On the Indifferentiability of Key-Alternating Ciphers

Elena Andreeva[1], Andrey Bogdanov[2], Yevgeniy Dodis[3], Bart Mennink[1], and John P. Steinberger[4]

[1] KU Leuven and iMinds, {elena.andreeva, bart.mennink}@esat.kuleuven.be
[2] Technical University of Denmark, a.bogdanov@mat.dtu.dk
[3] New York University, dodis@cs.nyu.edu
[4] Tsinghua University, jpsteinb@gmail.com

Abstract. The Advanced Encryption Standard (AES) is the most widely used block cipher. The high level structure of AES can be viewed as a (10-round) *key-alternating* cipher, where a t-round key-alternating cipher KA_t consists of a small number t of fixed permutations P_i on n bits, separated by key addition:

$$\mathrm{KA}_t(K, m) = k_t \oplus P_t(\dots k_2 \oplus P_2(k_1 \oplus P_1(k_0 \oplus m))\dots),$$

where $(k_0, \dots, k_t)$ are obtained from the master key K using some key derivation function.
For $t = 1$, KA_1 collapses to the well-known Even-Mansour cipher, which is known to be *indistinguishable* from a (secret) random permutation, if P_1 is modeled as a (public) random permutation. In this work we seek for stronger security of key-alternating ciphers — *indifferentiability from an ideal cipher* — and ask the question under which conditions on the key derivation function and for how many rounds t is the key-alternating cipher KA_t indifferentiable from the ideal cipher, assuming $P_1, \dots, P_t$ are (public) random permutations?
As our main result, we give an affirmative answer for $t = 5$, showing that the 5-*round key-alternating cipher* KA_5 *is indifferentiable from an ideal cipher*, assuming $P_1, \dots, P_5$ are five independent random permutations, and the key derivation function sets all rounds keys $k_i = f(K)$, where $0 \le i \le 5$ and f is modeled as a random oracle. Moreover, when $|K| = |m|$, we show we can set $f(K) = P_0(K) \oplus K$, giving an n-bit block cipher with an n-bit key, making only six calls to n-bit permutations $P_0, P_1, P_2, P_3, P_4, P_5$.
Keywords. Even-Mansour, ideal cipher, key-alternating cipher, indifferentiability.

1 Introduction

Block Ciphers. A block cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ takes a κ-bit key K and an n-bit input x and returns an n-bit output y. Moreover, for each key K the map $E(K, \cdot)$ must be a permutation, and come with an efficient inversion procedure $E^{-1}(K, \cdot)$. Block ciphers are central primitives in cryptography. Most importantly, they account for the bulk of data encryption and data authentication occurring in the field today, as well as play a critical role in the design of "cryptographic hash functions" [13, 36, 42, 50].

Indistinguishability. The standard security notion for block ciphers is that of (computational) *indistinguishability* from a random permutation, which states that no computationally bounded distinguisher $\mathcal{D}$ can tell apart having oracle access to the block cipher $E(K, \cdot)$ or its inverse $E^{-1}(K, \cdot)$ for a *random key* K from having oracle access to a (single) truly random permutation P and its inverse P^{-1}. This security notion is relatively well understood in the theory community, and is known to be implied by the mere existence of one-way functions, through a relatively non-trivial path: from one-way functions to pseudorandom generators [38], to pseudorandom functions (PRFs) [32], to pseudorandom permutations (PRPs) [45], where the latter term is also a "theory synonym" for the "practical notion" of a block cipher. Among these celebrated results, we explicitly note the seminal work of Luby-Rackoff [45], who proved that four (independently keyed) rounds of the Feistel network $(L', R') = (R, f(K, R) \oplus L)$, also known as the "Luby-Rackoff construction", are enough to obtain a PRP $E((K_1, K_2, K_3, K_4), (L_0, R_0))$ on n-bit inputs/outputs from four $n/2$-to-$n/2$-bits PRFs $f(K_1, R_0), \ldots, f(K_4, R_3)$. In fact, modulo a few exceptions mentioned below, the Luby-Rackoff construction and its close relatives were the *only theoretically-analyzed* ways to build a block cipher.

Is Indistinguishability Enough? Despite this theoretical success, practical ciphers — including the current block cipher standard AES — are built using very different means. One obvious reason is that the theoretical feasibility results above are generally too inefficient to be of practical use (and, as one may argue, were not meant to be). However, a more subtle but equally important reason is that a practitioner — even the one who understands enough theory to know what a PRP is — would not think of a block cipher as a synonym of a PRP, but as something *much stronger*!

For example, the previous U.S. block cipher standard DES had the following so called "key complementary" property $E(\bar{K}, \bar{x}) = \overline{E(K, x)}$, where $\bar{y}$ stands for the bitwise complement of the string y. Although such an equality by itself does not contradict the PRP property, though effectively reducing the key space by a half, it was considered undesirable and typically used as an example of something that a "good" block cipher design should definitely avoid. Indeed, AES is not known to have any simple-to-express relations between its inputs/outputs on related keys. Generally speaking though, related-key attacks under more complex related-key relations (using nonlinear functions on the master key) for AES were identified and received a lot of attention in the cryptanalytic community several years ago [6–8], despite not attacking the standard PRP security. In fact, the recent biclique cryptanalysis of the full AES cipher [11] in the single-key setting implicitly uses the similarity of AES computation under related keys.

Indeed, one of the reasons that practical block ciphers are meant to have stronger-than-PRP properties is that various applications (e.g. [10,23,29,33,36, 39,40,42,49,50,55]) critically rely on such "advanced properties", which are far and beyond the basic indistinguishability property. Perhaps the most important such example comes in the area of building good "hash functions", as many cryptographic hash functions, including the most extensively used SHA-1/2 and MD5 functions, use the famous block-cipher-based Davies-Meyer compression function $f(K,x) = E(K,x) \oplus x$ in their design.[5] This compression function f is widely believed to be collision-resistant (CR) if E is a "good-enough" block cipher (see more below), but this obviously does not follow from the basic PRP property. For example, modifying any good block cipher E to be the identity permutation on a single key K' clearly does not affect it PRP security much (since, w.h.p., a random key $K \neq K'$), but then $f(K',x) = x \oplus x = 0$ for all x, which is obviously not CR. While the example above seems artificial, we could instead use a natural and quite popular Even-Mansour (EM) [29] cipher $E(K,x) = P(K \oplus x) \oplus K$, where P is some "good-enough" public permutation. As we mention below, the EM cipher is known to be indistinguishable [29] assuming P is a public "random permutation", and, yet, the composed Davies-Meyer hash function $f(K,x) = E(K,x) \oplus x = P(K \oplus x) \oplus (K \oplus x)$ is certainly not CR, as any pair $(K,x) \neq (K',x')$ satisfying $K \oplus x = K' \oplus x'$ yields a collision.

Ideal Cipher Model. Motivated by these (and other) considerations, practitioners view a good block cipher as something much closer to an *ideal cipher* than a mere PRP, much like they view a good hash function much closer to a *random oracle* than a one-way (or collision-resistant) function. In other words, many important applications of block ciphers (sometimes implicitly) assume that E "behaves" like a family $\mathcal{IC}$ of 2^κ completely random and independent permutations $P_1, \ldots, P_{2^\kappa}$. More formally, an analysis in the ideal cipher model assumes that all parties, including the adversary, can make (a bounded number of) both encryption and decryption queries to the ideal block cipher $\mathcal{IC}$, for any given key K (not necessarily random!). Indeed, under such an idealistic assumption one can usually *prove* the security of most of the above mentioned applications of block ciphers [23,29,33,36,39,40,42,49,50,55], such as a simple and elegant proof that the Davies-Meyer compression function $f(K,x) = E(K,x) \oplus x$ is CR in the ideal cipher model (ICM) [55].

Of course, the ideal cipher model is ultimately a heuristic, and one can construct artificial schemes that are secure in the ICM, but insecure for any concrete block cipher [9]. Still, a proof in the ideal cipher model seems useful because it shows that a scheme is secure against generic attacks, that do not exploit specific weaknesses of the underlying block cipher. Even more important than potential applications, the ICM gives the block cipher designers a much "higher-than-PRP" *goal* that they should strive to achieve in their proposed designs, even though this goal is, theoretically-speaking, impossible to achieve. This raises an important question to the theory community if it is possible to offer some

[5] Where E is some particular block cipher; e.g., in the case if SHA-1/2, it was called SHACAL [34,35].

theoretical framework within which one might be able to evaluate the design of important block ciphers, such as AES, in terms of being "close" to an ideal cipher or, at least, resisting generic "structure-abusing" attacks.

INDIFFERENTIABILITY. One such framework is the so-called *indifferentiability* framework of Maurer et al. [46], popularized by Coron et al. [16] as a clean and elegant way to formally assess security of various idealized constructions of hash functions and block ciphers. Informally, given a construction of one (possibly) idealized primitive B (i.e., block cipher) from *another idealized* primitive A (i.e., random oracle), the indifferentiability framework allows one to formally argue the security of B in terms of (usually simpler) A. Thus, although one does not go all the way to building B from scratch, the indifferentiability proof illustrates the lack of "generic attacks" on B, and shows that any concrete attack must use something about the internals of any candidate implementation of A. Moreover, the indifferentiability framework comes with a powerful composition theorem [46] which means that most natural (see [51]) results shown secure in the "ideal-B" model can safely use the construction of B using A instead, and become secure in the "ideal-A" model.

For example, we already mentioned that the design of popular hash functions, such as SHA-1/2 and MD5, could be generically stated in terms of some underlying block cipher E. Using the indifferentiability framework, one can *formally* ask if the resulting hash function is indifferentiable from a random oracle if E is an ideal cipher. Interestingly, Coron et al. [16] showed a negative answer to this question. Moreover, this was not a quirk of the model, but came from a well-known (and serious) "extension" attack on the famous Merkle-Damgård domain extension [21, 47]. Indeed, an attack on indifferentiability usually leads to a serious real-world attack for some applications, and, conversely, the security proof usually tells that the high-level design of a given primitive (in this case hash function) does not have structural weaknesses. Not surprisingly, all candidates for the recently concluded SHA-3 competition were strongly encouraged to come with a supporting indifferentiability proof in some model (as we will expand on shortly).

RANDOM ORACLE VS. IDEAL CIPHER. Fortunately, Coron et al. [16] also showed that several simple tweaks (e.g., truncating the output or doing prefix-free input encoding) make the resulting hash function construction indifferentiable from a random oracle. Aside from formally showing that the ICM model "implies" the random oracle model (ROM) in theory, these (and follow-up [3, 14]) positive results showed that (close relatives of) *practically used* constructions are "secure" (in the sense of resisting *generic* attacks, as explained above).

From the perspective of this work, where we are trying to validate the design principle behind existing block ciphers, the opposite direction (of building an ideal cipher from a random oracle) is much more relevant. Quite interestingly, it happened to be significantly more challenging than building a PRP out of a PRF. Indeed, the most natural attempt is to use the already mentioned Feistel construction, that uses the given random oracles f to implement the required round functions.[6] However, unlike the standard PRF-PRP case, where four rounds were already sufficient [45], in the indifferentiability setting even five rounds are provably insecure [15, 16, 25]. On a high-level, the key issue is that in the latter framework the distinguisher can have direct access to all the intermediate round functions, which was provably impossible in the more restricted indistinguishability framework. As a step towards overcoming this difficulty, Dodis and Puniya [25] considered a variant of the indifferentiability framework called "honest-but-curious" (HBC) indifferentiability, where the adversary can only query the global Feistel construction, and get all the intermediate results, but cannot directly query the round functions. In this model, which turns out to be *incomparable* to "standard" indifferentiability [15], they showed that the Feistel construction with a super-logarithmic number of rounds (with random oracle round functions) is HBC-indifferentiable from a fixed ideal permutation. The elegant work of Coron et al. [15] (and later Seurin [54]) conjectured and attempted a "standard" indifferentiability proof for the Feistel construction with six rounds. Unfortunately, while developing several important techniques, the proof contained some non-trivial flaws. Fortunately, this result was later fixed by Holenstein et al. [37], who succeeded in proving that a fourteen-round Feistel construction can be used to build an ideal cipher from a random oracle.

Key-Alternating Ciphers. Despite this great theoretical success showing the equivalence between the random oracle and the ideal cipher models, the above results of [15, 37, 54] only partially address our main motivation of theoretically studying the soundness of the design of *existing* block ciphers. In particular, we notice that (from a high level) there are two major design principles for block ciphers. The "old school" approach is indeed Feistel-based, with many prominent ciphers such as DES, Blowfish, Camellia, FEAL, Lucifer, and MARS. However, it appears that all such ciphers use *rather weak* (albeit non-trivial) round functions, and (in large part) get their security by using *many more* rounds than theoretically predicted. So, while the theoretical soundness of the Feistel network is important philosophically, it is unclear that random oracle modeling of the round functions is realistic.

In fact, we already mentioned a somewhat paradoxical fact: while, in theory, the random oracle model appears much more basic and minimal than the highly structured ideal cipher model (much like a one-way function is more basic than a one-way permutation), in practice, the implication appears to be *totally reversed*. In particular, in practice it appears much more accurate to say that hash functions (or "random oracles") are built from block ciphers (or "ideal ciphers") than the other way around. Indeed, in addition to the widely used SHA-1/2 and

[6] The most natural modeling would give a *single* n-to-n-bit permutation from several $n/2$-to-$n/2$-bit random oracles. However, by prepending the *same* κ-bit key K to each such RO, one gets a candidate block cipher. We notice, though, that unlike the secret-key setting, it is (clearly) *not* secure to prepend several *independent* keys to each round function. We will come back to this important point when discussing the importance of key derivation in the indifferentiability proofs.

MD5 examples, other prominent block-cipher-based hash functions are recent SHA-3 finalists BLAKE [2] and Skein [30].

Perhaps most importantly for us, the current block cipher standard AES, as well as a few other "new school" ciphers (e.g., 3-Way, SHARK, Serpent, Present, and Square), are *not Feistel-based.* Instead, such ciphers are called *key-alternating ciphers*, and their design goes back to Daemen [18–20]. In general, a key-alternating cipher KA_t consists of a small number t of fixed permutations P_i on n bits, separated by key addition:

$$\mathrm{KA}_t(K, m) = k_t \oplus P_t(\ldots k_2 \oplus P_2(k_1 \oplus P_1(k_0 \oplus m))\ldots),$$

where the round keys $k_0, \ldots, k_t$ are derived from the master key K using some *key derivation* (aka "key schedule") function. For one round $t = 1$, the construction collapses to the well-known Even-Mansour (EM) [29] cipher. Interestingly, already in the standard "PRP indistinguishability" model, the analysis of the EM [29] (and more general key-alternating ciphers [12]) seems to require the modeling of P as a *random permutation* (but, on the other hand, does not require another computational assumption such as a PRF). With this idealized modeling, one can show that the Even-Mansour cipher is indistinguishable [29], and, in fact, its exact indistinguishablity security increases beyond the "birthday bound" as the number of round increases to 2 and above [12, 28].

Our Main Question. Motivated by the above discussion, we ask the main question of our work:

> *Under which conditions on the key derivation function and for how many rounds t is the key-alternating cipher KA_t indifferentiable from the ideal cipher, assuming $P_1, \ldots, P_t$ are random permutations?*

As we mentioned, one motivation for this question comes from the actual design of the AES cipher, whose design principles we are trying to analyze. The second motivation comes from the importance of having the composition theorem guaranteed by the indifferentiability framework. Indeed, we already saw a natural example where using the Even-Mansour cipher to instantiate the classical Davies-Meyer compression function gave a totally insecure construction, despite the fact that the Davies-Meyer construction was known to be collision-resistant in the ideal cipher model [55], and the EM cipher indistinguishable in the random permutation model [29]. The reason for that is the fact that the EM cipher is easily seen to be not indifferentiable from an ideal cipher. In contrast, if we were to use a variant of the key alternating cipher which *is* provably indifferentiable, we would be *guaranteed* that the composed Davies-Meyer function remains collision-resistant (now, in the random permutation model).

The third motivation comes from the fact that the direct relationship between the random permutation (RP) model and the ideal cipher model is interesting in its own right. Although we know that these primitives are equivalent through the chain "IC $\Rightarrow$ RP (trivial) $\Rightarrow$ RO [24, 26] $\Rightarrow$ IC [15, 37]", a direct "RP $\Rightarrow$ IC" implication seems worthy of study in its own right (and was mentioned as an open problem in [17]).[7] More generally, we believe that the random permutation model (RPM) actually deserves its own place alongside the ROM and the ICM. The reason is that both the block cipher standard AES and the new SHA-3 standard Keccak [4] (as well as several other prominent SHA-3 finalists Grøstl [31] and JH [56]) are most cleanly described using a (constant number of) *permutation(s)*. The practical reason appears to be that it seems easier to ensure that the permutation design does not lose any entropy (unlike an ad-hoc hash function), or would not have some non-trivial relationship among different keys (unlike an ad-hoc block cipher). Thus, we find the indifferentiability analyses in the RPM very relevant both in theory and in practice. Not surprisingly, there has been an increased number of works as of late analyzing various constructions in the RPM [5, 12, 24, 26, 44, 52, 53].

Our Main Result. As our main result, we show the following theorem.

Theorem 1. *The* 5*-round key-alternative cipher* KA_5 *is indifferentiable from an ideal cipher, assuming* $P_1, \ldots, P_5$ *are five independent random permutations, and the key derivation function sets all rounds keys* $k_i = f(K)$*, where* $0 \leq i \leq 5$ *and* f *is modeled as a* κ*-to-*n*-bits random oracle.*

A more detailed statement appears in Theorem 3. In particular, our indifferentiability simulator has provable security $O(q^{10}/2^n)$, running time $O(q^3)$, and query complexity $O(q^2)$ to answer q queries made by the distinguisher. Although (most likely) far from optimal, our bounds are (unsurprisingly) much better than the $O(q^{16}/2^{n/2})$ and $O(q^8)$ provable bounds achieved by following the indirect "random-oracle route" [37].

We also show a simple attack illustrating that a one- or even two-round KA_t construction is never indifferentiable from the ideal cipher (in the full version of this paper [1]). This should be contrasted with the simpler indistinguishability setting, where the 1-round Even-Mansour construction is already secure [29]. Indeed, as was the case with Merkle-Damgård based hash function design and the "extension attack", the Davies-Meyer composition fiasco of the 1-round EM cipher demonstrated that this lack of indifferentiability indeed leads to a serious real-world attack on this cipher.

Finally, we give some justification of why we used 5 rounds, by attacking several "natural" simulators for the 4-round construction.

Importance of Key Derivation. Recall, in the secret-key indistinguishability case, the key derivation function was only there for the sake of minimizing the key length, and having $t+1$ independent keys $k_0, \ldots, k_t$ resulted in the best security analysis. Here, the key K is *public* and controlled by the attacker. In particular, it is trivial to see that having $t+1$ independent keys is like having a one-round construction (as then the attacker can simply fix all-but-one-keys k_i), which we know is trivially insecure. Thus, in the indifferentiability setting it is very important that the keys are somehow correlated (e.g., equal).

[7] Indeed, our efficiency and security below are much better than following the indirect route through random oracle.

Another important property for the key derivation functions, at least if one wants to optimize the number of rounds, appears to be its *invertibility.* Very informally, this means that the only way to compute a valid round k_i is to "honestly compute" a key derivation function f on some key K first. In particular, in our analysis we use a random oracle as such a non-invertible key derivation function. We give some evidence of the importance of invertibility for understanding the indifferentiability-security of key-alternating ciphers by (1) critically using such non-invertibility in our analysis; and (2) showing several somewhat surprising attacks for the 3-round construction with certain natural "invertible" key schedules (e.g., all keys k_i equal to K for $\kappa = n$). We stress that our results do not preclude the use of invertible key schedules for a sufficiently large number of rounds (say, 10-12), but only indicate why having non-invertible key schedules is very helpful in specific analyses (such as ours) and also for avoiding specific attacks (such as our 3-round attacks). Indeed, subsequent to our work, Lampe and Seurin [43] showed that the 12-round key alternating cipher will all keys $k_i = K$ (for $\kappa = n$) is indeed indifferentiable from an ideal cipher, with security $O(q^{12}/2^n)$ and simulator query complexity $O(q^4)$ to answer q queries made by the distinguisher. Although using substantially more rounds and achieving noticeably looser exact security than this work, their result is closer to the actual design of the AES cipher, whose key schedule f is indeed easily invertible.

Instantiating the Key Derivation Function. Although we use a random oracle as a key derivation function (see above), in principle one can easily (and efficiently!) build the required random oracle from a random permutation [24,26], making the whole construction entirely permutation-based. For example, the most optimized "enhanced-CBC" construction from [24] will use only a single additional random permutation and make $\frac{2\kappa}{n} + O(1)$ calls to this permutation to build a κ-to-n-bit random oracle f.[8] Unsurprisingly, this instantiation will result in a cipher making a lot fewer calls to the random permutation (by a large constant factor) than following the indirect RP-to-RO-to-IC cycle.

Moreover, we can further optimize the most common case $\kappa = n$ as follows. First, [24] showed that $f(K) = P(K) \oplus P^{-1}(K)$ is $O(q^2/2^n)$-indifferentiable from an n-to-n-bit random oracle, which already results in a very efficient block cipher construction with 7 permutation calls. Second, by closely examining our proof, we observe that we do not need the full power of the random oracle f for key derivation. Instead, our proof only uses the "preimage awareness" [27] of the random oracle[9] and the fact that random oracle avoids certain simple combinatorial relations among different derived keys. In particular, we observe that the "unkeyed Davies-Meyer" function [24] $f(K) = P(K) \oplus K$ is enough for our analysis to go through. This gives the following result for building an n-bit ideal cipher with n-bit key, using only six random permutation calls.

[8] The indifferentiability security of this construction to handle q queries is "only" $O(q^4/2^n)$, but this is still much smaller than the bound in Theorem 3, and will not affect the final asymptotic security.

[9] Informally, at any point of time the simulator knows the list of all input-output pairs to f "known" by the distinguisher.

Theorem 2. *The following n-bit cipher with n-bit key is indifferentiable from an ideal cipher:*

$$E(K, m) = k \oplus P_5(k \oplus P_4(k \oplus P_3(k \oplus P_2(k \oplus P_1(k \oplus m))))),$$

where $k = P_0(K) \oplus K$ and $P_0, P_1, P_2, P_3, P_4, P_5$ are random permutations.

Overall, our results give the first theoretical evidence for the design soundness of key-alternative ciphers — including AES, 3-Way, SHARK, Serpent, Present, and Square — from the perspective of indifferentiability.[10]

2 Preliminaries

For a domain $\{0,1\}^m$ and a range $\{0,1\}^n$, a random oracle $\mathcal{R} : \{0,1\}^m \to \{0,1\}^n$ is a function drawn uniformly at random from the set of all possible functions that map m to n bits. For two sets $\{0,1\}^\kappa$ and $\{0,1\}^n$, an ideal cipher $\mathcal{IC}$: $\{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ is taken randomly from the set of all block ciphers with key space $\{0,1\}^\kappa$ and message and ciphertext space $\{0,1\}^n$. A random permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ is a function drawn randomly from the set of all n-bit permutations.

Key-Alternating Ciphers. A key-alternating cipher KA_t consists of a small number t of fixed permutations P_i on n bits separated by key addition:

$$\mathrm{KA}_t(K, m) = k_t \oplus P_t(\ldots k_2 \oplus P_2(k_1 \oplus P_1(k_0 \oplus m))\ldots),$$

where the round keys $k_0, \ldots, k_t$ are derived from the master key K using some key schedule f: $(k_0, \ldots, k_t) = f(K)$. The notion of key-alternating ciphers itself goes back to Daemen [18–20] and was used in the design of AES. However, it was Knudsen [41] who proposed to instantiate multiple-round key-alternating ciphers with randomly drawn, fixed and public permutations (previously, a single-round key-alternating construction was proposed by Even-Mansour [29]).

Indifferentiability. We use the notion of indifferentiability [16, 46] in our proofs to show that if a construction $\mathcal{C}^\mathcal{P}$ based on an ideal subcomponent $\mathcal{P}$ is indifferentiable from an ideal primitive $\mathcal{R}$, then $\mathcal{C}^\mathcal{P}$ can replace $\mathcal{R}$ in any system. As noticed in [51] the latter statement must be qualified with some fine print: since the adversary must eventually incorporate the simulator, the indifferentiability composition theorem only applies in settings where the adversary comes from a computational class that is able to "swallow" the simulator (e.g., the class of polynomial-time, polynomial-space algorithms); see [22, 51] for more details on the limitations of indifferentiability.

Definition 1. *A Turing machine $\mathcal{C}$ with oracle access to an ideal primitive $\mathcal{P}$ is called $(t_D, t_S, q, \varepsilon)$-indifferentiable from an ideal primitive $\mathcal{R}$ if there exists a*

[10] We also mention a complementary recent work of [48], who mainly looked at "weaker-than-indistinguishability" properties which can be proven about AES design.

simulator $\mathcal{S}$ with oracle access to $\mathcal{R}$ and running in time t_S, such that for any distinguisher D running in time at most t_D and making at most q queries, it holds that:

$$\mathrm{Adv}^{\mathrm{indif}}_{\mathcal{C},\mathcal{R},\mathcal{S}}(D) = \left|\Pr\left[D^{\mathcal{C}^{\mathcal{P}},\mathcal{P}} = 1\right] - \Pr\left[D^{\mathcal{R},\mathcal{S}^{\mathcal{R}}} = 1\right]\right| < \varepsilon.$$

Distinguisher D can query both its *left oracle* (either $\mathcal{C}$ or $\mathcal{R}$) and its *right oracle* (either $\mathcal{P}$ or $\mathcal{S}$). We refer to $\mathcal{C}^{\mathcal{P}}, \mathcal{P}$ as the *real world*, and to $\mathcal{R}, \mathcal{S}^{\mathcal{R}}$ as the *simulated world*.

3 Indifferentiability of KA_5

In this section we discuss our main result, namely that KA_5 with an RO key schedule is indifferentiable from an ideal cipher. In the statement below, KA_5 stands for a 5-round key-alternating cipher implemented with round functions $P_1, \ldots, P_5$ and key scheduling function f, with the round functions, their inverses, and the key scheduling function all being available for oracle queries by the adversary (and thus, also, all being implemented as interfaces by the simulator).

Theorem 3. *Let $P_1, \ldots, P_5$ be independent random n-bit permutations, and f be a random κ-to-n-bits function. Let D be an arbitrary information-theoretic distinguisher that makes at most q queries. Then there exists a simulator $\mathcal{S}$ such that*

$$\mathrm{Adv}^{\mathrm{indif}}_{KA_5,\mathcal{IC},\mathcal{S}}(D) \leq 320 \cdot 6^{10}\left(\frac{q^{10}}{2^n} + \frac{q^4}{2^n}\right) = O\left(\frac{q^{10}}{2^n}\right),$$

where $\mathcal{S}$ makes at most $2q^2$ queries to the ideal cipher $\mathcal{IC}$ and runs in time $O(q^3)$.

Our 5-round simulator $\mathcal{S}$ is given by the pseudocode in game G_1 (see Figures 1–4), and more precisely by the public functions f, P1, P1^{-1}, P2, P2^{-1}, ..., P5, P5^{-1} within G_1. Here f emulates the key scheduling random oracle, whereas P1, P1^{-1} emulate the random permutation P_1 and its inverse P_1^{-1}, and so on. Since the pseudocode of game G_1 is not easy to assimilate, a high-level description of our simulator is likely welcome. Furthermore, because the simulator is rather complex, we also try to argue the necessity of its complex behavior by discussing why some simpler classes of simulators might not work.

To describe the simulator-distinguisher interaction we use expressions such as "D makes the query $\mathrm{f}(K) \rightarrow k$" to mean that the distinguisher D queries f (which is implemented by the simulator) on input K, and receives answer k as a result. The set of values k for which the adversary has made a query of the form $\mathrm{f}(K) \rightarrow k$ for some $K \in \{0,1\}^\kappa$ is denoted $\mathcal{Z}$ (thus $\mathcal{Z}$ is a time-dependent set). If $\mathrm{f}(K) \rightarrow k$ then we also write K as "$f^{-1}(k)$"; here f and f^{-1} are internal tables maintained by the simulator to keep track of scheduled keys and their preimages (see procedure $\mathrm{f}(K)$ in Figure 1 for more details).

```
Game G_1

Random tapes: $p_1, \ldots, p_5, \{p_E[K] : K \in \{0,1\}^\kappa\}, r_f$

private procedure ReadTape($Table, x, p$)
    $y \leftarrow p(x)$
    if ($Table(x) \neq \perp$) then abort
    if ($Table^{-1}(y) \neq \perp$) then abort
    $Table(x) \leftarrow y$
    $Table^{-1}(y) \leftarrow x$
    return $y$

public procedure E($K, x$)
    if ($ETable[K](x) \neq \perp$) return $ETable[K](x)$
    $y \leftarrow$ ReadTape($ETable[K], x, p_E[K](\rightarrow, \cdot)$)
    return $y$

public procedure E$^{-1}$($K, y$)
    if ($ETable[K]^{-1}(y) \neq \perp$) return $ETable[K]^{-1}(y)$
    $x \leftarrow$ ReadTape($ETable[K]^{-1}, y, p_E[K](\leftarrow, \cdot)$)
    return $x$

public procedure f($K$)
    if $f(K) \neq \perp$ return $f(K)$
    $k \leftarrow r_f(K)$
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup k$
    $f(K) \leftarrow k$
    $f^{-1}(k) \leftarrow K$
    $KeyQueries \leftarrow KeyQueries \cup \{(K, k, ++qnum)\}$
    if ($qnum > 6q^2 + q$) then abort
    return $f(K)$

public procedure P1($x$)
    if ($P_1(x) \neq \perp$) return $P_1(x)$
    $y \leftarrow$ ReadTape($P_1, x, p_1(\rightarrow, \cdot)$)
    AddQuery($1, x, y, \rightarrow$)
    return $P_1(x)$

public procedure P1$^{-1}$($y$)
    PrivateP1$^{-1}$($y$)
    Cleanup()
    return $P_1^{-1}(y)$
```

```
Game G_1 (continued)

private procedure PrivateP1$^{-1}$($y$)
    if ($P_1^{-1}(y) \neq \perp$) return $P_1^{-1}(y)$
    $x \leftarrow$ ReadTape($P_1^{-1}, y, p_1(\leftarrow, \cdot)$)
    AddQuery($1, x, y, \leftarrow$)
    FreezeLeftValues($x, \perp$)
    $LeftQueue \leftarrow LeftQueue \cup (1^+, y)$
    return $P_1^{-1}(y)$

public procedure P2($x$)
    if ($P_2(x) \neq \perp$) return $P_2(x)$
    $y \leftarrow$ ReadTape($P_2, x, p_2(\rightarrow, \cdot)$)
    AddQuery($2, x, y, \rightarrow$)
    return $P_2(x)$

public procedure P2$^{-1}$($y$)
    if ($P_2^{-1}(y) \neq \perp$) return $P_2^{-1}(y)$
    $x \leftarrow$ ReadTape($P_2^{-1}, y, p_2(\leftarrow, \cdot)$)
    AddQuery($2, x, y, \leftarrow$)
    return $P_2^{-1}(y)$

public procedure P3($x$)
    PrivateP3($x$)
    Cleanup()
    return $P_3(x)$

private procedure PrivateP3($x$)
    if ($P_3(x) \neq \perp$) return $P_3(x)$
    $y \leftarrow$ ForcedP3($3^-, x$)
    if ($y \neq \perp$) then
        if ($y \in$ range($P_3$)) then abort
        $P_3(x) \leftarrow y$
        $P_3^{-1}(y) \leftarrow x$
        AddQuery($3, x, y, \perp$)
        $RightQueue \leftarrow RightQueue \cup (3^+, y)$
    else
        $y \leftarrow$ ReadTape($P_3, x, p_3(\rightarrow, \cdot)$)
        AddQuery($3, x, y, \rightarrow$)
    end if
    $LeftQueue \leftarrow LeftQueue \cup (3^-, x)$
    return $P_3(x)$
```

Fig. 1: The simulated world (first of four sets of procedures).

A triple (i, x, y) such that D has made the query $\mathrm{P}i(x) \rightarrow y$ or $\mathrm{P}i^{-1}(y) \rightarrow x$ is called an *i-query*, $i \in \{1, 2, 3, 4, 5\}$. Moreover, when the simulator "internally defines" a query $\mathrm{P}i(x) = y$, $\mathrm{P}i^{-1}(y) = x$ we also call the associated triple (i, x, y) an i-query, even though the adversary might not be aware of these values yet. (While this might seem a little informal, we emphasize that this section is, indeed, meant mainly as an informal overview.) A pair of queries (i, x_i, y_i), $(i+1, x_{i+1}, y_{i+1})$ such that $y_i \oplus k = x_{i+1}$ for some $k \in \mathcal{Z}$ is called *k-adjacent*. We also say that a pair of queries $(1, x_1, y_1)$, $(5, x_5, y_5)$ is *k-adjacent* if $k \in \mathcal{Z}$ and $\mathrm{E}(f^{-1}(k), x_1 \oplus k) = y_5 \oplus k$, where $\mathrm{E}(K, x)$ is the ideal cipher (and $\mathrm{E}^{-1}(K, y)$ its inverse). (Since $\mathcal{Z}$ is time-dependent, a previously non-adjacent pair of queries might become adjacent later on; of course, this is unlikely.) A sequence of queries

$$(1, x_1, y_1), (2, x_1, y_2), \ldots, (5, x_5, y_5)$$

```
Game $G_1$ (continued)

public procedure $P3^{-1}(y)$
    $PrivateP3^{-1}(y)$
    Cleanup()
    return $P_3^{-1}(y)$

private procedure $PrivateP3^{-1}(y)$
    if $(P_3^{-1}(y) \neq \perp)$ return $P_3^{-1}(y)$
    $x \leftarrow ForcedP3(3^+, y)$
    if $(x \neq \perp)$ then
        if $(x \in domain(P_3))$ then abort
        $P_3(x) \leftarrow y$
        $P_3^{-1}(y) \leftarrow x$
        AddQuery$(3, x, y, \perp)$
        $LeftQueue \leftarrow LeftQueue \cup (3^-, x)$
    else
        ReadTape$(P_3^{-1}, y, p_3(\leftarrow, \cdot))$
        AddQuery$(3, x, y, \leftarrow)$
    end if
    $RightQueue \leftarrow RightQueue \cup (3^+, y)$
    return $P_3^{-1}(y)$

public procedure $P4(x)$
    if $(P_4(x) \neq \perp)$ return $P_4(x)$
    $y \leftarrow$ ReadTape$(P_4, x, p_4(\rightarrow, \cdot))$
    AddQuery$(4, x, y, \rightarrow)$
    return $P_4(x)$

public procedure $P4^{-1}(y)$
    if $(P_4^{-1}(y) \neq \perp)$ return $P_4^{-1}(y)$
    $x \leftarrow$ ReadTape$(P_4^{-1}, y, p_4(\leftarrow, \cdot))$
    AddQuery$(4, x, y, \leftarrow)$
    return $P_4^{-1}(y)$

public procedure $P5(x)$
    $PrivateP5(x)$
    Cleanup()
    return $P_5(x)$

private procedure $PrivateP5(x)$
    if $(P_5(x) \neq \perp)$ return $P_5(x)$
    $y \leftarrow$ ReadTape$(P_5, x, p_5(\rightarrow, \cdot))$
    AddQuery$(5, x, y, \rightarrow)$
    FreezeRightValues$(y, \perp)$
    $LeftQueue \leftarrow LeftQueue \cup (5^-, x)$
    return $P_5(x)$
```

```
Game $G_1$ (continued)

public procedure $P5^{-1}(y)$
    if $(P_5^{-1}(y) \neq \perp)$ return $P_5^{-1}(y)$
    $x \leftarrow$ ReadTape$(P_5^{-1}, y, p_5(\leftarrow, \cdot))$
    AddQuery$(5, x, y, \leftarrow)$
    return $P_5^{-1}(y)$

private procedure FreezeLeftValues$(x_1, k^\star)$
    forall $k \in \mathcal{Z} \backslash \{k^\star\}$ do
        if $(x_1 \oplus k \in LeftFreezer)$ then abort
        $LeftFreezer \leftarrow LeftFreezer \cup \{x_1 \oplus k\}$
    end forall

private procedure FreezeRightValues$(y_5, k^\star)$
    ... // (symmetric to FreezeLeftValues)

private procedure ForcedP3$(i, z)$
    if $(i = 3^-)$ then
        $x_3 \leftarrow z$
        $candidate \leftarrow \emptyset$
        forall $k \in \mathcal{Z}$ do
            if $(x_3 \oplus k \notin range(P_2))$ continue
            $y_1 \leftarrow P_2^{-1}(x_3 \oplus k) \oplus k$
            if $(y_1 \notin range(P_1))$ continue
            $x_1 \leftarrow P_1^{-1}(y_1)$
            if $(x_1 \oplus k \in LeftFreezer)$ continue
            if $(candidate \neq \emptyset)$ then abort
            $candidate \leftarrow (k, x_1 \oplus k)$
        end forall // $(k)$
        if $(candidate = \emptyset)$ return $\perp$
        $(k, x) \leftarrow candidate$
        $y_5 \leftarrow E(f^{-1}(k), x) \oplus k$
        TallyEQuery$(f^{-1}(k), x, \rightarrow)$
        if $(y_5 \notin range(P_5))$ return $\perp$
        $y_4 \leftarrow P_5^{-1}(y_5) \oplus k$
        return $P4^{-1}(y_4) \oplus k$
    end if
    if $(i = 3^+)$ then
        ... // (symmetric to case $(i = 3^-)$)
    end if
    return $\perp$
```

Fig. 2: The simulated world (second of four sets of procedures).

for which there exists a $k \in \mathcal{Z}$ such that each adjacent pair is k-adjacent and such that the first and last queries are also k-adjacent is called a *completed k-path* or *completed k-chain*.

Consider first the simplest attack that a distinguisher D might carry out: D chooses a random $x \in \{0,1\}^n$ and a random $K \in \{0,1\}^\kappa$ (where $\{0,1\}^\kappa$ is the key space), queries $E(K, x) \rightarrow y$ (to its left oracle), then queries $f(K) \rightarrow k$, $P1(x \oplus k) \rightarrow y_1$, $P2(y_1 \oplus k) \rightarrow y_2$, $P3(y_2 \oplus k) \rightarrow y_3$, ..., $P5(y_4 \oplus k) \rightarrow y_5$ to the simulator, and finally checks that $y_5 \oplus k = y$. The simulator, having itself answered the query $f(K)$, can already anticipate the distinguisher's attack when the query $P2(y_1 \oplus k)$ is made, since it sees that a k-adjacency is about to be formed between a 1-query and a 2-query. At this point, a standard strategy

```
Game G_1 (continued)

private procedure ExistsPath($i, z, k$)
    if ($i = 1^+$) then
        $y_1 \leftarrow z$
        if ($y_1 \notin \text{range}(P_1)$) return false
        $x_1 \leftarrow \text{P1}^{-1}(y_1)$
        $(\ell, x) \leftarrow \text{ProbeForward}(2, 5, y_1 \oplus k, k)$
        if ($\ell \neq 5 \vee x \notin \text{domain}(P_5)$) return false
        if ($\text{E}(f^{-1}(k), x_1 \oplus k) \neq P_5(x) \oplus k$) then abort
        TallyEQuery($f^{-1}(k), x_1 \oplus k, \rightarrow$)
        return true
    end if
    if ($i = 3^-$) then
        $x_3 \leftarrow z$
        $(\ell_1, y) \leftarrow \text{ProbeBackward}(2, 1, x_3 \oplus k, k)$
        $(\ell_2, x) \leftarrow \text{ProbeForward}(3, 5, x_3, k)$
        if ($\ell_1 \neq 1 \vee y \notin \text{range}(P_1)$) return false
        if ($\ell_2 \neq 5 \vee x \notin \text{domain}(P_5)$) return false
        if ($\text{E}(f^{-1}(k), P_1^{-1}(y) \oplus k) \neq P_5(x) \oplus k$) then abort
        TallyEQuery($f^{-1}(k), P_1^{-1}(y) \oplus k, \rightarrow$)
        return true
    end if
    if ($i = 3^+$) then
        ... // (symmetric to case ($i = 3^-$))
    end if
    if ($i = 5^-$) then
        ... // (symmetric to case ($i = 1^+$))
    end if

private procedure ProbeForward($i, j, x_i, k$)
    // ($i, j \in \{1, 2, 3, 4, 5\}$, $i < j$)
    while $i < j$ do
        if ($P_i(x_i) = \perp$) break
        $x_i \leftarrow P_i(x_i) \oplus k$
        $i \leftarrow i + 1$
    end
    return ($i, x_i$)

private procedure ProbeBackward($i, j, y_i, k$)
    // ($i, j \in \{1, 2, 3, 4, 5\}$, $i > j$)
    while $i > j$ do
        if ($P_i^{-1}(y_i) = \perp$) break
        $y_i \leftarrow P_i^{-1}(y_i) \oplus k$
        $i \leftarrow i - 1$
    end
    return ($i, y_i$)
```

```
Game G_1 (continued)

private procedure EmptyQueue()
    do
        while $\neg$LeftQueue.empty()
            $(i, z) \leftarrow$ LeftQueue.pop()
            if ($i = 1^+$) then ProcessNew1Edge($z$)
            if ($i = 3^-$) then ProcessNew3$^-$Edge($z$)
        end while
        while $\neg$RightQueue.empty()
            $(i, z) \leftarrow$ RightQueue.pop()
            if ($i = 3^+$) then ProcessNew3$^+$Edge($z$)
            if ($i = 5^-$) then ProcessNew5Edge($z$)
        end while
    while ($\neg$LeftQueue.empty())

private procedure ProcessNew1Edge($y_1$)
    forall $k \in \mathcal{Z}$
        if (ExistsPath($1^+, y_1, k$)) then continue
        if ($y_1 \oplus k \notin \text{domain}(P_2)$) then continue
        CompletePath1$^+$($y_1, k$)
    end forall

private procedure ProcessNew3$^-$Edge($x_3$)
    forall $k \in \mathcal{Z}$
        if (ExistsPath($3^-, x_3, k$)) then continue
        if ($x_3 \oplus k \notin \text{range}(P_2)$) then continue
        CompletePath3$^-$($x_3, k$)
    end forall

private procedure ProcessNew3$^+$Edge($y_3$)
    ... // (symmetric to ProcessNew3$^-$Edge)

private procedure ProcessNew5Edge($x_5$)
    ... // (symmetric to ProcessNew1Edge)

private procedure Cleanup()
    EmptyQueue()
    LeftFreezer $\leftarrow \emptyset$
    RightFreezer $\leftarrow \emptyset$

private procedure AddQuery($i, x, y, dir$)
    Queries $\leftarrow$ Queries $\cup \{(i, x, y, dir, {+}{+}qnum)\}$
    if ($qnum > 6q^2 + q$) then abort
```

Fig. 3: The simulated world (third of four sets of procedures).

would be for the simulator to pre-emptively[11] complete a k-chain by answering (say) the queries $\text{P3}(y_2 \oplus k)$ and $\text{P4}(y_3 \oplus k)$ randomly itself, and setting the value of $\text{P5}(y_4 \oplus k)$ to $\text{E}(f^{-1}(k), x) \oplus k$ by querying E.

The distinguisher might vary this attack by building a chain "from the right" (by choosing a random y and querying $\text{P5}^{-1}(y \oplus k) \rightarrow x_5$, $\text{P4}^{-1}(x_5 \oplus k) \rightarrow x_4$, etc) or by building a chain "from the inside" (e.g., by choosing a random x_3 and querying $\text{P3}(x_3) \rightarrow y_3$, $\text{P2}^{-1}(x_3 \oplus k)$, $\text{P4}(y_3 \oplus k) \rightarrow y_4$, ...) or even by building a chain "from the left and right" simultaneously (the two sides meeting

[11] Pre-emption is generally desirable in order for the simulator to avoid becoming "trapped" in an over-constrained situation.

```
Game G_1 (continued)

private procedure CompletePath1^+$(y_1, k)$
    $x_1 \leftarrow P_1^{-1}(y_1)$
    $x_3 \leftarrow P_2(y_1 \oplus k) \oplus k$
    $x_4 \leftarrow$ PrivateP3$(x_3) \oplus k$
    $x_5 \leftarrow$ P4$(x_4) \oplus k$
    FinishPath1^+3^-$(x_1, x_5, k)$

private procedure CompletePath3^-$(x_3, k)$
    $x_2 \leftarrow P_2^{-1}(x_3 \oplus k)$
    $x_1 \leftarrow$ PrivateP1$^{-1}(x_2 \oplus k)$
    $x_4 \leftarrow P_3(x_3) \oplus k$
    $x_5 \leftarrow$ P4$(x_4) \oplus k$
    FinishPath1^+3^-$(x_1, x_5, k)$

private procedure FinishPath1^+3^-$(x_1, x_5, k)$
    if $(x_1 \oplus k \in \mathit{LeftFreezer})$ then
        $\mathit{fresh} \leftarrow$ true
        $\mathit{LeftFreezer} \leftarrow \mathit{LeftFreezer} \backslash \{x_1 \oplus k\}$
    else
        $\mathit{fresh} \leftarrow$ false
    end if
    $y_5 \leftarrow k \oplus \mathrm{E}(f^{-1}(k), x_1 \oplus k)$
    TallyEQuery$(f^{-1}(k), x_1 \oplus k, \rightarrow)$
    if $(x_5 \in \mathrm{domain}(P_5))$ then abort
    if $(y_5 \in \mathrm{range}(P_5))$ then abort
    $P_5(x_5) \leftarrow y_5$
    $P_5^{-1}(y_5) \leftarrow x_5$
    AddQuery$(5, x_5, y_5, \perp)$
    $\mathit{RightQueue} \leftarrow \mathit{RightQueue} \cup (5^-, x_5)$
    if $(\mathit{fresh})$ then
        FreezeRightValues$(y_5, k)$
    end if
```

```
Game G_1 (continued)

private procedure CompletePath3^+$(y_3, k)$
    ... // (symmetric to CompletePath3^-)

private procedure CompletePath5^-$(x_5, k)$
    ... // (symmetric to CompletePath1^+)

private procedure FinishPath5^-3^+$(y_5, y_1, k)$
    ... // (symmetric to FinishPath1^+3^-)

private procedure TallyEQuery$(K, z, dir)$
    if $(dir = \rightarrow)$ then
        if $(\mathit{TallyETable}[K](z) = \perp)$ then $++\mathit{Eqnum}$
        $\mathit{TallyETable}[K](z) \leftarrow t \leftarrow \mathrm{E}(K, z)$
        $\mathit{TallyETable}[K]^{-1}(t) \leftarrow z$
    end if
    if $(dir = \leftarrow)$ then
        ... // (symmetric to case $dir = \rightarrow$)
    end if
    if $(\mathit{Eqnum} > 2q^2)$ then abort
```

Fig. 4: The simulated world (fourth of four sets of procedures).

up somewhere in the middle). Given all these combinations, a natural strategy is to have the simulator complete chains whenever it detects *any* k-adjacency. We call this type of simulator *naïve*. The difficulty with the naïve simulator is that, as the path-completion strategy is applied recursively to queries created by the simulator itself, some uncontrollable chain reaction might occur that causes the simulator to create a superpolynomial number of queries, and, thus, lead to an unacceptable simulator running time and to an unacceptably watered-down security bound. Even if such a chain reaction cannot occur, the burden of showing so is on the prover's shoulders, which is not necessarily an easy task. We refer to the general problem of showing that runaway chain reactions do not occur as the problem of *simulator termination*.[12]

To overcome the naïve simulator's problematic termination, we modify the naïve simulator to be more restrained and to complete fewer chains. For this we

[12] Naturally, since the simulator can only create finitely many different i-queries, the simulator is, in general, guaranteed to terminate. Thus "simulator termination" refers, more precisely, to the problem of showing that the simulator only creates polynomially many queries per adversarial query. We prefer the term "termination" to "efficiency" because it seems to more picturesquely capture the threat of an out-of-control chain reaction.

use the "tripwire" concept. Informally, a tripwire is an ordered pair of the form $(i, i+1)$ or $(i+1, i)$ or $(1,5)$ or $(5,1)$ (for a 5-round cipher). "Installing a tripwire (i,j)" means the simulator will complete paths for k-adjacencies detected between positions i and j *and for which the j-query is made after the i-query.* (Thus, tripwires are "directed".) As long as no tripwires are triggered, the simulator does nothing; when a tripwire is triggered, the simulator completes the relevant chain(s), and recurses to complete chains for other potentially triggered tripwires, etc. The "naïve" simulator then corresponds to a tripwire simulator with all possible tripwires installed. The tripwire paradigm is essentially due to Coron et al. [15] even while the terminology is ours.

Restricting ourselves to the (fairly broad) class of tripwire simulators, conflicting goals emerge: to install enough tripwires so that the simulator cannot be attacked, while installing few enough tripwires (or in clever enough positions) that a termination argument can be made. Before presenting our own 5-round solution to this dilemma, we briefly justify our choice of five rounds.

Firstly, *no* tripwire simulator with 3 rounds is secure, since it turns out that the naïve 3-round simulator (i.e., with all possible tripwires) can already be attacked. Hence, regardless of termination issues, any 3-round tripwire simulator is insecure. Secondly, we focused on 4-round simulators with four tripwires, as proving termination for five or more tripwires seemed a daunting task. A particularly appealing simulator, here, is the 4-tripwire simulator

$$(1,4), (4,1), (2,3), (3,2)$$

whose termination can easily be proved by modifying Holenstein et al. termination argument [37], itself adapted from an earlier termination argument of Seurin [54]. Unfortunately it turns out this simulator can be attacked, making it useless. This attack as well as the above-mentioned attack on the 3-round naïve simulator can be found in the full version of this paper [1], where some other attacks on tripwire simulators are also sketched.

Ultimately, the only 4-round, 4-tripwire simulator for which we didn't find an attack is the simulator with the (asymmetric) tripwire configuration

$$(1,2), (3,2), (3,4), (1,4)$$

(and its symmetric counterpart). However, since we could not foresee a manageable termination argument for this simulator, we ultimately reverted to five rounds. Our 5-round simulator has tripwires

$$(2,1), (2,3), (4,3), (4,5)$$

(and no tripwires of the form $(1,5)$ or $(5,1)$), as sketched in Figure 5. This simulator has the advantage of having a clean (though combinatorially demanding) termination argument, and, as previously discussed, of having excellent efficiency and also better security than the state-of-the-art in "indifferentiable blockcipher" constructions.

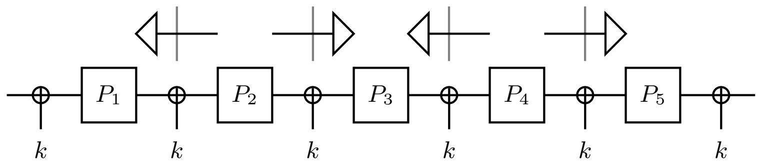

Fig. 5: Tripwire positions for our 5-round simulator. A directed arrow from column P_i to column P_j indicates a tripwire (i, j). The tripwires are $(2, 1)$, $(2, 3)$, $(4, 3)$ and $(4, 5)$.

Some more high-level description of the 5-round simulator. We have already mentioned that our 5-round simulator has tripwires

$$(2, 1), (2, 3), (4, 3), (4, 5).$$

To complete the simulator's description it (mainly) remains to describe how the simulator completes chains, once a tripwire is triggered, since there is some degree of freedom as to which i-query is "adapted" to fit E, etc. Quickly and informally, when a newly created 1-query or 3-query triggers respectively the $(2, 1)$ or $(2, 3)$ tripwire, the relevant path(s) that are completed have their 5-query adapted to fit E. (We note the same query may trigger the completion of several new paths.) Symmetrically, when a newly created 3-query or 5-query triggers a $(4, 3)$ or $(4, 5)$ tripwire, the completed paths have their 1-query adapted to fit E. We note that new 2-queries and 4-queries can never trigger a tripwire, due to the tripwire structure. Moreover, 2- and 4-queries are never adapted, and always have at least one "random endpoint". The latter property turns out to be crucial for various arguments in the proof. It also makes the implementation of the procedures P2(), P2^{-1}(), P4() and P4^{-1}() particularly simple, since these do nothing else than lazy sample and return.

The above "quick and informal" summary of the path-completion process is over-simplified because 3-queries can also, in specific situations, be adapted to complete a path. To gain some preliminary intuition about 3-queries, consider a distinguisher D that chooses values x and K and then makes the queries $\mathrm{f}(K) \to k$, $\mathrm{P1}(x \oplus k) \to y_1$, $\mathrm{P2}(y_1 \oplus k) \to y_2$, $\mathrm{E}(K, x) \to y$, $\mathrm{P5}^{-1}(y \oplus k) \to x_5$ and $\mathrm{P4}^{-1}(x_5 \oplus k) \to x_4$. So far, no tripwires have been triggered, but the adversary already knows (e.g., in the real world) that $\mathrm{P3}(y_2 \oplus k) = x_4 \oplus k$, even while the simulator has not yet defined anything internally about P3. Typically, such a situation where the adversary "already knows" something the simulator doesn't are dangerous for the simulator and can lead to attacks; in this case, it turns out the distinguisher cannot use this private knowledge to fool the simulator. It does mean, however, that the simulator needs to be on the lookout for such "pre-defined" 3-queries whenever it answers queries to P3(), P3^{-1}() or, more generally, whenever it makes a new 3-query internally.

In fact the code used by the simulator to answer 3-queries is altogether rather cautious and sophisticated, even slightly more so than the previous discussion might suggest. To gain further insight into the simulator's handling of 3-queries, consider a distinguisher D' that similarly chooses values x and K and then makes the queries $\mathrm{f}(K) \rightarrow k$, $\mathrm{P1}(x \oplus k) \rightarrow y_1$, $\mathrm{P2}(y_1 \oplus k) \rightarrow y_2$, $\mathrm{E}(K, x) \rightarrow y$ and $\mathrm{P5}^{-1}(y \oplus k) \rightarrow x_5$. (So D' makes all the same queries as the distinguisher D above except for the final query $\mathrm{P4}^{-1}(x_5 \oplus k)$, which is *not* made by D'.) At this point, the value $\mathrm{P3}(y_2 \oplus k)$ is not yet pre-defined by E and by the previous queries, since the query $\mathrm{P4}^{-1}(x_5 \oplus k)$ hasn't been made; if D' queries $\mathrm{P3}(y_2 \oplus k) \rightarrow y_3$, the simulator might conceivably sample y_3 randomly, and later use the freedom afforded by the missing P4 query to adapt the chain. If the simulator did this, however, the simulator would create a "non-random" 4-query (i.e., a 4-query that doesn't have at least one non-adapted, "random endpoint"), which would wreak havoc within the proof. Instead, when faced with the query $\mathrm{P3}(y_2 \oplus k)$, the simulator detects the situation above and starts by making the "missing" query $\mathrm{P4}^{-1}(x_5 \oplus k) \rightarrow x_4$ internally, thus giving the P4-query its required "random endpoint" (at x_4), and finally adapts $\mathrm{P3}(y_2 \oplus k)$ to $x_4 \oplus k$. It so turns out that, with high probability, the simulator is never caught trying to adapt P3() to two different values in this way.

The sets *LeftQueue* and *RightQueue* mentioned in the pseudocode are two queues of queries maintained by the simulator for the purpose of tripwire detection. When a new i-query is created, $i \in \{1, 3\}$, that the simulator believes might set off the $(2, 1)$ or $(2, 3)$ tripwire, the simulator puts this i-query into *LeftQueue*, to be checked later; similarly for $i \in \{3, 5\}$, the simulator puts a newly created i-query into *RightQueue* if it believes this new query might set off a $(4, 3)$ or $(4, 5)$ tripwire. (The same 3-query might end up in both *LeftQueue* and *RightQueue*.) As evidenced by the procedure EmptyQueue() in Fig. 3, *LeftQueue* and *RightQueue* are emptied sequentially and separately, which we choose to do mostly because it offers conceptual advantages within the proof. In the full version of this paper [1] we further discuss how the simulator might come to believe that a newly created i-query will likely *not* set off a tripwire (and thus not put this i-query into the relevant queue(s)), as well give a more detailed discussion of the pseudocode of the simulator.

Due to the space constraints, we similarly leave to the full version [1] a full formal indifferentiability proof of our construction, as well as all other results mentioned in the introduction (e.g., our attacks and the proof of Theorem 2).

References

1. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the Indifferentiability of Key-Alternating Ciphers. Cryptology ePrint Archive, Report 2013/061 (2013), full version of this paper
2. Aumasson, J., Henzen, L., Meier, W., Phan, R.: SHA-3 proposal BLAKE (2010), submission to NIST's SHA-3 competition
3. Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In: ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer-Verlag, Berlin (2006)

4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.: The KECCAK sponge function family (2011), submission to NIST's SHA-3 competition
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer-Verlag, Berlin (2008)
6. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds. In: Gilbert, H. (ed.) EUROCRYPT. LNCS, vol. 6110, pp. 299–319. Springer (2010)
7. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT. LNCS, vol. 5912, pp. 1–18. Springer (2009)
8. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO. LNCS, vol. 5677, pp. 231–249. Springer (2009)
9. Black, J.: The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In: FSE 2006. LNCS, vol. 4047, pp. 328–340. Springer-Verlag, Berlin (2006)
10. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer-Verlag, Berlin (2002)
11. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, vol. 7073, pp. 344–371. Springer (2011)
12. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.X., Steinberger, J., Tischhauser, E.: Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations. In: EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer-Verlag, Berlin (2012)
13. Brachtl, B., Coppersmith, D., Hyden, M., Matyas, S., Meyer, C., Oseas, J., Pilpel, S., Schilling, M.: Data authentication using modification detection codes based on a public one-way encryption function (March 1990), U.S.Patent No 4.908.861
14. Chang, D., Lee, S., Nandi, M., Yung, M.: Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In: ASIACRYPT 2006. LNCS, vol. 4284, pp. 283–298. Springer-Verlag, Berlin (2006)
15. Coron, J.S., Patarin, J., Seurin, Y.: The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In: CRYPTO 2008. LNCS, vol. 5157, pp. 1–20. Springer-Verlag, Berlin (2008)
16. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer-Verlag, Berlin (2005)
17. Coron, J.S., Dodis, Y., Mandal, A., Seurin, Y.: A Domain Extender for the Ideal Cipher. In: TCC 2010. LNCS, vol. 5978, pp. 273–289. Springer-Verlag, Berlin (2010)
18. Daemen, J., Govaerts, R., Vandewalle, J.: Correlation Matrices. In: Preneel, B. (ed.) FSE. LNCS, vol. 1008, pp. 275–285. Springer (1994)
19. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) IMA Int. Conf. LNCS, vol. 2260, pp. 222–238. Springer (2001)
20. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
21. Damgård, I.: A Design Principle for Hash Functions. In: CRYPTO '89. LNCS, vol. 435, pp. 416–427. Springer-Verlag, Berlin (1990)

22. Demay, G., Gaži, P., Hirt, M., , Maurer, U.: Resource-Restricted Indifferentiability. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 664–683. Springer (2013)
23. Desai, A.: The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search. In: CRYPTO 2000. LNCS, vol. 1880, pp. 359–375. Springer-Verlag, Berlin (2000)
24. Dodis, Y., Pietrzak, K., Puniya, P.: A New Mode of Operation for Block Ciphers and Length-Preserving MACs. In: EUROCRYPT. pp. 198–219 (2008)
25. Dodis, Y., Puniya, P.: On the Relation Between the Ideal Cipher and the Random Oracle Models. In: TCC 2006. LNCS, vol. 3876, pp. 184–206. Springer-Verlag, Berlin (2006)
26. Dodis, Y., Reyzin, L., Rivest, R., Shen, E.: Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6. In: Fast Software Encryption 2009. LNCS, vol. 5665, pp. 104–121. Springer, Heidelberg (2009)
27. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for Practical Applications. In: EUROCRYPT 2009. LNCS, vol. 5479, pp. 371–388. Springer-Verlag, Berlin (2009)
28. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: EUROCRYPT 2012. LNCS, Springer-Verlag, Berlin (2012)
29. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. In: ASIACRYPT '91. LNCS, vol. 739, pp. 201–224. Springer-Verlag, Berlin (1991)
30. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family (2010), submission to NIST's SHA-3 competition
31. Gauravaram, P., Knudsen, L., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.: Grøstl – a SHA-3 candidate (2011), submission to NIST's SHA-3 competition
32. Goldreich, O., Goldwasser, S., Micali, S.: How to Construct Random Functions. In: 25th Annual Symposium on Foundations of Computer Science, FOCS. pp. 464–479. IEEE Computer Society, West Palm Beach, Florida, USA (1984)
33. Granboulan, L.: Short Signatures in the Random Oracle Model. In: ASIACRYPT 2002. LNCS, vol. 2501, pp. 364–378. Springer-Verlag, Berlin (2002)
34. Handschuh, H., Naccache, D.: SHACAL. Submission to the NESSIE project (2000)
35. Handschuh, H., Naccache, D.: SHACAL : A Family of Block Ciphers. Submission to the NESSIE project (2002)
36. Hirose, S.: Some Plausible Constructions of Double-Block-Length Hash Functions. In: FSE 2006. LNCS, vol. 4047, pp. 210–225. Springer-Verlag, Berlin (2006)
37. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In: ACM Symposium on Theory of Computing, STOC. pp. 89–98. ACM, San Jose, CA, USA (2011)
38. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random Generation from one-way functions. In: ACM Symposium on Theory of Computing, STOC. pp. 12–24. ACM, Seattle, Washington, USA (1989)
39. Jonsson, J.: An OAEP Variant With a Tight Security Proof. Cryptology ePrint Archive, Report 2002/034 (2002)
40. Kilian, J., Rogaway, P.: How to Protect DES against Exhaustive Key Search (An Analysis of DESX). Journal of Cryptology 14(1), 17–35 (2001)

41. Knudsen, L.: Block Ciphers - The Basics (May 2011), eCRYPT II Summer School on Design and Security of Cryptographic Algorithms and Devices, Invited talk
42. Lai, X., Massey, J.: Hash Function Based on Block Ciphers. In: EUROCRYPT '92. LNCS, vol. 658, pp. 55–70. Springer-Verlag, Berlin (1992)
43. Lampe, R., Seurin, Y.: How to Construct an Ideal Cipher from a Small Set of Public Permutations. Cryptology ePrint Archive, Report 2013/255 (2013)
44. Lee, J., Hong, D.: Collision Resistance of the JH Hash Function. IEEE Transactions on Information Theory 58(3), 1992–1995 (2012)
45. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal of Computing 17(2), 373–386 (1988)
46. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer-Verlag, Berlin (2004)
47. Merkle, R.: One Way Hash Functions and DES. In: CRYPTO '89. LNCS, vol. 435, pp. 428–446. Springer-Verlag, Berlin (1990)
48. Miles, E., Viola, E.: Substitution-Permutation Networks, Pseudorandom Functions, and Natural Proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 7417, pp. 68–85. Springer (2012)
49. Phan, D.H., Pointcheval, D.: Chosen-Ciphertext Security without Redundancy. In: ASIACRYPT 2003. LNCS, vol. 2894, pp. 1–18. Springer-Verlag, Berlin (2003)
50. Preneel, B., Govaerts, R., Vandewalle, J.: Hash Functions Based on Block Ciphers: A Synthetic Approach. In: CRYPTO '93. LNCS, vol. 773, pp. 368–378. Springer-Verlag, Berlin (1993)
51. Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with Composition: Limitations of the Indifferentiability Framework. In: EUROCRYPT 2011. LNCS, vol. 6632, pp. 487–506. Springer-Verlag, Berlin (2011)
52. Rogaway, P., Steinberger, J.P.: Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In: Wagner, D. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 5157, pp. 433–450. Springer (2008)
53. Rogaway, P., Steinberger, J.P.: Security/Efficiency Tradeoffs for Permutation-Based Hashing. In: EUROCRYPT. pp. 220–236 (2008)
54. Seurin, Y.: Primitives et protocoles cryptographiques à sécurité prouvée. Ph.D. thesis, Université de Versailles Saint-Quentin-en-Yvelines, France (2009)
55. Winternitz, R.S.: A Secure One-Way Hash Function Built from DES. In: IEEE Symposium on Security and Privacy. pp. 88–90. IEEE Computer Society (1984)
56. Wu, H.: The Hash Function JH (2011), submission to NIST's SHA-3 competition