Advanced cryptology (GBX9SY06)

✧

Block ciphers

Pierre Karpman

December 17, 2020

Part I
Some models and provably-secure constructions

1 First definitions

Recall that a block cipher is a family of invertible mappings indexed by a key: $\mathcal{E} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, where both $\mathcal{E}$ and its inverse $\mathcal{E}^{-1}$ are "efficiently computable"*. In practice, we mostly care about ciphers for which $\mathcal{M} = \mathcal{C}$, meaning that $\mathcal{E}$ defines a family of *permutations* over $\mathcal{M}$. In the vast majority of cases, we also have $\mathcal{M} = \{0,1\}^n$ for some integer n, often equal to 64 or 128 (though smaller, larger, and intermediate values are possible). In some corner cases, however, it may be that $\mathcal{M}$ has a different structure, for instance all integers smaller than a certain N, or the set of valid credit-card numbers. This is the concern of *format-preserving encryption*, which will not be addressed in this course. Note that as long as cryptography is implemented on digital circuits, there is no similar incentive to take $\mathcal{K}$ of another form than $\{0,1\}^\kappa$. To summarize, we will use the following definition.

Definition 1 (Block ciphers)**.** A block cipher is a family of mappings $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ such that for all $k \in \{0,1\}^\kappa$, $\mathcal{E}(k,\cdot)$ is a permutation. The quantities n and κ are positive integers, called the block size (or length) and key size, respectively.

Block ciphers are important *primitives* in symmetric-key cryptography. When used with a suitable *mode of operation*, they allow to ensure the confidentiality and authenticity of data. We will not address the issue of modes in this course, but it is important to remember that they are essential; without a proper mode, a block cipher is mostly useless.

There are many constructions that satisfy Definition 1, but not all of them are useful in a cryptographic context. For instance, $\mathcal{E}$ such that for all k the mapping $\mathcal{E}(k,\cdot)$ is the identity will not be very useful *e.g.* when later trying to provide confidentiality. We thus need to express one or several security properties that should hold for a "good" block cipher. Intuitively and informally, we typically require that:

1. Given $x_0, \ldots, x_m, \mathcal{E}(k,x_0), \ldots, \mathcal{E}(k,x_m)$, it should be "hard" to find k, even if the x_is span the entire message space $\{0,1\}^n$. An attack violating this property is called a *key-recovery attack*†.

*As concrete block ciphers (usually) fix all their parameters, there is not much sense to argue about this efficiency in terms of asymptotic complexity, and we will not attempt to give a precise definition of what "efficient" means in this context.

†More generally, we may require that it should be hard to recover any k' s.t. $\mathcal{E}(k,\cdot) = \mathcal{E}(k',\cdot)$ on most inputs.

2. Given pairwise-distinct $x_0, \ldots, x_m, \mathcal{E}(k, x_0), \ldots, \mathcal{E}(k, x_m)$ and $y_0 \neq x_i$ (resp. $\mathcal{E}(k, y_0)$), it should be "hard" to learn information about $\mathcal{E}(k, y_0)$ (resp. y_0) (except that it is distinct from the $\mathcal{E}(k, x_i)$s (resp. x_is)). An attack violating this property is usually called a *distinguishing* attack.

The first of these informal properties is maybe the most obvious one to think of, but it is not sufficient in itself to capture all the desired properties of a block cipher. For instance, one may consider a cipher $\mathcal{E}$ for which key-recovery attacks are hard, but such that for all k, $\mathcal{E}(k, m) = m$ whenever m starts with a zero bit. It is quite clear that such an $\mathcal{E}$ will not be useful to provide confidentiality.

In order to be more precise about our requirements for "good" block ciphers, it is useful to first consider *ideal* block ciphers. This allows to set a standard for any security property we might be interested in, as these should be "maximally hard" for an ideal cipher. A concrete (non-ideal) cipher $\mathcal{E}$ is then considered to be secure if it is hard to decide if one is interacting with $\mathcal{E}$ or with an optimally-secure ideal cipher.

The definition of an ideal cipher is quite simple: it is simply a block cipher $\mathfrak{E}$ such that for all k, $\mathfrak{E}(k, \cdot)$ is a permutation uniformly drawn at random among all permutations over the same domain. Thus, all keys of $\mathfrak{E}$ define completely independent mappings that are all equally likely to be selected. Fixing the notation, we have the following definition.

Definition 2 (Ideal block cipher)**.** Let Π_n denote the set of all $2^n!$ permutations over $\{0,1\}^n$. For any finite set $\mathcal{S}$, we write $x \xleftarrow{\$} \mathcal{S}$ the uniform sampling of x over all elements of $\mathcal{S}$. Then, an ideal block cipher is a cipher $\mathfrak{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ such that for all $k \in \{0,1\}^\kappa$, $\mathfrak{E}(k, \cdot) \xleftarrow{\$} \Pi_n$.

Note that not all ideal ciphers may be "useful". For instance, when randomly sampling an ideal cipher, there is a $1/(2^n! \times 2^\kappa)$ chance that $\mathfrak{E}$ be defined as the not-so-useful identity-everywhere cipher. However, we claim that the odds of picking an "insecure" cipher are small enough for this eventuality to be ignored altogether. Also more importantly, ideal ciphers are by definition immune to distinguishing attacks as defined above; indeed given pairwise-distinct $x_0, \ldots, x_m$, $\mathfrak{E}(k, x_0), \ldots, \mathfrak{E}(k, x_m), y_0 \neq x_i$ then $\Pr[\mathfrak{E}(k, y_0) = a]$ is equal to 0 if $a = \mathfrak{E}(k, x_i)$ for some i and $1/(2^n - (m+1))$ otherwise, where the probability is computed over the sampling of $\mathfrak{E}(k, \cdot)$. In other words, the only thing that an adversary knows about the evaluation of $\mathfrak{E}(k, \cdot)$ on inputs that it didn't query comes from the fact that $\mathfrak{E}(k, \cdot)$ is a permutation.

As already said above, one may use the notion of an ideal cipher to define the security of a concrete one by how easy it is for an adversary to decide if he is interacting with an ideal cipher or not. One way to formulate this is with the notion of *pseudorandom permutation* (PRP). Informally, this notion is expressed by having an adversary interacting with an *oracle* $\mathcal{O}$ which is randomly chosen to be either of $\mathcal{E}(K, \cdot)$ or $\mathfrak{E}(K, \cdot)$ for a random key K (as $\mathfrak{E}$ is ideal, the latter case is simplified by dropping the key and simply asking that $\mathfrak{E}$ be a uniformly chosen permutation). Then one considers what is the best *advantage* over a random choice that an adversary has of deciding how $\mathcal{O}$ was instantiated.

It seems clear that the advantage should in fact be a function of the *data complexity* (the number of queries to $\mathcal{O}$) and of the *time complexity* (where the unit is generally the time it takes to evaluate $\mathcal{E}$) of the adversary. For instance, this accounts for the fact that any cipher (even an ideal one) of key size κ can be broken by exhaustive key search in time 2^κ, and yet could still be secure if the adversary only has more limited resources. We then define the PRP security of $\mathcal{E}$ through the following PRP advantage function (see *e.g.* [BKR00, BR]).

Definition 3 (PRP advantage)**.** The *PRP advantage* of $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ is

defined as:

$$\mathrm{Adv}^{\mathrm{PRP}}_{\mathcal{E}}(q,t) := \max_{A_{q,t}} \Big| \Pr\left[A^{\mathcal{O}}_{q,t}(\cdot) = 1 : \mathcal{O} \xleftarrow{\$} \Pi_n\right] - \Pr\left[A^{\mathcal{O}}_{q,t}(\cdot) = 1 : \mathcal{O} = \mathcal{E}(K,\cdot), K \xleftarrow{\$} \{0,1\}^\kappa\right] \Big|,$$

where $A^{\mathcal{O}}_{q,t}$ denotes an adversary A who makes at most q queries to its oracle $\mathcal{O}$, runs in time at most t, and returns a unique bit.

A similar notion of *strong* PRP (SPRP) can be defined, where the adversary is additionally granted access to the inverse of $\mathcal{O}$.

Now we can use Definition 3 to formulate some requirements about the security of a "good" block cipher $\mathcal{E}$. As, we would ideally want $\mathcal{E}$ to behave as close as possible to an ideal cipher $\mathfrak{E}$ (defined over the same domains), *i.e.* we wish that $\mathrm{Adv}^{\mathrm{PRP}}_{\mathcal{E}}(q,t) \approx \mathrm{Adv}^{\mathrm{PRP}}_{\mathfrak{E}}(q,t)$ for all q, t (for instance, we would like to have $\mathrm{Adv}^{\mathrm{PRP}}_{\mathcal{E}}(1,t) \approx t/2^\kappa$). By abuse of terminology, we usually say of such an hypothetical cipher that "it is a PRP".

Note that breaking PRP security does not even require to recover the key k when $\mathcal{O} = \mathcal{E}(k,\cdot)$, which was one of the informal goals stated at the beginning of this section. As recovering the key *does* however allow to break PRP security, focusing only on the latter does not weaken the requirements on $\mathcal{E}$, and allows to capture the second informal goal of resisting distinguishing attacks.

The notion of PRP is useful to express desirable properties for concrete block ciphers. However, it is in itself useless to actually evaluate their security. That it is to say, the definition provides very little insight into how to compute *e.g.* $\mathrm{Adv}^{\mathrm{PRP}}_{\text{AES-128}}$. It is in fact the goal of cryptanalysis in general, by finding explicit attacks, to lower-bound the advantage function for specific ciphers and complexities.

Finally, the PRP definition is not without limitations; while it is suitable to characterise the security of a block cipher used in a context where there is a single unknown key sampled uniformly at random, it is inadequate in settings where the adversary may for instance choose the key as in PGV constructions for compression functions, or when unknown *related-keys* are involved.

2 An ideal construction

We now present a generic construction of block ciphers due to Even and Mansour [EM91, EM97], which is very simple and yet of considerable interest. Let $\mathcal{P} : \{0,1\}^n \to \{0,1\}^n$ be a "public" (*i.e.* not secret) permutation, then one simply defines the Even-Mansour cipher built from $\mathcal{P}$ as $\mathcal{E}(k_1||k_2, m) := \mathcal{P}(m \oplus k_1) \oplus k_2$, for all keys $(k_1||k_2)$ of $2n$ bits and all messages m. In fact, we can also go for an even simpler definition by taking $k_1 = k_2$, thence obtaining $\mathcal{E}(k,m) = \mathcal{P}(m \oplus k) \oplus k$. What is remarkable is that both of these constructions can be proven to be good PRPs, assuming that $\mathcal{P}$ is itself a "good" permutation. Intuitively, this implies for instance that $\mathcal{P}$ should not admit efficient distinguishers, in line with our second intuitive requirement for a block cipher. However, this is not an easy notion to formalize; the approach taken by Even and Mansour to prove their construction is then to only allow an adversary to make black-box oracle queries to a uniformly sampled permutation $\mathcal{P}$. A possible interpretation is then to consider this number of queries itself to denote the "time complexity" of the attack.

2.1 Proof sketch

The security theorem proved by Even and Mansour does not use the notion of PRP, which was not formalised at the time. Instead, it proves that the success probability of an adversary for the *existential forgery problem* (EFP) is upper-bounded by $O(DT/2^n)$,

where D is the number of queries to $\mathcal{E}$ (with an unknown fixed random key) and T the number of black-box queries to $\mathcal{P}$ or its inverse. In other words, the scheme achieves security up to the birthday bound against black box adversaries, and any attack violating the bound for a concrete instantiation must exploit structural properties of $\mathcal{P}$. The proof was also adapted to a PRP setting for a slightly more general construction by Kilian and Rogaway [KR01]. We give a sketch of the proof in this setting.

The idea is the following: we define two *games* corresponding respectively to the real or ideal scenario of the PRP definition. In both games the adversary makes queries to a $\mathcal{P}^{\pm}$ and an $\mathcal{E}^{\ddagger}$ oracle whose answers are provided by a simulator; the adversary wins if it is able to tell which game it is playing. To upper-bound the corresponding advantage in function of the number of queries, we define the simulators of each game to behave identically up to the occurrence of so-called *bad* events. An upper-bound on the probability of such occurrences then allows to conclude.

The simulator in the ideal game is quite simple: it answers any query to $\mathcal{P}^{\pm}$ uniformly from the set of still available (pre)images, and the same for queries to $\mathcal{E}$ (since it is also a uniformly sampled permutation, independent from $\mathcal{P}$). The simulator of the real game does the same, except that it must now ensure the consistency of the answers with the fact that $\mathcal{E}(k, m) = \mathcal{P}(m \oplus k) \oplus k$ (taking here the one-key variant) for some randomly sampled k. To do so it itself draws k at random and simply applies the definition, *e.g.* if one has queried $\mathcal{E}(k) = \mathcal{P}(0) \oplus k$ then a future query to $\mathcal{P}(0)$ must return $\mathcal{E}(k) \oplus k$ instead of drawing the image at random (so this would then be a bad event). In more details and following [KR01], the real simulator does as follows:

— It selects $k \xleftarrow{\$} \{0,1\}^n$ and let $\mathcal{P}$ and $\mathcal{E}$ be *a priori* undefined and ***bad*** be unset.

— On query $\mathcal{E}(m)$:

1. Draw c uniformly from the as yet undefined images for $\mathcal{E}$.
2. If $\mathcal{P}(m \oplus k)$ is defined as y then define $\mathcal{E}(m)$ as $y \oplus k$ and set ***bad***.
3. If $\mathcal{P}^{-1}(c \oplus k)$ is defined then set ***bad*** and go to 1.
4. Define $\mathcal{E}(m)$ as c.

— On query $\mathcal{P}(x)$

1. Draw y uniformly from the as yet undefined images for $\mathcal{P}$.
2. If $\mathcal{E}(x \oplus k)$ is defined as c then define $\mathcal{P}(x)$ as $c \oplus k$ and set ***bad***.
3. If $\mathcal{E}^{-1}(y \oplus k)$ is defined then set ***bad*** and go to 1.
4. Define $\mathcal{P}(x)$ as y.

— On query $\mathcal{P}^{-1}(y)$

1. Draw x uniformly from the as yet undefined preimages for $\mathcal{P}$.
2. If $\mathcal{E}^{-1}(y \oplus k)$ is defined as m then define $\mathcal{P}^{-1}(y)$ as $m \oplus k$ and set ***bad***.
3. If $\mathcal{E}(x \oplus k)$ is defined then set ***bad*** and go to 1.
4. Define $\mathcal{P}^{-1}(y)$ as x.

It is quite clear that this simulator deviates from an ideal one exactly when ***bad*** is set, since this corresponds to the situations where the (pre)image to be defined is not sampled uniformly. We make this explicit by also including the bad events in the definition of the ideal simulator as follows:

[‡]One could also allow an $\mathcal{E}^{-1}$ oracle to evaluate SPRP security. It in fact doesn't change much of the result.

— It selects $k \xleftarrow{\$} \{0,1\}^n$ and let $\mathcal{P}$ and $\mathcal{E}$ be *a priori* undefined and ***bad*** be unset.

— On query $\mathcal{E}(m)$:

1. Draw c uniformly from the as yet undefined images for $\mathcal{E}$.
2. If $\mathcal{P}(m \oplus k)$ is defined then set ***bad***.
3. If $\mathcal{P}^{-1}(c \oplus k)$ is defined then set ***bad***.
4. Define $\mathcal{E}(m)$ as c.

— On query $\mathcal{P}(x)$

1. Draw y uniformly from the as yet undefined images for $\mathcal{P}$.
2. If $\mathcal{E}(x \oplus k)$ is defined as c then set ***bad***.
3. If $\mathcal{E}^{-1}(y \oplus k)$ is defined then set ***bad***.
4. Define $\mathcal{P}(x)$ as y.

— On query $\mathcal{P}^{-1}(y)$

1. Draw x uniformly from the as yet undefined preimages for $\mathcal{P}$.
2. If $\mathcal{E}^{-1}(y \oplus k)$ is defined as m then set ***bad***.
3. If $\mathcal{E}(x \oplus k)$ is defined then set ***bad***.
4. Define $\mathcal{P}^{-1}(y)$ as x.

To make the computation of an upper-bound on ***bad*** happening easier, we then rewrite it as:

— Let $\mathcal{P}$ and $\mathcal{E}$ be *a priori* undefined and ***bad*** be unset.

— On query $\mathcal{E}(m)$:

1. Draw c uniformly from the as yet undefined images for $\mathcal{E}$.
2. Define $\mathcal{E}(m)$ as c.

— On query $\mathcal{P}(x)$

1. Draw y uniformly from the as yet undefined images for $\mathcal{P}$.
2. Define $\mathcal{P}(x)$ as y.

— On query $\mathcal{P}^{-1}(y)$

1. Draw x uniformly from the as yet undefined preimages for $\mathcal{P}$.
2. Define $\mathcal{P}^{-1}(y)$ as x.

— Once all queries have been made, select $k \xleftarrow{\$} \{0,1\}^n$ and set ***bad*** if:

- $\exists\, x$ s.t. $\mathcal{P}(x)$ and $\mathcal{E}(x \oplus k)$ are both defined, or
- $\exists\, y$ s.t. $\mathcal{P}^{-1}(y)$ and $\mathcal{E}^{-1}(y \oplus k)$ are both defined.

One may check that the last step indeed accounts for all the bad events of the previous definition. Finally, for every pair of query (x, m) (resp. (y, m)) to $\mathcal{P}$ (resp. $\mathcal{P}^{-1}$) and $\mathcal{E}$ there is only one induced bad k, *viz.* $x \oplus m$ (resp. $\mathcal{P}^{-1}(y) \oplus m$), so the total number of bad keys is $\leq DT$ and the probability to set ***bad*** over a uniform choice for k is $\leq DT/2^n$; the fact that the two games only differ on bad events makes this an upper-bound on the advantage too, and the result follows. We refer the reader to [KR01] for more details.

2.2 Generic attacks

An important and common question that arises in the context of security proofs such as the one above is the question of *tightness*. That is, we are interested in knowing if the actual security of a construction might be better than what the proof provides, or if there exists an attack with a complexity matching the provable bound. Ideally, we would like all proofs to be tight. If the status of a proof is not known, it is a natural research problem to either find an attack matching the bound of the proof or a better proof matching less efficient attacks.

In the case of the Even-Mansour construction, it was quickly found out by Daemen that the proof is indeed tight [Dae91]. In fact, it is quite straightforward to extract an "optimal" attack strategy from the proof sketch as given above; this will also be a key recovery attack so even if the proof was only concerned with distinguishers it is not harder to reach this "stronger" objective. Several variants of this strategy now generically called *slide attack* exist (see *e.g.* [DKS12]), and we sketch two of them. The first one works as follows:

1. Pick N distinct random values x_i, query $\mathcal{E}(x_i)$ and $\mathcal{P}(x_i)$, and insert their sums $y_i := \mathcal{E}(x_i) \oplus \mathcal{P}(x_i)$ along with x_i in a table.
2. For all i, j such that $y_i = y_j$, return $x_i \oplus x_j$ as a key candidate.

First, let us show that if the unknown key k is equal to $x_i \oplus x_j$ for some (i, j), the test in step 2) will indeed return k as a candidate. This is straightforward, as by definition we have:

$$y_i = \mathcal{P}(x_i \oplus k) \oplus k \oplus \mathcal{P}(x_i) = \mathcal{P}(x_j) \oplus k \oplus \mathcal{P}(x_j \oplus k) = y_j.$$

Now what is the probability (in function of N) that $k = x_i \oplus x_j$ for some i, j? This is exactly the same as searching for a collision in a random sequence where "equality" is defined as having (absolute) difference k, so the expected number of solutions is $\Theta(N^2/2^n)$ and the success probability of the attack (where we define a "success" as any outcome where k is among the returned candidates) then matches the bound of the security proof since $T = D = N$. However we can go further and show that the attack can still be (somewhat) efficiently implemented if one additionally considers its actual time[§] and memory complexity (which are not accounted for in the theorem). First, by using a suitable data structure (*e.g.* a hash table), one only needs $O(N)$ memory to store the N (x_i, y_i) pairs, and every collision can be detected in average constant time. Thus, we only need to look at the expected number of possible additional false positives at step 2),[¶] which are the key candidates returned when $y_i = y_j$ while $x_i \oplus x_j \neq k$. If we make the reasonable assumption that $x \mapsto \mathcal{E}(x) \oplus \mathcal{P}(x)$ behaves as a random function the expected number of collisions is again $\Theta(N^2/2^n)$, which is much smaller than N for $N \ll 2^n$. We thus only expect to search k from a few false positives, are equivalently to encounter a few of them before finding the correct value k and stopping the attack. Putting everything together, this attack has a time, memory, and data complexity of N, and a success probability $\min(\Theta(N^2/2^n), 1)$; note also that here one in fact only requires "known-plaintext" access to the $\mathcal{E}$ oracle, which is a somewhat milder requirement to the "chosen-plaintext" actually granted in the PRP definition.

We only describe, and do not analyse the second attack. Unlike the first, it actually requires the keys k_1 and k_2 to be identical (the previous did not, even though it was

[§]We referred to the black-box queries to $\mathcal{P}$ as the "time complexity". While it is indeed reasonable to consider this as a *lower-bound* on the time complexity of an attack that does not exploit structural properties of $\mathcal{P}$, an actual attack algorithm such as the one above might imply additional processing beyond computations of $\mathcal{P}$.

[¶]Since a large number of false positives to examine would lead to an inefficient "real" implementation of the attack.

presented in this simplified case), but it allows to trade queries to $\mathcal{E}$ for queries to $\mathcal{P}$. That is, it succeeds with probability $\min(\Theta(DT/2^n), 1)$ with D and T free to take any value, whereas the previous attack required $D = T$. This attack works as follows:

1. Pick D distinct random values x_i, query $\mathcal{E}(x_i)$, and insert $y_i := \mathcal{E}(x_i) \oplus x_i$ along with x_i in a table.
2. Pick T distinct random values x'_i, query $\mathcal{P}(x'_i)$, and insert $y'_i := \mathcal{P}(x'_i) \oplus x'_i$ along with x'_i in a table.
3. For all i, j such that $y_i = y'_j$, return $x_i \oplus x'_j$ as a key candidate.

One may remark that here one is in fact simply searching for a collision between two functions. It is thus possible to implement this search in a memoryless fashion, though doing this in a straightforward way will require to evaluate both functions the same number of time and thus adding again the constraint $D = T$.

We refer the reader to [DKS12] for a more detailed discussion on multiple variants of the slide attacks.

2.3 Generalisations

There are several ways to generalise the Even-Mansour construction beyond the simple variant taking $k_1 = k_2$ that we already considered. One direction is to consider a block cipher $\mathcal{E}' : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ instead of a public permutation $\mathcal{P}$. This gives the "FX" construction $\mathcal{E}(k||k_1||k_2, m) := \mathcal{E}'(k, m \oplus k_1) \oplus k_2$, which was first suggested by Rivest and then analysed by Kilian and Rogaway [KR01]. Using a method similar to the one of Even and Mansour, they proved that the PRP advantage (this time exactly in the sense of Definition 3) of an adversary attacking $\mathcal{E}$ is upper-bounded by $DT/2^{\kappa+n-1}$, where D and T again represent the number of queries to the unknown oracle $\mathcal{O}$ and the "time complexity" of black-box accesses to the unkeyed $\mathcal{E}'$ respectively.

Another natural way to generalise the scheme is to compose it with independent instantiations of itself, *i.e.* defining $\mathcal{E}(k_1||k_2||\ldots||k_{r+1}, m)$ as $\mathcal{P}_r(\ldots\mathcal{P}_1(m \oplus k_1) \oplus k_2)\ldots) \oplus k_{r+1}$. Chen and Steinberger showed that one needed $O(2^{\frac{rn}{r+1}})$ queries to reach a constant advantage in a PRP game [CS14], while Lampe and Seurin showed an *indifferentiability* property for the twelve-round iterated scheme that uses distinct permutations $\mathcal{P}_1, \ldots, \mathcal{P}_{12}$ but equal keys $k_1 = \ldots = k_{13}$ [LS13]. However, interestingly, the simplest and most economical way to compose an Even-Mansour scheme, *i.e.* taking r identical permutations and $r+1$ equal keys does *not* work, in the sense that it does not provide much more security than the original non-iterated scheme. Let us show why.

Consider $\mathcal{E}(k, m) := \mathcal{P}(\ldots\mathcal{P}(m \oplus k)\ldots) \oplus k$ for an arbitrarily large number of applications of $\mathcal{P}$. We first observe that if two values x and x' are s.t. $x' = x \oplus k$, then $\mathcal{E}(\mathcal{P}(x')) = \mathcal{P}(\mathcal{E}(x)) \oplus k$, and by symmetry $\mathcal{E}(\mathcal{P}(x)) = \mathcal{P}(\mathcal{E}(x')) \oplus k$. It follows that:

$$\mathcal{E}(\mathcal{P}(x)) \oplus \mathcal{P}(\mathcal{E}(x)) = \mathcal{P}(\mathcal{E}(x')) \oplus k \oplus \mathcal{P}(\mathcal{E}(x)) = \mathcal{E}(\mathcal{P}(x')) \oplus \mathcal{P}(\mathcal{E}(x')).$$

Thus one can recover the key of $\mathcal{E}$ with probability ≈ 1 by picking $2^{n/2}$ random values x and looking for collisions for the function $x \mapsto \mathcal{E}(\mathcal{P}(x)) \oplus \mathcal{P}(\mathcal{E}(x))$, each of them suggesting a candidate value for k. Unlike the attacks on the non-iterated Even-Mansour scheme that we have considered so far this one requires *chosen plaintexts*, but this requirement can in fact be relaxed by modifying the attack to instead consider collisions between the lists $\{\mathcal{P}(\mathcal{E}(x)) \oplus x\}$ and $\{\mathcal{E}(x) \oplus \mathcal{P}^{-1}(x)\}$). However neither allows to trade data queries for time queries, and no such attack is currently known in this setting.

3 Related-key attacks

So far we focused on attacks from adversaries who could only access a single oracle, for instance corresponding to $\mathcal{E}(k, \cdot)$ for an unknown key k. A way to increase an adversary's power is then to allow access to a family of oracles, for instance corresponding to $\{\mathcal{E}(\varphi(k), \cdot),\ \varphi \in \Phi\}$, still for an unknown key k, and Φ a set of *related-key functions* $\{0,1\}^\kappa \to \{0,1\}^\kappa$. That is to say, the adversary is given access to $\mathcal{E}$ with several unknown keys that are related one to another through the elements of Φ. We call *related-key attack* an attack that corresponds to this model.

The security requirements associated with related-key attacks are the same as for the *single-key* case. For instance, we can informally state that it should be hard to recover the unknown k for a cipher for which we are given related-key oracle access. It is also possible to define a related-key (S)PRP notion that generalises the one of Definition 3, as was done by Bellare and Kohno [BK03]. We let again $\mathfrak{E}$ denote an ideal cipher, such that for all k, $\mathfrak{E}(k, \cdot) \xleftarrow{\$} \Pi_n$ and write $\mathfrak{E} \xleftarrow{\$} \Pi_n^\kappa$ the uniform sampling of such a cipher. We define a related-key oracle $\mathcal{E}_{\mathrm{RK}(\cdot,K)}(\cdot)$ for a block cipher $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ that takes a first input $\varphi : \{0,1\}^\kappa \to \{0,1\}^\kappa$ and a second input $m \in \{0,1\}^n$, and returns $\mathcal{E}(\varphi(K), m)$. We then have the following.

Definition 4 (Related-key PRP advantage restricted to Φ)**.** The *related-key PRP advantage* of $\mathcal{E}$ with respect to the related-key class Φ is defined as:

$$\mathrm{Adv}^{\text{PRP-RKA}}_{\Phi,\mathcal{E}}(q,t) := \max_{A_{q,t}} \left| \Pr\left[A_{q,t}^{\mathfrak{E}_{\mathrm{RK}(\cdot,K)}(\cdot)} = 1 : K \xleftarrow{\$} \{0,1\}^\kappa, \mathfrak{E} \xleftarrow{\$} \Pi_n^\kappa\right] - \Pr\left[A_{q,t}^{\mathcal{E}_{\mathrm{RK}(\cdot,K)}(\cdot)} = 1 : K \xleftarrow{\$} \{0,1\}^\kappa\right] \right|.$$

One can see that this definition is parameterized by the allowed related-key class Φ. In fact, this parameterization plays a major role in how secure a cipher can be w.r.t. Definition 4: there are classes Φ relatively to which *no* cipher (even an ideal one) attains any meaningful level of security. For instance, assume that Φ includes a constant function $\varphi : x \mapsto c$; then an adversary may simply query its related-key oracle on φ and an arbitrary m and compare the result with $\mathcal{E}(c, m)$; it succeeds with advantage close to one by answering 1 if the two values match and 0 otherwise.

There are less trivial related-key classes for which similar problems occur. One of the perhaps less intuitive examples is to consider $\Phi = \varphi^\oplus \cup \varphi^+$, where $\varphi^\oplus = \{x \mapsto x \oplus \Delta,\ \Delta \in \{0,1\}^\kappa\}$ and $\varphi^+ = \{x \mapsto x + \Delta,\ \Delta \in \{0,1\}^\kappa\}$ (with $+$ denoting here the addition modulo 2^κ). Remarkably, non-trivial security *is* achievable w.r.t. $\varphi^\oplus$ or φ^+ in isolation. We will not explore this formally, rather focusing on the intuition.

There are two key properties for Φ to allow non-trivial security. The first is that it be *collision-resistant*, and the second that it be *output-unpredictable*. Collision resistance means that it is hard to find two functions φ, φ' such that $\varphi(K) = \varphi'(K)$ for a uniform K. This is the property that does not hold for $\varphi^\oplus \cup \varphi+$ and that can be exploited to mount easy attacks using this class. However, it does hold trivially for, say $\varphi^\oplus$, as $\Delta \mapsto K \oplus \Delta$ is a permutation for any K. The output unpredictability property means that it is hard to guess $\varphi(K)$ for a uniform K; more generally, it should be hard to guess any of the elements of $\{\varphi(K), \varphi \in X \subseteq \Phi\}$ relatively to the size of X. It is easy to see that this property does not hold if Φ includes constant mappings, but that it does for $\Phi = \varphi^\oplus$.

We now illustrate the fact that if many related-key queries with different functions φ are allowed, the security of any cipher degrades significantly. Consider a simplified attack setting where the adversary interacts with a known cipher $\mathcal{E}$ with unknown key k and tries to recover k. To do so, he is allowed oracle access to $\mathcal{E}(\varphi(k), \cdot)$, $\varphi \in \varphi^\oplus$. A good (in fact

basically optimal) attack that does not exploit any properties of $\mathcal{E}$ consists in querying $\mathcal{E}(k \oplus \Delta, 0)$ for $2^{\kappa/2}$ randomly selected Δ and putting the results in a list $\mathcal{L}$ along with the value Δ. Then the adversary tries $2^{\kappa/2}$ candidates k' for the key, each time computing $\mathcal{E}(k', 0)$. Any match with an element of $\mathcal{L}$ suggests a value for k, and one expects such a match with high probability.

3.1 Provably-secure constructions

Even when Φ is restricted to *meaningful* functions, the related-key model gives significantly more power to the adversary. The collision attack sketched above is an example; another one is to consider some very simple attacks on the Even-Mansour scheme. Let Φ include at least one function of the form $x \mapsto x \oplus \Delta$, then an adversary can distinguish an Even-Mansour scheme from an ideal cipher by querying $\mathcal{E}(k \oplus \Delta, \Delta)$, $\mathcal{E}(k, 0)$, and checking if the two values only differ by Δ.

This attack does not contradict the security proof of Even and Mansour, as it requires related-key queries that are not covered by the single-key proof. It does show however that a good cipher in the single-key setting might be terribly broken in a related-key setting, even w.r.t. "meaningful" related-key classes.

It is quite easy to see that the above attack also applies to iterated Even-Mansour schemes when independent keys are used at every round. A slightly more involved but also very efficient attack exists on the two-round scheme that uses identical keys. However, it was proven (in an ideal permutation model similar to the one of the single-key proof) that using three rounds or more with identical keys leads to a construction that is related-key secure w.r.t. $\varphi^{\oplus}$ [CS15, FP15].

3.2 Tweakable block ciphers

The fact that single-key and related-key security may be widely disconnected is part of the reason why not all concrete block ciphers are designed to be related-key secure. This comes usually with the argument that resisting related-key attacks (for some a priori defined classes) would add some computational overhead and that the model is altogether unrealistic, as most real-life protocols only use block ciphers with a single key and then do not require related-key security. We will not delve into this debate whose implications are rather complex, instead recalling an example of a constructive use of related-key secure ciphers.

First, let us recall the definition of *tweakable block ciphers*. These are simply block ciphers that take a second parameter called a *tweak*, such that all distinct pairs of keys and tweaks define (in principle) independent permutations. The difference between the key and the tweak is that the latter is public and may be freely chosen by the adversary. Using simplified expressions for the domains, we have the following.

Definition 5 (Tweakable block ciphers)**.** A tweakable block cipher is a family of mappings $\widetilde{\mathcal{E}} : \{0,1\}^{\kappa} \times \{0,1\}^{\theta} \times \{0,1\}^{n} \to \{0,1\}^{n}$ such that for all $k \in \{0,1\}^{\kappa}$, $t \in \{0,1\}^{\theta}$, $\mathcal{E}(k, t, \cdot)$ is a permutation.

Tweakable block ciphers are useful in that they allow to "diversify" a fixed-key instance without selecting a new, independent key. In other words, two communicating parties may first secretly share a secret key k with which to use $\widetilde{\mathcal{E}}$, and then publicly agree on a new tweak for each message to be exchanged. The concept of tweakable ciphers was formalised by Liskov et al. [LRW11], but already implicitly used for instance by Rogaway et al. to define an efficient mode of operation [RBBK01].

A simple way to build a tweakable block cipher $\widetilde{\mathcal{E}}$ from a "regular" cipher $\mathcal{E}$ is to define $\widetilde{\mathcal{E}}(k, t, \cdot)$ as $\mathcal{E}(k \oplus t, \cdot)$. The *single-key* security of this construction fully reduces

to the related-key security of $\mathcal{E}$ w.r.t. $\varphi^{\oplus}$, which may be non-trivial.[‖] For instance, this construction provably achieves a meaningful level of security in the ideal permutation model if $\mathcal{E}$ is a three-round iterated Even-Mansour scheme with identical keys. Note however that because of generic collision attacks, the security is limited w.r.t. the number of different tweaks for which $\widetilde{\mathcal{E}}$ is queried; the design of similar generic constructions that are secure beyond the birthday-bound is a rather active research topic.

3.3 Key-dependent message security

We conclude this section with a very brief mention of a different security notion that has some similarity with related-key security in the sense that it grants additional power to the adversary, *viz. key-dependent message security*. This is concerned with settings where one is *knowingly* given the encryption under the key k of k itself, *i.e.* one is given $\mathcal{E}(k, k)$. This is not captured by the PRP-like definitions since in this case the adversary has to explicitly specify the values on which it queries the oracle, and cannot ask for it to be evaluated on an input it doesn't know. Even without a formal definition for key-dependent message security it is also quite clear that it does not coincide with, say, PRP security since the Even-Mansour construction is trivially broken if one knows that $c = \mathcal{E}(k, k)$: indeed k can be immediately recovered in that case as $c \oplus \mathcal{P}(0)$, but it is equally clear that nothing would have been learned on k had $\mathcal{E}$ been an ideal cipher.

Part II
Some structural attacks

4 Divide & conquer attacks

A historically and practically important technique used to attack block ciphers is the so-called Meet-in-the-middle attack (MiTM; not to be confused with the *Man*-in-the-middle attack). This was first mentioned in the seventies by Diffie and Hellman [DH77] to show that composing two block ciphers with different keys did not increase the security as much as one could naively think; namely, the expected time complexity for a constant success probability of the best generic attack only doubles, while one could have assumed that it goes to the square. In a nutshell, using the notation $\mathcal{E}_k$ for $\mathcal{E}(k, \cdot)$ when $\mathcal{E}$ is a block cipher, a MiTM attack on $\mathcal{E}_{k_1} \circ \mathcal{E}_{k_0}$ simply works as follows:

1. Obtain the ciphertext c corresponding to a known plaintext m for the unknown keys k_0 and k_1.
2. Create a list L of pairs $(k', \mathcal{E}(k', m))$ for all possible keys k'.
3. For all possible k'', compute $x := \mathcal{E}^{-1}(k'', c)$. If x is found as the second element of a pair $(k', \cdot)$ of L, output (k'', k') as a key candidate for (k_1, k_0).

If $\kappa := |k| > n := |m|$, one may use more than one plaintext-ciphertext pair in order to avoid an exponentially-growing number of false positives. Apart from that, this algorithm minimizes the time complexity in generically attacking "double-$\mathcal{E}$". Note however that if one restricts the running time of the adversary to t, the above attack succeeds with probability at most $t^2/2^{2\kappa}$, which is generally lower than the generic complexity of $t/2^{\kappa}$ one can achieve in attacking $\mathcal{E}$ alone. It can in fact be proven that this gap cannot be filled [ABCV98]

[‖] It is interesting to note that even if $\mathcal{E}$ is related-key secure w.r.t. $\varphi^{\oplus}$, $\widetilde{\mathcal{E}}$ admits trivial related-key attacks w.r.t. this same class.

The basic property exploited in the above algorithm is that a collision between two a priori random lists suggests a candidate value for the key. If the lists are of size N and N', this allows to test up to $O(N \cdot N')$ candidates for a certain condition, allowing for a quadratic gain in time complexity. This feature appears in many other collision-based "MiTM" attacks, such as the slide attacks on Even-Mansour constructions or the many-related-keys attack on any block cipher.

MiTM attacks do not only apply in such generic settings. For instance, they can be useful in exploiting weaknesses in the key-schedule of a concrete iterated block cipher.

Definition 6 (Iterated block cipher)**.** An *iterated block cipher* is a block cipher $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ that can be defined as the composition of a *round function* $\rho : \mathbb{N} \times \{0,1\}^{\kappa'} \times \{0,1\}^n \to \{0,1\}^n$ whose *round-keys* are generated by a *key-schedule* or *key-expansion* algorithm $\Gamma : \mathbb{N} \times \{0,1\}^\kappa \to \{0,1\}^{\kappa'}$. That is, the r-round $\mathcal{E}(k, \cdot)$ is equal to $\rho(r, \Gamma(r,k), \cdot) \circ \ldots \circ \rho(0, \Gamma(0,k), \cdot)$.

The huge majority of actual block ciphers are iterated.

One can see that the double-encryption construction $\mathcal{E}_{k_1} \circ \mathcal{E}_{k_0}$ can be redefined as a new block cipher $\mathcal{E}'_k$ with twice as much rounds as $\mathcal{E}$ and a "bipartite" key-schedule Γ' s.t. $\Gamma'(0 \leq i \leq r, k) = \Gamma(i, k_0)$ and $\Gamma'(r < i \leq 2r+1, k) = \Gamma(i-r-1, k_1)$. While one does not expect a concrete design to exhibit such a strong independence property for the two halves of its key schedule, weaker cases of independence remain possible. Suppose for instance that a block cipher $\mathcal{E}$ is such that with high probability (over the keys and the plaintexts), a certain subset $\mathcal{S}$ of the bits of the intermediate ciphertext after r_f rounds, *i.e.* the image of $\rho(r_f, \cdot, \cdot) \circ \ldots \rho(0, \cdot, \cdot)$, only depends on a subset $\mathcal{K}_0$ of the key bits. Assume further that with high probability (over the keys and the ciphertexts), this same subset only depends on a subset $\mathcal{K}_1$ of the key bits when computed as a partial decryption, *i.e.* as the image of $\rho^{-1}(r_f - 1, \cdot, \cdot) \circ \ldots \circ \rho^{-1}(r, \cdot, \cdot)$. Then if $\mathcal{K}_0 - \mathcal{K}_1 \neq \emptyset$ and $\mathcal{K}_1 - \mathcal{K}_0 \neq \emptyset$, one can mount a MiTM attack that independently searches for the bits of either set difference. This kind of approach is for instance the basis for the currently best attacks on the lightweight block cipher KATAN [CDK09, FM14].

5 Statistical attacks & distinguishers

We now introduce an altogether different approach that is very successful in attacking many block ciphers, namely *statistical*, distinguisher-based attacks. This denomination typically regroups *differential* and *linear* cryptanalysis, which are related in many ways. We will mostly focus on differential attacks, but also briefly mention the linear case. both of these approaches were successfully used to attack DES in the early nineties [BS90, Mat93].

A joint feature of these two strategies is they that try to quantitatively express "how closely" the attacked mapping is from being linear, but for two different characterisations of linearity. Informally the differential characterisation uses the fact that if $\mathcal{F}$ is linear then one always has $\mathcal{F}(a+b) = \mathcal{F}(a) + \mathcal{F}(b)$ for any a, b, while the linear one exploits the fact that a linear function $\mathcal{F}$ can be represented by a matrix.

Note on signature types. Fundamentally, block ciphers operate on binary data that does not correspond to any abstract element such as a vector (in the mathematical sense), a finite field element, etc. This is why we typically have $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$. However, in the same way as concrete designs sometimes benefit from algebraic constructions by seeing their inputs as mathematical objects, attacks may also be defined as operating on, say, vectors of $\mathbb{F}_2^n$ rather than binary strings of length n. When this is the case, one needs to agree on a mapping from one set to another. Fortunately, this is usually straightforward.

5.1 The Differential case

Definition 7 (Differential)**.** Let $\mathcal{E}$ be a block cipher, a *differential* is a pair $(\Delta \neq 0, \delta)$ of input and output *differences* for $\mathcal{E}$, according to some group law $+$ (generally taken to be the addition in $\mathbb{F}_2^n$, *i.e.* the XOR or $\oplus$).

An input (k, m) to $\mathcal{E}$ is said to *verify* the differential (Δ, δ) if $\mathcal{E}(k, m+\Delta) - \mathcal{E}(k, m) = \delta$. If the key is fixed, we may also call *differential pair* for (Δ, δ) an ordered pair $((m, c), (m', c'))$ of two plaintext-ciphertext pairs for $\mathcal{E}(k, \cdot)$ s.t. $m' - m = \Delta$ and $c' - c = \delta$. Note that over $\mathbb{F}_2^n$, addition coincides with subtraction, and the following expressions can be simplified. We always consider such a case from now on.

The usefulness of the notion of differential comes from the fact that it is often a good distinguisher between mappings that are "ideally random" or not. That is, one will exploit statistical properties of $x \mapsto \mathcal{P}(x) \oplus \mathcal{P}(x \oplus \Delta)$ (for some Δ) in the hope of deciding whether $\mathcal{P} = \mathcal{E}(k, \cdot)$ for some unknown key k or if it is a random permutation. This can be done by considering the *differential probability* of the differential.

Definition 8 (Differential probability)**.** The *differential probability* of a differential (Δ, δ) w.r.t. a permutation $\mathcal{P}$ is the probability of obtaining a differential pair for for (Δ, δ) for $\mathcal{P}$:

$$\mathrm{DP}^{\mathcal{P}}(\Delta, \delta) := \Pr_{m \in \{0,1\}^n}[\mathcal{P}(m) \oplus \mathcal{P}(m \oplus \Delta) = \delta].$$

Note that the differential probability is a function of both the differential itself and the permutation. In particular, this means that for a block cipher, we may (and usually do) have $\mathrm{DP}^{\mathcal{E}(k,\cdot)}(\Delta, \delta) \neq \mathrm{DP}^{\mathcal{E}(k' \neq k,\cdot)}(\Delta, \delta)$, for an arbitrary (Δ, δ). This leads to the notion of *expected differential probability* for block ciphers, which is simply the average over k of the differential probability of $\mathcal{E}(k, \cdot)$:

$$\mathrm{EDP}^{\mathcal{E}}(\Delta, \delta) := 2^{-\kappa} \sum_{k \in \{0,1\}^\kappa} \mathrm{DP}^{\mathcal{E}(k,\cdot)}(\Delta, \delta).$$

A convenient (unfortunately not necessarily true) hypothesis is that any fixed-key DP for $\mathcal{E}$ is "close" to the EDP. In order to distinguish $\mathcal{E}$ from a random permutation $\mathcal{P}$, then one only needs the difference between $\mathrm{EDP}^{\mathcal{E}}(\Delta, \delta)$ and $\mathrm{DP}^{\mathcal{P}}(\Delta, \delta)$ to be sufficiently large for a given (known) (Δ, δ). This begs the question of what is the expected value of $\mathrm{DP}^{\mathcal{P}}$ for a random differential. In the non-injective case, replacing $\mathcal{P}$ by a random function $\mathcal{F}$, the answer is easy. By definition, $\mathcal{F}(x) \xleftarrow{\$} \{0,1\}^n$ for all x, thus $\Pr[\mathcal{F}(x) \oplus \mathcal{F}(x \oplus \Delta) = \delta] = \Pr[\mathcal{F}(x) = \delta \oplus \mathcal{F}(x \oplus \Delta)] = 2^{-n}$. The injective case is more complex, but one can show that the number of differential pairs is approximately drawn according to a Poisson distribution of mean and variance 2^{-1} [O'C95, DR07]. This means that the expected DP is also equal to 2^{-n}; note however that it can only take values multiple of 2^{-n+1} (as differential pairs are symmetric and thus come by two). It is also worthwhile to remark that if $\mathcal{P}$ is *linear* w.r.t. the difference operation (here $\oplus$), then by definition, $\mathrm{DP}^{\mathcal{P}}(\Delta, \delta) = 1$ if $\mathcal{P}(\Delta) = \delta$, 0 otherwise.

We now have everything we need to describe a differential distinguisher on a block cipher. This is done in algorithm 1. This distinguisher only succeeds with some probability, that can among other things depend on the actual value of $\mathrm{DP}^{\mathcal{E}(k,\cdot)}$ for the random key k. The multiplicative constant on line 3 can be set to ensure that at least one differential pair is found with high probability. However, if the EDP is close to 2^{-n}, selecting a large constant may decrease the success probability, by increasing the likelihood that a differential pair is found for a random permutation (this issue may be slightly reduced by having the algorithm returning zero as soon as a differential pair is found). Finally, the data cost is of $2 \cdot \mathrm{mult}/p = \Theta(p^{-1})$ chosen plaintexts, which directly dictates the time cost as well. The memory cost depends on up to what point one ensures that the m on line 5 are

Input: $\mathcal{O} \xleftarrow{\$} \{\mathcal{P}, \mathcal{E}(k \xleftarrow{\$} \{0,1\}^\kappa, \cdot)\}$; (Δ, δ) of $\mathrm{EDP}^{\mathcal{E}} = p$
Output: 1 if $\mathcal{O} = \mathcal{P}$, 0 otherwise

```
1  begin
2      count := 0
3      mult := 10 /* Can be set to other values                    */
4      for i := 0; i < mult/p do
5          m ←$ {0,1}^n /* Without replacement, also removing m ⊕ Δ  */
6          if O(m) ⊕ O(m ⊕ Δ) = δ then
7              count := count + 1
8          end
9      end
10     if count = 0 then
11         return 1
12     end
13     else
14         return 0
15     end
16 end
```

Algorithm 1: Differential distinguisher for $\mathcal{E}$

indeed sampled uniformly and without replacement (where one should also exclude $m \oplus \Delta$ if m has already been tried): using a memoryless uniform sample *with replacement* will lead to collisions and thence a noticeable loss of efficiency when $1/p \geq 2^{n/2}$; storing all the m's requires memory; generating the m's as the images of a pseudorandom permutation applied to a running counter requires negligible memory and loses at most a factor two from checking a pair only once.

Finding a distinguisher is enough to break the PRP security of a block cipher. However, when possible, it is even better for an attack to recover some key material. It is also possible to do so with differential attacks, by using a distinguisher as a test for the possible values of (part of) the unknown key.

We describe this process in a case where the block size n is equal to half of the key size κ and where round keys are as long as the block. We also assume that given one n-bit round key r_k there are 2^n easy-to-enumerate $2n$-bit master keys which have r_k as a round key (for the appropriate round). A simple key schedule for which this statement is true is $\Gamma(i, k_1||k_0) := k_{(i \mod 2)}$. In that case, using a divide-and-conquer approach, an attacker might enumerate all 2^n possible values for the last round key and, if he is able to uniquely determine the right one thanks to a distinguisher, the full key can be determined by again enumerating all possible 2^n values and checking them w.r.t. a few plaintext-ciphertext pairs.

The crucial step is the first one, that is trying to distinguish the right guess for the last round key from all the other ones. In a differential attack, this might be done by having a "good" distinguisher up to the before-last round. Say that the expected probability of the differential used in this distinguisher is $p \gg 2^{-n}$, *i.e.* $\mathrm{EDP}^{\mathcal{E}/(r-1)} = p$, where $\mathcal{E}/(r-1)$ denotes $\mathcal{E}$ reduced to $r-1$ rounds. One assumes that the EDP of the same differential is much smaller after r rounds, *i.e.* $\mathrm{EDP}^{\mathcal{E}/r} \ll p$. Going further, we assume that composing $\mathcal{E}/r$ with *the inverse of the round function with a random round key* does not increase the EDP of the differential (ideally increasing it); that is, if we let $\mathcal{E}'(k, \cdot) := \rho^{-1}(r, k' \xleftarrow{\$} \{0,1\}^{\kappa'}, \cdot) \circ \mathcal{E}/r(k, \cdot)$, we assume that $\mathrm{EDP}^{\mathcal{E}'} \lll p$. With these assumptions, the attacker may then run the following algorithm 2, at the end of which a further 2^n candidates for the entire key need to be tried. The time complexity of this

Input: $\mathcal{O} = \mathcal{E}(k \xleftarrow{\$} \{0,1\}^\kappa, \cdot)\}$; (Δ, δ) of $\mathrm{EDP}^{\mathcal{E}/(r-1)} = p$
Output: A candidate for the last round key

```
1  begin
2      max := 0
3      cand := 0
4      forall k' ∈ {0,1}^n do
5          count := 0
6          mult := 10
7          for i := 0; i < mult/p do
8              m ←$ {0,1}^n
9              c_0 := ρ^{-1}(r, k', (O(m)))
10             c_1 := ρ^{-1}(r, k', (O(m ⊕ Δ)))
11             if c_0 ⊕ c_1 = δ then
12                 count := count + 1
13             end
14         end
15         if count > max then
16             max := count
17             cand := k'
18         end
19     end
20     return cand
21 end
```

Algorithm 2: Differential key-recovery attack for $\mathcal{E}$

algorithm is $\Theta(2^n \cdot p^{-1})$; its memory and data complexity depend on whether one is able (or willing) to reuse data for different key guesses; its success probability depends on the validity of all the involved hypotheses (which unfortunately might be hard to test). Note that in practice, one does not need to return a single candidate: it is perfectly reasonable to keep all candidates whose counter is above a certain threshold. Of course, this requires some additional memory and subsequent tests for the correct full key, so it is still best to keep this number quite low.

The basic approach as sketched above is unlikely to be used as is in an actual attack. For instance, the process used to guess (and eliminate) (part of) a round key may be more complex; several differentials may be used; *truncated* differentials may be employed; some early-abort strategies may be deployed; etc. We do not describe these in these notes and refer an interested reader to *e.g.* [KR11, Din14] for examples.

So far, we have assumed that a suitable differential (Δ, δ) was known to the attacker. Most of the time, finding such a differential is in fact the crux of the attack. We will not address this problem, but briefly mention one of the starting points to do so, that is finding differential *characteristics* (or trails).

In all generality, computing the differential probability $\mathrm{DP}^{\mathcal{P}}(\Delta, \delta)$ for $\mathcal{P} : \{0,1\}^n \to \{0,1\}^n$ and a single differential requires to enumerate all possible inputs to $\mathcal{P}$, *i.e.* with time complexity $\Theta(2^n)$ and negligible memory; computing *all* differentials can be done in time and memory $\Theta(2^{2n})$. When n is large (for instance because $\mathcal{P}$ is the round function of a block cipher), this exponential complexity is intractable. However, if $\mathcal{P}$ is a small component used as part of a round function, for instance an 8-bit S-box, the associated complexity is perfectly reasonable. If $\mathcal{E}$ is a typical substitution-permutation-network block cipher, this fact can be used to efficiently compute the differential probability of its round function for an arbitrary differential: the DP of the S-boxes are explicitly computed,

while the DP of the linear layer is known. This is not directly useful, as attacking one round of a block cipher is usually not an impressive feat. However, one may hope to chain several one-round differentials to obtain a characteristic.

Definition 9 (Differential characteristic)**.** Let us write $\Delta \xrightarrow{\rho} \Delta'$ the fact that (Δ, Δ') is a differential for the round function ρ (where we drop the round index and the key for simplicity). An r-round *differential characteristic* for ρ is an $(r+1)$-tuple $(\Delta_0, \ldots, \Delta_r)$ s.t. $\Delta_0 \xrightarrow{\rho} \Delta_1 \xrightarrow{\rho} \ldots \xrightarrow{\rho} \Delta_r$.

Note that if an input follows a characteristic[**] $(\Delta_0, \ldots, \Delta_r)$, then it is also a differential pair for the r-round differential (Δ_0, Δ_r). While the converse is not true in general, one sometimes assumes that if a characteristic has a high probability of being followed by a random input, relatively to the block size, then it is dominating the other characteristics that lead to the same differential and the probability of the latter is close to the one of the former. This hypothesis is quite tempting, especially as estimating the probability of a characteristic is much easier than for a differential, even if it requires its own hypotheses.

Let us consider a characteristic $C := (\Delta_0, \ldots, \Delta_r)$. If all round keys of ρ are independent and random, the probability that an input follows C is equal to the product of the one-round differential probabilities, that is equal to $\Pi_{0 \leq i < r} \mathrm{DP}^{\rho}(\Delta_i, \Delta_{i+1})$. In practice, for want of a better model, we assume the same even if the round keys are not independent (which is more often the case than not); this is the *Markov assumption* (for block ciphers) [LMM91]. Using this assumption, one may use various ways to find r-round characteristics and their associated probabilities, for instance "by hand", or using Matsui's branch-and-bound algorithm [Mat94].

5.2 The Linear case

Linear cryptanalysis presents many similarities with the differential approach. The main difference comes from the nature of the statistical property exploited in the underlying distinguishers: in the linear case, one is interested in how *biased* is a linear equation in the input and output bits of a permutation.

Definition 10. Walsh transform The *Walsh transform* $\mathcal{W}^{\mathcal{P}} : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}$ of $\mathcal{P} : \{0,1\}^n \to \{0,1\}^n$ is defined extensively by:

$$\mathcal{W}^{\mathcal{P}}(a, b) := \sum_{m \in \{0,1\}^n} -1^{\langle b, \mathcal{P}(m) \rangle + \langle a, m \rangle},$$

where $\langle x, y \rangle$, $x, y \in \{0,1\}^n$ denotes the $\{0,1\}$-valued dot product between x and y when seen as vectors of $\mathbb{F}_2^n$, and when the sum is computed over $\mathbb{Z}$.

One can note the similarity of the expressions defining $\mathcal{W}^{\mathcal{P}}(a, b)$ and $\mathrm{DP}^{\mathcal{P}}(\Delta, \delta)$: both involve an expression relating the input and output of $\mathcal{P}$, that is evaluated over all its possible inputs. A difference however is that in the linear case, we only consider a single value $\mathcal{P}(m)$ at a time whereas the differential looks at pairs of input related in a certain way. A consequence of this is that linear attacks may only rely on *known* plaintexts, versus chosen plaintexts for differential ones.

The equation defining the Walsh transform can be defined in words as counting how often a given *linear approximation* $\langle a, m \rangle = \langle b, \mathcal{P}(m) \rangle$ with *linear masks* (a, b) holds, versus how often it does not, when m ranges over all possible values. In the extreme case where $\mathcal{P}$ is linear, this equality is either always true, or it is true exactly half of the time, and $\mathcal{W}^{\mathcal{P}}(\cdot, \cdot) \in \{2^n, 0\}$.[††] On the other hand, if the approximation with masks (a, b) holds

[**] Defined in the obvious way.

[††] The proof is left as an exercise.

approximately the same number of time that it does not, then $\mathcal{W}^{\mathcal{P}}(a, b)$ is small but not exactly 0.

The *correlation* $C_{\mathcal{P}}(a, b)$ of a linear approximation (a, b) for $\mathcal{P}$ is defined directly from the Walsh transform as $C_{\mathcal{P}}(a, b) = \mathcal{W}^{\mathcal{P}}(a, b)/2^n$; this quantity thus ranges from -1 to 1. One can show that for a non-trivial approximation (a, b) and $n \geq 5$, the distribution of $C(a, b)$ over all n-bit permutations can be approximated by a normal distribution $\mathcal{N}(0, 2^{-n})$ [DR07, BT13]. This means that if a certain $\mathcal{P}$ is such that $C = C_{\mathcal{P}}(a, b)$ is sufficiently away from 0 for a given approximation (a, b), one may be able to distinguish it from a random permutation in a way similar to algorithm 1: an adversary may count how many times the approximation holds over sufficiently many $D := \Theta(C^2)$ input values, and decide that he is interacting with $\mathcal{P}$ if this is sufficiently away from $D/2$. Finally, to continue the analogy, such distinguishers can also be used in key-recovery attacks, and linear approximations can also be chained over several rounds in a way similar to differential characteristics.

6 Algebraic attacks

We will for a moment step back from n-to-n-bit mappings, and consider the n-to-one case of *Boolean functions*.

Definition 11 (Boolean function)**.** A *Boolean function* with n variables is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Note that we could have alternatively used a signature $f : \{0, 1\}^n \to \{0, 1\}$, which would be equivalent for most purposes. Our choice was motivated by the fact that we will precisely treat Boolean functions as algebraic objects. However, we will nonetheless adopt a compact representation for vectors of $\mathbb{F}_2^n$, writing $011\ldots$ for the vector $[0, 1, 1, \ldots]$.

A natural way to represent an n-variable Boolean function f is to define its "truth table", that is, to list all its 2^n possible inputs and their associated evaluation. This simply requires a string s_f of 2^n bits, where the value of the i^{th} bit of the string $s[i]$ is given by the evaluation of f on the vector that maps to i as a binary integer.

Example 1. The 3-variable Boolean function f given by $f(000) = 1$, $f(001) = 0$, $f(010) = 0$, $f(011) = 1$, $f(100) = 1$, $f(101) = 1$, $f(110) = 0$, $f(111) = 1$, can be represented by the truth table `10011101`, read from left to right (*i.e.* the bit of smallest index is on the left).

With this representation in mind, it is particularly obvious that there are 2^{2^n} distinct n-variable Boolean functions: one for each 2^n-bit string.

An important fact about Boolean functions is that they possess another "natural" representation, as multivariate polynomials over $\mathbb{F}_2$.

Definition 12 (Algebraic normal form and degree of a Boolean function)**.** The *algebraic normal form* (ANF) of the n-variable Boolean function f is the unique polynomial $g \in \mathbb{F}_2[X_0, X_1, \ldots, X_{n-1}]/\langle X_i^2 - X_i\rangle_{i<n}$ such that for all $x \in \mathbb{F}_2^n$, $f(x) = \text{eval}(g, x[0], x[1], \ldots, x[n-1])$ (where we may also write $g(x[0], x[1], \ldots, x[n-1])$ for the evaluation $\text{eval}(g, x[0], x[1], \ldots, x[n-1])$).

The *degree* of f is the degree of its ANF g.

We can prove existence and unicity with these statements: 1) any polynomial of $R := \mathbb{F}_2[X_0, X_1, \ldots, X_{n-1}]/\langle X_i^2 - X_i\rangle_{i<n}$ maps to an n-variable Boolean function defined by its evaluation; 2) this mapping is injective, as two distinct such polynomials g, g' define different Boolean functions, since their difference $g - g'$ is not the zero polynomial and no non-zero polynomial of R identically evaluates to zero; 3) this mapping is bijective, as the two sets have the same cardinality.

We now have at least two ways of representing a Boolean function: by its truth table and by its ANF. However, while going from the ANF to the truth table is easy, we still need an efficient way of computing the ANF of a function given its truth table. Fortunately, it turns out that doing so is easy too.

First, stating the obvious, computing the ANF means retrieving the value of the coefficient in front of all possible 2^n monomials (which is either one or zero). This is trivial for the constant monomial (written $g_{00\dots0}$): the ANF g of f has a constant term iff $f(00\dots0) = 1$. Indeed, by definition, $g(0,0,\dots,0) = g_{00\dots0} = f(00\dots0)$. This is not too hard either for the n degree-one monomials $X_0, \dots X_{n-1}$. Say that we want to compute the coefficient $g_{10\dots0}$ in front of X_0: we just need to evaluate f on $10\dots0$ and add the result (modulo two) to $g_{00\dots0}$; this is again simply by definition, as $f(10\dots0) = g(1,0,\dots,0) = g_{10\dots0} + g_{00\dots0}$, so $g_{10\dots0} = f(10\dots0) + g_{00\dots0} = f(10\dots0) + f(00\dots0)$. We then simply proceed inductively for higher-degree monomials. For instance, $g_{110\dots0}$ is given by $f(110\dots0) + f(100\dots0) + f(010\dots0) + f(000\dots0)$, and more generally $g_u = \sum_{v \preccurlyeq u} f(v)$, where $a \preccurlyeq b$, $a, b \in \mathbb{F}_2^n$ if $\forall i, b[i] = 0 \Rightarrow a[i] = 0$. The mapping $u \mapsto g_u$ is sometimes called the *Möbius transform(ation)* in analogy with Möbius' inversion formula.

Example 2. The ANF of the Boolean function f of Example 1 is given by $g = 1 + X_1 + X_2 + X_0X_2 + X_0X_1X_2$.

This approach calls for three major comments. The first is that to compute a coefficient g_u, of a degree-d monomial X_u, we are in effect differentiating d times the d-variate polynomial g' obtained from g where we only "keep" the indeterminates that appear in X_u. This polynomial is of degree at most d, with only X_u as a possible degree-d monomial. Thus, the result of the differentiation is either a non-zero constant polynomial (*i.e.* 1) if g' is of degree exactly d, or the zero function. This in turns indicates if $X_u \in g'$, *i.e.* the value of g_u.

Formally, the notion of differentiation that we use is defined as follows.

Definition 13 (Derivative of a Boolean function)**.** The *derivative* of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *along* $\Delta \in \mathbb{F}_2^n$ is defined as $\partial f / \partial \Delta := x \mapsto f(x) + f(x + \Delta)$. The order-$d$ derivative $\partial^d f / \partial \Delta_0, \dots \Delta_{d-1}$ is defined as $\partial(\dots(\partial f / \partial \Delta_0)\dots) / \partial \Delta_{d-1}$.

The second comment is related to the complexity of computing the ANF of f from its truth table. A naïve implementation might simply compute the value of each coefficient independently; there are 2^n of them, and the coefficient of a monomial of degree d requires 2^d evaluations of f. The total cost of this approach is thus $\sum_{0 \leq i \leq n} \binom{n}{i} 2^i = 3^n$ calls to f. However, one can do much better by observing that computing the coefficient of g_u computes as intermediate results the values of all of the g_v's, $v \preccurlyeq u$. In particular, the computation of $g_{111\dots1}$ also computes on the way the values of *all* the other $2^n - 1$ coefficients of the ANF. Thus, we only need to compute $g_{111\dots1}$ while computing all intermediate values only once and storing them along the way. This can be done conveniently in a recursive way: first compute the ANF of the two degree-$(n-1)$ functions $f^{(1)} := g(1, \cdot, \cdot, \dots, \cdot)$ and $f^{(0)} := g(0, \cdot, \cdot, \dots, \cdot)$, obtaining $g^{(1)}$ and $g^{(0)}$, then return $g = X_0\, g^{(1)} + g^{(0)}$. This algorithm can be transformed into an efficient iterative and in-place procedure [Jou09], given as algorithm 3. The inner loop 5–10 is executed n times by the outer loop. Its i^{th} execution takes 2^{n-i} executions of the innermost loop 6–8 that itself performs 2^i elementary computations. The total complexity of algorithm 3 is thus $n2^n$ elementary operations. This compares very favourably with the cost of $3^n \approx 2^{1.58n}$ of the naïve algorithm, even for small values of n.

The third and last comment we make is that the Möbius transform is its own inverse, *i.e.* an involution. One can prove this statement recursively by seeing that $f(00\dots0) =$

Input: The truth table s of an n-variable Boolean function f
Output: The table s is overwritten with the coefficients of g, the ANF of f

```
1  begin
2      for i := 0; i < n do
3          L := 2^i
4          R := 0
5          while R < 2^n do
6              for j := 0; j < L do
7                  s[L + R + j] := s[L + R + j] ⊕ s[R + j]
8              end
9              R := R + 2L
10         end
11     end
12 end
```

Algorithm 3: Fast Möbius transform computation (iterative version)

$g(0, 0, \ldots, 0)$ and that

$$\begin{aligned} & g(1, 0, \ldots, 0) = f(10\ldots0) + f(00\ldots0) \\ \Leftrightarrow\ & g(1, 0, \ldots, 0) = f(10\ldots0) + g(0, 0, \ldots, 0) \\ \Leftrightarrow\ & f(10\ldots0) = g(1, 0, \ldots, 0) + g(0, 0, \ldots, 0). \end{aligned}$$

One should notice the relation between a Möbius and a Fourier transform: both allow to interpolate a function from its values and vice-versa. The Walsh transform of Definition 10 could also be seen as a kind of Fourier transform, and can also be computed with a fast algorithm.

We now go back to n-to-n-bit mappings $\mathcal{P}$. As any such mapping can be written as the collection of n n-to-one-bit Boolean functions, all of the results obtained so far in this section extend smoothly. In particular, one may define the ANF of $\mathcal{P}$ as the collection of the ANFs of its constituent Boolean functions (*e.g.* projected on the canonical basis). This allows to naturally define the degree of $\mathcal{P}$ to be the maximal degree of its n Boolean functions (obtained in turn thanks to their respective ANFs); note however that the *minimum* degree among all constituent Boolean functions is *not* invariant from the basis used for the projection. An important fact is that if $\mathcal{P}$ is a permutation, then it is of degree at most $n - 1$. This is simply a consequence of the fact that the coefficient of the unique degree-n monomial (for each of the projected functions) is given by a projection of $\bigoplus_{x \in \{0,1\}^n} \mathcal{P}(x) = 0$.

The degree of a mapping $\mathcal{P}$ can be used as a distinguisher. A random mapping should be of maximal degree (*i.e.* $n - 1$) with high probability: one ANF of its constituent functions may be drawn at random (with the constraint that it is not of degree n), and the probability that all the coefficients of the n degree-$(n - 1)$ monomials is zero is 2^{-n}. Thus, if a concrete mapping is of a lower degree, we can use this as a distinguishing criterion, provided that we have an efficient test for the degree.

The simplest way to compute the degree of $\mathcal{P}$ is to compute its ANF. However, this is exponential in the block size of $\mathcal{P}$, and thus quickly becomes intractable in a cryptographic context. If the degree d to be tested for is not too high, an efficient alternative is to differentiate $\mathcal{P}$ at order $d + 1$ along $\Delta_0, \ldots, \Delta_d$ in a way that is equivalent to computing one coefficient g_u of the ANFs of each of the constituent functions of $\mathcal{P}$, *i.e.* by having all the Δ_is of weight one and disjoint support (*e.g.* $\Delta_0 = 100\ldots$, $\Delta_1 = 0100\ldots$, etc.) and evaluating the resulting function on an arbitrary point. Doing thusly, if $\mathcal{P}$ is of degree d, the result is necessarily zero over all the n functions of $\mathcal{P}$; on the other hand, if it is

of degree (much) larger than d, the result is zero iff X_u is not a given (sub)monomial in the ANF of any of the functions of, say, a random permutation. By repeating the test a few time with different monomials, one can distinguish the high and low degree cases with high probability. Note that the fast Möbius transform can be used both to compute a single test and to efficiently bundle several ones into the test for an even higher-degree monomial.

Algebraic attacks such as the above are somewhat less common than differential and linear cryptanalysis, but they can nonetheless be very effective. Some examples are given by attacks on the Trivium stream cipher [DS09, FV13], reduced-round Keccak [BCC11], the ASASA construction [MDFK15, BK15], or by attacks using the "division property" framework to show that some specific output bis are the image of low-degree functions [Tod15, BC16].

Some of these attacks use a particularly nice theorem due to Boura, Canteaut and De Cannière, that provides a non-trivial upper-bound for the degree of an iterated mapping. Quite trivially, if ρ is of degree d, then ρ^r is of degree less than d^r. This already provides a lower-bound on the necessary number of rounds for, say, an iterative cipher, to resist the above attack. However, what Boura et al. showed is that if ρ uses "small" S-boxes of non-maximal degree, the degree of ρ^r may be much lower. We state a particular case of their theorem as Theorem 1

Theorem 1 ([BCC11])**.** *Let $\rho : \{0,1\}^n \to \{0,1\}^n$ be a function corresponding to the concatenation of m smaller invertible S-boxes of size $n_0 \geq 3$ and degree at most $n_0 - 2$, then for any function $\rho' : \{0,1\}^n \to \{0,1\}^n$, we have:*

$$\deg(\rho' \circ \rho) \leq n - \frac{n - \deg(\rho')}{n_0 - 2}$$

For instance, if ρ uses 32 degree-2 4-bit S-boxes, the naïve bound tells us that at least 7 rounds are necessary to reach the maximal degree $127 < 2^7$, while Theorem 1 gives a much higher number of at least 12 rounds (the successive upper-bounds being 2, 4, 8, 16, 32, 64, 96, 112, 120, 124, 126, 127).

A well-known example of distinguisher that can be thought of in an algebraic way is the square/integral/saturation distinguisher on 3-round AES. Recall that this distinguisher works by fixing the input to fifteen of the sixteen AES S-boxes to a constant and summing the 256 3-round ciphertexts obtained when the input to the last S-box ranges over all possible values. The distinguishing criterion is then for this sum to be zero. In other words, one is simply computing the presence of certain degree-8 monomials that are guaranteed to be absent in the ANF of 3-round AES, unlike in the case of a random function.

Finally, one should also mention the relation between the "deterministic" version of algebraic attacks that we have introduced and their statistical counterparts of *higher-order differentials*. The astute reader will have noticed that differential cryptanalysis exploits statistical properties of order-one derivatives along the input differences Δ, where the fact that (Δ, δ) is a high-probability differential means that this derivative is biased towards δ. There is no reason why this could not be generalized to higher-order derivatives as well, exploiting the bias of, say, $x \mapsto \mathcal{P}(x) \oplus \mathcal{P}(x \oplus \Delta_0) \oplus \mathcal{P}(x \oplus \Delta_1) \oplus \mathcal{P}(x \oplus \Delta_0 \oplus \Delta_1)$. Because each round of differentiation may decrease the degree of the resulting mapping, it may become more biased towards certain values than higher-degree ones (for instance, a degree zero mapping is indeed very biased towards its constant value!). A drawback of this approach, however is that the evaluation of a derivative grows exponentially in its order.

Part III
Elements of S-box design

7 Substitution boxes

A large number of block ciphers use *substitution boxes* (or S-boxes) as components. Those are simply functions with small (binary) domains that are extensively defined and that may be used in various stages of an algorithm.

Definition 14 (S-box)**.** An *S-box* is a (usually injective) mapping $S : \{0,1\}^b \to \{0,1\}^{b'}$, where one usually has $b = b' \in \{4, 8\}$.

Early ciphers did not necessarily use injective S-boxes (most famously DES), or even fixed ones (*e.g.* Blowfish, where large S-boxes are generated as part of the key schedule). However, as this is now much less common, we will focus on the "modern" usage of S-boxes which fit the typical profile of Definition 14, that is 4 or 8-bit permutations that are explicitly defined by a table. Beyond DES and Blowfish-like S-boxes, this excludes among others the 3-bit S-box of PrintCipher [KLPR10], the 5-bit S-box of SHA-3 [Nat15] and the 16-bit S-box of Whirlwind [BNN+10]. As an example, we give the table of the third S-box of Serpent [BAK98] in Table 1.

Table 1: Serpent S6 S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	2	C	5	8	4	6	B	E	9	1	F	D	3	A	0

S-boxes are often used as the only non-linear component of a cipher. It is thus important that they are themselves sufficiently resistant against attacks that exploit non-linearity such as typically differential, linear and algebraic cryptanalysis. Because of the limited size of their inputs and outputs, S-boxes can be analysed "exhaustively" with respect to the properties underlying these attacks. For instance, while it is intractable to compute the differential probability of a differential for a large (say, 128-bit) permutation, doing so for an 8-bit S-box is very easy. The same can be said of linear approximations and algebraic normal forms. One can adapt the definitions of these notions to the specific case of S-boxes in the following way.

Definition 15 (Differential uniformity of an S-box)**.** Let S be an n-bit S-box. We define its *difference distribution table* (or DDT) as the function δ_S defined extensively by:

$$\delta_S(a, b) := \#\{x \in \mathbb{F}_2^n \,|\, S(x) + S(x + a) = b\}.$$

The *differential uniformity* $\mathcal{D}$ of S is defined as:

$$\max_{(a,b)\neq(0,0)} \delta_S(a, b).$$

Put another way, an n-bit S-box with differential uniformity $\mathcal{D}$ has a maximum differential probability of $\mathcal{D}/2^n$ over its inputs, for any differential.

Definition 16 (Linearity of an S-box)**.** Let S be an n-bit S-box. We define its *linear approximation table* (or LAT) as the function ℓ_S defined extensively by:

$$\ell_S(a, b) := \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle b,S(x)\rangle+\langle a,x\rangle}.$$

The *linearity* $\mathcal{L}$ of S is defined as:

$$\max_{(a,b)\neq(0,0)} |\ell_S(a,b)|.$$

This means that the absolute correlation of any linear approximation of an n-bit S-box of linearity $\mathcal{L}$ is upper-bounded by $\mathcal{L}/2^n$. For example, the S-box of Table 1 has differential uniformity 4 and linearity 8. It turns out that those are in fact optimal for a 4-bit S-box [LP07]

Intuitively, it seems to be the case that an S-box with low differential uniformity and linearity is necessary to design an efficient and secure cipher: anytime a non-trivial difference (say) is present on the inputs of an S-box (one then says that the S-box is *(differentially) active*), their outputs will verify any difference with probability at most $\mathcal{D}/2^n$, which is useful knowledge for a designer. Consequently, 8-bit S-boxes with low uniformity (for instance 4) and linearity seem to be systematically better than 4-bit ones, for instance, because a single-bit input difference to a cipher activates a single S-box. The same arguments can be made w.r.t. the ANF: an 8-bit S-box may have a maximal degree of 7 versus 3 for a 4-bit one, and the Boura et al. upper-bound on the degree of iterated round functions grows comparatively faster with large S-boxes as well.

Despite the above arguments, small 4-bit S-boxes are also used intensively along larger 8-bit ones. In fact, the choice of the size of an S-box is a good example of the difficulty of finding the "right" trade-off in cryptographic designs. A first point is that good 8-bit S-boxes are systematically much more expensive to implement than good 4-bit ones (whether the cost function is hardware circuit, lookup tables, etc.). A second point is that S-boxes are never used in isolation; a good 8-bit S-box used with a poor diffusion layer in an SPN may result in a weaker round function that a 4-bit S-box with a good diffusion layer. All in all, there are many settings where a small S-box is as good a choice as a large one, if not better.

Even when the size of an S-box is fixed, several strategies are available to design an S-box. These include random generation, optimal (or near-optimal) generation, or building the S-box as a small block cipher, from even smaller components. We will now describe each of these approaches.

7.1 Random generation of S-boxes

A simple way to define an S-box is simply to draw all of its entries at random. Given a sufficiently good PRNG, this is elementary if the S-box does not need to be invertible. In the more likely case where the S-box is a permutation, a less trivial algorithm such as the Fisher-Yates/Knuth uniform shuffle can be used. Assuming uniform randomness is available and on input N this algorithm uniformly samples a permutation among all $N!$ permutations of N elements. In effect, this defines an ideal cipher over a small domain. We give a description in algorithm 4. Note that if line 3 of algorithm 4 is replaced with $s \xleftarrow{\$} \{0, \ldots, N-1\}$, one obtains something similar to the shuffle used *e.g.* in RC4. This modified shuffle is not uniform, as it defines N^N equally likely permutations, and $N! \nmid N^N$.

This strategy for generating S-boxes is not very good. While a uniformly chosen permutation is nearly guaranteed to offer good resistance against differential, linear, and algebraic attacks, it is almost certainly possible to do better w.r.t. to the first two for the typical S-box sizes. Furthermore, a random S-box will likely be expensive to implement, as it is by definition unstructured.

An alternative "random" approach is to pseudo-randomly generate a circuit or a sequence of instructions implementing a permutation, and then to check if the result has good cryptographic properties [UDI+11]. If one only considers "efficient" circuits (w.r.t. some given cost function), one is guaranteed to obtain an S-box with an efficient implementation, at condition that the algorithm terminates. This approach was quite successful

Input: A table T of size N holding all N numbers $0 \dots N-1$ each exactly once.
Output: A permutation of $0 \dots N-1$ uniformly selected among the $N!$ possible ones.

```
1 begin
2 |  for i := 0; i < N − 1 do
3 |  |  s ←$ {i, . . . , N − 1}
4 |  |  Swap T[i] and T[s]
5 |  end
6 |  return T
7 end
```

Algorithm 4: Uniform shuffle

at finding 4-bit S-boxes with optimal differential and linear properties and very low (in fact minimal) cost, *e.g.* using only four linear (XOR) and four non-linear (not XOR) two-to-one gates.

7.2 (Near-)Optimal generation

Ideally, one would want to know S-boxes that have (near-)optimal resistance against some classes of attacks. For instance, this could mean having a differential uniformity as close to the minimum of two as possible. Let us first consider the impact of this approach for 4-bit S-boxes. There are only $16! \approx 2^{44.2}$ possible 4-bit S-boxes, which is not a small amount by any means, but still possible to exhaustively enumerate. If the important characteristics of an S-box (say, differential uniformity, linearity and maximum degree on all output bits) can be tested efficiently enough, then it should be possible to sample all possible S-boxes and detect which ones are the best.

This type of exhaustive search has been performed several times [BCBP03, LP07, Saa11], but in a more clever way that is significantly more efficient. The idea is to notice that properties such as differential uniformity and linearity are invariant by composition with an affine mapping. That is, for any S-box S and any $M_i, M_o \in \mathrm{GL}_4(\mathbb{F}_2), \boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^4$, the S-box $S' : x \mapsto M_o \times S(M_i \times x \oplus \boldsymbol{a}) \oplus \boldsymbol{b}$ is such that $\mathcal{D}_S = \mathcal{D}_{S'}$ and $\mathcal{L}_S = \mathcal{L}_{S'}$. Thus, one only needs to enumerate permutations of 16 elements up to affine equivalence and to keep classes with the best cryptographic properties; each of these classes can then be compactly represented by a single S-box[‡‡]. If one then wishes to test additional criteria, for an S-box (and assuming that optimal resistance to differential and linear cryptanalysis remains important, which is usually the case), it is enough to do so on members of the known "optimal" classes. All in all, there are 16 such classes (cf. *e.g.* [LP07]) of 4-bit S-boxes of differential uniformity 4 and linearity 8. Note however that the degree of an S-box is not invariant by affine composition, neither have to be some global properties of a cipher such as its maximum expected differential probability [CR15]. The notions of optimality and equivalence of S-boxes described above should be understood as relative to some specific criteria, and not as absolute.

For 8-bit S-boxes, restricting to affine-equivalent classes is not enough to make a similar exhaustive search manageable. Yet, it is possible to explicitly design S-boxes with very good cryptographic properties. In fact, such an approach is also possible for smaller sizes, including 4-bit S-boxes. It is however quite more interesting for larger S-boxes because of the limitations of exhaustive enumeration.

An efficient way to design very good n-bit S-boxes is to use as a basis the inversion map ι of $\mathbb{F}_{2^n}^\times$, extended in zero by $\iota(0) = 0$. One can show that the resulting S-box (using

[‡‡]It is worth pointing out that the inverse of an S-box S is not necessarily in the same class as S

a suitable mapping from $\mathbb{F}_{2^n}$ to $\{0,1\}^n$) has differential uniformity 4 if n is even and 2 if it is odd, and linearity $2^{n/2+1}$ (in the even case) [Nyb93]. For instance, this means that a 4-bit S-box generated by this process is optimal w.r.t. to these criteria. In the 8-bit case, this means that one can easily generate S-boxes of differential uniformity 4 and linearity 32. The AES S-box, as well as many others, was designed in such a way.

An important remark is that directly taking ι as an S-box can lead to attacks if the S-box is used within an $\mathbb{F}_{2^n}$-linear round function. In that case, the whole round function admits a simple algebraic description over that field, which can be exploited [JK97]. This should be taken as a further warning that statistical approaches, while important to consider, are by no means the only way to attack a block cipher. In practice, it is however easy to avoid such algebraic attacks, *e.g.* by composing ι with a binary matrix that has no simple algebraic expression over the larger field $\mathbb{F}_{2^n}$. This is for instance what is done in the AES S-box.

It is remarkable that for n odd, an S-box generated from such an invert mapping has the lowest possible differential uniformity. This is not quite the case for n even, and one knows that the theoretical lower-bound cannot be reached for *e.g.* $n = 4$. Some isolated examples of differentially-optimal mappings for some even values of n do exist [BDMW10], but it is in general an open problem to determine the best reachable differential uniformity for all cases. As far as 8-bit S-boxes are concerned, this question remains unanswered.

From an implementation perspective, S-boxes based on the inversion are not lightweight; this is particularly true in the 8-bit case. While this is not an issue if the S-box is implemented as a table lookup, this approach is most of the time not the one to be preferred. Even if the prevalence of the AES means that much effort has been spent developing efficient implementations of its S-box, it remains quite more expensive than more structured alternative, which we describe next.

7.3 S-boxes as small block ciphers

It is possible to rethink entirely the design of S-boxes as trying to build small secure block ciphers with efficient implementations. While this statement is ignoring some subtleties (for instance consequences of the fact that S-boxes are unkeyed), this view opens the way to many design possibilities. First, considering that 4-bit S-boxes can be efficiently designed *from scratch*, one only needs to consider larger cases. Then, smaller S-boxes may precisely be used as sub-components of higher-level constructions that for instance take a 4-bit S-box as input and define an 8-bit one. Finally, these constructions may be borrowed from block cipher round-function design, such as *e.g.* Feistel structures. We now take this as an opportunity to catalogue many such constructions which have successfully been used in block cipher and S-box design.

7.3.1 Feistel structure

The Feistel structure is altogether one of the oldest in block cipher design. A 3-round Feistel can be used to define a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 as:

1. $x_1^L := x_0^R; \quad x_1^R := S_0(x_0^R) \oplus x_0^L;$
2. $x_2^L := x_1^R; \quad x_2^R := S_1(x_1^R) \oplus x_1^L;$
3. $x_3^L := x_2^R; \quad x_3^R := S_2(x_2^R) \oplus x_2^L;$

and $S(x_0^L || x_0^R) = x_3^L || x_3^R$.

An interesting property of the Feistel structure is that it can be proven that in a block cipher context, three rounds amplify the resistance to differential and linear cryptanalysis by squaring the differential uniformity and the squared correlation (which is a function of

the linearity) of the worst of the three small S-boxes. However, this is a statement that applies to a *keyed* context when each entry to a small S-box is masked with a random key, and where the amplification is obtained for the MEDP and MELP of the resulting cipher. That is, *on average*, an 8-bit S-box built from a 3-round Feistel using smaller components will be much more resistant to statistical attacks than the small S-boxes themselves.

Taken literally, this result is not very useful, as an unkeyed S-box needs to fix all parameters, and guarantees in the average case are thus meaningless. However, it can be seen as quite encouraging: one can randomly sample many distinct instantiations of this construction, evaluate the characteristics of the result, and stop whenever a satisfactory one has been found. In fact, it is very easy to find the desired amplification properties by using this process. In the 4-to-8 case, this means that one can easily design 8-bit S-boxes with differential uniformity 16 and linearity 64 by using three optimal 4-bit S-boxes as $S_{0,1,2}$. This is quite weaker than, what one would get with an inverse mapping, but not too expensive to implement if the 4-bit S-boxes are well-chosen.

It is interesting to remark that because the Feistel structure does not require the round function to be invertible, one is not restricted to injective 4-bit S-boxes. This observation may lead to better S-boxes, as non-injective mappings may be found that have stronger differential properties. This was used by Canteaut et al. to design an 8-bit S-box from a 3-round Feistel that is very cheap to implement and that has differential uniformity only 8 [CDL15]. However, the linearity of this S-box is 64, which gives it an uneven resistance to typical differential and linear cryptanalysis. Yet there are some contexts where an increased resistance to differential cryptanalysis specifically may be useful.

7.3.2 Misty structure

The Misty structure was first used in the Misty block cipher [Mat97], that also pioneered the recursive approach of building S-boxes as small block ciphers. This structure is very close to a Feistel one, but requires the round function to be invertible. A 3-round Misty benefits from the same amplification properties as a 3-round Feistel. Such a structure, when building a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 can be defined as:

1. $x_1^L := x_0^R; \quad x_1^R := S_0(x_0^L) \oplus x_0^R;$
2. $x_2^L := x_1^R; \quad x_2^R := S_1(x_1^L) \oplus x_1^R;$
3. $x_3^L := x_2^R; \quad x_3^R := S_2(x_2^L) \oplus x_2^R;$

and $S(x_0^L || x_0^R) = x_3^L || x_3^R$. All in all, the Misty structure also allows to build good large S-boxes from smaller ones, similarly as the Feistel case. Because of invertibility constraints of the small S-boxes, it is however not possible to reach particularly low differential uniformity as in the Feistel case [CDL15].

7.3.3 Lai-Massey structure

Our last example is the Lai-Massey structure, that was introduced in the block cipher IDEA [LMM91]. Similarly as Misty, it requires invertible S-boxes. In its simplest form, this structure builds a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 as:

1. $x_1 := S_0(x_0^R \oplus x_0^L);$
2. $x_2^L := S_1(x_0^L \oplus x_1);$
3. $x_2^R := S_1(x_0^R \oplus x_1);$

and $S(x_0^L||x_0^R) = x_2^L||x_2^R$. A slightly more complex variant using five S-boxes was used in the (second version of the) 8-bit S-box of the WHIRLPOOL hash function [BR03], which lead to a good differential uniformity of 8 and linearity of 56. When using only three small S-boxes, a typical Lai-Massey 8-bit S-box has differential uniformity and linearity comparable to the average Feistel or Misty case, that is 16 and 64 respectively.

It is interesting to remark that Lai-Massey S-boxes lead to a very simple *impossible differential* pattern: if a pair of inputs to S has the same difference on the two halves x_0^L and x_0^R, there is no difference in the output of S_0 and thus there is *always* a difference in the input (and thence output) of S_1 and S_2. The probability of all differentials of the form $\Delta||\Delta \to 0||\delta$ and $\Delta||\Delta \to \delta||0$ is therefore equal to zero. While there is no obvious way to exploit this property in an attack, is should again serve as a useful reminder that as important as they can be, a few key properties are unlikely to capture all important behaviours of a cryptographic mapping.

8 Implementing S-boxes

By definition (at least in our restricted sense), an S-box is an explicit permutation with a small domain. Consequently, a straightforward way to compute the application of an S-box is to define a table that stores the image of x at index x. This table is easy to compute from the definition of the S-box (usually itself given in this very form, see Table 1).

This approach is relatively efficient, but it is not without drawbacks. Mostly, it imposes some minimum size requirement for the code of any algorithm using the S-box, which might not be negligible if an 8-bit S-box is used on constraint platforms, and it requires to access secret-dependent memory locations, which depending on the architecture might lead to the so-called "timing attacks" [TOS10].

We will briefly present two alternative ways of computing S-box applications, that may not always be applicable but that have some merits.

8.1 Parallel table lookups with vector instructions

Vector instructions are CPU instructions that operate on large registers (*e.g.* 128 or 256 bits) that are logically thought of as holding multiple small elements (*e.g.* sixteen 8-bit values). Current CPUs support many instructions that take such registers as input and that compute a single instruction with all the vectors elements as operands. This paradigm is commonly known as SIMD: "single instruction, multiple data".

An example of SIMD instructions on Intel/AMD processors is `PADDD`, which takes two 128-bit `xmm` registers (or possibly a 128-bit memory location for the second operand) logically seen as vectors `x` and `y` of four 32-bit values, and that updates the entries of `x` to hold the values `x[0] + y[0]`, `x[1] + y[1]`, etc.

A SIMD instruction that is very useful to implement many cryptographic algorithms is `PSHUFB`, which computes a parallel table-lookup of a 4-to-8-bit table on sixteen 4-bit values. Seeing its 128-bit operands x and y as vectors of sixteen bytes, the result of calling `PSHUFB` x y is to update x as follows:

$$x'[i] = \begin{cases} x[\lfloor y[i] \rfloor_4] & \text{if the most significant bit of } y[i] \text{ is not set} \\ 0 & \text{otherwise} \end{cases}, \ 0 \leq i < 16$$

where $\lfloor \cdot \rfloor_4$ denotes truncation to the four least significant bits. One can see that by taking x to be a table T of sixteen arbitrary 8-bit "images", and by taking y to be any vector of sixteen 4-bit "indices" $(a, b, c, \ldots)$, calling `PSHUFB` x y will update x to x' as $x'[0] = x[y[0]] = T[a]$, $x'[1] = x[y[1]] = T[b]$, etc. By taking T to be the images of an S-box, a single call to `PSHUFB` performs a parallel application of this S-box on 128 bits of data. This is obviously particularly useful to implement block ciphers using 4-bit S-boxes,

but the approach can be successfully extended to a larger S-box as well if the S-box can be efficiently expressed in terms of smaller tables. This is for instance the case of the 8-bit S-boxes of subsection 7.3, but also to some extent of the one of the AES [Ham09].

Like most SIMD instructions, `PSHUFB` may be called in C by using compiler intrinsics, in this case `_mm_shuffle_epi8`. The following Figure 1 is an example implementation of an 8-bit S-box using a Lai-Massey structure, using C intrinsics.

```
__m128i sub(__m128i x)
{
  __m128i xlo, xhi, xmid;

  __m128i LO_MASK = _mm_set1_epi8(0x0f);
  __m128i LO_SBOX = _mm_set_epi32(0x01030605, 0x0b090e0d, 0x0802070c,
  ↪  0x0f040a00);
  __m128i HI_SBOX = _mm_set_epi32(0x10306050, 0xb090e0d0, 0x802070c0,
  ↪  0xf040a000);

  xhi  = _mm_srli_epi16(x, 4);
  xhi  = _mm_and_si128(xhi, LO_MASK);
  xlo  = _mm_and_si128(x, LO_MASK);
  xmid = _mm_xor_si128(xlo, xhi);

  xmid = _mm_shuffle_epi8(LO_SBOX, xmid);
  xlo  = _mm_xor_si128(xlo, xmid);
  xhi  = _mm_xor_si128(xhi, xmid);

  xlo = _mm_shuffle_epi8(LO_SBOX, xlo);
  xhi = _mm_shuffle_epi8(HI_SBOX, xhi);
  x   = _mm_xor_si128(xlo, xhi);

  return x;
}
```

Figure 1: SSE implementation of an 8-bit S-box using compiler intrinsics.

8.2 Bitsliced implementations

S-boxes being typically small and applied many times in parallel, they are a typical target for vectorization. This can be done as in the above, by using dedicated vector instructions, but also by considering the very usual bitwise operations such as `^`, `&` and `|` in C as operating on vectors of w bits, where w is the bit-length of the operands (typically up to 64). That is, by definition, computing `a ^ b` with `a` and `b` 64-bit registers means computing 64 parallel 1-bit XORs.

This basic property can be used to easily vectorize any Boolean function, as soon as one knows how to compute it explicitly in terms of its input variables. For instance, on a 64-bit architecture, it is not costlier to compute the function $(a, b, c) \mapsto a \cdot b + c$ 64 times than only once: the C code `r = a&b ^ c` results in the bit i of `r` being equal to the above function applied to the bit i of `a`, `b` and `c`.

The biggest hurdle to using bitsliced implementations of S-boxes in practice is to represent the S-box inputs in a suitable way. For instance, if S must be applied to a string of consecutive bits in memory, each input must first be split as bits of the same

position in several registers, which may be an expensive process. Consequently, bitsliced implementations are usually used either to implement the encryption of many inputs to the block cipher in parallel (in which case the input format conversion is simpler), as in the case of DES for which this technique was originally introduced in cryptography [Bih97a], or for algorithms that were explicitly designed to take advantage of such techniques, such as Serpent or SHA-3.

Unlike implementations by table lookups (parallel or not), the efficiency of a bitsliced implementation is highly dependent on the one of computing its associated Boolean functions in an explicit way. In other words, one want to find efficient logical circuits implementing the S-box. Note that as already hinted at in subsection 7.1, this can be done either by first specifying an S-box and then searching for efficient circuits (see *e.g.* [Osv00] in the original context of Serpent), or by generating efficient circuits and testing if they make good S-boxes [UDI+11]. In Figure 2, we give a bitsliced implementation of the S-box of Table 1 (together with the required data input and output transformation).

Finally, let us remark that an explicit way of computing an S-box can always be derived from its algebraic normal form. However, if not further simplified, it will usually be rather inefficient.

```
uint8_t serpent_s6(uint8_t x)
{
        uint8_t r0 = x & 1;
        uint8_t r1 = (x >> 1) & 1;
        uint8_t r2 = (x >> 2) & 1;
        uint8_t r3 = (x >> 3) & 1;
        uint8_t r4;

        r2 ^= 1;
        r4  = r3;

        r3 &= r0;
        r0 ^= r4;

        r3 ^= r2;
        r2 |= r4;

        r1 ^= r3;
        r2 ^= r0;

        r0 |= r1;
        r2 ^= r1;

        r4 ^= r0;
        r0 |= r3;

        r0 ^= r2;
        r4 ^= r3;

        r4 ^= r0;
        r3 ^= 1;

        r2 &= r4;

        r2 ^= r3;

        return (r0 | (r1 << 1) | (r4 << 2) | (r2 << 3));
}
```

Figure 2: Bitsliced implementation of the Serpent S6 S-box.

References

[ABCV98] William Aiello, Mihir Bellare, Giovanni Di Crescenzo, and Ramarathnam Venkatesan. Security amplification by composition: The case of doubly-iterated, ideal ciphers. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 390–407. Springer, 1998.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[BC16] Christina Boura and Anne Canteaut. Another View of the Division Property. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 654–682. Springer, 2016.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Biham [Bih03], pages 33–50.

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of keccak and *Luffa*. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.

[BDMW10] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. An apn permutation in dimension six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[Bih97a] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings* [Bih97b], pages 260–272.

[Bih97b] Eli Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*. Springer, 1997.

[Bih03] Eli Biham, editor. *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*. Springer, 2003.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In Biham [Bih03], pages 491–506.

[BK15] Alex Biryukov and Dmitry Khovratovich. Decomposition attack on SASASASAS. *IACR Cryptology ePrint Archive*, 2015:646, 2015.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[BNN+10] Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptography*, 56(2-3):141–162, 2010.

[BR] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. Lecture notes available at http://cseweb.ucsd.edu/~mihir/cse207/index.html.

[BR03] Paulo S.L.M. Barreto and Vincent Rijmen. The Whirlpool Hashing Function, May 2003. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BT13] Andrey Bogdanov and Elmar Tischhauser. On the wrong key randomisation and key equivalence hypotheses in matsui's algorithm 2. In Moriai [Mor14], pages 19–38.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Clavier and Gaj [CG09], pages 272–288.

[CDL15] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of lightweight s-boxes using feistel and MISTY structures. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 373–393. Springer, 2015.

[CG09] Christophe Clavier and Kris Gaj, editors. *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*. Springer, 2009.

[CR15] Anne Canteaut and Joëlle Roué. On the behaviors of affine equivalent sboxes regarding differential and linear attacks. In Oswald and Fischlin [OF15], pages 45–74.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014.

[CS15] Benoit Cogliati and Yannick Seurin. On the provable security of the iterated even-mansour cipher against related-key and chosen-key attacks. In Oswald and Fischlin [OF15], pages 584–613.

[Dae91] Joan Daemen. Limitations of the even-mansour construction. In Imai et al. [IRM93], pages 495–498.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer*, 10(6):74–84, 1977.

[Din14] Itai Dinur. Improved differential cryptanalysis of round-reduced speck. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2014.

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in cryptography: The even-mansour scheme revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2012.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[EM91] Shimon Even and Yishay Mansour. A construction of a cioher from a single pseudorandom permutation. In Imai et al. [IRM93], pages 210–224.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *J. Cryptology*, 10(3):151–162, 1997.

[FM14] Thomas Fuhr and Brice Minaud. Match box meet-in-the-middle attack against KATAN. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2014.

[FP15] Pooya Farshim and Gordon Procter. The related-key security of iterated even-mansour ciphers. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 342–363. Springer, 2015.

[FV13] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Moriai [Mor14], pages 502–517.

[Ham09] Mike Hamburg. Accelerating AES with vector permute instructions. In Clavier and Gaj [CG09], pages 18–32.

[Hel94] Tor Helleseth, editor. *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*. Springer, 1994.

[IRM93] Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors. *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *Lecture Notes in Computer Science*. Springer, 1993.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Biham [Bih97b], pages 28–40.

[Jou09] Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. Printcipher: A block cipher for ic-printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[KR01] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *J. Cryptology*, 14(1):17–35, 2001.

[KR11] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LP07] Gregor Leander and Axel Poschmann. On the classification of 4 bit s-boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[LRW11] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. *J. Cryptology*, 24(3):588–613, 2011.

[LS13] Rodolphe Lampe and Yannick Seurin. How to construct an ideal cipher from a small set of public permutations. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 444–463. Springer, 2013.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Helleseth [Hel94], pages 386–397.

[Mat94] Mitsuru Matsui. On correlation between the order of s-boxes and the strength of DES. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In Biham [Bih97b], pages 54–68.

[MDFK15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2015.

[Mor14] Shiho Moriai, editor. *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*. Springer, 2014.

[Nat15] National Institute of Standards and Technology. FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015.

[Nyb93] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Helleseth [Hel94], pages 55–64.

[O'C95] Luke O'Connor. On the distribution of characteristics in bijective mappings. *J. Cryptology*, 8(2):67–86, 1995.

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.

[Osv00] Dag Arne Osvik. Speeding up serpent. In *AES Candidate Conference*, pages 317–329, 2000.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205. ACM, 2001.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4×4-bit s-boxes. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2011.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[TOS10] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptology*, 23(1):37–71, 2010.

[UDI+11] Markus Ullrich, Christophe De Cannière, Sebastian Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4×4-bit S-boxes. In *SKEW 2011*, 2011.