Chapter 2 - Differential cryptanalysis.

James McLaughlin

1 Introduction.

Differential cryptanalysis, published in 1990 by Biham and Shamir [5, 6], was the first notable cryptanalysis technique to be discovered outside the world's intelligence agencies. It was the first technique to allow an attack on DES faster than exhaustive search [7]; although as with linear cryptanalysis the number of chosen plaintext/ciphertext pairs required meant that exhaustive search was still more feasible in practice.

Differential cryptanalysis is very similar to linear cryptanalysis; a "differential characteristic" is built up to cover some of the rounds by calculating individual characteristics for various S-boxes and then joining these together. In this case, we are trying to find a situation where, for some pair of plaintexts (P_i, P_j) and their corresponding ciphertexts (C_i, C_j), the "output difference" $C_i \oplus C_j$ will take a particular value ΔY with sufficiently high probability assuming that the "input difference" $P_i \oplus P_j$ takes some particular value ΔX. To build up the corresponding differential characteristic, we join together the individual S-box characteristics into characteristics for individual rounds (as we did for linear cryptanalysis), and then join together the round characteristics by letting the output difference from one round be the input difference into the next. For a particular set of key bits, we partially decrypt the cipher and, for each possible value of these bits, maintain a count of how many times the correct output difference occurred.

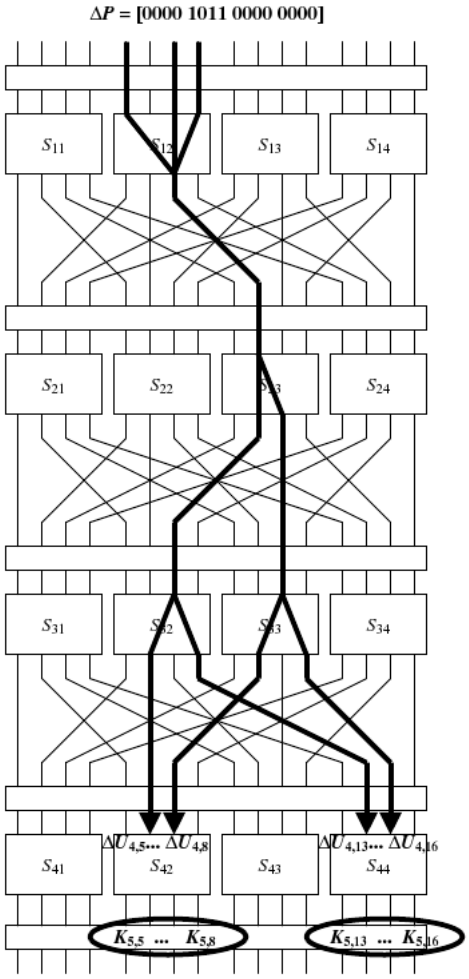
Definition 1.1. For each S-box, we construct a table. Each column corresponds to a given output difference, each row to a given input difference. Each entry in the table is the number of pairs of S-box inputs with input difference corresponding to the row that map to pairs with the column's output difference.

We call these tables *difference distribution tables*. For example, this is the difference distribution table for the 4x4 S-box in [10]:

		Output Difference															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Difference	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
	2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
	3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
	4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
	5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
	6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
	7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
	8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
	9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
	A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
	B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
	C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
	D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
	E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
	F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

Because of the need to have pairs of plaintexts with the given input difference, differential cryptanalysis is a *chosen-plaintext* attack, unlike linear cryptanalysis which was a *known-plaintext* attack. This makes it harder to carry out, as the cryptanalyst has to be able to obtain ciphertexts corresponding to specified pairs of plaintexts instead of just random plaintexts. (In fact, variants of the attack employ even more complicated sets of plaintexts related in specific ways, such as quartets $(P_1, P_2 = P_1 \oplus \Delta X_1, P_3 = P_1 \oplus \Delta X_2, P_4 = P_2 \oplus \Delta X_2)$.)

As stated, by joining together the differential characteristics, we obtain a *differential* for the rounds in which they feature, $(\Delta X, \Delta Y)$. Let y denote the number of output bits affected by ΔY. We are trying to choose and link up differential characteristics so that ΔY will be the output difference for those y bits significantly more often than $1/2^y$ of the time, given that the input difference is ΔX.



(A differential characteristic linked across several rounds of the sample SPN in [10].)

An important factor in choosing and linking together differential characteristics is that, for ciphers like DES where the key interacts with the data only by being xored with it before entering the S-box, it will have absolutely no effect on the characteristic! Think about an S-box with six input bits and four output bits. Let's assume we're working with input difference 010000 (in other words, inputting pairs that differ only in bit x_2) and output difference y_1.

Where the superscripted number identifies whether a data value corresponds to the first or second input in the pair, input difference 010000 means that $x_2^1 \oplus x_2^2 = 1$, and $x_i^1 \oplus x_i^2 = 0$ for all other i. So, where the corresponding data bits are d_2^1 and d_2^2, and the corresponding key bit is k_2, $(d_2^1 \oplus k_2) \oplus (d_2^2 \oplus k_2) = 1$, which reduces to $(d_2^1 \oplus d_2^2) = 1$ - the key bit has no effect!

In general, differential cryptanalysis has not been as limited as linear cryptanalysis in terms of the rounds that can be covered by characteristics. A given attack, designed for maximum effectiveness, may cover only the first $r - 1$ rounds [10], it may cover the first $r - 3$ rounds ([5] section 4), it may cover rounds 2 to $r - 2$ and use the first round differently [7] ... All this varies depending on the cipher and any tweaks made by the cryptanalyst to optimize the attack for it, although we will say that substitution-permutation networks, since their round function acts on the whole data block, will probably require more rounds to be covered as the diffusion throughout the remaining rounds will be faster.

The overall differential characteristic is considered not to be the differential $(\Delta X, \Delta Y)$, but

the set of input and output differences for each of the various rounds of the cipher, accompanied for each round by the set of S-boxes involved and the input/output difference we chose for each. In practice, it's the set of individual characteristics that we used to build it. Even if the differences specified by the differential characteristic do not occur, the differential may still occur (that is, the output difference may be ΔY for input difference ΔX), so the probability of the overall characteristic is a lower bound for the probability of the differential.

2 The attack.

2.1 Obtaining ciphertexts.

We begin by obtaining the ciphertexts corresponding to our chosen plaintexts, by whatever means are necessary.

Definition 2.1. A pair $((P_i, C_i), (P_j, C_j))$ such that $P_i \oplus P_j = \Delta X$ is called a *right pair* if $C_i \oplus C_j = \Delta Y$. If this is not the case, it is called a *wrong pair*.

How many chosen plaintexts do we need? Assume that we're counting the number of times the expected output difference occurs for each TPS candidate. Let p denote the probability that a given pair with input difference ΔX is a right pair. For the correct TPS, we need the expected output difference to occur at least once, so we need at least one right pair to be present. We expect one right pair to be present in $1/p$ pairs, so $2/p$ plaintexts would be a reasonable lower bound to start with. (Not all differential attacks work like this, though - the "memoryless" variant exploits the quartet structure to require only $1/p$ plaintexts.)

Deriving a better figure is rather complex. We will explain it in more detail after we have discussed some "preprocessing" done on the (P_i, C_i) pairs before the main attack.

2.1.1 Discarded pairs

Before the main attack, we can discard certain pairs which cannot possibly be right pairs. Consider the way in which our differential characteristic travels through various S-boxes of the cipher, and in particular the ones it *doesn't* pass through in the final round covered by the characteristic. If the characteristic holds, then since there wasn't any difference in the input to these S-boxes, there shouldn't be any difference in output either. We trace this effect through the final, uncovered round(s) of the cipher to identify ciphertext bits that shouldn't differ.

Hence any chosen plaintext pair in which some of these bits differ in the corresponding ciphertexts should be immediately discarded. The differential characteristic couldn't have held. We can't check the plaintext for anything like this, though, as we chose the plaintexts so that they only differed in the right places.

It may be possible to discard more pairs based on the S-boxes which *are* involved in the final round of the characteristic. For the input differences specified, certain output differences occur with probability zero - these are called *impossible differentials*. If it's possible to spot that these have occurred without a partial decryption (and this is sometimes, but not always, the case) we can discard the pairs in which they occurred.

These will discard pairs which are certainly wrong pairs. Shamir and Biham also describe in [5] a method of discarding pairs which have high probability of being wrong pairs. This seems in practice not to have discarded too many right pairs, and to have helped improve the effectiveness of the attack.

For a given pair, look at the actual output differences that have occurred for the S-boxes involved in the final round of the characteristic. For the predicted input differences for these S-boxes, and the observed output differences, we multiply together the corresponding values in their difference distribution tables to obtain what is known as the *weight* of the pair. [5] states without proof that a right pair will typically have a higher weight than a wrong pair, and that for the attack on DES presented in that paper eliminating all pairs with weight below 8192 "discarded about 97% of the wrong pairs" while leaving "almost all of the right pairs".

There may be ways to discard more pairs for a given cipher, but in any case we discard all the pairs we can and are left with a set which should contain a higher concentration of right pairs.

2.1.2 Suggested subkeys.

Definition 2.2. We say that a pair of chosen plaintexts "suggests" a given value for the TPS if the correct output difference occurs for that pair and that TPS candidate.

In working out how many chosen plaintexts we will need, the average (mean) number of TPSes suggested by each pair is also important.

As an example of how to calculate this, let's make the simplifying assumption that our characteristic covers all rounds except the final round. Consider a final-round S-box that is affected by our TPS, and the expected input difference to it. We'll refer to the subset of the TPS bits corresponding to this S-box as the TPSS(target partial sub-subkey). For every possible output difference from that S-box, let us total up the number of pairs with the expected input difference that can give rise to it, then divide by the total number of possible output differences.

Lemma 2.3. *For conventional S-boxes such as those used in DES or AES, this figure is* $2^{|input\ bits|-|output\ bits|}$. *We do not provide a proof here, but with knowledge of the properties of difference distribution tables it is easy to prove.*

This value is the number of TPSSes suggested by the S-box. This is because, for a randomly chosen output difference in the ciphertext, it is the average number of keys that could have turned the input pair into one of the ones that could result in that difference.

We multiply the figures we have obtained for each S-box, and obtain a, the average number of target partial subkeys suggested by each pair.

2.1.3 The signal to noise ratio.

Definition 2.4. Let p be the probability of the differential characteristic used in the attack, a be the average number of TPS candidates suggested by each pair, b be the ratio |non-discarded pairs|/|total number of pairs|, and m be the number of bits in the TPS.

The *signal to noise ratio*, denoted S/N, is

$$\left(\frac{2^m \times p}{a \times b}\right).$$

Using S/N, which we now have enough information to work out, we can obtain a better estimate for the number of pairs we need. In [5], Biham and Shamir state that experiments indicate an S/N between 1 and 2 requires between $40/p$ and $80/p$ chosen plaintexts, and that the higher the signal to noise ratio, the less right pairs we need. S/N is in fact a rearrangement of an equation for (number of right pairs present)/(number of times an average TPS gives

the correct output difference), and hence a lower bound for (number of times the right key is counted)/(number of times an average key is counted.) Hence, if it's too far below 1, the attack will fail.

(This doesn't necessarily mean that variant attacks like *impossible differential cryptanalysis* will fail, though.)

2.2 After we have obtained the pairs.

2.2.1 Counting on every possible TPS.

The most basic way to carry out a differential cryptanalytic attack is, for each pair of known plaintexts, to partially decrypt their corresponding ciphertexts with every possible TPS. We allocate an integer variable to each TPS candidate, initialise it to 0 at the start, and increment it by 1 whenever the expected output difference occurs for its corresponding subkey.

We either accept the TPS with the largest count as the correct one, or (if we are not so sure that it has a commanding lead) start off by assuming that it's correct, but switch to the one with the next highest count if it turns out not to be, and so on.

Note that it has to be the largest count, not the one that deviated the most from |plaintext pairs|/2. This is an important difference between differential and linear cryptanalysis.

2.2.2 The clique method.

The clique method was introduced in [5] to deal with TPSes so large that the amount of memory required to assign a counter to each one wasn't feasible. It is, however, only feasible if a relatively small number of pairs are being analysed (though since a large TPS will result in a large signal to noise ratio, that may not be an unreasonable assumption.)

Let us associate with each pair some form of data structure to keep track of the TPS values it suggests. If a is low enough, a linked list would seem to be a good way to do this; however Biham and Shamir did not make this assumption, and constructed an alternate data structure using less memory but which would result in false positives for some candidate keys. In practice this doesn't seem to have been a problem for them, though.

For each S-box affected by the TPS bits, we allocate $2^{\text{TPS bits affecting that S-box}}$ of storage to each pair. Typically this will be $2^{\text{input bits}}$ per S-box, one for each possible sub-TPS affecting it. We initialise them to zero.

Whenever a particular TPS is suggested by the pair, for the sub-TPS corresponding to each S-box we set its bit to 1. So, by concatenating sub-TPSes whose corresponding bits have been set to 1, the idea is that we thus reconstruct a suggested key. This is where the false positives issue becomes relevant - if two keys are suggested, and if five S-boxes are involved, we may have to set two bits to 1 for every S-box. This means that we have in practice recorded up to $2^{|\text{involved S-boxes}|} = 32$ keys as suggested when only two were! The reason this didn't cause problems may have been that no individual false positive of this sort was suggested for particularly many pairs, but this is not stated explicitly.

We then need to find out which key is recorded as having been suggested by the most pairs. Again, a linked list of objects (each object being a pair and its associated data on suggested keys) would seem to be a useful data structure to use.

For the first pair in our list, we iterate through the list until we find another pair such that they both suggest a common key. We then continue to iterate through the list looking for other pairs that suggest this key. We keep track of how many suggested it, then look for pairs that

have different keys in common with the first... Eventually, we reach the bottom of the list, and start looking for pairs which suggest keys that the second pair also suggested.

The more pairs that suggest a particular TPS candidate, the more likely we consider it to be the correct one.

2.2.3 When the TPS is too long for the first method, and there are too many pairs for the clique method.

In this case, we will have to use the conventional method, except that we don't attack the full TPS because we aren't counting on all the S-boxes. In other words, we keep counters for every possible value of a smaller TPS, defined by removing bits from the actual TPS corresponding to S-boxes we decided not to count on - call them *redundant* S-boxes.

We will need a particularly high number of pairs for this to work, as the reduced TPS size will reduce S/N. We can mitigate this somewhat by checking, for the redundant S-boxes, whether impossible input/output difference combinations have occurred and using this fact to discard more pairs, however.

After recovering the sub-TPS, we may be able to use the redundant S-boxes in a second attack to recover the rest of the TPS, or we may simply proceed to an exhaustive search on all the key bits we have not yet found.

2.2.4 An entirely different approach - "memoryless" attacks.

Biham and Shamir were not able to turn any of the above approaches into an attack on DES faster than exhaustive search. However, in [7], they were able to come up with a new way to perform the attack that worked with longer TPSes than any of the preceding methods, and to use this in a more powerful attack on DES. It was also intended to use less memory, to be highly parallelisable, and to produce results fast enough to deal with frequent key changes.

The basic idea is that the TPS should be very close in size to the actual key, and that as soon as a given TPS is suggested by one of the pairs, all possible values of the remaining key bits should be tried to see if one of them gives us the correct key. If none of them do, we resume the attack.

The attack requires a much more complex set of chosen plaintexts than before - instead of a set of pairs with the required input difference, huge "structures" are defined containing several (2^{13} in the attack on DES) pairs with that input difference, all of them related in some particular way. (Biham and Shamir later optimised the attack by using quartets instead of pairs.) In the example given, each structure had an extremely low chance of containing even one right pair, and 2^{35} structures were needed to give the attack a 58% probability of succcess.

Working out how the pairs in a structure should be related depends heavily on the differential characteristic being used, the cipher being attacked, and any other tweaks made to the method - in the attack on DES, for instance, the characteristic began at the second round and only a fraction of the pairs in each structure had the required difference going into this round. It's something the cryptanalyst will have to work out for herself for each individual attack.

3 Variants of the differential cryptanalytic attack

There have been several variants of differential cryptanalysis, and attacks building on the basic concept, since it was first introduced. We briefly discuss some of these variants here; *truncated*

differential cryptanalysis, *higher-order* differential cryptanalysis, and *multiplicative/hybrid* differentials [9]. Although we do not discuss it here, we also draw the reader's attention to another variant, *impossible* differential cryptanalysis [13, 3, 2].

Attacks building on the concept of differential cryptanalysis include *boomerang attacks* [20], *amplified boomerang attacks* [11] and *rectangle attacks* [4]. Again, however, we do not cover these here.

3.1 Truncated differential cryptanalysis

In a conventional differential attack, ΔX and ΔY are completely defined. However, it may be possible to carry out attacks in which we do not need to know the full output difference, just some of the bits in it. For example, given a DES S-box, instead of working with output difference 0110, we might be interested in all output differences in which bit 3 changed and bit 4 did not, and be uninterested in the left-hand bits. So, instead of the differential $(\Delta X, 0110)$, we would instead have the *truncated* differential $(\Delta X, ??10)$

Similarly, we might have discovered for a given Serpent S-box that output difference 0011 occurs with high probability as long as the input difference is either 0111 or 1111 (As far as we know, this is not in fact the case for any of Serpent's S-boxes.) and decide to use both of these input differences. This would give us the truncated differential $(?111, 0011)$.

Both of these are special cases of truncated differential characteristics, in which only some of the bits in the input/output differences are specified. Truncated differential cryptanalysis does, however, generalise even further than this - it is not even necessary to specify specific bits. A truncated differential is defined as a pair(information determining a subset of all input differences ΔX, information determining a subset of all possible output differences ΔY.) For example, the truncated input difference might be ??00, and we might be interested in whether the first and last bits of the output difference XOR to 1.

Truncated differential cryptanalysis was first defined by Lars Knudsen in [12], in which it was used to attack 6-round DES. The "partial differential" cryptanalysis of [8] is in fact an example of truncated differential cryptanalysis.

3.2 Higher-order differential cryptanalysis

Higher-order differential cryptanalysis was first defined by Lai in [14], and developed further by Knudsen in [12]. To understand higher-order differentials, it is first necessary to understand the concept of *derivatives*:

Definition 3.1. The (first-order) derivative of a Boolean function $f(x)$, with respect to a vector s, is defined as $\Delta_s f(x) = f(x+s) - f(x)$. Usually, we will be working over $GF(2)$ and so this will equate to $\Delta_s f(x) = f(x) \oplus f(x \oplus s)$. This generalises directly to the case of multiple-output Boolean functions.

Definition 3.2. The definitions of higher-order derivatives are defined recursively from the above definition - so $\Delta^i_{(a_1,\ldots,a_i)} f(x) = \Delta_{a_i}(Delta^{i-1}_{(a_1,\ldots,a_{i-1})} f(x))$.

For example, consider the second order derivative $\Delta^2_{a_1,a_2} f(x) = \Delta_{a_2}(\Delta_{a_1} f(x))$:

$$\begin{aligned}
&\Delta_{a_2}(\Delta_{a_1} f(x)) \\
&= \Delta_{a_2}(f(x+a_1) - f(x)) \\
&= f(x+a_1+a_2) - f(x+a_2) - f(x+a_1) + f(x).
\end{aligned}$$

(As before, since most ciphers operate over $GF(2)$, this will usually equate to $f(x \oplus a_1 \oplus a_2) \oplus f(x \oplus a_2) \oplus f(x \oplus a_1) \oplus f(x)$.)

We then also rely on the following results:

Lemma 3.3. *If the cipher operates over $GF(2)$, and if the entries in the vector $(a_1, \ldots, a_i)$ are not linearly independent, $\Delta^i_{(a_1,\ldots,a_i)} f(x) = 0$.*

Lemma 3.4. *Let $deg(f)$ denote the algebraic degree of f. Then $deg(\Delta_a f(x)) \leq (deg(f(x)) - 1)$. Note that if f is the zero function, for which the degree is undefined in general but usually defined as $-\infty$, we may have to treat it as a special case - or at least avoid confusion by noting that that $-\infty \leq (-\infty - 1)$.)*

Corollary 3.5. *If $\Delta^i_{(a_1,\ldots,a_i)} f(x)$ is not a constant, then f has algebraic degree $> i$.*

(The above corollary is used in [12] as the basis of an algorithm that, given a Boolean function on multiple outputs (such as a block cipher) returns a lower bound for its algebraic degree.)

We now address the question of how to use this in cryptanalysis. Because any r-th derivative of a multiple-output Boolean function with algebraic degree r is a constant, a "higher-order differential" with probability 1, using chosen-plaintext structures each of size 2^r, is defined for any round function into which plaintext is input directly. (I.E we assemble the 2^r chosen inputs to the function specified by $\Delta^i_{(a_1,\ldots,a_r)} f(x)$, ensuring that all a_k are independent. The XOR of their outputs is a constant with probability 1, and we need to know beforehand what this constant is). If the decryption of the final rounds will allow us to tell if the sum of the outputs was the predicted constant, then we can carry out partial decryptions for the various TPS values, and eliminate any TPS for which the correct output XOR did not result.

Unfortunately, this does not scale very well as the number of rounds increases. For instance, although Knudsen is able to attack an arbitrary 5-round Feistel cipher in this fashion in [12], there seems no way to extend the attack as described to a Feistel cipher with six or more rounds.

To defeat higher-order differential attacks, cryptographers are advised to avoid using Boolean functions of low algebraic degree as S-boxes or round functions.

3.3 Multiplicative and hybrid differentials

Multiplicative differential cryptanalysis [9] works with pairs $(x, x' = ax)$ - so we multiply x by a, instead of xoring it with a bitstring ΔX, to obtain x'.

Prior to [9]'s publication, several ciphers such as IDEA [15], Nimbus [16], xmx [17] and MultiSwap [19] utilised scalar multiplication modulo some value m. There were various reasons for this:

- IDEA's designers believed that mixing different operations over different groups, which were "algebraically incompatible", would provide a high level of security. The other operations they used were XOR and addition, and to this day there has been no successful attack on the full cipher, just reduced-round variants.

- xmx's designers were attempting to produce a fast, compact cipher with as much cross-platform portability as possible. Like the designers of TEA [18] and Salsa20 [1], they believed that avoiding S-boxes and permutations in favour of simple operations that all processors would be able to carry out quickly would be the best way to do this.

- [9] points out that scalar multiplication is hard to attack with traditional differential cryptanalysis using pairs $(x, x + \Delta X)$ or $(x, x \oplus \Delta X)$.

Borisov, Chew, Johnson and Wagner exploited the fact that m was typically equal to $2^{\text{size of block}}$ or $2^{\text{size of block}} - 1$) to come up with ways to use multiplicative differentials in the cryptanalysis of such schemes. They also looked at generalisations of this - for example the inputs might be values mod m, but the outputs might only be mod q for some $q < m$.

The key to multiplicative differential cryptanalysis is that we are still representing the numbers involved as z-strings of bits, and the modulus used is often either the largest value such a bitstring can represent or that value + 1. This leads to relationships such as the following, which can be exploited by cryptanalysts:

- Where $m = 2^l - 1$, $-x \bmod m = (x \oplus 11 \ldots 1) = (x \oplus n)$.
- Where $m = 2^l$, $(2^k)x \bmod m = (x \ll k)$.
- Again, where $m = 2^l$, reversing the bits transforms $(x, 2x)$ to $(x, x/2)$.
- Where $m = 2^l$ and x is odd, $-x \bmod m = (x \oplus 11 \ldots 10) = (x \oplus (n - 2))$.

Multiplicative differential cryptanalysis also makes use of the following result:

Lemma 3.6. *The bitstring representation of any positive integer m can be expressed as a sequence of strings of the form (111 . . . 1) or (100 . . . 0). For instance, 30777 = 111100000111001 = (111, 100000, 11, 100, 1).*

Interestingly, [9] also demonstrates a *truncated* multiplicative differential!

3.3.1 Hybrid differentials

Where the blocks on which a cipher operates are split into sub-blocks, we may want to use different types of differential on different sub-blocks. [9] presented an example where a 64-bit block *abcd* was split into four 16-bit sub-blocks (a, b, c, d), a multiplicative differential applied to a and d, and a conventional differential to b and c. This resulted in the chosen input pair $(a, b, c, d), (a' = a \times k, b' = b \oplus 5, c' = c \oplus 5, d' = d \times k)$.

[9] referred to these as "hybrid differentials".

References

[1] D.J. Bernstein. Salsa20 design. `http://cr.yp.to/snuffle/design.pdf`.

[2] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In M. Wiener, editor, *Advances in Cryptology - Crypto '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 12–23. IACR, Springer, August 1999.

[3] E. Biham, A. Biryukov, and A. Shamir. Miss in the middle attacks on IDEA and Khufu. In L.R. Knudsen, editor, *Proceedings of the Sixth International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. IACR, Springer, March 1999.

[4] E. Biham, O. Dunkelman, and N. Keller. The rectangle attack - rectangling the Serpent. In B. Pfitzmann, editor, *Advances in Cryptology - Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. IACR, Springer, 2001.

[5] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. Technical Report CS90-16, Weizmann Institute of Science, July 1990. `http://www.cs.technion.ac.il/~biham/Reports/Weizmann/cs90-16.ps.gz`.

[6] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems (extended abstract). In A.J. Menezes and S.A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. IACR, Springer, 1990.

[7] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In E.F. Brickell, editor, *Advances in Cryptology - Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. IACR, Springer, 1992.

[8] A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 85–99. IACR, Springer, May 1998.

[9] N. Borisov, M. Chew, R. Johnson, and D. Wagner. Multiplicative differentials. In J. Daemen and V. Rijmen, editors, *Proceedings of the Ninth International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 17–33. IACR, Springer, February 2002.

[10] H.M. Heys. A tutorial on linear and differential cryptanalysis. Technical Report CORR 2001-17, University of Waterloo, March 2001. Available online, with errata, at `http://www.engr.mun.ca/~howard/Research/Papers/index.html`.

[11] J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In B. Schneier, editor, *Proceedings of the Seventh International Workshop on Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. IACR, Springer, April 2000.

[12] L.R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Proceedings of the Second International Workshop on Fast Software Encryption (FSE 1994)*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. IACR, Springer, December 1994.

[13] L.R. Knudsen. DEAL - a 128-bit block cipher. Hosted on CiteSeerX, February, revised May 1998. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7982`.

[14] X. Lai. Higher order derivatives and differential cryptanalysis. In R.E. Blahut, D.J. Costello Jr, U. Maurer, and T. Mittelholzer, editors, *Communications and Cryptography - Two Sides of One Tapestry*, pages 227–233. Kluwer Academic Publishers, 1994. Scanned copy online at `http://cr.yp.to/cubeattacks.html`.

[15] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In I.B. Damgård, editor, *Advances in Cryptology - Eurocrypt '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. IACR, Springer, May 1990.

[16] A.W. Machado. The Nimbus cipher, October 2000. `https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/nimbus.zip`.

[17] D. M'Raïhi, D. Naccache, J. Stern, and S. Vaudenay. xmx - a firmware-oriented block cipher based on modular multiplications. In E. Biham, editor, *Proceedings of the Fourth International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 166–171. IACR, Springer, January 1997. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.3608`.

[18] R.M. Needham and D.J. Wheeler. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *Proceedings of the Second International Workshop on Fast Software Encryption (FSE 1994)*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. IACR, Springer, December 1994.

[19] Beale Screamer. Microsoft's digital rights management scheme - technical details, October 2001. `http://cryptome.org/beale-sci-crypt.htm`.

[20] D. Wagner. The boomerang attack. In L.R. Knudsen, editor, *Proceedings of the Sixth International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. IACR, Springer, March 1999.