathsf{I}}\big],$$

where the game $\mathrm{CR}^{\mathcal{A}}_{\mathrm{C},\mathsf{I}}$ is defined in Fig. 6 (left). Here, adversary $\mathcal{A}$ is required to return messages $M, M' \in \{0,1\}^{\mathrm{m}}$ that are distinct.

PRF-CPA*-security* [43]. The PRF-CPA advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\text{prf-cpa}}_{\mathrm{C},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\big[\mathrm{PRF\text{-}CPA}^{\mathcal{A}}_{\mathrm{C},\mathsf{I}}\big] - 1,$$

where the game $\mathrm{PRF\text{-}CPA}^{\mathcal{A}}_{\mathrm{C},\mathsf{I}}$ is defined in Fig. 6 (right). This notion is usually called PRF-security in the literature; we choose a slightly different terminology for naming consistency with other notions.

Game $\text{PRP-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$:

$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $b \twoheadleftarrow \{0,1\}$; $K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $\mathsf{P} \leftarrow \mathsf{C}^{\mathbf{O}}(K^*, \cdot)$
if $b = 0$: $\mathsf{P} \twoheadleftarrow \mathrm{Perm}(\mathrm{n})$
$b' \twoheadleftarrow \mathcal{A}^{\mathbf{O},\mathsf{P},\mathsf{P}^-}$; return $(b = b')$

Figure 7: Game defining the PRP-CCA**-security of a construction** C **of a block cipher with key and block lengths** k **and** n **from oracles O.**

We now move on to the security notions for block ciphers. Fix a construction $\mathsf{C}^{\mathbf{O}} \in \mathrm{Block}(\mathrm{k},\mathrm{n})$ of a block cipher with access to oracles $\mathbf{O}$ provided by a model MOD with initialization function $\mathsf{I} := \mathsf{I}_{\mathrm{MOD}}$, and let $\mathcal{A}$ be an adversary in the game formalizing the security notion. The placeholder $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA}\}$ indicates the adversarial capabilities considered in the model. We begin with the notion of PRP-security.

PRP-ATK-*security* [43, 55]. The PRP-ATK advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\text{prp-atk}}_{\mathsf{C},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{PRP-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}\right] - 1\,,$$

where the game $\text{PRP-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}} := \text{PRP-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$ is given in Fig. 7 (top). If ATK = CPA, we require that $\mathcal{A}$ never queries the oracle P^-. Again, PRP-CPA and PRP-CCA are usually called PRP and SPRP, but renamed here for naming consistency.

The more advanced security models we consider allow the adversary to obtain encryptions under keys K that depend on the challenge key K^* (RK-CPA-security), or encryptions under K^* of plaintexts M that depend on K^* (KD-CPA-security). Here, the relation between K^* and K (resp., M) is chosen by the adversary, who queries its oracle on a derivation function ϕ (resp., ψ) from a set of allowed relations Φ (resp., Ψ). The actual encryption key K used (resp., the message M being encrypted) by the oracle is then $K = \phi(K^*)$ (resp., $M = \psi(K^*)$). A combination of these notions, called KC-CPA-security, is also possible, where the adversary can query functions ξ from a set Ξ which specify pairs $(K, M) = \xi(K^*)$, and then obtain an encryption of M under K. In the CCA setting, the above also applies to decryptions and ciphertexts.

RK/KD/KC functions. For the purpose of this work, we consider (1) the set Φ of RK-derivation functions $\phi(K^*) := K^* \oplus \Delta$ for $\Delta \in \{0,1\}^{\mathrm{k}}$; (2) the set Ψ of KD-derivation functions $\psi(K^*) := \sum_{i=1}^{r} \alpha_i K_i^* \oplus \Delta$ for $\alpha_i \in \{0,1\}$ and an offset Δ, where K_i^* are components of K^*; and (3) the set Ξ of KC-derivation functions which consist of RK-derivation functions ϕ in the first (key) coordinate and KD-derivation functions ψ in the second (message) coordinate.

RK-ATK-*security* [16, 20, 51]. The RK-ATK advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\text{rk-atk}}_{\mathsf{C},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{RK-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}\right] - 1\,,$$

where the game $\text{RK-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}} := \text{RK-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$ is given in Fig. 8 (top). We require that $\mathcal{A}$ only queries functions $\phi^e, \phi^d \in \Phi$. If ATK = CPA, $\mathcal{A}$ is never allowed to query the oracle RKDec.

Game $\text{RK-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$:

$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $b \twoheadleftarrow \{0,1\}$
$K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $\mathsf{E} \leftarrow \mathsf{C}^{\mathbf{O}}$
if $b = 0$:
  $\mathsf{E} \twoheadleftarrow \mathrm{Block}(\mathrm{k},\mathrm{n})$
$b' \twoheadleftarrow \mathcal{A}^{\mathbf{O},\text{RKEnc},\text{RKDec}}$
return $(b = b')$

Proc. $\text{RKEnc}(\phi^e, M)$:

$K \leftarrow \phi^e(K^*)$; $C \leftarrow \mathsf{E}(K, M)$
return C

Proc. $\text{RKDec}(\phi^d, C)$:

$K \leftarrow \phi^d(K^*)$; $M \leftarrow \mathsf{E}^-(K, C)$
return M

Game $\text{KD-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$:

$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $b \twoheadleftarrow \{0,1\}$
$K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $\mathsf{E} \leftarrow \mathsf{C}^{\mathbf{O}}$
if $b = 0$:
  $\mathsf{E} \twoheadleftarrow \mathrm{Block}(\mathrm{k},\mathrm{n})$
$\mathsf{CL} \leftarrow [\,]$
$b' \twoheadleftarrow \mathcal{A}^{\mathbf{O},\text{KDEnc},\text{KDDec}}$
return $(b = b')$

Proc. $\text{KDEnc}(\psi^e)$:

$M \leftarrow \psi^e(K^*)$; $C \leftarrow \mathsf{E}(K^*, M)$
$\mathsf{CL} \leftarrow \mathsf{CL} : C$; return C

Proc. $\text{KDDec}(\psi^d)$:

$C \leftarrow \psi^d(K^*)$; if $C \in \mathsf{CL}$: return $\perp$
$M \leftarrow \mathsf{E}^-(K^*, C)$; return M

Game $\text{KC-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$:

$\mathbf{O} \twoheadleftarrow \mathsf{I}$; $b \twoheadleftarrow \{0,1\}$
$K^* \twoheadleftarrow \{0,1\}^{\mathrm{k}}$; $\mathsf{E} \leftarrow \mathsf{C}^{\mathbf{O}}$
if $b = 0$:
  $\mathsf{E} \twoheadleftarrow \mathrm{Block}(\mathrm{k},\mathrm{n})$
$\mathsf{ML} \leftarrow [\,]$; $\mathsf{CL} \leftarrow [\,]$
$b' \twoheadleftarrow \mathcal{A}^{\mathbf{O},\text{KCEnc},\text{KCDec}}$
return $(b = b')$

Proc. $\text{KCEnc}(\xi^e)$:

$(K, M) \leftarrow \xi^e(K^*)$
if $(K, M) \in \mathsf{ML}$: return $\perp$
$C \leftarrow \mathsf{E}(K, M)$; $\mathsf{CL} \leftarrow \mathsf{CL} : (K, C)$
return C

Proc. $\text{KCDec}(\xi^d)$:

$(K, C) \leftarrow \xi^d(K^*)$
if $(K, C) \in \mathsf{CL}$: return $\perp$
$M \leftarrow \mathsf{E}^-(K, C)$; $\mathsf{ML} \leftarrow \mathsf{ML} : (K, M)$
return M

Figure 8: Games defining the SEC-CCA**-security of a construction** C **of a block cipher with key and block lengths** k **and** n **from oracles O. Here,** $\mathrm{SEC} \in \{\mathrm{RK}, \mathrm{KD}, \mathrm{KC}\}$**.**

KD-ATK-*security* [22, 24]. The KD-ATK advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\text{kd-atk}}_{\mathsf{C},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{KD-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}\right] - 1\,,$$

where $\text{KD-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}} := \text{KD-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$ is given in Fig. 8 (center). We require that $\mathcal{A}$ only queries functions $\psi^e, \psi^d \in \Psi$. If ATK = CPA, $\mathcal{A}$ is never allowed to query the oracle KDDec.

KC-ATK-*security* [30]. The KC-ATK advantage of $\mathcal{A}$ against C is

$$\mathrm{Adv}^{\text{kc-atk}}_{\mathsf{C},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{KC-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}\right] - 1\,,$$

where the game $\text{KC-ATK}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}} := \text{KC-CCA}^{\mathcal{A}}_{\mathsf{C},\mathsf{I}}$ is given in Fig. 8 (bottom). We require that adversary $\mathcal{A}$ only queries functions $\xi^e, \xi^d \in \Xi$. If ATK = CPA, $\mathcal{A}$ is never allowed to query the oracle KCDec.

4 Indistinguishability up to Collisions

In this section, we further expand on the single steps of our automated proof strategy outlined in the Introduction. Throughout, let C be a construction with access to oracles $\mathbf{O}$ provided by an idealized model MOD, whose queries are specified by an oracle configuration $\mathbf{OC}$, for which we want to prove SEC-ATK-security for any SEC and ATK presented above. Without loss of generality, we

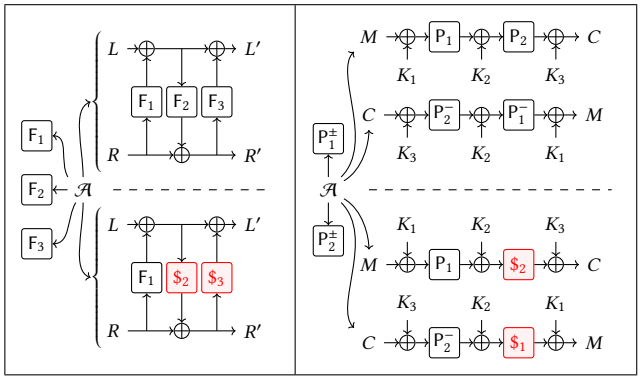

Figure 9: Left: Oracle replacement for the 3-round Feistel permutation. Right: Oracle replacement for the 2-round Even–Mansour cipher. In both figures, the real construction is represented on top, and the one with modified oracles at the bottom. Direct oracles are never modified, and are available in both settings.

Game Rand-CPA$^{\mathcal{A}}_{\mathsf{C},\mathbf{OC}',\mathsf{I}}$:	Proc. $\textsc{Ch}(M)$:
$\mathbf{O} \leftarrow\!\!\!\!\leftarrow \mathsf{I}$ $b \leftarrow\!\!\!\!\leftarrow \{0,1\}$; $b' \leftarrow\!\!\!\!\leftarrow \mathcal{A}^{\mathbf{O},\textsc{Ch}}$ return $(b = b')$	if $b = 1$: return $\mathsf{C}[\mathbf{OC}']^{\mathbf{O}}(M)$ $C \leftarrow\!\!\!\!\leftarrow \{0,1\}^{\mathrm{n}}$; return C
Game Rand-CCA$^{\mathcal{A}}_{\mathsf{C},\mathbf{OC}^{\pm\prime},\mathsf{I}}$:	Proc. $\textsc{ChEnc}(M)$:
$\mathbf{O} \leftarrow\!\!\!\!\leftarrow \mathsf{I}$; $b \leftarrow\!\!\!\!\leftarrow \{0,1\}$ $b' \leftarrow\!\!\!\!\leftarrow \mathcal{A}^{\mathbf{O},\textsc{ChEnc},\textsc{ChDec}}$ return $(b = b')$	if $b = 1$: return $\mathsf{C}[\mathbf{OC}']^{\mathbf{O}}(M)$ $C \leftarrow\!\!\!\!\leftarrow \{0,1\}^{\mathrm{n}}$; return C
	Proc. $\textsc{ChDec}(C)$:
	if $b = 1$: return $\mathsf{C}^{-}[\mathbf{OC}^{-\prime}]^{\mathbf{O}}(C)$ $M \leftarrow\!\!\!\!\leftarrow \{0,1\}^{\mathrm{n}}$; return M

Figure 10: Top: Definition of the randomizability game for a construction C and modified oracle configuration $\mathbf{OC}'$ with one-sided access. Bottom: Definition of the randomizability game for a construction C and modified oracle configurations $\mathbf{OC}'$ and $\mathbf{OC}^{-\prime}$ with two-sided access.

assume that an adversary $\mathcal{A}$ playing this game never queries any oracle twice on the same input.[4] Furthermore, for every two-sided oracle provided by the game (i.e., every invertible oracle in $\mathbf{O}$, plus the inverse construction if ATK = CCA), we assume that $\mathcal{A}$ does not make a forward query and then a backward query on the result, or vice versa. Denote by G_0 the game corresponding to the real world of the considered SEC-ATK-game.

Oracle replacement. As its first step, AlgoROM tries to replace some of the oracle gates in C (and C^{-}, if the considered security model grants two-sided access to the construction) with forgetful random oracles \$, which on every invocation return a fresh random string of the appropriate length, even when called twice on the same input. The algorithm's goal is to replace sufficiently many oracle gates with \$ so that the outputs of the resulting construction (and the inverse, in case of two-sided access) are fully randomized.

To denote oracle replacement, we substitute the corresponding entry in $\mathbf{OC}$ with \$. For example, consider the three-round Feistel permutation with configuration $\mathbf{OC} = (\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3)$. Then $\mathbf{OC}' = (\mathsf{F}_1, \$_2, \$_3)$ means that the oracles in the last two rounds are replaced with FROs (see Fig. 9 (left)). Similarly, given the two-round Even–Mansour cipher with $\mathbf{OC} = (\mathsf{P}_1, \mathsf{P}_2)$ and $\mathbf{OC}^{-} = (\mathsf{P}_2^{-}, \mathsf{P}_1^{-})$, then $\mathbf{OC}' = (\mathsf{P}_1, \$_2)$ and $\mathbf{OC}^{-\prime} = (\mathsf{P}_2^{-}, \$_1)$ mean that the last permutation (both in the forward and backward direction) is replaced with an FRO (see Fig. 9 (right)). Notice that when ATK = CCA, we allow to modify the oracle configuration independently for the direct construction and its inverse.

We remark that this notation does *not* mean that the original construction is considered in an "augmented" idealized model, where FROs are added to $\mathbf{O}$ and certain gates are re-wired to invoke \$. Rather, it denotes a modified construction $\mathsf{C}[\mathbf{OC}']$ in the original model where certain oracle gates in C are replaced with FRO gates, akin to changes done in game-hopping proofs. In particular, the oracles offered directly to the adversary are never changed.

[4] This is not always without loss of generality (e.g., in the CPA-model for randomized encryption), but for the models and primitives considered here, it is.

This oracle replacement step is automated in our proofs, where AlgoROM replaces $\mathbf{OC}_i$ with \$ and then checks if the construction outputs are indeed random (see next step). As for which oracle gates to replace, we proceed via a "guided exhaustive search": Note that most natural cryptographic constructions would be trivially randomized if all gates were replaced with \$. However, AlgoROM tries to replace sufficiently many gates so as to randomize the construction, but not so many that collisions between replaced and not-replaced oracles can be trivially found, as this would lead to a large distinguishing advantage between $\mathsf{C}[\mathbf{OC}]$ and $\mathsf{C}[\mathbf{OC}']$. For typical constructions the replaced rounds will be some of the "innermost" calls, as these are the "hardest to control" for the adversary (assuming two-sided access to the construction). However, experiments show that this is not always the case.

To give some examples, when running experiments for Feistel-based constructions, AlgoROM replaces two consecutive rounds. For Even–Mansour ciphers, a single round is sufficient (see Fig. 9). For substitution-permutation networks, depending on the width of the construction, many replacements may be needed.

Randomization. Once AlgoROM has fixed a candidate oracle configuration $\mathbf{OC}'$ (and $\mathbf{OC}^{-\prime}$, in case of two-sided access) to replace $\mathbf{OC}$ (and $\mathbf{OC}^{-}$), it attempts to prove that the modified construction is indeed randomized. That is, it seeks to show that for every input to $\mathsf{C}[\mathbf{OC}']$ (and $\mathsf{C}^{-}[\mathbf{OC}^{-\prime}]$), the outputs are uniformly random. Formally, this is captured by the Rand-ATK advantage of an adversary $\mathcal{A}$ against C and $\mathbf{OC}'$ (and $\mathbf{OC}^{-\prime}$), defined as

$$\mathrm{Adv}^{\text{rand-atk}}_{\mathsf{C},\mathbf{OC}^{(\pm)\prime},\mathsf{I}}(\mathcal{A}) := 2 \cdot \Pr\left[\text{Rand-ATK}^{\mathcal{A}}_{\mathsf{C},\mathbf{OC}^{(\pm)\prime},\mathsf{I}}\right] - 1\,,$$

where the games $\text{Rand-ATK}^{\mathcal{A}}_{\mathsf{C},\mathbf{OC}^{(\pm)\prime},\mathsf{I}}$ are given in Fig. 10. AlgoROM recognizes an oracle replacement as randomizing if this advantage is 0. If the candidate replacement is not randomizing, it is discarded and the next one is tested.

Randomization is checked as follows: The outputs of $\mathsf{C}[\mathbf{OC}']$ (and $\mathsf{C}^{-}[\mathbf{OC}^{-\prime}]$) are functions of the randomness of the FROs. AlgoROM tries to invert the expressions of the outputs of the

scheme and re-express (parts of) the FRO outputs in terms of the construction outputs (i.e., it "solves for \$"). If it succeeds, this establishes a one-to-one correspondence between the randomness of the FROs and the construction outputs. Since FRO outputs are random by definition, this means that the construction outputs must be random too. In the constructions that we consider, this correspondence can be established via simple manipulation of expressions; see Section 2 for a concrete example.

Bad-event definition. Once AlgoROM has fixed an oracle configuration $\mathbf{OC}'$ (and $\mathbf{OC}^{-\prime}$, if ATK = CCA) which results in a randomized construction, it proceeds with examining the SEC-ATK-game for $\mathrm{C}[\mathbf{OC}']$, which we denote by G_1. Note, however, that moving from G_0 to G_1 comes at the cost of an event Bad (defined in both games) which captures the inconsistencies in oracle responses between the two games. We describe this event next.

To define the event Bad, it is useful to think of the oracles offered by the model as implemented via lazy sampling. In G_0, every oracle gate maintains its own table, which (partially) implements the primitive it is associated to: If, upon an oracle call, the gate finds that query stored in its own table, it returns the corresponding answer. If not, it first checks the tables of *all other* gates implementing the same primitive, to see if a reply for that query has already been set elsewhere. If so, the corresponding value is again returned (but not added to the gate's own table); if not, a fresh and correctly distributed value is generated, and the gate's table is updated accordingly. Lazy sampling in G_1 works analogously, except that the FRO gates no longer check or update any tables, but simply return random values of the correct length.

With this description of G_0 and G_1 in mind, the event Bad we seek to define is the event that, in the execution of G_1, two oracle gates which in G_0 do implement the same primitive, and of which at least one is replaced in G_1, are called on colliding inputs. (This also includes direct queries to the primitive oracles, which are never replaced but may still account for collisions with oracle gates in the construction.) Indeed, for such a query the oracles in G_0 return consistent answers, whereas those in G_1 will likely not.

More precisely, denote by $\mathrm{Bad}^{(\mathrm{p},ie)}$ the event that a collision occurs between the input of the i-th oracle gate as a result of a query to $\mathrm{C}[\mathbf{OC}']$, and a direct query to the primitive P which this gate used to implement in $\mathrm{C}[\mathbf{OC}]$. Similarly, let $\mathrm{Bad}^{(je,ke)}$ be the event that a collision occurs between inputs of the j-th and the k-th oracle gates as a result of two queries to $\mathrm{C}[\mathbf{OC}']$. Then we have $\mathrm{Bad} = \bigvee \mathrm{Bad}^{(\mathrm{p},ie)} \vee \bigvee \mathrm{Bad}^{(je,ke)}$, with $i \in \mathsf{L}^{(\mathrm{p,e})}$ and $(j,k) \in \mathsf{L}^{(\mathrm{e,e})}$, where

$$\mathsf{L}^{(\mathrm{p,e})} := \{i \in [t] : \mathbf{OC}'_i = \$\}\,,$$
$$\mathsf{L}^{(\mathrm{e,e})} := \{(j,k) \in [t]^2 : \mathbf{OC}'_j = \$ \wedge \mathbf{OC}_j = \mathbf{OC}_k\}\,.$$

Observe that if P is invertible, then P and P^- are counted as separate oracles in this setting. Also notice that for the pairs $(j,j) \in \mathsf{L}^{(\mathrm{e,e})}$, the corresponding queries to $\mathrm{C}[\mathbf{OC}']$ have to be different (otherwise such a collision can be trivially triggered), whereas they don't have to be different for all other pairs $(j,k) \in \mathsf{L}^{(\mathrm{e,e})}$ with $j \neq k$.

If ATK = CCA, we must add collisions that arise from querying the construction twice in the backward direction, as well as once in each direction. That is, we also consider events $\mathrm{Bad}^{(\mathrm{p},id)}$, $\mathrm{Bad}^{(jd,kd)}$ and $\mathrm{Bad}^{(re,sd)}$ for $i \in \mathsf{L}^{(\mathrm{p,d})}$, $(j,k) \in \mathsf{L}^{(\mathrm{d,d})}$, and $(r,s) \in \mathsf{L}^{(\mathrm{e,d})}$, where

$$\mathsf{L}^{(\mathrm{p,d})} := \{i \in [t] : \mathbf{OC}_i^{-\prime} = \$\}\,,$$
$$\mathsf{L}^{(\mathrm{d,d})} := \{(j,k) \in [t]^2 : \mathbf{OC}_j^{-\prime} = \$ \wedge \mathbf{OC}_j^{-} = \mathbf{OC}_k^{-}\}\,,$$
$$\mathsf{L}^{(\mathrm{e,d})} := \left\{(r,s) \in [t]^2 : \begin{array}{r}(\mathbf{OC}'_r = \$ \wedge \mathbf{OC}_r = \mathbf{OC}_s^{-}) \vee \\ (\mathbf{OC}_r^{-\prime} = \$ \wedge \mathbf{OC}_r^{-} = \mathbf{OC}_s)\end{array}\right\}.$$

Finally, to account for adaptivity, observe that each event $\mathrm{Bad}^{(\mathrm{p},ie)}$ (and $\mathrm{Bad}^{(\mathrm{p},id)}$, if present) is actually a disjunction of two bad events, one each depending on whether the primitive is queried before or after the construction while generating a collision. The same holds for all other bad events with indices $j \neq k$ (resp., $r \neq s$).

By the fundamental lemma of game-playing, the distinguishing advantage between games G_0 and G_1 is then upper-bounded by

$$\begin{aligned}\sum \Pr\left[\mathrm{Bad}^{(\mathrm{p},ie)}\right] + \sum \Pr\left[\mathrm{Bad}^{(je,ke)}\right] + \sum \Pr\left[\mathrm{Bad}^{(\mathrm{p},id)}\right] \\ + \sum \Pr\left[\mathrm{Bad}^{(jd,kd)}\right] + \sum \Pr\left[\mathrm{Bad}^{(re,sd)}\right],\end{aligned} \tag{1}$$

with indices ranging over the sets defined above.

Collision game formation. In its next step, AlgoROM bounds the probability of each bad event (in G_1) defined above. To that end, each such event is first converted into a collision game, which an adversary C can win if and only if $\mathcal{A}$ triggers the corresponding event. This is done as follows: For each bad event, we compute the expressions corresponding to the colliding queries, which are then equated to form the winning condition of the collision game.

Note that the oracle for the construction (and its inverse, if present) is randomized in G_1. We denote by $\mathrm{Adv}^{\mathrm{coll}}_{\mathrm{C}}(C)$ the sum of all advantages in these collision games, ranging over all bad events considered in (1).

Switching. Finally, if the ideal world in the SEC-ATK-security game for $\mathrm{C}[\mathbf{OC}]$ offers construction oracles that are not uniformly random functions, we need to bound the difference between G_1 (which has randomized construction oracles) and this game. In the case of random permutation oracles (the only case considered in this work), this can be done via the RP/RF switching lemma. Note that this step is redundant if the construction oracles in the ideal world of the SEC-ATK-security game implement random functions.

Collecting all terms defined above, we obtain the following result.

Theorem 4.1 (Indistinguishability up to collisions). *Let* C *be a construction in the model* MOD*, whose oracle queries are specified by an oracle configuration* $\mathbf{OC}$*. Then for any adversary* $\mathcal{A}$ *in the* SEC-ATK*-security game for* C *there exist adversaries* $\mathcal{B}$*,* C *and* $\mathcal{D}$ *with*

$$\begin{aligned}\mathrm{Adv}^{\text{sec-atk}}_{\mathrm{C,I}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{rand-atk}}_{\mathrm{C},\mathbf{OC}^{(\pm)\prime},\mathrm{I}}(\mathcal{B}) + \mathrm{Adv}^{\mathrm{coll}}_{\mathrm{C}}(C) \\ + \mathrm{Adv}^{\mathrm{rp/rf\text{-}atk}}_{\mathrm{n}}(\mathcal{D})\,,\end{aligned}$$

where $\mathbf{OC}'$ *(and* $\mathbf{OC}^{-\prime}$*, if* ATK = CCA*) is the oracle replacement chosen by* AlgoROM*, and* n *is the block length of* C*. Here,* $\mathcal{B}$ *makes the same number of primitive and construction queries as* $\mathcal{A}$*,* C *makes a number of queries proportional to all queries of* $\mathcal{A}$*, and* $\mathcal{D}$ *makes the same number of queries as* $\mathcal{A}$ *makes to* C*.*

Game $R\text{-Coll}_{\mathsf{I}}^{\mathcal{A}}$:	Game $\text{Toy}_n^{\mathcal{A}}$:
$\mathbf{O} \twoheadleftarrow \mathsf{I}$ for $i = 1$ to n, either: i) $V_i \twoheadleftarrow \{0,1\}^n$ give V_i to $\mathcal{A}^{\mathbf{O}}$ ii) $V_i \twoheadleftarrow \mathcal{A}^{\mathbf{O}}$ return $R^{\mathbf{O}}(V_1, \ldots, V_n)$	$\mathsf{F} \twoheadleftarrow \text{Func}(n, n)$ $C_1 \twoheadleftarrow \{0,1\}^n$; $X_1 \twoheadleftarrow \mathcal{A}^{\mathsf{F}}(C_1)$ $C_2 \twoheadleftarrow \{0,1\}^n$; $X_2 \twoheadleftarrow \mathcal{A}^{\mathsf{F}}(C_2)$ return $\big(X_1 \oplus X_2 \oplus C_2 =$ $\mathsf{F}(C_1 \oplus X_2) \oplus \mathsf{F}(C_2)\big)$

Figure 11: Left: Template for a collision game. Right: A concrete example.

Collider$(R, <)$:
$E \leftarrow R(V_1, \ldots V_n)$; $\sigma \leftarrow <$; $\mathcal{F} \leftarrow (E, \sigma)$ return IsWinnable$(\mathcal{F})$
IsWinnable$(\mathcal{F})$:
$\Omega \leftarrow$ set of rules applicable to $\mathcal{F}$ if $\Omega = \emptyset$: return true pick *rule* $\in \Omega$ if $\big(\textit{rule}(\mathcal{F}) = \bot\big)$: return false return $\big(\exists \mathcal{F}' \in \textit{rule}(\mathcal{F}) : \text{IsWinnable}(\mathcal{F}')\big)$

Figure 12: Decision procedure for the winnability of a collision game. Procedure IsWinnable **is calls itself recursively. Neither the order in which rules are picked in** IsWinnable**, nor that in which subfamilies $\mathcal{F}'$ are analyzed in the recursive call, have been specified. Completeness, soundness and termination are not affected by this order, though concrete performance may be different.**

5 Analyzing Collision Games

In this section, we provide a procedure for deciding whether or not an adversary can win a general class of so-called *collision games*. We start with formal definitions and then give a concrete example.

Collision game. A collision game consists of a sequence of steps and a winning condition. Steps can be of two types: (i) A value is sampled uniformly at random and given to the adversary (who is stateful), or (ii) The adversary outputs a value. The adversary wins the game if certain (in)equalities between the values are met. Fig. 11 (left) describes the general form of a collision game, where R is a relation in the XOR-language describing the winning condition. Alternatively, a collision game can be seen as a tuple $(R, <)$ where $<$ is a total order on a sequence of formal adversary variables $X_1, \ldots, X_k$ and challenger variables $C_1, \ldots, C_\ell$. We define the advantage of an adversary $\mathcal{A}$ in winning such a game as

$$\text{Adv}^{\text{coll}}(\mathcal{A}) := \Pr\big[R\text{-Coll}_{\mathsf{I}}^{\mathcal{A}}\big] .$$

We say that a collision game is *winnable* if there exists an adversary against the game that has a non-negligible advantage. Otherwise, we say that the game is *unwinnable*. To simplify our presentation, in the remainder of this section we focus on collision games that involve a single random oracle in $\text{Func}(n, n)$. We refer to the end of the section for an explanation of how our decision procedure can be generalized to several oracles, including (keyed) permutations.

For the sake of performance, in our decision procedure we seek to express several collision games at once, e.g., games where the order of some steps is not yet specified. To that end, we introduce the notion of a *family of collision games*.

Family of collision games. A pair (E, σ) is called a family of collision games if E is a set of equalities and inequalities in the XOR-language on certain formal variables, and σ is a strict partial order on the variables.

We use the convention that adversary variables are named X_i and challenger variables C_j, and we let V and W be variables of either type. The family of collision games is defined as the set of all games whose order of variables is compatible with the partial order σ. Concretely, we say that a collision game $(R, <)$ belongs to family (E, σ) if relation R coincides with (in)equalities E, and the total order $<$ is compatible with the partial order σ. Formally, $R(V_1, \ldots, V_n) = E$ and $V < W \implies W \not<_\sigma V$. For example, the following family of games contains the Toy game depicted in Fig. 11 (right):

$$E = \{X_1 \oplus X_2 \oplus C_2 = \mathsf{F}(C_1 \oplus X_2) \oplus \mathsf{F}(C_2)\} ,$$
$$\sigma\colon C_1 <_\sigma X_1 \text{ and } X_1 <_\sigma X_2 .$$

Note that the partial order σ does not relate X_2 and C_2. Thus this family also contains another collision game (distinct from the Toy game), where C_2 is given to the adversary *after* it outputs X_2.

We assume that collision games are *purified* in the sense that expressions that are input to F consist of a single variable. To purify a game, AlgoROM recursively replaces equations of the form $\mathsf{F}(e) \oplus e' = 0$, where e and e' are expressions, by $\mathsf{F}(X) \oplus e' = 0$ and $X = e$.

In Fig. 12 we describe our Collider procedure for deciding when a given collision game is winnable. The procedure starts with an input collision game and forms an initial family containing only this game. It then (heuristically) picks a *rule* from the set of *collider rules* Ω defined in Fig. 13, each transforming a family of games into a set of families or $\bot$, and applies it to the given family. If no rule can be picked, the procedure returns "winnable," and in case of $\bot$, it outputs "unwinnable." If neither is the case, Collider recursively applies this procedure to each family of games in the set, and returns "winnable" as long as one of them is declared "winnable," and "unwinnable" otherwise. Note that the choice of *rule* is not non-deterministic: Once a rule is picked, only that rule is applied in that step.

Collider properties. We say that Collider is sound if it always outputs *false* when given an unwinnable collision game. We say that Collider is complete if it always outputs *true* when given a winnable collision game. We say that Collider always terminates if, for any collision game, it outputs a value in finitely many steps.

The remainder of this section is dedicated to the following theorem. We here present the theorem in the asymptotic language, and discuss concrete bounds in the proof.

Theorem 5.1 (Collider). Collider *is sound, complete, and always terminates.*

Peeling: $\dfrac{(\{F(V) \oplus e \bowtie 0\} \cup E, \sigma)}{(\{C \oplus e \bowtie 0\} \cup E[F(V) \mapsto C], \sigma \text{ s.t. } V < C)}$
for fresh C, if $\forall W \neq V$ s.t. $F(W) \in \text{terms}(E) : (V \neq W) \in E$

Assignment: $\dfrac{(\{X \oplus e = 0\} \cup E, \sigma)}{(\{X = e\} \cup E[X \mapsto e], \sigma)}$
if $e := V_1 \oplus \cdots \oplus V_k$ with $\forall i \in [k] : (V_i <_\sigma X) \wedge (X \in \text{vars}(E)) \wedge (F(X) \notin \text{terms}(E))$

Contradiction: $\dfrac{(\{0 \neq 0\} \cup E, \sigma)}{\bot}$

Unpredictability: $\dfrac{\left(\{C \oplus \bigoplus_{i=1}^{k} V_i = 0\} \cup E, \sigma\right)}{\bot}$
if $\forall i \in [k] : V_i <_\sigma C$

Partitioning: $\dfrac{(E, \sigma)}{(E[V \mapsto W], \sigma[V \mapsto W]) \mid (\{V \neq W\} \cup E, \sigma)}$
with $(V = W), (V \neq W) \notin E$

Ordering: $\dfrac{(E, \sigma)}{(E, \sigma \text{ s.t. } V < W) \mid (E, \sigma \text{ s.t. } W < V)}$
if V, W belong to a common equation, $V \not<_\sigma W$, $W \not<_\sigma V$, and $(V \neq W) \in E$

Figure 13: Collider rules Ω. Here $\bowtie \in \{=, \neq\}$. We use X for variables controlled by the adversary and C for random variables sampled in the game. V and W are used for both X and C. Expressions e and e' are in the XOR-language, and $E[e \mapsto e']$ means replacing any occurrence of expression e with expression e' in all equations in E.

Fig. 13 describes the formal semantics of the collider rules. Every such rule expresses how a single family of games (on top of the horizontal line) is replaced by one or more families (below the line), as long as the side conditions are met.

The following lemma guarantees that each rule is sound and complete in the sense that the original family contains a winnable game if and only if at least one of the resulting families does.

Lemma 5.2 (Winnability preservation). *Every rule in Fig. 13, transforming a family of games into one or more families, is such that the original family contains a winnable game if and only if at least one of the resulting families does.*

Proof. We inspect each rule individually.

Peeling: This rule produces a single family of collision games. The oracle query X to F is replaced by a fresh random variable C. The dependency $X <_\sigma C$ is introduced to model the fact that query X must correspond to a game step that precedes the step where the adversary learns C. Clearly, an adversary against the new family of games can be turned into an adversary against the original family. The converse is also true assuming a probability loss, due to the fact that $\mathcal{A}$ has oracle access to F. Say we have an adversary $\mathcal{A}$ against the original family of games, that can win at least one of the games in the family with probability δ. We build an adversary $\mathcal{B}$ that can win at least one of the games in the new family with probability δ/q, where q is the number of queries performed by $\mathcal{A}$ to its oracle. (There is a one-to-one correspondence between the games of the original family and those of the new one. $\mathcal{B}$ will succeed in the game corresponding to the one that $\mathcal{A}$ wins.) First, let us assume w.l.o.g. that $\mathcal{A}$ never repeats a query and that it performs all the queries in E to its oracle F (possibly at the end of the game, when all the unpredictable variables have been given to the adversary). $\mathcal{B}$ will work as follows:
(1) Guess index $i \in [q]$.
(2) Run the game where $\mathcal{A}$ is successful, answering $\mathcal{A}$'s queries with $\mathcal{B}$'s own oracle F, until $\mathcal{A}$ performs the i-th query Q.
(3) Use Q as the value that $\mathcal{B}$ must choose on the game step introduced after the peeling rule, and get C, the value of the fresh variable created during the rule transition.
(4) Answer the i-th query with C, and continue the execution of $\mathcal{A}$.

If the guess i was correct, the value that $\mathcal{A}$ finally chooses for variable V will be Q, thus $\mathcal{B}$ has perfectly simulated $\mathcal{A}$'s game and $\mathcal{B}$ will be successful if $\mathcal{A}$ is.[5] (Guessing the relevant query is necessary because $\mathcal{A}$ may potentially decide to choose what value to give to V based on all query answers.) We conclude that $\mathcal{B}$ will win with probability at least δ/q.
Since q is polynomial, either both families contain a winnable game, or neither does. However, observe that this loss magnifies with every application of the *peeling* rule. The loss will remain polynomial if this rule is applied a constant number of times.

Assignment: This rule captures the fact that if an equation of the form $X \oplus e = 0$ is present, and the value of e is fully determined before the step where X is picked by the adversary, then without loss of generality, we can restrict attention to adversaries that set X to e. Thus, this rule produces a single family of collision games that is identical to the original one. Note that in any possible collision game in the original family, the step where X is produced must follow the steps involving all other variables $V_1, \ldots, V_k$, given the side condition. Since equation $X \oplus V_1 \oplus \cdots \oplus V_k = 0$ must be satisfied, we can assume without loss of generality that the adversary has instantiated the value of X as $V_1 \oplus \cdots \oplus V_k$, a value that is available when choosing X. Therefore, the replacement $X \mapsto V_1 \oplus \cdots \oplus V_k$ is sound. Note that equation $X = e$ is kept, to possibly keep track of a solution to the game. However, variable X disappears from the remaining equations. A side condition of this rule is that X appears in more than one equation. This prevents us from applying this rule more than once (which would prevent termination).

Contradiction: Clearly a family of games containing an equation of the form $0 \neq 0$ does not contain a single winnable game.

Unpredictability: This rule models the fact that an equality involving a random variable that is not "under the control" of the adversary will be satisfied only with negligible probability. Note that all games in the original family contain an equation $C \oplus V_1 \oplus \cdots \oplus V_k = 0$, where variable C is sampled uniformly

[5] Note that the side condition of the peeling rule guarantees that, if $\mathcal{A}$ is successful, Q is different from all other queries to F.

after the value of all other variables V_i has been chosen by $\mathcal{A}$ (because their game step precedes the one where C is provided). The probability that this equation is satisfied is thus $1/2^n$, where n is the output length of the oracle.

Partitioning: This rule produces two families of collision games: One having unified variables $X \mapsto X'$, and the other containing a new equation $X \neq X'$. This is exclusive and exhaustive, and thus, if the original family contains a winnable game, at least one of the resulting families will. For the sake of termination, this rule has the side condition that both $(X = X')$ and $(X \neq X')$ do not appear in the equations of the original family. This prevents us from applying the rule more than once.

Ordering: This rule imposes an order on two incomparable steps of the original family of games. We assume both possible orders in the resulting families, so the rule is sound and complete. For the sake of termination, this rule cannot be applied if σ already imposes an ordering on the variables.

We incur a loss of factor q for each application of the peeling rule, and a loss of factor 2 each time we use partitioning or ordering. Furthermore, we pick up an additive loss of $1/2^n$ (with 2^n the size of the domain and range of the RO), in the unpredictability rule. Assuming that there are at most t function calls in the winning condition of the initial game with at most s variables, we obtain a final bound of at most $O(t2^{2s}q^t/2^n)$, which is asymptotically negligible. □

Proof (of Theorem 5.1). We prove termination, soundness, and completeness separately.

Termination. We show that, on input any collision game (with a finite number of steps, equations and oracle queries), Collider always produces an output in finitely many iterations.

Let μ be the function that maps a family of collision games into $\mathbb{N}^4$, defined as $\mu(E, \sigma) = (n_f, n_\sigma, n_{\bowtie}, n_v)$, where:

- n_f is the number of oracle queries in the equations E;
- n_σ is the number of pairs of variables that are not comparable through σ;
- $n_{\bowtie}$ is the number of pairs of variables (V, W) s.t. $(V = W)$, $(V \neq W) \notin E$;
- n_v is the number of variables that appear in more than one equation in E.

We define $\mu(\bot) = (0, 0, 0, 0)$. Consider the lexicographic order over $\mathbb{N}^4$. By definition $\mu(\mathcal{F}) \geq (0, 0, 0, 0)$ for any family $\mathcal{F}$. Termination follows from the fact that μ is strictly decreasing with respect to every rule in Fig. 13, which can be checked by inspection of every rule.

Note that in the worst-case scenario, Collider will explore all possible ordered partitions of the game variables. For an upper bound k on the number of variables, the algorithm may take time in the order of $k!/2(\log 2)^{k+1}$, the k-th ordered Bell number. The total number of game variables can be upper-bounded by the number of steps in the game plus the number of oracle queries in the equations.

We next show the soundness and completeness of our algorithm. Recall that soundness requires that the algorithm does not introduce additional solutions: if the output is *true*, the game is indeed winnable. On the other hand, completeness guarantees that the algorithm does not miss any solutions: if the output is *false*, the game is indeed unwinnable.

Soundness. We now show that if Collider outputs *true*, then the given game must be winnable. Note that the family of collision games given on the first call to IsWinnable only contains one collision game, the one of interest. This is because σ is set to be the total order $<$ on the existing game variables. Note that Collider outputs *true* if we eventually arrive at a family of collision games where no rule is applicable. We will see that there must be a winnable collision game in such a family. Combined with Lemma 5.2, this implies soundness.

Now consider a family of collision games (E, σ) where no rule is applicable. In these games, σ can be made a total order arbitrarily and all variables can be made different. Since all variables are different, all occurrences of $F(V)$ can be removed by peeling. Furthermore, every equation will have a leading variable which can be found and removed from other equations by assignment. More concretely, all equalities in E must be of the form $V_1 \oplus \cdots \oplus V_k = 0$ where there is a "leading" variable V_j which, according to σ, comes after all other variables in the equation. Also, such a leading variable does not appear in any other equation. Furthermore, all inequalities are of the form $V_1 \oplus \cdots \oplus V_k \neq 0$ where each V_i is not a leading variable of any equation. We claim that the following adversary wins any collision game in the family with overwhelming probability. First, instantiate all non-leading variables uniformly and independently at random. Then, following the order imposed by σ, instantiate the leading variables so that they satisfy their corresponding equation. (Note that all the information necessary to do so will be available.) This mechanism clearly satisfies all equalities in the game. All the inequalities will also be satisfied with overwhelming probability. This is because the probability that one individual inequality is violated can be upper-bounded by $1/2^n$. By the union bound, if there are n_i inequalities, the probability that the described adversary is successful is $1 - n_i/2^n$, which is overwhelming.

Completeness. Finally we show completeness: If Collider outputs *false*, then the given game must be unwinnable. As in the proof of soundness, note that the initial call to IsWinnable is a family of games that only contains the collision game of interest. Thus completeness follows directly from Lemma 5.2 and the fact that the two collider rules that output $\bot$ do so from a family of games that does not contain a single winnable game. This observation is obvious for the *contradiction* rule. For the *unpredictability* rule, note that all games in the original family contain an equation $C \oplus V_1 \oplus \cdots \oplus V_k = 0$, where variable C is sampled uniformly after the value of all other variables V_i is fixed. The probability that this occurs is negligible and out of the control of the adversary. □

Analyzing collisions is the main component of the proof strategy used by AlgoROM. As a result, the corollary below follows from Theorem 5.1.

Corollary 5.3. AlgoROM *is sound and complete with respect to the proof technique, and always terminates. That is, if a construction has a collision-based game-hopping proof, then* AlgoROM *will find it, and it will never classify an insecure scheme as secure.*

We emphasize that this corollary does not entail that the tool is complete with respect to classifying a scheme as either secure or insecure: There may exist constructions that are secure, but for which the indistinguishability-up-to-collision proof technique, which AlgoROM automates, cannot be applied.

An example. We present a concrete example to illustrate the rules in Fig. 13. Consider the Toy game in Fig. 11 (right). As described in Fig. 12, after purification, we start with the family of collision games below, which only contains the Toy game at this point. We represent the strict partial order σ as an acyclic graph, where there is a path between nodes V and W if and only if $V <_\sigma W$.

$$E = \left\{ \begin{array}{c} X_1 \oplus X_2 \oplus C_2 = \\ \mathsf{F}(X_3) \oplus \mathsf{F}(C_2)\,, \\ X_3 = C_1 \oplus X_2 \end{array} \right\} \qquad \begin{array}{ccccc} C_1 & \to & X_1 & \to & C_2 \to X_2 \\ & & & & X_3 \end{array}$$

We first apply the *partitioning* rule on X_3 and C_2 and get two new systems, one where we apply replacement $X_3 \mapsto C_2$, and one with an additional inequality $X_3 \neq C_2$. Let us analyze the former:

$$E = \left\{ \begin{array}{c} X_1 \oplus X_2 \oplus C_2 = 0\,, \\ C_2 = C_1 \oplus X_2 \end{array} \right\} \qquad C_1 \to X_1 \to C_2 \to X_2$$

We can now apply the *assignment* rule to the first equation and leading variable X_2 (given that $X_1 <_\sigma X_2$ and $C_2 <_\sigma X_2$). We get:

$$E = \left\{ \begin{array}{c} X_2 = X_1 \oplus C_2\,, \\ C_1 \oplus X_1 = 0 \end{array} \right\} \qquad C_1 \to X_1 \to C_2 \to X_2$$

Now, we apply the *assignment* rule on the second equation and leading variable X_1 (given that $C_1 <_\sigma X_1$):

$$E = \left\{ \begin{array}{c} X_2 = C_1 \oplus C_2\,, \\ X_1 = C_1 \end{array} \right\} \qquad C_1 \to X_1 \to C_2 \to X_2$$

At this point no more rules can be applied, which means that all collision games in this family are winnable. Indeed, each of the two equalities has a *leading* variable that goes after all other variables in its equation and which does not appear in any other equation. An adversary could instantiate all variables that do not lead any equation uniformly at random (in this case there are no such variables), and then instantiate the leading variables in order as their corresponding equation dictates. In this case, the adversary chooses $X_1 := C_1$ and then $X_2 := C_1 \oplus C_2$. We conclude that the original collision game is also winnable.

If the branch with $X_3 \mapsto C_2$ had resulted in an abortion, we would still need to explore the other branch with $X_3 \neq C_2$. Such a branch could be winnable even if the previous one was not, due to the presence of inequalities in the equations. However, for this specific example, this other branch is not winnable. We invite the reader to verify this fact.

Supporting permutations. We conclude by discussing how our rules can be extended to support multiple oracles including permutations and block ciphers. Having multiple oracles is easy and would simply require having one peeling rule per oracle. For permutations, we need to account for the fact that they can also be executed backwards. This allows the adversary to choose the value of $P(e)$ at the price of not being able to choose the value for e, e.g., in order to set $P(e) := X$ one can set $e := P^-(X)$. Permutations can be captured with a non-deterministic purification process (leading to multiple collision games that must all be analyzed). In particular, an expression like $P^\pm(e) \oplus e' = 0$ would be purified in both the following ways (for a fresh variable X): (i) $P^\pm(X) \oplus e' = 0$ and $X = e$; or (ii) $P^\mp(X) = e$ and $X \oplus e' = 0$. This logic can be easily extended to capture block ciphers, as well as multi-arity oracles.

6 Selected Experimental Results

We next give a selection of results obtained through AlgoROM and discuss how they relate to prior works; see Table 1,[6] where entries typeset in green represent new results obtained using our tool. We evaluate AlgoROM by corroborating existing results, and use it to prove constructions secure which, to the best of our knowledge, have not been studied before. We have experimented with many configurations using the tool; those in Table 1 are the ones that AlgoROM found to be secure while minimizing keying material and round complexity. Note that AlgoROM is flexible and can be used to analyze many more schemes and configurations.

As mentioned in the introduction, for several applications, many existing works focus on the case of independent round functions and permutations, despite this being a less accurate model of constructions used in practice, which often use identical round functions. We believe that this is because, in the latter setting, the number of collisions to be analyzed grows quadratically (rather than linearly) in the number of rounds, resulting in an overwhelming number of cases to be resolved by hand. Furthermore, the complexity of the equations also grows with the number of rounds, making their manual analyses even more error-prone. In that sense, we believe that pen-and-paper proofs that use indistinguishability-up-to-collision would be very difficult, if not practically impossible, for most constructions analyzed by our tool, not to mention that they are error-prone and less reliable.[7]

We note, however, that manual proofs may still be possible via novel proof techniques that, for example, exploit the specific structure of a particular scheme (even if the tool runs for a long time). For example, a proof might first identify a useful property of, say, 4 contiguous rounds of Feistel in a Feistel-based construction, and then exploit this property to prove the security of the full scheme.

AlgoROM is written in OCaml, and its full source code is available under

`https://gitlab.com/ambrona/algorom` .

The tool contains a series of early-decision and termination rules to speed up the analysis. All experiments were run on a 10-core 3.2GHz Apple M1 Pro and 16GB of RAM, though AlgoROM only uses a single core. Collider is the computationally most intensive procedure, especially with multiple rounds of queries. Other demanding steps include projections (e.g., in SPN), and RKA/KDM functions with multiple key components. We give a selection of timings for our experiments below. In Appendix A, we provide a sample output produced by our tool when run on 3-round KAF with different round functions and the same key in the CPA model.

[6] AlgoROM can produce a detailed output explaining various steps of the analysis. A full proof transcript, however, often involves a large number of case distinctions, and thus we focus on the final results delivered.

[7] To exemplify this growth in complexity, observe that AlgoROM takes time in the order of milliseconds to confirm known results from the literature, whereas it takes much longer for the more challenging constructions not studied before (KC-CPA security of 10-round KAF took approximately 165 hours).

Table 1: Summary of our main CPA and CCA-security results. "O" stands for oracle, "S" indicates using the same function in each round, and "D" different round functions. New results are highlighted in green, and non-highlighted results are contained in the literature. We have presented key schedules with minimal keying material. See the text for a selection of timings. The exponents in the 2-SP column denote the number of rounds, plus one. A diamond ◊ indicates no result, capping the experiment run-time to one week.

	O	LR	KAF	EM	2-SP
IND-CPA	S	$[K,0,K\oplus\Delta]$	$[K,0,K,K]$	$[K,K]$	$[[K_1,K_2]^4]$
	D	$[K,0,K]$	$[K,0,K]$	$[K,K]$	$[[K_1,K_2]^4]$
RK-CPA	S	$[K,0,K,K\oplus\Delta]$	$[K,0,0,K,K]$	$[K,K,K]$	$[[K_1,K_2]^5]$
	D	$[K,0,K]$	$[K,0,0,K]$	$[K,K,K]$	$[[K_1,K_2]^5]$
KD-CPA	S	$[K,0,K\oplus\Delta]$	$[K,0,0,0,0,0,0,K\oplus\Delta]$	$[K,K,K\oplus\Delta,K\oplus\Delta]$	$[[K_1,K_2]^5]$
	D	$[K,0,K]$	$[K,0,0,0,K]$	$[K,K,K]$	$[[K_1,K_2]^5]$
KC-CPA	S	$[K,0,K,K\oplus\Delta]$	$[K,0,0,0,0,0,0,K,K\oplus\Delta]$	$[K,K,K,K\oplus\Delta,K]$	◊
	D	$[K,0,K]$	$[K,0,0,0,K,K]$	$[K,K,K,K]$	$[[K_1,K_2]^7]$
IND-CCA	S	$[K,0,0,K\oplus\Delta]$	$[K,0,K,K]$	$[K,K]$	$[[K_1,K_2]^5]$
	D	$[K,0,0,K]$	$[K,0,0,K]$	$[K,K]$	$[[K_1,K_2]^5]$
RK-CCA	S	$[K,0,K,K\oplus\Delta]$	$[K,0,0,0,K,K]$	$[K,K,K,K]$	$[[K_1,K_2]^7]$
	D	$[K,0,0,K]$	$[K,0,0,0,K,K]$	$[K,K,K,K]$	$[[K_1,K_2]^6]$
KD-CCA	S	$[K,0,0,K\oplus\Delta]$	$[K,0,0,0,0,0,0,K\oplus\Delta]$	$[K,K,K\oplus\Delta,K\oplus\Delta]$	$[[K_1,K_2]^7]$
	D	$[K,0,0,K]$	$[K,0,0,0,0,K]$	$[K,K,K]$	$[[K_1,K_2]^6]$
KC-CCA	S	$[K,0,K,K\oplus\Delta]$	◊	$[K,K,K,K\oplus\Delta,K,K,K]$	◊
	D	$[K,0,0,K]$	$[K,K,0,K,K,0,K,K]$	$[K,K,K,K,K]$	◊

LR ciphers. AlgoROM proves the following results for LR ciphers.

IND: This notion was studied in [56] for identical round functions, showing that the construction with key schedule $[K_1, K_2, K_3]$ is CPA-secure (resp., $[K_1, K_2, K_3, K_4]$ is CCA-secure). AlgoROM confirms these results and shows that they extend, with identical (resp., independent) round functions to the simple key schedule $[K, 0, K \oplus \Delta]$ (resp., $[K, 0, K]$) for CPA-security, and to $[K, 0, 0, K \oplus \Delta]$ (resp., $[K, 0, 0, K]$) for CCA-security.

RK: This notion was studied in [10] for identical round functions, showing that the construction with key schedule $[K_1, K_2, K_2]$ is CPA-secure (resp., $[K_1, K_2, K_1, K_2]$ is CCA-secure) for claw-free and switch-free RK-derivation functions. AlgoROM confirms these results, when restricting the RK-functions to key offsets (i.e., XOR-RKA-security). It finds that they extend, with identical round functions to the key schedule $[K, 0, K, K \oplus \Delta]$ for CCA-security, and with independent functions to $[K, 0, K]$ for CPA-security and to $[K, 0, 0, K]$ for CCA-security. The results for identical round functions are new.

KD: To the best of our knowledge, LR ciphers have not yet been studied in this setting. AlgoROM proves CPA-security with identical (resp., independent) round functions and key schedule $[K, 0, K \oplus \Delta]$ (resp., $[K, 0, K]$), and CCA-security with key schedule $[K, 0, 0, K \oplus \Delta]$ (resp., $[K, 0, 0, K]$).

KC: AlgoROM proves that the construction with identical round functions and key schedule $[K, 0, K, K \oplus \Delta]$ is CCA-secure, and the one with independent round functions and key schedule $[K, 0, K]$ is CPA-secure (resp., $[K, 0, 0, K]$ is CCA-secure). To the best of our knowledge, these results are new.

AlgoROM analyzes the LR configurations shown in Table 1 in a matter of milliseconds. The hardest case is the KC-CCA security analysis, which took 53ms. All constructions are proven by replacing the 2nd and 3rd round functions.

KAF ciphers. We obtain the following results for KAF ciphers.

IND: This notion was studied in [53] for independent round functions and keys. AlgoROM proves CCA-security with identical round functions and key schedule $[K, 0, K, K]$, and CPA-security (resp., CCA-security) with independent round functions and key schedule $[K, 0, K]$ (resp., $[K, 0, 0, K]$).

RK: This notion was studied in [45] for identical round functions and correlated round keys. AlgoROM proves CPA-security with identical (resp., independent) round functions and key schedule $[K, 0, 0, K, K]$ (resp., $[K, 0, 0, K]$), and CCA-security with key schedule $[K, 0, 0, 0, K, K]$ in both settings. To the best of our knowledge, these results are new.[8]

KD: This notion was studied in [38] for identical round functions and correlated round keys, showing that a 4-round construction is CCA-secure. The authors require the KD-derivation functions to be offset-free, claw-free and XOR-offset-free. Our results, although less general, include the case of XOR-combinations of keys, which escapes their requirements. AlgoROM proves CCA-security for identical round functions and key schedule $[K, 0, 0, 0, 0, 0, 0, K \oplus \Delta]$, and CPA-security (resp., CCA-security) for independent functions and key schedule $[K, 0, 0, 0, K]$ (resp., $[K, 0, 0, 0, K, K]$). All these results are new.

KC: AlgoROM proves CPA-security for identical (resp., independent) round functions and key schedule $[K, 0, 0, 0, 0, 0, 0, 0, K, K \oplus \Delta]$ (resp., $[K, 0, 0, 0, K, K]$), and CCA-security for independent round functions and key schedule $[K, K, 0, K, K, 0, K, K]$. All these results are new.

Analyzing KAF constructions turns out to be somewhat challenging for AlgoROM. (RK-)CPA/CCA analysis is in the order of milliseconds. KD-CPA/CCA took 4s. The most complex case (which is also the hardest among all experiments performed) is the KC-CPA security analysis of the 10-round KAF with identical round functions, which approximately took 165 hours. We have not been able to prove CCA results in the setting of identical round functions (limiting tests to at most one week). With further optimizations, or a multi-core extension, KC-CCA-security with identical round functions may be within reach.

EM ciphers. We obtain the following results for EM ciphers.

IND: It is well-known that the 1-round construction with key schedule $[K, K]$ is CCA-secure [35]. AlgoROM confirms this.

RK: This notion was studied in [29, 40] for independent round permutations, showing that the construction with key schedule $[K, K, K]$ (resp., $[K, K, K, K]$) is XOR-RKA-CPA-secure (resp., CCA-secure). AlgoROM confirms these results, and proves the same configurations secure with identical round permutations. The latter results are new.

[8] We note that [45] achieve their results under conditions that are incomparable to ours. Their most efficient construction has fewer rounds, but a more complex (and highly non-linear) key schedule, and uses whitening keys, which we don't. When restricting to linear key schedules, their general result does not include the very simple key schedule we have found.

KD: This notion was studied in [39] for independent round permutations, showing that the construction with key schedule $[K_1, K_2, K_3]$ (resp., $[K_1, K_2]$) is CCA-secure for offset-free (and claw-free) KD-derivation functions. They also study the case of identical round functions, proving that $[K_1, K_2, K_3]$ gives a CCA-secure construction for claw-free and XOR-offset-free KD-derivation functions. AlgoROM confirms the former set of results; note that, again, the XOR-combinations of keys escapes the latter result. AlgoROM proves CCA-security for identical (resp., independent) round functions and key schedule $[K, K, K \oplus \Delta, K \oplus \Delta]$ (resp., $[K, K, K]$). These results are new.

KC: This notion was studied in [30] for independent round permutations, showing that the construction with key schedule $[K, K, K, K]$ is CCA-secure when the KC-derivation functions are restricted to offsets [30, Theorem 2]. The authors use bad-event analysis to replace the 3rd permutation in encryption and the 1st (furthermost) permutation in decryption with FROs.

Using the tool, however, we have identified a collision not considered in [30]. AlgoROM reports that a collision can be triggered after these replacements by querying encryption on $(K \oplus \Delta, M)$ to get a ciphertext C, and then decryption on $(K \oplus \Delta \oplus C' \oplus C, C')$ with a $C' \neq C$. Indeed, the encryption query results in the 3rd permutation being queried on $Q \coloneqq \mathsf{P}_2(\mathsf{P}_1(M \oplus K \oplus \Delta) \oplus K \oplus \Delta)$ in the forward direction, with the resulting ciphertext being $C = \mathsf{P}_3(Q) \oplus K \oplus \Delta$. The decryption query results in the 3rd permutation being queried on $C' \oplus K \oplus \Delta \oplus C' \oplus C = K \oplus \Delta \oplus C$ in the *backwards* direction. Expanding C we get $K \oplus \Delta \oplus \mathsf{P}_3(Q) \oplus K \oplus \Delta = \mathsf{P}_3(Q)$, i.e., this backwards query to the 3rd permutation should result in Q. This is not guaranteed in the ideal world and is therefore a collision whose probability must be bounded, but was not considered in [30]. (In fact, the proof strategy suggested in [30] also fails for RK-CCA security.)

In a similar vein, it may be that also other proofs involve missing cases which, although not invalidating the stated theorems, have been overlooked in the analyses. The tool, on the other hand, exhaustively lists all cases that need to be considered. This further highlights the possibility of errors in hand-made proofs, and how an automatic tool can help in avoiding them.

AlgoROM proves CPA-security with identical (resp., independent) round functions and key schedule $[K, K, K, K \oplus \Delta, K]$ (resp., $[K, K, K, K]$), and CCA-security with $[K, K, K, K \oplus \Delta, K, K, K]$ (resp., $[K, K, K, K, K]$). All these results are new.

AlgoROM analyzes the listed EM constructions in a matter of millisecond. The hardest case is the KC-CCA-security analysis, which took 127ms. Different proofs use different replacements. For example, the KC-CCA proof replaces the 3rd round permutation in encryption and the 2nd round permutation in decryption.

SP networks. AlgoROM proves the following results for SP networks. Note that, due to the restrictions in our grammar, the class of diffusion layers that we consider is confined to binary matrices. Furthermore, the complexity of SPNs grows quickly and we are able to analyze only width-2 and width-3 constructions, the latter only for IND-security. Except for the configurations in the IND case, all other results that we obtain are new.

IND: AlgoROM proves CCA-security of a width-2 SPN with four rounds, repeating round keys $[K_1, K_2]$, identical round permutations, and alternating diffusion with matrices

$$M_1 \coloneqq \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad M_2 \coloneqq \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

With one less round we get CPA-security. AlgoROM can also handle width-3 SPNs, proving CCA-security with 5 rounds, repeating keys $[K_1, K_2, K_3]$, distinct permutations, and diffusion matrices

$$M_1 \coloneqq \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \; M_2 \coloneqq \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \text{ and } \; M_3 \coloneqq \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

RK: AlgoROM proves the first CPA-security (resp., CCA-security) result for SPNs, for a 4-round (resp., 6-round), width-2 SPN with identical permutations, key schedule and diffusion layers as above. In the CCA case, we can reduce the number of rounds to 5 with $5 \cdot 2 = 10$ different round permutations.

KD: AlgoROM proves that the same construction that is proven RK-ATK-secure is also KD-ATK-secure, with ATK $\in \{\text{CPA}, \text{CCA}\}$.

KC: The analysis of SPNs in the KC-setting is complex. AlgoROM is only able to show CPA-security with different permutations in 6 rounds.

Analyzing SPNs is challenging for AlgoROM, and we focused on width-2 and width-3 constructions. The IND-CPA/CCA analyses of width-2 constructions are in the order of milliseconds. RK-CCA-security took approximately 400s, and KD-CCA 700s. The KC-CPA analysis of the 6-round width-2 SPN with different permutations took about 4300s. The CCA-security analysis of the width-3 SPN with different (resp., identical) round permutations took 104s (277s).

Other experiments. We have used AlgoROM to experiment with other novel designs, including SP networks with missing S-boxes, which are relevant for *MPC-friendly* symmetric primitives [4, 44]. As an example, the tool shows that 4-round SP of width 2 (with the above 2×2 diffusion layers) is IND-CCA secure even if 5 out of the 8 S-boxes are removed (the 2nd round left, 3rd round right, and 4th round left S-boxes alone are sufficient for security). We stress that AlgoROM can be used to explore alternative design patterns for many more existing schemes (e.g., the Hades strategy [44]).

We have also applied our tool to the CR and RK/KD/KC-CPA-security of the PGV hashes, as well as Merkle–Damgård with a fixed number of message blocks (and no strengthening) and the Sponge construction. In particular, the tool confirms the collision-resistance security of the 12 secure PGV modes [21, 62].

7 Conclusion and Future Directions

Automation is a prominent direction in cryptologic research, and there has been a long series of works in this area. There are important motivations for this: avoiding mistakes that creep in when proofs are done by hand, enabling proofs for constructions that are simply too complex and laborious to be analyzed manually, and exploring richer design spaces. AlgoROM touches on all these aspects, and we believe it is a valuable first step for the automation of meaningful proofs of security in standard (i.e., computational) idealized models. Specifically, we have used the tool to identify a flaw in a previous proof [30], we have proven several new security results,

and we have explored the design space of existing constructions for new round-function and round-key configurations in different security models. The analyses that AlgoROM performs can involve a large number of complex case distinctions that exceed what is possible by hand. (Even manually verifying proof transcripts produced by the tool can become intractable.) As noted above, this is especially the case when analyzing practical constructions that use the same round function or permutation in every round.

We believe that our approach can be extended to study other primitives or security notions. Authenticated encryption, for example, is a natural next step. As mentioned above, the tool currently supports constructions with an arbitrary but *fixed* number of rounds. Extending it to deal with an unbounded number of rounds may be possible by introducing new logic (e.g., inductive reasoning).

Another natural extension of our work would be to automate other proof techniques (beyond the game-based approach) such as the H-coefficient and the expectation methods [26, 49, 61], which can potentially deliver tighter bounds. We believe that the techniques underlying Collider can be adapted to do so.

Acknowledgments

We thank all our reviewers for their time and valuable feedback. Most of this work was performed while Miguel Ambrona was employed by NTT Laboratories. Pooya Farshim was supported in part by EPSRC grant EP/V034065/1. Patrick Harasser was funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297.

References

[1] Martín Abadi and Phillip Rogaway. 2002. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Journal of Cryptology* 15, 2 (March 2002), 103–127. https://doi.org/10.1007/s00145-001-0014-7

[2] William Aiello, Mihir Bellare, Giovanni Di Crescenzo, and Ramarathnam Venkatesan. 1998. Security Amplification by Composition: The Case of Doubly-Iterated, Ideal Ciphers. In *CRYPTO'98 (LNCS, Vol. 1462)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 390–407. https://doi.org/10.1007/BFb0055743

[3] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. 2013. Using SMT solvers to automate design tasks for encryption and signature schemes. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 399–410. https://doi.org/10.1145/2508859.2516718

[4] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 430–454. https://doi.org/10.1007/978-3-662-46800-5_17

[5] Miguel Ambrona, Gilles Barthe, Romain Gay, and Hoeteck Wee. 2017. Attribute-Based Encryption in the Generic Group Model: Automated Proofs and New Constructions. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 647–664. https://doi.org/10.1145/3133956.3134088

[6] Miguel Ambrona, Gilles Barthe, and Benedikt Schmidt. 2016. Automated Unbounded Analysis of Cryptographic Constructions in the Generic Group Model. In *EUROCRYPT 2016, Part II (LNCS, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Heidelberg, 822–851. https://doi.org/10.1007/978-3-662-49896-5_29

[7] Miguel Ambrona, Pooya Farshim, and Patrick Harasser. 2024. AlgoROM. https://gitlab.com/ambrona/algorom

[8] R. Anderson and M. Kuhn. 1997. Low Cost Attacks on Tamper Resistant Devices. In *5th Security Protocols Workshop (LNCS, Vol. 1361)*. Springer, Heidelberg, 125–136.

[9] Benny Applebaum. 2016. Garbling XOR Gates "For Free" in the Standard Model. *Journal of Cryptology* 29, 3 (July 2016), 552–576. https://doi.org/10.1007/s00145-015-9201-9

[10] Manuel Barbosa and Pooya Farshim. 2015. The Related-Key Analysis of Feistel Constructions. In *FSE 2014 (LNCS, Vol. 8540)*, Carlos Cid and Christian Rechberger (Eds.). Springer, Heidelberg, 265–284. https://doi.org/10.1007/978-3-662-46706-0_14

[11] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. 2013. Fully automated analysis of padding-based encryption in the computational model. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 1247–1260. https://doi.org/10.1145/2508859.2516663

[12] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. 2014. Automated Analysis of Cryptographic Assumptions in Generic Group Models. In *CRYPTO 2014, Part I (LNCS, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Heidelberg, 95–112. https://doi.org/10.1007/978-3-662-44371-2_6

[13] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. 2015. Strongly-Optimal Structure Preserving Signatures from Type II Pairings: Synthesis and Lower Bounds. In *PKC 2015 (LNCS, Vol. 9020)*, Jonathan Katz (Ed.). Springer, Heidelberg, 355–376. https://doi.org/10.1007/978-3-662-46447-2_16

[14] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. 2015. Automated Proofs of Pairing-Based Cryptography. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 1156–1168. https://doi.org/10.1145/2810103.2813697

[15] Mathieu Baudet, Véronique Cortier, and Steve Kremer. 2005. Computationally Sound Implementations of Equational Theories Against Passive Adversaries. In *ICALP 2005 (LNCS, Vol. 3580)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, Heidelberg, 652–663. https://doi.org/10.1007/11523468_53

[16] Mihir Bellare and Tadayoshi Kohno. 2003. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In *EUROCRYPT 2003 (LNCS, Vol. 2656)*, Eli Biham (Ed.). Springer, Heidelberg, 491–506. https://doi.org/10.1007/3-540-39200-9_31

[17] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. https://doi.org/10.1145/168588.168596

[18] Mihir Bellare and Phillip Rogaway. 2006. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT 2006 (LNCS, Vol. 4004)*, Serge Vaudenay (Ed.). Springer, Heidelberg, 409–426. https://doi.org/10.1007/11761679_25

[19] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2007. Sponge Functions. ECRYPT hash workshop.

[20] Eli Biham. 1994. New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract). In *EUROCRYPT'93 (LNCS, Vol. 765)*, Tor Helleseth (Ed.). Springer, Heidelberg, 398–409. https://doi.org/10.1007/3-540-48285-7_34

[21] John Black, Phillip Rogaway, and Thomas Shrimpton. 2002. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Heidelberg, 320–335. https://doi.org/10.1007/3-540-45708-9_21

[22] John Black, Phillip Rogaway, and Thomas Shrimpton. 2003. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In *SAC 2002 (LNCS, Vol. 2595)*, Kaisa Nyberg and Howard M. Heys (Eds.). Springer, Heidelberg, 62–75. https://doi.org/10.1007/3-540-36492-7_6

[23] Florian Böhl, Gareth T. Davies, and Dennis Hofheinz. 2014. Encryption Schemes Secure under Related-Key and Key-Dependent Message Attacks. In *PKC 2014 (LNCS, Vol. 8383)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 483–500. https://doi.org/10.1007/978-3-642-54631-0_28

[24] Jan Camenisch and Anna Lysyanskaya. 2001. An Identity Escrow Scheme with Appointed Verifiers. In *CRYPTO 2001 (LNCS, Vol. 2139)*, Joe Kilian (Ed.). Springer, Heidelberg, 388–407. https://doi.org/10.1007/3-540-44647-8_23

[25] Brent Carmer and Mike Rosulek. 2016. Linicrypt: A Model for Practical Cryptography. In *CRYPTO 2016, Part III (LNCS, Vol. 9816)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 416–445. https://doi.org/10.1007/978-3-662-53015-3_15

[26] Shan Chen and John P. Steinberger. 2014. Tight Security Bounds for Key-Alternating Ciphers. In *EUROCRYPT 2014 (LNCS, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, Heidelberg, 327–350. https://doi.org/10.1007/978-3-642-55220-5_19

[27] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. 2013. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science* 492 (2013), 1–39.

[28] Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. 2018. Provable Security of (Tweakable) Block Ciphers Based on Substitution-Permutation Networks. In *CRYPTO 2018, Part I (LNCS, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 722–753. https://doi.org/10.1007/978-3-319-96884-1_24

[29] Benoit Cogliati and Yannick Seurin. 2015. On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 584–613. https://doi.org/10.1007/978-3-662-46800-5_23

[30] Aisling Connolly, Pooya Farshim, and Georg Fuchsbauer. 2019. Security of Symmetric Primitives against Key-Correlated Attacks. *IACR Trans. Symm. Cryptol.* 2019, 3 (2019), 193–230. https://doi.org/10.13154/tosc.v2019.i3.193-230

[31] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. 2008. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *ACM CCS 2008*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, 371–380. https://doi.org/10.1145/1455770.1455817

[32] Ivan Damgård. 1988. Collision Free Hash Functions and Public Key Signature Schemes. In *EUROCRYPT'87 (LNCS, Vol. 304)*, David Chaum and Wyn L. Price (Eds.). Springer, Heidelberg, 203–216. https://doi.org/10.1007/3-540-39118-5_19

[33] Ivan Damgård. 1990. A Design Principle for Hash Functions. In *CRYPTO'89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, 416–427. https://doi.org/10.1007/0-387-34805-0_39

[34] Yevgeniy Dodis, Jonathan Katz, John Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. 2017. Provable Security of Substitution-Permutation Networks. Cryptology ePrint Archive, Report 2017/016. https://eprint.iacr.org/2017/016.

[35] Orr Dunkelman, Nathan Keller, and Adi Shamir. 2012. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 336–354. https://doi.org/10.1007/978-3-642-29011-4_21

[36] Shimon Even and Oded Goldreich. 1983. On the Power of Cascade Ciphers. In *CRYPTO'83*, David Chaum (Ed.). Plenum Press, New York, USA, 43–50.

[37] Shimon Even and Yishay Mansour. 1997. A Construction of a Cipher from a Single Pseudorandom Permutation. *Journal of Cryptology* 10, 3 (June 1997), 151–162. https://doi.org/10.1007/s001459900025

[38] Pooya Farshim, Louiza Khati, Yannick Seurin, and Damien Vergnaud. 2021. The Key-Dependent Message Security of Key-Alternating Feistel Ciphers. In *CT-RSA 2021 (LNCS, Vol. 12704)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, 351–374. https://doi.org/10.1007/978-3-030-75539-3_15

[39] Pooya Farshim, Louiza Khati, and Damien Vergnaud. 2017. Security of Even–Mansour Ciphers under Key-Dependent Messages. *IACR Trans. Symm. Cryptol.* 2017, 2 (2017), 84–104. https://doi.org/10.13154/tosc.v2017.i2.84-104

[40] Pooya Farshim and Gordon Procter. 2015. The Related-Key Security of Iterated Even-Mansour Ciphers. In *FSE 2015 (LNCS, Vol. 9054)*, Gregor Leander (Ed.). Springer, Heidelberg, 342–363. https://doi.org/10.1007/978-3-662-48116-5_17

[41] Horst Feistel. 1973. Cryptography and computer privacy. *Scientific American* 228, 5 (1973), 15–23.

[42] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. 2016. Automated Proofs of Block Cipher Modes of Operation. *J. Autom. Reason.* 56, 1 (2016).

[43] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. How to Construct Random Functions (Extended Abstract). In *25th FOCS*. IEEE Computer Society Press, 464–479. https://doi.org/10.1109/SFCS.1984.715949

[44] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. 2020. On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In *EUROCRYPT 2020, Part II (LNCS, Vol. 12106)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 674–704. https://doi.org/10.1007/978-3-030-45724-2_23

[45] Chun Guo. 2019. Understanding the Related-Key Security of Feistel Ciphers From a Provable Perspective. *IEEE Transactions on Information Theory* 65, 8 (2019).

[46] Shai Halevi and Hugo Krawczyk. 2007. Security under key-dependent inputs. In *ACM CCS 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM Press, 466–475. https://doi.org/10.1145/1315245.1315303

[47] Shoichi Hirose. 2006. Some Plausible Constructions of Double-Block-Length Hash Functions. In *FSE 2006 (LNCS, Vol. 4047)*, Matthew J. B. Robshaw (Ed.). Springer, Heidelberg, 210–225. https://doi.org/10.1007/11799313_14

[48] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. 2015. Automated Analysis and Synthesis of Authenticated Encryption Schemes. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 84–95. https://doi.org/10.1145/2810103.2813636

[49] Viet Tung Hoang and Stefano Tessaro. 2016. Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security. In *CRYPTO 2016, Part I (LNCS, Vol. 9814)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 3–32. https://doi.org/10.1007/978-3-662-53018-4_1

[50] Russell Impagliazzo and Steven Rudich. 1989. Limits on the Provable Consequences of One-Way Permutations. In *21st ACM STOC*. ACM Press, 44–61. https://doi.org/10.1145/73007.73012

[51] Lars R. Knudsen. 1993. Cryptanalysis of LOKI91. In *AUSCRYPT'92 (LNCS, Vol. 718)*, Jennifer Seberry and Yuliang Zheng (Eds.). Springer, Heidelberg, 196–208. https://doi.org/10.1007/3-540-57220-1_62

[52] Steve Kremer and Laurent Mazaré. 2007. Adaptive Soundness of Static Equivalence. In *ESORICS 2007 (LNCS, Vol. 4734)*, Joachim Biskup and Javier López (Eds.). Springer, Heidelberg, 610–625. https://doi.org/10.1007/978-3-540-74835-9_40

[53] Rodolphe Lampe and Yannick Seurin. 2015. Security Analysis of Key-Alternating Feistel Ciphers. In *FSE 2014 (LNCS, Vol. 8540)*, Carlos Cid and Christian Rechberger (Eds.). Springer, Heidelberg, 243–264. https://doi.org/10.1007/978-3-662-46706-0_13

[54] Moses Liskov, Ronald L. Rivest, and David Wagner. 2002. Tweakable Block Ciphers. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Heidelberg, 31–46. https://doi.org/10.1007/3-540-45708-9_3

[55] Michael Luby and Charles Rackoff. 1986. How to Construct Pseudo-Random Permutations from Pseudo-Random Functions (Abstract). In *CRYPTO'85 (LNCS, Vol. 218)*, Hugh C. Williams (Ed.). Springer, Heidelberg, 447. https://doi.org/10.1007/3-540-39799-X_34

[56] Michael Luby and Charles Rackoff. 1988. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.* 17, 2 (1988).

[57] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. 2014. Automated Analysis and Synthesis of Block-Cipher Modes of Operation. In *CSF 2014 Computer Security Foundations Symposium*, Anupam Datta and Cedric Fournet (Eds.). IEEE Computer Society Press, 140–152. https://doi.org/10.1109/CSF.2014.18

[58] Ian McQuoid, Mike Rosulek, and Lawrence Roy. 2021. Batching Base Oblivious Transfers. In *ASIACRYPT 2021, Part III (LNCS, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Heidelberg, 281–310. https://doi.org/10.1007/978-3-030-92078-4_10

[59] Catherine Meadows. 2020. Symbolic and Computational Reasoning About Cryptographic Modes of Operation. Cryptology ePrint Archive, Report 2020/794. https://eprint.iacr.org/2020/794.

[60] Ralph C. Merkle. 1990. One Way Hash Functions and DES. In *CRYPTO'89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, 428–446. https://doi.org/10.1007/0-387-34805-0_40

[61] Jacques Patarin. 2009. The "Coefficients H" Technique (Invited Talk). In *SAC 2008 (LNCS, Vol. 5381)*, Roberto Maria Avanzi, Liam Keliher, and Francesco Sica (Eds.). Springer, Heidelberg, 328–345. https://doi.org/10.1007/978-3-642-04159-4_21

[62] Bart Preneel, René Govaerts, and Joos Vandewalle. 1994. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *CRYPTO'93 (LNCS, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Heidelberg, 368–378. https://doi.org/10.1007/3-540-48329-2_31

[63] Itsaka Rakotonirina, Miguel Ambrona, Alejandro Aguirre, and Gilles Barthe. 2022. Symbolic Synthesis of Indifferentiability Attacks. In *ASIACCS 22*, Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako (Eds.). ACM Press, 667–681. https://doi.org/10.1145/3488932.3497759

[64] Claude E. Shannon. 1949. Communication theory of secrecy systems. *Bell Systems Technical Journal* 28, 4 (1949), 656–715.

[65] Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332. https://eprint.iacr.org/2004/332.

[66] Dominique Unruh. 2010. The impossibility of computationally sound XOR. Cryptology ePrint Archive, Report 2010/389. https://eprint.iacr.org/2010/389.

A Sample Output

The source code of AlgoROM is available under [7]. `main.ml` is the main test file. `test_lr.ml`, `test_kaf.ml`, `test_em.ml`, and `test_sp.ml`, respectively, contain our experiments for the LR, KAF, EM, and SPN ciphers. Our other experiments, including those for hash function, are included under `test_des.ml`.

We give a sample output produced by AlgoROM when run on 3-round KAF with different round functions and the same key in the PRP-CPA model. We have cleaned the output to make it more readable. Note that this is a simple example. Often, the outputs produced by the tool are too overwhelming to be read by a human.

```
Analyzing Enc then Enc, with oracles in queries [2, 3]

queries on call Enc(L, R):
  * R + k to F1
  * F1(R + k) + L + k to F2$2
  * F2$2(F1(R + k) + L + k) + R + k to F3$3

queries on call Enc(L', R')
  * R' + k to F1
  * F1(R' + k) + L' + k to F2$2
  * F2$2(F1(R' + k) + L' + k) + R' + k to F3$3

Unpredictable variables: [k, #ct1, #ct2]
Adversarial variables: [L, R, L', R', X]

Analyzing collision - Type: Direct (2)
  query1: F1(R + k) + L + k
  query2: X
  winning_condition:
     eqs: F1(R + k) + L + k + X = 0
   ineqs:
  no solution

Analyzing collision - Type: Direct (3)
  query1: F2$2(F1(R + k) + L + k) + R + k
  query2: X
  replacement $2 -> #ct1 + R
  winning_condition:
     eqs: #ct1 + k + X = 0
   ineqs:
  no solution

Analyzing collision - Type: Double (2, 2)
  query1: F1(R + k) + L + k
  query2: F1(R' + k) + L' + k
  winning_condition:
     eqs: F1(R + k) + F1(R' + k) + L + L' = 0
   ineqs: [L + L', R + R'] != [0, 0]
  no solution

Analyzing collision - Type: Double (3, 3)
  query1: F2$2(F1(R + k) + L + k) + R + k
  query2: F2$2(F1(R' + k) + L' + k) + R' + k
  replacement F2$2 -> #ct1 + R
  winning_condition:
     eqs: R' + #ct1 + F2$2'
   ineqs: [L + L', R + R'] != [0, 0]
   order: L < #ct1, R < #ct1
  no solution

 Secure! Proven with $'s on rounds [2, 3]
9 ms: KAF[ (F1, k) (F2, k) (F3, k) ] is provably CPA secure :)
```