Brute Force Cryptanalysis

Aron Gohr
aron.gohr@gmail.com

Abstract. The topic of this contribution is the cryptanalytic use of *spurious keys*, i.e. non-target keys returned by exhaustive key search. We show that the counting of spurious keys allows the construction of distinguishing attacks against block ciphers that are generically expected to start working at (marginally) lower computational cost than is required to find the target key by exhaustive search. We further show that *if* a brute force distinguisher does return a strong distinguishing signal, fairly generic optimizations to random key sampling will in many circumstances render the cost of detecting the signal *massively* lower than the cost of exhaustive search.
We then use our techniques to quantitatively characterize various non-Markov properties of round-reduced Speck32/64. We fully compute, for the first time, the ciphertext pair distribution of 3-round Speck32/64 with one input difference Δ without any approximations and show that it differs markedly from Markov model predictions; we design a perfect distinguisher for the output distribution induced by the same input difference for 5-round Speck32/64 that is efficient enough to process millions of samples on an ordinary PC in a few days; we design a generic two-block known-plaintext distinguisher against Speck32/64 and show that it achieves 58 percent accuracy against 5-round Speck, equivalent e.g. to a linear distinguisher with ≈ 50 percent bias.
Turning our attention back to differential cryptanalysis, we show that our known-plaintext distinguisher automatically handles the 5-round output distribution induced by input difference Δ as well as the perfect differential distinguisher, but that no significant additional signal is obtained from knowing the plaintexts. We then apply the known-plaintext brute force distinguisher to 7-round Speck32/64 with fixed input difference Δ, finding that it achieves essentially the same distinguishing advantage as state-of-the-art techniques (neural networks with key averaging). We also show that our techniques can precisely characterize non-Markov properties in longer differential trails for Speck32/64.

1 Introduction and Goals of This Work

Brute force attacks on block ciphers commonly encounter two problems, namely impractical computational complexity and the problem of *spurious keys*. For instance, exhaustive search for a key K such that $E_K(P) = C$ may find many solutions, of which only one is usually the key sought by the adversary. The standard approach to this problem is to eliminate all spurious keys by requesting more ciphertext. The standard approach to the cost problem is not to use brute

force, or at least to view it only as the yardstick against which *real* cryptographic attacks must be measured.

In this paper, we instead use the counting of spurious keys to distinguish the output of block ciphers from random data; and we show that in various situations of interest, very good estimates of the number of these spurious keys can be obtained at a computational cost that is much lower than that of exhaustive key search.

As an application, we analyze the behaviour of low-round versions of one specific cipher (Speck32/64) with no or very mild approximating assumptions. In particular, we calculate the distribution of ciphertext pairs induced by some particularly suitable input difference exactly for a few rounds. We show that the ciphertext pair distribution of 3-round and 5-round Speck32/64 deviates fairly strongly from predictions of the Markov model of its differential transitions; we show that the output pair distribution of many longer trails of Speck32/64 likewise exhibits nonuniformity within the output difference classes. We then exhibit a known-plaintext distinguisher with very high bias for 5-round Speck32/64, show that it automatically exploits chosen plaintext inputs near-perfectly, and use it to show that in one particular situation of interest, an adversary probably would need to find a weakness in the key schedule of Speck32/64 to make use of knowledge of the plaintext inputs to the cipher.

1.1 Related Work

Generalizations of Differential Cryptanalysis Several works have considered various generalizations of differential cryptanalysis. One of the earliest proposals in this direction were fairly generic statistical attacks suggested by Vaudenay [Vau96]; here, plaintext and ciphertext data is jointly projected to some smaller space by a suitable hash function h and the distribution of the hash values is analyzed for deviations from the value distribution induced by applying h to uniformly distributed values. Similarly, Harpes' and Massey's *partitioning cryptanalysis* [HM97] divides the plaintext and ciphertext set of a cipher under study into well-chosen equivalence classes and follows the transition probabilities of these equivalence classes through the cipher. Wagner [Wag04] gave a formal framework called *commutative diagram cryptanalysis* for this class of attacks and showed how various attacks discussed in previous literature can be seen to fit into this framework. Some of the techniques we use in the present paper to make cryptanalysis by the sampling of spurious keys practical (for instance, the construction of *decryption equivalence classes* in section 2 and their subsequent use throughout the paper) fit into this framework as well.

While Wagner's work had already discussed all-in-one approaches to differential cryptanalysis as one instantiation of his commmutative diagram cryptanalysis framework and established some of their properties (for instance, a link between all-in-one differential and all-in-one linear attacks), all-in-one differential cryptanalytic attacks do not seem to have been used on any block cipher prior to Albrecht and Leander's attacks on round-reduced Katan [AL12]. Later, all-in-one attacks were developed by Gohr against Speck32/64 [Goh19]. The main

purpose of all-in-one attacks in Gohr's paper was to establish a strong baseline against which machine-learning attacks on the same target ciphers could be compared. The same paper also introduced a distinguisher that exploits the nonuniformity in the ciphertext pair distribution of 5-round Speck32/64 perfectly for one particular input difference. The present paper improves on this distinguisher by making its use practical on large test sets and by providing strong heuristic evidence that no significant additional signal can be obtained by exploiting knowledge of the plaintext pairs by attacks that do not use weaknesses of the key schedule.

Universal Cryptographic Attacks It is universally understood (at least since [Sha49]) that most practical cryptographic systems can be broken by a *brute force attack* given an attacker of unlimited computational capacity. In the case of a simple known-message attack, the adversary will intercept a ciphertext C and know by some other means the underlying plaintext message M. They know, in this setting, that $C = E_K(M)$, where E_K is a family of (maybe probabilistic) permutations and K is a secret key taken from some key space $\mathcal{K}$. The standard brute force attack then allows them to identify a small subset $S \subseteq \mathcal{K}$ such that $E_K(M) = C$ for each $K \in S$ simply by trying all possible values of K. For typical ciphers, S will in expectation consist of only a single element when the amount of intercepted ciphertext exceeds the key size of the cipher. When only a plaintext distribution is known, the adversary still learns some *a posteriori* probability distribution on the key and plaintext space.

Most of the subsequent cryptographic literature has considered this type of attack merely as a baseline to *real* cryptographic attacks: if the adversary has no *other* ideas, they may resort to the application of brute force attacks. Since arguably the main design goal of modern ciphers is that the adversary should have no other ideas available, a significant amount of work has been done on quantifying the real-world costs of brute force attacks (see e.g. [BG12, CKL$^+$21, Ber05]).

Implicitly, much cryptanalytic work assumes the *wrong-key randomisation hypothesis*, which can be viewed as saying that the brute force attack does not extract any cryptographically interesting information from some ciphertext *before* the right key is found. Under this assumption, it is indeed only for engineering aspects and as a baseline that the study of brute force attacks is interesting.

This paper explores a different aspect of brute force attacks, namely that they allow the cryptanalyst to *perfectly* exploit the signal induced by some input structure without the need to understand exactly what properties of the input text are still extractable from the ciphertext. Using brute force search in this way allows the cryptanalyst to design an input structure they hope will produce an exploitable output signal and then test rigorously how much distinguishing advantage an unbounded attacker could obtain against the tested primitive. To the best of our knowledge, this use of optimized brute force search as an all-in-one distinguisher was first made explicit in [Goh19] to upper-bound the performance of differential distinguishers against five-round Speck32/64; it is, however, implicit already in [Sha49], since a perfect distinguisher between two

sources can simply be viewed as a plaintext recovery attack against a keyed probabilistic source that sends a single bit by either encrypting some plaintext with known structure or outputs a random block of bits. Perfect distinguishers have also been studied subsequently in various other works, e.g. Vaudenay's decorrelation theory [Vau03]. However, they seem to have mostly been used as purely theoretical tools, most likely due to their apparent inability to produce attacks more efficient than naive brute force key search.

Engineering aspects and optimizations of brute force key search have, of course, been extensively studied in the literature; this includes, for instance, previous work on the efficient implementation of brute force search [CKL+21, BG12, KPP+06, Ber05] or time-memory tradeoffs [Hel80]. These papers essentially look at the concrete cost of running a brute force key recovery attack against some algorithm when nothing else is believed to work; in contrast, the present paper looks at brute force as a way to get an idea whether something else *might* work to recover some cryptanalytic signal by establishing whether the desired signal can in principle be recovered at all.

Non-Markov Properties of Speck Non-Markov properties were first found in Speck by Biryukov, Velichkov and Le Corre in the context of work to improve automatic differential trail search for ARX primitives [BVLC16]. Gohr later showed using neural cryptanalysis that some non-Markov properties are very widespread for Speck32/64 [Goh19]. In particular, they showed using the *real difference experiment* that the output pair distribution of Speck32/64 can for at least eight rounds reliably be distinguished at fairly low data requirements (on the order of 2^{16} ciphertext pairs) from a version of itself in which output pairs are randomized within their difference class. Benamira et al. were later able to recover distinguishers with quite similar performance characteristics as the neural distinguishers used by Gohr using different methods [BGPT21].

Block Cipher Cryptanalysis Without the Markov Assumption In most of the literature on differential cryptanalysis, the Markov assumption is understood to be strictly speaking wrong for many ciphers, but still treated as a model assumption that is useful and has no viable alternative (e.g. [BVLC16]). One very notable exception is the cryptanalysis of hash functions [WY05, DCR06, Leu13, SBK+17, LP19]. In hash collision search, the adversary has the goal of creating two messages $M_1 \neq M_2$ such that $h(M_1) = h(M_2)$, where h is the hash function under study. In the calculation of any efficiently computable h, a relatively small internal hash function state will be manipulated for a relatively small number of steps until an output is produced, giving rise to intermediate states $M_i = h_0(M_i), h_1(M_i), h_2(M_i), \ldots, h_n(M_i) = h(M_i)$. In a nutshell, differential cryptanalysis is then used to control the differences $h_i(M_0) \oplus h_i(M_1)$ for chosen messages M_0 and M_1, the aim being that $h_n(M_0) \oplus h_n(M_1) = 0$. In these attacks, high-probability differentials over parts of the hash computation are combined with careful control of the exact values of messages and internal state to minimize the amount of trial and error until a conforming pair is found.

Efficient High-Performance Distinguishers against Speck32/64 Various types of differential all-in-one distinguishers have been proposed for Speck32/64. Table 1 gives an overview; briefly, neural networks, stacked ensembles of simpler models, key-averaging based techniques, and variations of brute force key search have all been used to model the output pair distribution of round reduced versions of Speck32/64 with greater accuracy than is achievable by computing the difference distribution fully under the Markov assumption. Of these, the neural network,

Table 1. Qualitative overview of differential distinguishers for reduced Speck32/64 with input difference $\Delta = (0x40, 0x0)$.

Rounds	Type	Accuracy	Reference
5-8	DDT (Markov)	0.911, 0.758, 0.591, 0.512	[Goh19]
5-8	Neural network	0.929, 0.788, 0.616, 0.514	[Goh19]
5-7	Average Key Rank	0.9298, 0.7879, 0.6028	[BGPT21]
5-6	M-ODT ensemble + LGBM	0.923, 0.779	[BGPT21]
5	Optimized brute force search	≈ 0.95	[Goh19]

average key rank and M-ODT based distinguishers are notable for reaching a roughly similar level of performance both in terms of distinguishing power and execution speed on a standard platform (although the low memory footprint might make the neural network approach more efficient in a specialized hardware setting).

1.2 Main Contributions

In the present work, we first introduce a general framework for brute force distinguishers for block ciphers. We then discuss general conditions under which an approximative evaluation of a brute force distinguisher is possible at a cost substantially lower than brute force key search.

As a concrete application, we show that a brute force distinguisher using some conceptually straightforward performance optimizations can be used in practice to exploit the ciphertext pair distribution of 5-round Speck32/64 perfectly when the input distribution consists of plaintext pairs with difference $(0x40, 0x0)$. Our distinguisher is efficient enough that testing millions of samples is easily practical on a normal PC. We also show that knowledge of the plaintext does not yield a significant additional signal in this situation.

To obtain our five-round distinguisher, we first compute the ciphertext pair distribution of 3-round Speck with our input difference completely. We show that this distribution has a very compact description (it can be stored, for instance, in a 50 MB zip file), and that the distribution of output pairs is in most possible output difference classes very non-uniform. We further compute the 3-round difference distribution exactly and show that about half of the values of the exact model differ from the predictions of the corresponding Markov model, with around 20 percent of values different enough that the errors of the Markov

model are easily apparent after sampling on the order of 10^8 output pairs. To verify our models, we compare the predicted output distributions of both models against empirical sampling of appropriate versions of the cipher.

We show that our techniques can also be used to compute the ciphertext pair distribution induced by specific longer differential trails for Speck32/64. Examining some published trails from the literature, we find that the output pairs produced by these trails are often distributed in a highly non-uniform way within the given output difference class.

2 Preliminaries

2.1 Conventions and Notations

Basic Notions In this paper, E will denote a block cipher, i.e. a family $\{E_K : \mathcal{P} \to \mathcal{C}, K \in \mathcal{K}\}$ of bijections between finite sets $\mathcal{P}$ (the *plaintext set*) and $\mathcal{C}$ (the *ciphertext set*) that is parameterized by one parameter called the *key* which is chosen from a finite set $\mathcal{K}$ called the *key space*. Unless something else is specified, $\mathcal{P}, \mathcal{C}$ and $\mathcal{K}$ will in this paper be the sets of bit strings of some fixed *block size* l_P and *key size* l_K respectively; for Speck32/64, we have $l_P = 32, l_K = 64$.

Random Experiments When a value r is drawn from some set S according to the probability distribution $\mathcal{D}$, we will write $r \xleftarrow{\mathcal{D}} S$. If $\mathcal{D}$ is the uniform distribution, we simply write $r \leftarrow S$.

Dealing With Multi-Block Plaintext and Ciphertext We will by abuse of notation extend the domain and codomain of a block cipher to cover multiple blocks by using the cipher in ECB mode. The adversary is in this paper allowed to query an oracle to obtain $E_K(P)$ for some (maybe multi-block) P of their choosing; the oracle will depending on the outcome of a random coin $r \leftarrow \{0, 1\}$ either return $E_K(P)$ or a value drawn at random from $\mathcal{C}$. The adversary *wins* if they can recover r better than by random guessing.

Bitwise Operations We will use the usual notations for bitwise addition, rotation, modular addition and standard dot product over $\mathbb{F}_2$. In this paper, $\oplus$ will be bitwise addition of two bit vectors, $W \lll c$ and $W \ggg c$ will be left and right rotation of a w-bit word W by c steps respectively, $\boxplus$ will be addition of two w-bit numbers modulo 2^w. All operations are extended component-wise to arbitrary sequences of values of the appropriate bitsize. For vectors $v, w \in K^n$ for some field K, $v\dot{w} := \sum_{i=1}^{n} v_i w_i$ will denote their standard dot product.

Difference Classes When a pair of group elements g_1, g_2 in a group G have difference $\delta = g_1 g_2^{-1}$, we say that the pair (g_1, g_2) is in the *difference class* δ. When nothing else is specified, the group operation in question will simply be bitwise addition.

Iterative Block Ciphers Block ciphers are usually built by applying in alternating steps a simple key dependent function (such as bitwise addition of keying data to the cipher state) and a fixed round permutation. Both steps together form the *round function* of the cipher. The *subkeys* used in single rounds of the cipher are derived from a master key by means of a *key schedule*.

In this paper, we say that a cipher *uses the free key schedule* if the subkeys are chosen independently and uniformly at random from the set of possible subkey values.

Sets Sets are always regarded as finite in this paper. For a set M, the cardinality of M will be denoted by $Card(M)$. In particular, all functions appearing in this work are regarded to have upper limits on the bit-length of any inputs, although these limits may not always be specified.

Decryption Equivalence When E is a block cipher and C is a ciphertext (possibly spanning multiple blocks), we set $N_{keys}(P, C) := Card(\{K \in \mathcal{K} : E_K(P) = C\})$. Further we call

$$\{(N_{keys}(P, C), P) : P \in \mathcal{P}\}$$

the *decryption set* of C. When E is an iterative block cipher, i.e. when it is the composition of a sequence of round functions, we will use the name notions to talk about the decryption sets of intermediate cipher states.

When two (multi-block) ciphertexts $C_1, C_2 \in \mathcal{C}$ have the same decryption set, we say that C_1 and C_2 are *decryption equivalent*. Decryption equivalence of C_1 and C_2 is also denoted by $C_1 \sim C_2$. Note that decryption equivalence of two ciphertexts implies that even an information theoretic adversary cannot tell which of them was used to encrypt a known plaintext if they do not have some information about the key. The set of all C' with $C' \sim C$ is called the decryption equivalence class of C and denoted $[C]$.

2.2 The Speck Family of Ciphers

Speck was proposed in 2013 by Beaulieu et al. [BSS$^+$] as a family of lightweight ciphers intended for efficient software implementation. It has since been subjected to a significant amount of cryptanalysis; the best known attacks not dependent on weak key classes are differential attacks [Din14, SHY16, Goh19, BGL$^+$21].

The Speck ciphers are characterized by two parameters, namely the key and block size employed. Speck(b/k) denotes Speck with a block size of b bits and a key size of k bits. Hence, Speck(b/k) takes as input a b-bit plaintext P and a k-bit key K. The plaintext and key are represented as a tuple of w-bit words, where $w = b/2$; in all variants, $k \in \{2w, 3w, 4w\}$. First, a nonlinear key schedule is used to derive a sequence of subkeys $K_0, K_1, \ldots, K_{r-1}$ from K, where r is the number of round transformations R to be used in encryption. Then, the ciphertext C is obtained from P by setting $C = K_{r-1} \oplus R(K_{r-2} \oplus R(\ldots R(K_0 \oplus R(P))))$. The K_i are b-bit values of the form $K_i = K'_i || K'_i$, where K'_i is a w-bit value.

The round transformation R is an ARX function and can be given by $R(P_l||P_r) := ((P_l \ggg \alpha) \boxplus P_r)||(((P_l \ggg \alpha) \boxplus P_r) \oplus (P_r \lll \beta))$, where α, β are rotation constants. The version of Speck that will be used for the experiments in this paper is Speck32/64 with the free schedule, so we have $w = 16, \alpha = 7, \beta = 2$. However, none of these details will matter much to our experiments except as regards the choice of input difference for differential distinguishers.

3 Brute-Force Distinguishers for Block Ciphers

In this section, we will give a framework for brute force distinguishers for block ciphers, illustrate the framework with some examples, and discuss under which conditions brute force distinguishers may succeed at a work factor much lower than brute force key search.

Finally, we discuss implications of the framework for selective key search policies and the wrong-key randomisation hypothesis.

3.1 Goals

Key recovery attacks against iterative block ciphers usually have three parts. First, a round-reduced version of the cipher under study is attacked by a *distinguisher*: this is an algorithm which, given some knowledge on the cipher inputs, can find non-random behaviour in the reduced cipher output. Second, partial key search is performed on the remaining rounds of the cipher, generating a list of final subkey candidates that transform the observed cipher output into non-random output for the reduced cipher. This needs to either be done for all rounds, removing the rounds of the cipher one by one to recover all subkeys, or the key schedule has to be reversed in order to recover the master key from already recovered subkeys. Often, in a final step some remaining key candidates need to be tested on additional input-output pairs to remove non-target key candidates.

In the basic setting of known or chosen plaintext attacks, a distinguisher against a block cipher can itself be viewed as made of two components, namely a randomized algorithm $\mathcal{G}$ that generates input to the cipher, and another algorithm $\mathcal{E}$ that tries to exploit the induced output distribution of the cipher, possibly with some knowledge of the specific input values that were generated by $\mathcal{G}$.

When designing attacks against block ciphers, $\mathcal{G}$ and $\mathcal{E}$ have to be finely tuned to work together: after all, $\mathcal{E}$ must as well as possible be able recognize, based on the ciphertext, some structure in the plaintext generated by $\mathcal{G}$. The design of both stages of an efficient distinguishing attack is often highly nontrivial. Naturally, this means that many questions are usually not answered by a given attack, for instance:

1. How far away is $\mathcal{E}$ from exploiting the signal generated by $\mathcal{G}$ perfectly? Could there be another algorithm $\mathcal{E}'$ that finds more structure in the block cipher output?

2. Is $\mathcal{G}$ chosen well? Would a stronger distinguishing signal survive to the cipher output if some other plaintext generating algorithm were used?
3. In many cases, $\mathcal{E}$ will not use all of the information about the plaintext values tried that is available after running $\mathcal{G}$: for instance, differential distinguishers routinely ignore the input values entirely, only forcing them to be good starting points of some differential trail. Could a better attack be developed by using more knowledge about the plaintext?

On the highest level, this paper aims to help answer such questions by decoupling the development of the plaintext generation and ciphertext exploitation algorithms in known or chosen plaintext attacks. To this end, we will show that in a wide range of settings, proof-of-concept distinguishers against small block ciphers can be constructed generically once $\mathcal{G}$ is fixed. These distinguishers are furthermore expected to be close to exploiting the output distribution perfectly, allowing for the establishment of sound and nontrivial upper bounds for the success rates of distinguishers against reduced small block ciphers exploiting specific input structures.

These points will be reinforced by various practical experiments on reduced Speck32/64. The techniques used, however, have a generic theoretical foundation and are therefore not specific to Speck.

3.2 Basic Concepts and Examples

Let E be a cipher with key space $\mathcal{K}$, plaintext set $\mathcal{P}$ and ciphertext set $\mathcal{C}$.

Definition 1. *Let $E_K : \mathcal{P} \to \mathcal{C}$ be a cipher and $f : \mathcal{P} \to [0,1]$ be a discrete probability mass function on the plaintext set. We say that f is a* plaintext signal with advantage δ for E *if a computationally unbounded adversary $\mathcal{A}$ has a win-rate of $0.5 \cdot (1+\delta)$ in the following game:*

1. $r \leftarrow \{0,1\}$.
2. $P \xleftarrow{f} \mathcal{P}$.
3. $K \leftarrow \mathcal{K}$.
4. *If $r = 0$, then $C \leftarrow \mathcal{C}$; else $C := E_K(P)$.*
5. *$\mathcal{A}$ gets C as input and guesses r.*

Given these notations, we have the following:

Lemma 1. *Let f be a plaintext signal and let C be a ciphertext. Assume that keys are uniformly distributed and sampled independently from the plaintext.*

The posterior probability for $r = 1$ given the observed ciphertext C is then given by

$$\mathbb{P}(r = 1|C) = \frac{p_1}{p_0 + p_1},$$

where $p_0 = 1/Card(\mathcal{P})$ and $p_1 = \sum_{K \in \mathcal{K}} f(E_K^{-1}(C))/Card(\mathcal{K})$.

Proof. p_0 is by the rules of the game the likelihood of sampling C in the case $r = 0$. p_1 is the likelihood of sampling C in the case $r = 1$. The claim follows by applying Bayes' theorem.

All of the following examples are well-known and only given to illustrate the use of the notations introduced:

Example 1. **(No Free Lunch)** Let f be the uniform distribution on $\mathcal{P}$. Since E_K is a bijection from $\mathcal{P}$ to $\mathcal{C}$ for every K, the distribution of C in the game given is then independent of r. Consequently, the advantage of any adversary is zero.

Example 2. **(Diagonal Signal)** Let E_K be a block cipher with block size b extended by ECB mode to inputs longer than one block. Let $\mathcal{P} = \{0,1\}^{2b}$ and let

$$f(P_0||P_1) = \begin{cases} 2^{-b} \text{ if } P_0 \oplus P_1 = 0 \\ 0 \text{ otherwise.} \end{cases}$$

Then an adversary can simply test if $C = C_0||C_0$ for some $C_0 \in \{0,1\}^b$. This strategy fails only if by chance a $C \leftarrow \mathcal{C}$ is of this form, i.e. in one out of 2^{b+1} samples seen. Consequently, this plaintext signal has advantage $\geq 1 - 2^{-b}$ (the unbounded adversary might do better than the simple strategy suggested).

Remark 1. The preceding example shows that plaintext signals with very high advantage can be cryptographically uninteresting. One could of course *fix* this by generating C in the $r = 0$ case by setting $C := \pi(P)$ with $\pi \leftarrow S_{2^b}$ freshly sampled and domain-extended by ECB mode. In this paper, we choose not to do so, for the following reasons:

1. While the plaintext signal exhibited in Example 2 is *uninteresting* from the point of view of learning any cipher internals, it does show that ECB mode (or more generally any mode that allows an adversary to completely control the non-key inputs to a block cipher) has significant weaknesses. This seems like a desirable property of a security game.
2. Indeed, many block cipher modes of operation that do *not* allow the adversary to control the non-key inputs to the primitive in question start to show significant plaintext leakage as the volume of plaintext processed approaches the birthday bound (see e.g. [LS18]). This suggests that the plaintext signal of Example 2, while useless e.g. for key recovery, should not *just* be viewed as a pathology of a weak mode of operation.

Example 3. **(Differential Attack)** Let $\mathcal{P}$ and $\mathcal{C}$ be as in the previous example but set

$$f(P_0||P_1) = \begin{cases} 2^{-b} \text{ if } P_0 \oplus P_1 = \delta, \\ 0 \text{ otherwise} \end{cases}$$

for some fixed $\delta \neq 0$. This plaintext signal corresponds to an all-in-one differential attack exploiting the induced ciphertext pair distribution. It has advantage $2a-1$ when a is the accuracy of a perfect adversary. For 5-round Speck32/64 and $\delta = (0x40, 0x0)$ [Goh19] therefore showed an advantage of ≈ 0.9.

Example 4. **(All-in-One Known Plaintext Attack)** Let $P_0 \in \mathcal{P}$ be a fixed plaintext, $f(P_0) = 1$ and $f(P') = 0$ for all $P' \neq P_0$. Let further $C \in \mathcal{C}$ and denote by $N_{keys}(C)$ the number of $K \in \mathcal{K}$ such that $E_K(P_0) = C$. In the game of Definition 1, the probability that C will be sampled is then $p_0 = 1/Card(\mathcal{C})$ when $r = 0$ and $p_1 = N_{keys}(C)/Card(\mathcal{K})$ when $r = 1$. Given C as input, an unbounded adversary will therefore obtain a posterior probability that $r = 1$ of $\mathbb{P}(r = 1) = p_1/(p_0 + p_1)$. When $Card(\mathcal{K}) = Card(\mathcal{C})$, this implies $\mathbb{P}(r = 1) > 0.5$ exactly if $N_{keys} > 1$, and in this case unoptimized brute force search will find *some* K satisfying $E_K(P_0) = C$ with an expected work factor of $\leq 2^{-2}Card(\mathcal{K})$ single-block encryptions or decryptions.

Example 4 uses the signal from a single known plaintext optimally if all keys are sampled. When the amount of plaintext is too low to rule out all wrong keys, the signal returned will, however, strongly depend on *which* plaintext is being used. This will be very evident when we present empirical results using round-reduced Speck32/64.

Remark 2. **(Limitations of the Framework)** While the attacks use the information given to the adversary optimally (when the key space is sampled exhaustively), it is still possible that better attacks can be constructed at the same data cost:

1. The methods here discussed can only produce useful distinguishers that use a fairly small amount of data. For a typical block cipher, beyond a few blocks of plaintext-ciphertext pairs the advantage of a known-plaintext attack in the security game of Definition 1 will become overwhelming, but the amount of computation necessary to carry the attack out also becomes identical to standard brute force key search (since in the real case only one key remains contributing to the recovered signal). Still, an adversary using the techniques in this paper can of course combine information from several observations by treating them as independent instantiations of the same distinguishing problem.
2. The plaintext signals used may also not be the optimal ones.
3. Attacks that use both a plaintext and a ciphertext signal may be stronger than those that only use a plaintext signal.
4. Finally, attacks that choose the inputs to be queried adaptively may extract an even stronger signal than any attack that fits into the framework here discussed.

We leave these possible improvements and extensions to the techniques discussed in this paper to further research.

3.3 Optimizing Brute-Force Distinguishers

Naively, brute force distinguishers require sampling a significant part of the key space to recover a good approximation to the target signal. This is not feasible

outside of toy ciphers, especially if one hopes to use brute force distinguishers to help in the *identification* of useful plaintext signals.

In the sequel, we will for this reason discuss several fairly generic ways to speed the evaluation of brute force distinguishers up significantly. In the next section, we will then apply some of these ideas to two brute force distinguishers against round-reduced Speck32/64; for 5-round Speck32/64, this will result in a distinguisher that can in our implementation process around two ciphertext pairs per core per second on an ordinary PC without any loss of power compared to a full sampling of the key space.

Ignore the Key Schedule Replacing the actual key schedule of the cipher under study with an assumption of independent and uniformly distributed subkeys will often allow the adversary to build equivalence classes of ciphertexts. Usually, the set of possible subkeys forms a group (most commonly by bitwise addition of subkeys) and the operation that mixes key and (partially encrypted/decrypted) plaintext gives an action of that group on $\mathcal{P}$. When the subkeys for all rounds are uniformly and independently chosen, ciphertexts C and C' are decryption equivalent whenever they are on the same orbit of the final subkey addition. Depending on the size of the orbits and the size of the output space, this can greatly simplify statistical sampling of the output distribution.

Example 5. Let E_K be an Even-Mansour cipher [EM97] with b bit block size and independent subkeys K_1 and K_2, i.e. $E_K(P) := K_2 \oplus F(K_1 \oplus P)$ with F some fixed permutation. Assume that the plaintext signal is given by a two-block fixed plaintext $P = P_0||P_1$ and that the observed ciphertext is $C = C_0||C_1$. Then the problem of determining the number of keys that link P and C is equivalent to determining the probability of the differential transition $P_0 \oplus P_1 \to C_0 \oplus C_1$ because ciphertexts within the same difference class are always decryption equivalent for this cipher. This reduces the sampling effort required for finding solutions by a factor of 2^b compared to naive sampling.

Use Meet-in-the-Middle Sampling Working with independent subkeys also has the advantage that the resulting cipher becomes vulnerable to meet-in-the-middle attack. This means that the adversary can (naively) use n samplings from the plaintext distribution, $2n$ partial encryptions and decryptions, $O(n)$ memory, the sorting of two arrays of n entries and a final step to determine the size of the intersection of two sorted arrays to find out how many of n^2 tested (not necessarily unique) keys connect the plaintext signal to the observed ciphertext.

Example 6. Let E_K be a two-round Even-Mansour cipher with b bits block size and let the plaintext signal be again given by two fixed plaintext blocks. By Example 5, in order to construct a key (K_0, K_1, K_2) that connects $P_0||P_1$ and $C_0||C_1$ it suffices to find K_0 and K_2 such that $F(K_0 \oplus P_0) \oplus F(K_0 \oplus P_1) = F^{-1}(K_2 \oplus C_0) \oplus F^{-1}(K_2 \oplus C_1)$. After trying n_0 values for K_0 and n_2 values for K_2, we expect to obtain $\approx n_0 n_2 / 2^b$ collisions if C_0 and C_1 are sampled uniformly at random in such a way that they are independent of each other and of the P_i.

Remark 3. Meet-in-the-middle sampling can in principle use the standard known optimizations of the generic meet in the middle attack, principally collision detection using parallelizable memoryless methods [VOW99]. However, care needs to then be taken to remove pseudocollisions from the results and to keep track of the number of keys tested versus the number of matches found. Our experiments in the next section that use meet-in-the-middle sampling just use the simple approach of performing the meet-in-middle matching in memory.

Remark 4. Example 6 treats meet-in-the-middle sampling as if keys were sampled uniformly at random from the set of all keys. While this is true for the very first key that is sampled, subsequently sampling is biased, since the set of keys covered by sampling is always the cartesian product of the partial keys tried below and above the meet-in-the-middle point in the cipher. Heuristically, we expect this sampling bias to have the least effect on the performance of the resulting distinguisher when the meet-in-the-middle point is chosen such that the number of rounds on both sides is approximately the same, but we did not perform experiments to test this.

Selective Sampling Brute force distinguishers aim at estimating the expected probability weight of decryptions of a given ciphertext under uniformly sampled random keys as given by the plaintext signal[1]. However, that does not imply that a brute force distinguisher has to sample keys uniformly at random.

If parts of the key space are sampled with higher density (because they yield more plausible-looking plaintext), then they have to be downweighted in the final tallying step so that an unbiased estimate of the probability weight of uniformly random decryptions of the given ciphertext is obtained.

Example 7. Assume a differential distinguisher for the block cipher E, i.e. plaintexts of the form $P, P \oplus \delta$, where P is a single block. Assume further that it is known that a particularly strong differential signal is expected if E follows a differential path $\delta \to \delta_1 \to \ldots \delta_r$ for some initial rounds. Assume further that it is known that the transition $\delta \to \delta_1$ depends to a large degree on k bit conditions on the first subkey. Then a simple non-uniform sampling policy might consist of trying keys that fulfill the bit conditions half the time and trying random other keys the other half of the time. If finally n_b conforming keys are found in the path where the bit conditions are fulfilled and n_r keys are found in the random path with N trials in each path, an unbiased estimate of the proportion of *good* keys (i.e. keys that decrypt the observed ciphertext pair to the known input difference) under uniform sampling is $2^{-k}\frac{n_b}{N} + (1 - 2^{-k})\frac{n_r}{N}$.

Memoizing Intermediate Results If keys are chosen independently at random, it is also useful to remember if some equivalence class of partially decrypted ciphertexts C' has been successfully linked to a valid plaintext during past sampling;

[1] This assumes that the cipher uses uniformly random key generation. If key generation uses a non-uniform distribution, then the probability weight under that distribution is the one of interest.

if other keys connect the observed ciphertext to C', the information that a link to a valid plaintext exists can be propagated back to the source immediately in this case, without a need to sample the same path again potentially many times.

Modelling the Cipher It is of course also possible to replace part of the cipher with an approximation. For instance, if a cipher has $n+1$ rounds and the plaintext signal is given by a fixed plaintext P, one could *estimate* the number of keys connecting some partially decrypted ciphertext C' with P through the first n rounds instead of determining the number of good lower subkeys by sampling. This leads to *key-averaging distinguishers* as in [Goh19, BGPT21].

Sampling with Constraints In some situations, hard constraints may be available that come from other sources, for instance cryptanalysis already performed on the target, from side channels or from the cryptanalytic question that the adversary wants to resolve. This can be viewed as a special case of using a selective sampling policy where unpromising branches of the search tree can be cut entirely instead of being downweighted during the sampling phase. In this case, again the size of the unsampled part of the key space has to be taken into account when estimating the proportion of good keys.

Example 8. Suppose that E is an iterative b-bit block cipher with independent and uniformly distributed k-bit subkeys, $k < b$, where the subkeys act on the cipher state by bitwise addition on a fixed part of the state. Suppose further that E has a known differential trail

$$\delta_0 \to \delta_1 \to \delta_2 \to \dots \delta_n.$$

Suppose further that the resulting cipher does not fulfill the Markov property and suppose that a pair of inputs P_0, P_1 and outputs C_0, C_1 has been observed that fulfills $P_0 \oplus P_1 = \delta_0$ and $C_0 \oplus C_1 = \delta_n$. It is then a reasonable question whether the observed output pair could have followed the known differential trail.

The set of two-block cipher states after each round can be partitioned into 2^{2b-k} equivalence classes. Since the bitwise addition of the subkeys to each block of the two-block message is compatible with the constant difference classes, the cipher state after round i falls into one of 2^{b-k} equivalence classes if the cipher follows the predefined trail.

The adversary can calculate the distribution of the output pairs exactly by executing the following algorithm:

1. Set $N(0, c) := 1$ for all equivalence classes of input pairs.
2. Set $N(i, c') := 0$ for all i and all equivalence classes c' of intermediate cipher states that do not belong to the chosen trail.
3. Set $N(i+1, c) := \sum_{k \in sk} N(i, E_{i+1}^{-1}(c, k))$, where sk is the set of subkeys for round $i+1$ and $E_i^{-1}(c, k)$ denotes the ciphertext equivalence class reached by decrypting a representative of c by subkey k. The choice of representative does not matter here, but has to be fixed before evaluating the sums in question.

All values $N(n, c)$ for c compatible to the output difference can then be computed with $n \cdot 2^b$ one-round decryptions and the same number of summations using memory for $n \cdot 2^{b-k}$ multiple-precision integers (or fixed-precision floating point numbers, if approximate results are sufficient). $N(n, c)$ gives the number of keys that link decryption equivalence class c in δ_n to the input difference δ_0.

3.4 Brute Force Distinguishers and Key Search

Effects of Wrong Key Decryption Gohr found in [Goh19] that a key recovery attack against 11-round and 12-round Speck32/64 using machine learning based differential distinguishers could be sped up significantly by locating the right key using information gained from evaluating the distinguisher response to wrong keys. Brute force cryptanalysis gives a theoretical basis to the expectation that nonrandom behaviour should be visible (to an unbounded adversary, at least) in wrong-key decryptions whenever it is information-theoretically possible to distinguish ciphertext from random text given an amount of ciphertext lower than the unicity distance of the underlying plaintext distribution and encryption method: after all, brute force cryptanalysis is capable of finding the signal in question (at least with sampling of the full key space) and the *main idea* of brute force cryptanalysis is to look at wrong-key decryptions.

There is, however, no generic expectation as to the form of any wrong-key decryption signal .

Selective Key Search Standard modern cryptanalytic attacks mostly use a plaintext signal where all plaintexts with nonzero probability of being sampled are equiprobable. For instance, differential attacks and all attacks that assume full knowledge of the plaintext are of this form when translated into the framework used in the present paper. Set $p := f(P)$ for any plaintext P with $f(P) \neq 0$. Then, $\mathbb{P}(r = 1|C) = \frac{c_1 \cdot N_{keys}(C)}{c_1 \cdot N_{keys}(C) + c_2}$, where $N_{keys}(C)$ is the number of keys that connect C to a possible plaintext and where $c_1 = p \cdot Card(\mathcal{K})^{-1}$ and $c_2 = Card(\mathcal{C})^{-1}$. When $N_{keys}(C)$ can be efficiently computed (or at least efficiently estimated) and when E is an iterative block cipher, it is tempting to use to resulting distinguisher to peel off the encryption round by round and derive a probability distribution on the possible full keys. However, while this is expected to indeed yield a non-uniform posterior probability distribution on the outer subkeys, the probability mass for any $S \subseteq \mathcal{K}$ is still given by

$$\mathbb{P}(r = 1|C \text{ observed and } K \in S) = \frac{c_1(S) \cdot N_{keys}(C, S)}{c_1(S) \cdot N_{keys}(C, S) + c_2},$$

where $N_{keys}(C, S)$ is the number of *good* keys in S and where $c_1(S) = p \cdot Card(S)$. In particular, setting S to be a set with just a single element shows that all keys that have not been ruled out are a posteriori equiprobable. Our distinguisher does not point us towards the exact key that was used to generate C if C was indeed sampled from the ciphertext distribution. In particular, sorting the keys for instance by the posterior likelihood of the outermost subkey does not improve the expected position of the *true* key in the list of possible keys.

Remark 5. The situation changes when the adversary can combine distinguisher signals from several brute force distinguishers. In this case, the standard method of removing one cipher round at a time using the combined distinguisher signals may be quite applicable.

Remark 6. When the correct key can be recognised by running some additional tests, the above argument does also not show that the *computational cost* of solving for the true key is not reduced by sorting the subkeys at each search node by their posterior probability under the used real-or-random test. The reason for this is that generating the valid completions of some partial expanded key (i.e. the expanded subkeys that connect the observed ciphertext with a valid plaintext) may have a computational cost that does not primarily depend on the number of valid completions found.

4 Experimental Results and Applications

4.1 Overview and Strategy

In this section, we apply the techniques previously developed to Speck32/64 with independent subkeys. First, we combine Example 8 and the guess-and-determine technique for efficiently finding the subkeys fullfilling a single-round differential transition $\delta_0 \to \delta_1$ developed by Dinur [Din14] to efficiently count the number of keys fulfilling any given differential trail of Speck32/64. We use this to calculate the output pair distributions for some trails in the literature, finding that output pairs for many of these trails are far away from being uniformly distributed in their difference classes.

We then note that 3-round Speck32/64 with input difference $\Delta := (0x40, 0x0)$ has only 42149 possible output differences and that each of these output differences is linked by a unique differential trail to the fixed input difference. For each of these trails, we calculate the distribution of the output pairs. We compare these results both to empirical sampling and to the Markov model computed in [Goh19]. We find that empirical sampling matches the theoretical results of our model perfectly, but that the Markov model contains widespread mistakes that are easily visible to our empirical sampling. We verify that the Markov model correctly predicts the output distribution of a version of the cipher that has been changed to be Markov, implying that the errors in the model output are not due to problems in the implementation of the Markov model, but due to failure of the Markov assumption for the cipher under study.

In order to construct an efficient perfect distinguisher for five-round Speck32/64 with input difference Δ, we then use the same technique to connect a given output pair to all $3 - round$ output differences. We use the precomputed 3-round distribution to infer from this the number of keys that link our observed output to the given input difference. This gives a five-round distinguisher that can process millions of examples in a few days on a normal PC.

Finally, we develop a generic known-plaintext distinguisher for various reduced versions of Speck32/64 that is based on meet-in-the-middle sampling. We

show that with reasonable sampling settings, this distinguisher achieves state-of-the-art performance for 5-round and 7-round Speck32/64. We show that knowing the input does not help this distinguisher solve the 5-round problem by randomizing the input pairs without changing the input difference; this does not lead to a loss of performance when the input difference is Δ. Applying our distinguisher to two-block five-round Speck32/64 without a fixed input difference, we show that our distinguisher exhibits a bias that is roughly equivalent to finding a linear distinguisher with a 50 percent bias over these five rounds.

4.2 Calculating the Output Pair Distribution for Differential Trails in Speck32/64

Problem Statement Let E be an iterative block cipher. Suppose we are trying to construct a differential attack against it. We might then build a differential trail

$$\delta_0 \to \delta_1 \to \ldots \to \delta_n$$

for E that we hope has a relatively high transition probability. The standard approach to determining the transition probability of such a trail is to compute probabilities p_i of the single-round differential transitions $\delta_i \xrightarrow{p_i} \delta_{i+1}$ and treat successive round transitions as independent. If E is a Markov cipher, then this approach is theoretically justified; otherwise, it may yield wrong probabilities for the whole path while simultaneously failing to predict the distribution of ciphertext pairs in the output difference class. Since Speck is known not be Markov, a more precise modelling of its differential trails is desirable.

Relevant Properties of Speck32/64 We are going to use the following properties of Speck, all of which are well-known:

1. Speck is an iterative block cipher. A round consists of the application of a round function $R : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w \times \{0,1\}^w$ followed by the addition of an l-bit subkey K to both halves of the cipher state.
2. Given a two-round differential trail $\delta_0 \to \delta_1 \to \delta_2$, knowledge of δ_0 and δ_2 is sufficient to derive δ_1.

A Simple Approach A generic strategy to calculate both the true path probability and the output pair distribution in this setting has already been discussed in Example 8. Essentially, it relies on the following ideas:

1. We keep the assumption of independent and uniformly distributed subkeys because it allows us to easily identify sets of decryption equivalent cipher states.[2]

[2] One may be tempted to say that it allows identifying the decryption equivalence classes of the cipher states. While this is very likely true in practice, we do not have proof that the decryption equivalence classes for Speck32/64 are never larger than the single-round decryption equivalence classes we use in our experiments.

2. Concretely, we treat output pairs as equivalent if their difference is a valid Speck32/64 subkey.
3. We can then recursively count the number of keys that connect each equivalence class of cipher states appearing in our trail to the original input difference while caching any results for reuse.

Decryption Equivalence Classes For Speck32/64, the single-round decryption equivalence classes of cipher states are of size 2^{16}, since a subkey is a 16-bit bitstring. Denote for a single-block ciphertext C the left 16-bit word by C^l and the right 16-bit word by C^r. Then addition of a subkey K maps (C^l, C^r) to $(C^l \oplus K, C^r \oplus K)$. Any pair C_0, C_1 of ciphertexts is therefore under the assumption of independent and uniformly distributed subkeys decryption equivalent to

$$(0, C_0^l \oplus C_0^r), (C_0^l \oplus C_1^l, C_0^l \oplus C_1^r).$$

Writing $C := (C_0, C_1)$, $\Delta(C) := C_0 \oplus C_1$ and $\delta(C_0) := C_0^l \oplus C_0^r$, we can also derive the decryption equivalence class of C from knowledge of the pair $(\delta(C_0), \Delta(C))$. This representation is convenient in the present context, as $\Delta(C)$ is constant for each round in a given differential trail and the decryption equivalence classes of intermediate cipher states inside a differential trail can therefore be uniquely identified when $\delta(C_0)$ and the round number i are known.

Remark 7. That these ciphertext equivalence classes are useful (and that they are, in the terms of this paper, decryption equivalence classes) for Speck32/64 was already mentioned in [Goh19], where it was noted that neural network based distinguishers for Speck32/64 make use of these equivalence classes. The same equivalence classes are also used implicitly by the average key rank and M-ODT distinguishers proposed by [BGPT21].

Assuming that $N_{keys}([C_i]))$ are known for all 2^{16} equivalence classes in round i of a differential trail, calculating the distribution of the next round using the naive approach takes 2^{16} single-round trial decryptions per equivalence class; then, we have to look up whether the resulting decryptions fit the trail difference. If so, $\delta(C_0)$ is computed for the obtained decryption, the number of keys linking this equivalence class to the desired input difference is looked up and added. All in all, executing the algorithm given in Example 8 for an $r+1$-round trail would take $2^{32} \cdot r$ single-round decryptions and table lookups as well as memory for $2^{16} \cdot r$ numerical values of the desired precision.

Optimization of the Key Sampling Stage This is entirely practical, but becomes problematic if the goal is to do this for a large number of trails. We therefore use Dinur's guess-and-determine strategy for enumerating the solutions of single-round differential transitions for Speck [Din14] to quickly count the number of keys that connect a decryption equivalence class in round $i+1$ to the trail difference for round i. Processing a trail takes a few seconds on an 8-core PC in our implementation; for instance, the output distribution of trail 3 from table 3 in [Din14] is calculated in about 7 seconds on our machine.

Converting the Results to a Ciphertext Pair Distribution For any C in the output difference class of a differential trail T, we obtain the number $N_{keys}(C)$ of keys that link C to the input difference δ_0 of T. We have $N_{keys}(C) = \sum_{\Delta(P)=\delta_0} N_{keys}(P, C)$, and $N_{keys}(P, C)/Card(\mathcal{K})$ is equal to the probability of obtaining the ciphertext C given plaintext P. For the probability $\mathbb{P}(C|\Delta(P) = \delta_0)$ of obtaining ciphertext C given a plaintext P chosen uniformly at random from the set of plaintexts with $\Delta(P) = \delta_0$ we get

$$\mathbb{P}(C|\Delta(P) = \delta_0) = \frac{N_{keys}(C)}{Card(\mathcal{K}) \cdot Card(\Delta^{-1}(\delta_0))},$$

and these are the same for all C within the same single-round decryption equivalence class. Hence, the probability of obtaining the ciphertext C given a plaintext with $\Delta(P) = \delta_0$ under the condition that $\Delta(C) = \delta_n$ is given by

$$\mathbb{P}(C|\Delta(C) = \delta_n, \Delta(P) = \delta_0) = \frac{N_{keys}(C)}{\sum_{C' \in \Delta^{-1}(\delta_n)} N_{keys}(C')}$$

and the probability of obtaining some ciphertext C within the same single-round decryption equivalence class is of the same form, except that the sum in the denominator runs over only one representative of each class in this case.

Results We tried our methods on trails 1-4 of Table 3 in [Din14], which were originally proposed in [ALLW14] and [BRV14]. For trails 1-3, the distribution of decryption equivalence classes predicted at the trail output is markedly non-uniform; trail 4, on the other hand, is predicted to have a uniform output pair distribution.

Trail inputs and outputs as well as number of rounds covered are given in Table 2. For three of these trails, our model predicts a fairly large number of impossible output pairs. Most significantly, for trail 3 around 40 percent of the output pairs in its output difference class are predicted to be impossible if the trail is followed.

For trails 1 and 3, we checked the predictions of our model by direct sampling from the output distribution given by encrypting suitable plaintext pairs for the given number of rounds using random keys, where the real key schedule was used for key expansion. Since both trails were expected to have a high probability compared to other trails that might contribute to the same output difference, we expected the output distribution to be close but not necessarily identical to predictions. For trail 1, our aim was to check the pair distribution entirely; for trail 3, we wanted to see if the prediction of a large number of impossible output pairs would hold.

For trail 3, two out of 77 conforming output pairs were in decryption equivalence classes that were predicted to be impossible for the trail; the large number of predicted impossible pairs was further found to be due to the last transition of the trail, and no transitions to impossible output classes were found in a few hundred thousand trials fulfilling that transition.

Table 2. Results on single trails of Speck32/64. The trail IDs given are as in [Din14]. Input and output differences as well as round numbers of these trails are reported for the reader's convenience. p_{not} is the proportion of output pairs in the output difference class of the trail that our model predicts cannot appear as trail output. Empirical sampling used 10^9 samples from the ciphertext pair distribution, of which only the samples with the output difference of the trail were kept; their numbers are reported in the Samples column. A dash indicates that no empirical verification was done. Impossible Samples gives the number of samples found that are predicted to be incompatible with the trail (but which may be produced by other trails ending with the same output difference).

Trail ID	Trail Input	Trail Output	Rounds	$\log_2(p_{\text{not}})$	Samples	Impossible Samples
1	(0x211,0xa04)	(0x850a,0x9520)	6	-7	121831	0
2	(0xa60,0xa205)	(0x850a,0x9520)	7	-7	-	-
3	(0xa60, 0xa205)	(0x802a, 0xd4a8)	8	-1.19	77	2
4	(0x8054, 0xa900)	(0x40, 0x542)	9	$-\infty$	-	-

For trail 1, we divided the 2^{16} decryption equivalence classes within the output difference class into groups based on their predicted likelihood of appearing in the cipher output. This partitioned the 121831 samples we obtained into 65 groups. For each of these groups, we calculated the mean number of observed ciphertext pairs in each decryption equivalence class in the group as well as the mean predicted by our model. Both values were in near-perfect agreement (correlation approximately 0.999); see Figure 1 for details.

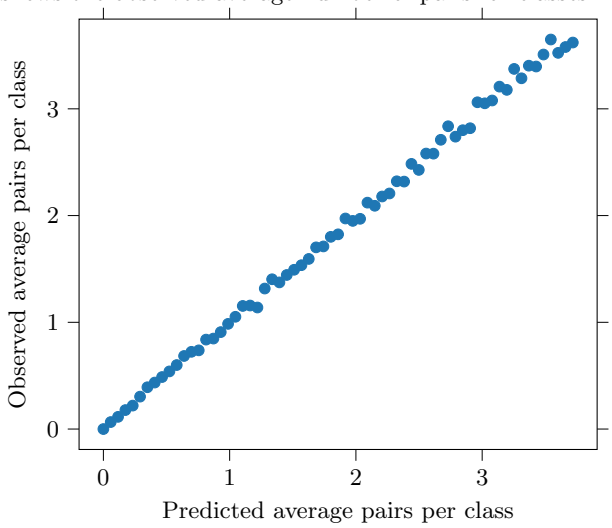
4.3 Calculating the Ciphertext Pair Distribution for 3-Round Speck32/64

In this section, we use our techniques to fully calculate the ciphertext pair distribution of Speck32/64 with input difference $\Delta = (0x40, 0x0)$. We show that it is highly compressible, with most decryption equivalence classes in possible output difference classes being impossible. We also show that the predicted likelihoods of many output difference classes differ significantly from those predicted by the standard Markov model. We confirm our findings by empirically sampling the 3-round distribution of both original Speck32/64 and a version of the cipher that has been changed to be Markov.

Preliminaries The following well-known observations are important for what follows:

1. Given a trail $\delta_0 \to \delta_1 \to \delta_2$ with $\delta_i = (\delta_i^l, \delta_i^r)$ for Speck, one obtains $\delta_1^r = (\delta_2^l \oplus \delta_2^r) \ggg \beta$ and $\delta_1^l = (\delta_0^r \lll \beta) \oplus \delta_1^r$.
2. The input difference $\Delta := (0x40, 0x0)$ transitions deterministically to $(0x8000, 0x8000)$ for Speck32/64.
3. In particular, if C_0, C_1 is a pair of ciphertexts for Speck32/64 reduced to five rounds and $N_{keys}(C_0, C_1)$ is the number of keys linking the ciphertext pair C_0, C_1 to the input difference Δ, then $N_{keys}(C_0, C_1)$ is also the number of

Fig. 1. A more detailed look at the data for trail 1. In total, 121831 output pairs conforming to the output difference of trail 1 were observed among 10^9 trials using the input difference of the trail. Decryption equivalence classes were grouped according to their expected frequency in the output. The x-axis shows the theoretically expected number of samples for decryption equivalence classes in each group, and the y-axis shows the observed average number of pairs for classes in that group in our test.



keys linking C_0, C_1 to the input difference $(0x8000, 0x8000)$ at the output of round 1. Hence, as far as the differential properties of five-round Speck32/64 with input difference Δ are concerned, five-round Speck32/64 can be treated

4. For four rounds or less the subkeys of Speck32/64 can be treated as chosen independently and uniformly at random if the master key has been chosen uniformly at random.

Calculating the Pair Distribution This implies that for 3-round Speck32/64, there is exactly one differential trail $\Delta \to \delta_1 \to \delta_2 \to \delta_3$ with potentially nonzero transition probability for any vector $\delta_3 \in \{0,1\}^{32}$.

We used the code published as supplementary material to [Goh19] to calculate all possible output differences of 3-round Speck32/64 with input difference Δ as well as the probabilities of the corresponding trails under the Markov assumption. It turns out that there are 42149 possible output differences.

Since a single trail connects each of these differences to our input difference, we then calculated the output pair distribution for each of these trails.

Results

Impossible Decryption Equivalence Classes We find that roughly 93 percent of differences in the possible output classes are in fact impossible; the whole distri-

bution, comprised in principle of $42149 \cdot 2^{16}$ floating-point values, can be losslessly compressed to about 50 megabytes.

Probabilities of Difference Classes Since each output difference comes from a single trail, any deviation between the Markov model and the actual output distribution has to come from round-to-round dependencies. In particular, independent round transitions would imply that all transition probabilities would be of the form 2^{-k} due to the differential properties of addition [LM01].

In total, our model predicts different 3-round transition probabilities for the chosen input difference for about 42.7 percent of all output differences. The most serious deviation between the two distributions occurs for the output difference (0x7e94,0x829f), which has an exact probability of 2^{-11}, but is predicted to appear with probability 2^{-9} under the Markov model; in empirical sampling, we found it in 48556 out of 10^8 output pairs.

Roughly 19.4 percent of output differences are in the Markov model predicted at probabilities that are three standard deviations from the values predicted by the exact model at this sample size. It is therefore easy to practically see the differences between the two models. Figure 2 gives a more detailed qualitative overview.

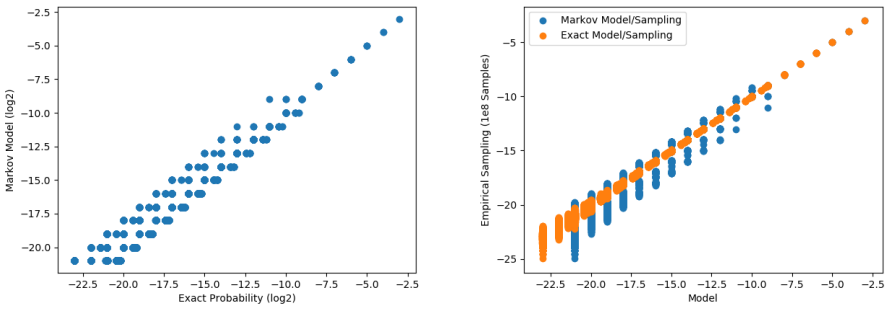


Fig. 2. A qualitative comparison between the differential distributions for 3-round Speck as obtained by exact calculation, the Markov model, and empirical sampling. *(Left)* Scatter plot showing the probabilities of all output differences for 3-round Speck32/64 with input difference $\Delta = (0x40, 0x0)$ as well as the corresponding Markov model predictions. Most of the points in the scatter plot can be resolved into horizontal point clouds at higher magnification. *(Right)* Comparison of both models with the results of sampling 10^8 outputs from the 3-round output distribution. No statistically significant deviations were observed between the predictions of the exact model and the results of sampling. Scatter points for the Markov model/sampling comparison partially obscured by the scatter plot for the exact model.

Testing the Models We subjected both the Markov model and the exact model to empirical testing at 10^8 samples. The Markov model predictions were clearly

incompatible with the results of sampling from the real cipher. The largest deviation between the values of our difference counters and the predictions of the exact model, however, was of size approximately 4.5σ when the standard deviation is calculated using the predicted probability, which is in the expected range given that we have 42149 counters to evaluate. Also smaller deviations between counters and predictions appeared at the expected rate.

In order to test the correctness of the Markov model, we designed a version of 3-round Speck32/64 that we theoretically expected to follow it. To this end, we inserted an additional 32-bit whitening key at the input to the third round. The first round transition is deterministic, so a uniform distribution of plaintext pairs within the input difference class gives for every value of the involved subkey rise to a uniform distribution of first-round output pairs within the difference class prescribed by the differential trail. The first round can therefore be ignored for the purposes of calculating the second round difference distribution. Our additional whitening key makes sure that ciphertext pairs entering the third round are likewise evenly distributed within the round-2 output difference class of the trail under consideration and that therefore the Markov assumption is fulfilled for the third round and therefore for the entire modified cipher. Indeed, sampling from this cipher yielded results that were within experimental error margins in agreement with the Markov model.

4.4 An Efficient Perfect Distinguisher for 5-Round Speck32/64

Efficiently Extending the Perfect Distinguisher to Five Rounds Given a five-round output pair C_0, C_1, we execute the following algorithm:

1. Set a counter $count := 0$.
2. For every possible third-round output difference Δ_3, do:
   (a) Determine the unique two-round trail $\Delta_3 \to \Delta_4 \to C_0 \oplus C_1$ that is the only possible link between Δ_3 and C_0, C_1.
   (b) Test if the individual transitions in the trail $\Delta_3 \to \Delta_4 \to C_0 \oplus C_1$ are possible. If yes, do:
      – Use Dinur's two-round attack [Din14] to enumerate all pairs of subkeys k for the last two rounds such that decrypting C_0, C_1 with the subkey pair k yields a partial decryption C'_0, C'_1 with $C'_0 \oplus C'_1 = \Delta_3$.
      – Calculate the decryption equivalence class of C'_0, C'_1 and look up the number $N(C'_0, C'_1)$ that connect it to the desired input difference in a precomputed decryption class distribution table for 3-round Speck32/64.
      – Update *count* to $count + N(C'_0, C'_1)$.
3. Return *count* as the number of keys linking C_0, C_1 to the input difference Δ.

For a randomly generated pseudo-ciphertext pair C, we expect to count $N_0 := 2^{32}$ keys. Given C, we therefore obtain a posterior probability p_{real} of C having been generated as a real pair in our game of $p_{real} = N_{keys}(C)/(N_{keys}(C) + N_0)$, where $N_{keys}(C)$ is the number of keys linking C to the input difference Δ.

Implementation Issues To keep memory requirements low, we keep the precomputed 3-round pair distribution in memory in compressed form and only decompress the distribution data for a single 3-round difference class when it is needed. This costs some speed, but means that we hold only around 55 Megabytes of precomputed data in memory instead of around 20 Gigabytes, as a naive implementation would do.

Performance of the Resulting Distinguisher We generated one million ciphertext pairs, of which roughly half were generated uniformly at random, while the rest were sampled from the real five-round output distribution. We then calculated the posterior real-or-random probabilities for all of these ciphertext pairs as described. Our distinguisher returned the result corresponding to ground truth in 944782 of these cases, corresponding to an overall accuracy of ≈ 94.5 percent, within error margins in agreement with the results previously reported in [Goh19].

Exploitation of Deep Round-to-Round Dependencies We used this data set to test whether the perfect distinguisher gains a significant signal from exploitation of events in the first two rounds of encryption. To this end, we ran a version of our distinguisher that omitted the calculation of decryption equivalence classes for round-3 output and instead used just the exact probabilities of intermediate output differences encountered at round 3 to approximate the number of keys linking round 3 candidate decryptions to the input difference. This distinguisher disagreed with the perfect distinguisher on 6775 pairs, with the perfect distinguisher returning the right label in 3952 or roughly 58 percent of these cases.

Some of the disagreements between these two very similar distinguishers were quite strong. The largest absolute difference in the test set was obtained for the ciphertext pair (0xdb07/0x3e18, 0xdfe4/0x3949), which was generated from the random distribution and given a posterior probability of being real of 97 percent by the simplified distinguisher, but recognised as random with 99 percent confidence by the perfect distinguisher.

4.5 A Known-Plaintext Brute Force Distinguisher for Reduced Speck32/64

Methods We also practically tried using meet-in-the-middle sampling to construct distinguishers against round-reduced Speck32/64. To distinguish r rounds of Speck32/64 with a two-block known plaintext P and ciphertext C, we encrypted P for $r_1 := \lceil r/2 \rceil$ rounds and decrypted C for $r_2 := r - r_1$ rounds. The obtained partial encryptions and decryptions were then matched by decryption equivalence class. Note that this matching by decryption equivalence class had the effect that the last subkey used encrypting towards the meet-in-the-middle point automatically became irrelevant.

Sampling was run with 10^8 keys in both directions and all collisions were counted. To this end, we kept the values obtained by partial encryption (after normalizing for decryption equivalence) in memory. After generation of the

desired number of examples, this array was sorted. Matching of the results of partial decryption was then performed by binary search over this sorted array.

Finally, the number of keys (in the setting where subkeys are independent and uniformly distributed) linking the given input pair with the given ciphertext pair was estimated from the result by assuming that sampling with n keys in this way covers to a good approximation $2^{16} \cdot n^2$ out of the total $2^{16 \cdot r}$ keys and that the tested keys can be viewed as uniformly distributed among all keys.

A Known Plaintext Distinguisher for Five-Round Speck32/64 We first used the resulting generic distinguisher to construct a two-block known-plaintext distinguisher against 5-round Speck32/64.

Plaintext pairs were thus sampled uniformly at random from the set of 64-bit bitstrings and encrypted by random keys sampled uniformly using the real Speck32/64 key schedule for five rounds. Half of the samples were then randomized by replacing the obtained ciphertext with a 64-bit bitstring chosen uniformly at random. Meet-in-the-middle sampling with 10^8 independent and uniformly distributed subkeys was then used to distinguish real from random samples. On a test set of size 1000 pairs, the ground truth label was returned in 581 cases.

As a point of comparison, assume that some single-mask linear distinguisher against 5-round Speck32/64 has a bias of ϵ, i.e. that we have

$$|\mathbb{P}(\lambda \cdot P \oplus \mu E_K(P) = 0) - \mathbb{P}(\lambda \cdot P \oplus \mu E_K(P) = 1)| = \epsilon,$$

where the probability is averaged over all choices of P and K and where $\lambda, \mu \in \{0,1\}^{32}$. Given two blocks of plaintext P_0, P_1 and corresponding ciphertext C_0, C_1, the optimal decision rule for an adversary using this linear property is then to assume that the given sample is from the real distribution if $\lambda(P_0 \oplus P_1) = \mu(C_0 \oplus C_1)$. Setting $p = 0.5 + \epsilon/2$, the probability of this event is $p^2 + (1-p)^2$ if P_0, P_1, C_0, C_1 is from the real distribution and 0.5 otherwise. Hence, a distinguisher of this kind achieves an overall accuracy a on two-block data of

$$a = \frac{\epsilon^2}{4} + 0.5,$$

which implies that the accuracy observed for our brute force distinguisher would only be reached at a $\epsilon \approx 0.56$. It would be interesting to know how close all-in-one linear distinguishers can come for these five rounds, assuming that the nonlinear component of the first round is removed; we leave this as a problem for further research.

Treating Differential Distinguishers as Known-Plaintext Distinguishers The meet-in-the-middle known-plaintext distinguisher will automatically exploit *chosen* plaintext very well at the sampling budget we used. We tested this by running our known-plaintext distinguisher without any changes on test data consisting of plaintext pairs P_0, P_1 with difference $\Delta = (0x40, 0x0)$ and corresponding ciphertext pairs C_0, C_1. We tested this with encryption for five and for seven rounds,

in both cases using the real Speck32/64 key schedule. We then ran the following experiments:

1. First, we tried to distinguish 5-round Speck32/64 with input difference Δ from random output. To this end, we generated a test set consisting of 1000 plaintext pairs and corresponding ciphertext exactly as before in the known-plaintext setting, but fixed the difference between plaintexts of each plaintext pair to Δ. We then ran both our meet-in-the middle distinguisher with a search budget of 10^8 samples for encryption and decryption and the perfect differential distinguisher on the ciphertext samples. In this experiment, the same decision was reached by both distinguishers in 985 cases; ground truth was matched in 950 cases by the perfect output pair distribution distinguisher and in 945 cases by the meet-in-the middle distinguisher. This shows that our meet-in-the-middle distinguisher is very close to exploiting the output distribution of 5-round Speck32/64 with the chosen input difference perfectly.
2. Next, we switched out the real plaintext pairs in our sample against freshly generated random plaintext pairs with input difference Δ. By this test, we hoped to find out whether the meet-in-the-middle distinguisher is able to gain some advantage from knowing the plaintext instead of just the difference between both inputs of a plaintext pair. No such advantage was detected in our trial. Indeed, accuracy even slightly increased, with ground truth being matched in 949 cases now and agreement with the distinguisher based on perfect knowledge of the output distribution rising to 993 cases.
3. We suspected that this apparent lack of an ability to exploit knowledge of the plaintext inputs in the preceding experiment was due to the fact that input pairs with input difference Δ always transition to the round-1 difference $(0x8000, 0x8000)$. Randomizing the inputs without changing the input difference therefore has no influence on the state difference after the first round when the input difference is Δ. We therefore repeated the previous two experiments with the input difference $(0x2800, 0x10)$ and found that for this input difference, randomizing the input data has a dramatic impact on the performance of our meet-in-the-middle distinguisher. With non-randomized plaintexts, our distinguisher matched ground truth in 859 out of 1000 trials; this is a reasonable level of performance, given that [BGPT21] report an accuracy of roughly 76 percent for the same problem using a neural distinguisher following the network architecture given in [Goh19]. After randomization of the input pairs (again keeping the input difference), the vast majority of the items in the randomized-input dataset where classified as random, irrespective of whether the output pair was generated uniformly at random or by sampling from the output distribution of 5-round Speck32/64 with input difference (0x2800,0x10). Out of our 1000 samples, only 201 were now classified as real, and ground truth was matched in only 568 cases.[3]

[3] The neural distinguisher reported by Benamira et al. is expected to achieve a lower accuracy than our meet-in-the-middle distinguisher simply because it uses less in-

4. Finally, we tested our meet-in-the-middle distinguisher also against 7-round Speck32/64, again using input pairs with fixed input difference Δ. We tested the 7-round version of our known-plaintext meet-in-the-middle distinguisher on a test set consisting of 4000 fixed-difference input pairs with corresponding output pairs. For comparison, we also ran a key averaging distinguisher based on a 6-round neural distinguisher on the output pairs; key averaging was performed for 1000 randomly selected keys using the six-round neural network and the key averaging implementation given in the supplementary data to [Goh19]. The search budget for the meet-in-the-middle distinguisher was again set to 10^8 samples. Both approaches achieved roughly the same empirical accuracy on the test set, showing that our meet-in-the-middle distinguishers work well on the 7-round problem.

5 Conclusions

The basic idea we have examined in this work is simple: namely, that direct counting of spurious keys, i.e. of non-target keys that decrypt some observed ciphertext to a plaintext that is consistent with our knowledge about the plaintext, yields a generic way of constructing all-in-one distinguishers for block ciphers. We have shown in this work that while the distinguishers so obtained are relatively expensive in terms of the computation time and memory required, the approach can be made quite practical for some interesting classes of distinguishers for small block ciphers when some relatively mild simplifying modifications to the cipher under study are allowed.

In this study, the main approximation of this kind we use is the replacement of the key schedule with uniformly distributed and independent choice of subkeys. In addition, we treat meet-in-the-middle sampling as an approximation to uniform sampling from the key space so extended. For five-round Speck with only the input difference known, we use some further optimizations to obtain a relatively efficient key-counting model for a situation where the independent-subkeys assumption is actually true and where our model hence becomes exact.[4] The models we consider yield various insights into optimal exploitation of the studied plaintext signals, including the slightly surprising finding that knowing the exact plaintext values does not seem to be useful to an all-in-one adversary trying to distinguish Speck32/64 with the input difference $(0x40, 0x0)$ from random data, assuming at least that the adversary sees only two output blocks and that they do not exploit the key schedule.

formation: it does not know the input pair. Using our framework, it is however straightforward to construct a brute force distinguisher that only uses the output pair and the input pair difference and thus solves the same problem. To this end, we simply constructed a meet-in-the-middle distinguisher for a version of Speck32/64 that has an initial 32-bit whitening key. This achieved 808 correct classifications for our 1000 input-output samples.

[4] Note that our implementation of this model does, however, still contain some errors in the computed ciphertext pair distribution due to the use of floating-point arithmetic.

We expect that the techniques developed in this paper will be generically useful in the study of small block ciphers, allowing for the refinement of the security assessment of round-reduced versions of such ciphers. They can also help to assess how much of the available signal is being seen by other approaches to distinguishing the same distributions, such as machine learning models. Likewise, the design of small good plaintext structures should profit from a generic, theoretically justified approach of testing them, even if such testing is relatively expensive.

Brute force cryptanalysis is capable of automatically and reliably solving a number of problems that would potentially need some human intervention if other techniques for automatic cryptanalysis are used: for instance, in the simple known-plaintext setting, brute force distinguishers will automatically circumvent arbitrary fixed initial permutations in a block cipher. Another attractive property of brute force distinguishers is that well-known cryptographic attacks are naturally obtained from them when additional modelling steps are justified: for instance, for Markov ciphers, the brute force attack using two-block input and output is immediately equivalent to an all-in-one differential attack using the input difference given by the two input blocks. Also key-averaging attacks such as in [Goh19, BGPT21] may be viewed simply as brute force attacks where part of the key sampling has been replaced by a heuristic model counting the number of good keys for some smaller part of the cipher under study.

For all these virtues, brute force distinguishers remain, on account of their genericity, fundamentally a blunt tool that can only produce relatively slow distinguishers. However, we expect that their further study will improve our understanding of other, more efficient attack modes on block ciphers and thereby ultimately improve the security guarantees offered especially by small and lightweight symmetric constructions.

Acknowledgments A significant part of this work was done when the author was working at BSI; their support was instrumental in making this paper possible. My thanks go out also to Friederike Laus, Le Van Schröer and Ernst Schulte-Geers for useful comments on an earlier version of this manuscript.

References

[AL12] Martin R Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In *International Conference on Selected Areas in Cryptography*, pages 1–15. Springer, 2012.

[ALLW14] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced simon and speck. In *International Workshop on Fast Software Encryption*, pages 525–545. Springer, 2014.

[Ber05] Daniel J Bernstein. Understanding brute force. In *Workshop Record of ECRYPT STVL Workshop on Symmetric Key Encryption, eSTREAM report*, volume 36, page 2005. Citeseer, 2005.

[BG12] Alex Biryukov and Johann Großschädl. Cryptanalysis of the full aes using gpu-like special-purpose hardware. *Fundamenta Informaticae*, 114(3-4):221–237, 2012.

[BGL+21] Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Conditional differential-neural cryptanalysis. *IACR Cryptol. ePrint Arch*, 719:2021, 2021.

[BGPT21] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. *IACR Cryptol. ePrint Arch*, 287:2021, 2021.

[BRV14] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers simon and speck. In *International Workshop on Fast Software Encryption*, pages 546–570. Springer, 2014.

[BSS+] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. *IACR eprint report*.

[BVLC16] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in arx: Application to block cipher speck. In *International Conference on Fast Software Encryption*, pages 289–310. Springer, 2016.

[CKL+21] Anupam Chattopadhyay, Mustafa Khairallah, Gaëtan Leurent, Zakaria Najm, Thomas Peyrin, and Vesselin Velichkov. On the cost of asic hardware crackers: A sha-1 case study. In *The Cryptographer's Track at the RSA Conference 2021*, 2021.

[DCR06] Christophe De Canniere and Christian Rechberger. Finding sha-1 characteristics: General results and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2006.

[Din14] Itai Dinur. Improved differential cryptanalysis of round-reduced speck. In *International Workshop on Selected Areas in Cryptography*, pages 147–164. Springer, 2014.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of cryptology*, 10(3):151–161, 1997.

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In *Annual International Cryptology Conference*, pages 150–179. Springer, 2019.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[HM97] Carlo Harpes and James L Massey. Partitioning cryptanalysis. In *International Workshop on Fast Software Encryption*, pages 13–27. Springer, 1997.

[KPP+06] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with copacobana–a cost-optimized parallel code breaker. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 101–118. Springer, 2006.

[Leu13] Gaëtan Leurent. Construction of differential characteristics in arx designs application to skein. In *Annual Cryptology Conference*, pages 241–258. Springer, 2013.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In *International Workshop on Fast Software Encryption*, pages 336–350. Springer, 2001.

[LP19] Gaëtan Leurent and Thomas Peyrin. From collisions to chosen-prefix collisions application to full sha-1. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 527–555. Springer, 2019.

[LS18] Gaëtan Leurent and Ferdinand Sibleyras. The missing difference problem, and its applications to counter mode encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 745–770. Springer, 2018.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of arx block ciphers with application to speck and lea. In *Australasian Conference on Information Security and Privacy*, pages 379–394. Springer, 2016.

[Vau96] Serge Vaudenay. An experiment on des statistical cryptanalysis. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 139–147, 1996.

[Vau03] Serge Vaudenay. Decorrelation: a theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, 2003.

[VOW99] Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12(1):1–28, 1999.

[Wag04] David Wagner. Towards a unifying view of block cipher cryptanalysis. In *International Workshop on Fast Software Encryption*, pages 16–33. Springer, 2004.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 19–35. Springer, 2005.