Optimized Interpolation Attacks on LowMC

Itai Dinur[1(✉)], Yunwen Liu[2], Willi Meier[3], and Qingju Wang[2,4]

[1] Département d'Informatique, École Normale Supérieure, Paris, France
dinur@di.ens.fr

[2] Department of Electrical Engineering,
ESAT/COSIC, KU Leuven and iMinds, Leuven, Belgium

[3] FHNW, Windisch, Switzerland

[4] Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai, China

Abstract. LowMC is a collection of block cipher families introduced at Eurocrypt 2015 by Albrecht et al. Its design is optimized for instantiations of multi-party computation, fully homomorphic encryption, and zero-knowledge proofs. A unique feature of LowMC is that its internal affine layers are chosen at random, and thus each block cipher family contains a huge number of instances. The Eurocrypt paper proposed two specific block cipher families of LowMC, having 80-bit and 128-bit keys.

In this paper, we mount interpolation attacks (algebraic attacks introduced by Jakobsen and Knudsen) on LowMC, and show that a practically significant fraction of 2^{-38} of its 80-bit key instances could be broken 2^{23} times faster than exhaustive search. Moreover, essentially all instances that are claimed to provide 128-bit security could be broken about 1000 times faster. In order to obtain these results we optimize the interpolation attack using several new techniques. In particular, we present an algorithm that combines two main variants of the interpolation attack, and results in an attack which is more efficient than each one.

Keywords: Block cipher · LowMC · High-order differential cryptanalysis · Interpolation attack

1 Introduction

LowMC is a collection of block cipher families designed by Albrecht et al. and presented at Eurocrypt 2015. The cipher is specifically optimized for practical instantiations of multi-party computation, fully homomorphic encryption, and zero-knowledge proofs. In such applications, non-linear operations result in a heavy computational penalty compared to linear ones. The designers of LowMC

Q.Wang—The fourth author is in part supported by the National Natural Science Foundation of China (no. 61472250), Major State Basic Research Development Program (973 Plan, no. 2013CB338004).

took an extreme approach, combining very dense affine layers with simple non-linear layers that have algebraic degree of 2.

Perhaps the most distinctive feature of LowMC is that its affine layers are chosen at random, and thus each block cipher family contains a huge number of instances. As this may enable a malicious party to instantiate LowMC with a hidden backdoor, its designers propose to use the Grain stream cipher [3] as a source of pseudo-random bits in order to restrict the freedom available in the LowMC instantiation. The designers also mention that it is possible to use any sufficiently random source to generate the affine layers, and this source does not necessarily need to be cryptographically secure.

The Eurocrypt paper proposed two specific block cipher families of LowMC, having 80-bit and 128-bit keys. The internal number of rounds in each family was set in order to guarantee a security level that corresponds to its key size. For this purpose, the resistance of LowMC was evaluated against a variety of well-known cryptanalytic attacks. One of the main considerations in setting the internal number of rounds was to provide resistance against algebraic attacks (such as high-order differential cryptanalysis [7]). Indeed, LowMC is potentially susceptible to algebraic attacks due to the low algebraic degree of its internal round, but the designers argue that LowMC has sufficiently many rounds to resist such attacks.

In this paper, we evaluate the resistance of LowMC against algebraic attacks and refute the designers' claims regarding its security level. Our results are given in Table 1, and show that a fraction of 2^{-38} of the LowMC 80-bit key instances could be broken in about 2^{57} time, using 2^{39} chosen plaintexts. The probability of 2^{-38} is practically significant, namely, a malicious party can easily find weak instances of LowMC by running its source of pseudo-random bits with sufficiently many seeds, and checking whether the resultant instance is weak (which can be done efficiently using basic linear algebra).

For LowMC with 128-bit keys, we describe an attack that breaks a fraction of 2^{-122} of its instances in time 2^{86} using 2^{70} chosen plaintexts. We note that this specific attack does not violate the formal security claims of the LowMC designers, as they do not consider attacks that apply to less than 2^{-100} of the instances as valid. Nevertheless, the designers of LowMC allow to instantiate it using a pseudo-random source that is not cryptographically secure. Our result shows that this is risky, as using an over-simplified source for pseudo-randomness may give a malicious party additional control over the LowMC instantiation, and allow finding weak instances much faster than exhaustively searching for them in 2^{122} time.

Finally, we describe an attack that can break essentially all LowMC instances with 128-bit keys. Although the attack is significantly slower than the weak-instance attack, it is still about 1000 times faster than exhaustive search, and uses 2^{73} chosen plaintexts.

All of our results were obtained using the interpolation attack, which is an algebraic attack introduced by Jakobsen and Knudsen in 1997 [4]. In an interpolation attack, the attacker considers some intermediate encryption value b as

Table 1. Attacks on LowMC

Instance Family	Number of Rounds	Section	Rounds Attacked	Fraction of Instances	Data[†]	Time[††]	Memory[†††]
LowMC-80	11	6.1	9	1	2^{35}	2^{38}	2^{35}
		6.2	10	1	2^{39}	2^{57}	2^{39}
		6.3	all (11)	2^{-38}	2^{39}	2^{57}	2^{39}
LowMC-128	12	7.1	11	1	2^{70}	2^{86}	2^{70}
		7.1	all (12)	2^{-122}	2^{70}	2^{86}	2^{70}
		7.2	all (12)	1	2^{73}	2^{118}	2^{80}

[†] Given in chosen plaintexts.
[††] Given in LowMC encryptions.
[†††] Given in 256-bit words.

a polynomial in the ciphertext bits. The aim of the attacker is to interpolate the algebraic normal form (ANF) of b by recovering its unknown coefficients, and this typically allows to recover the secret key using ad-hoc techniques.

In order to recover the unknown coefficients, the attacker allocates a variable for each one of them. Assuming that b has a low-degree representation in terms of the plaintext bits, the attacker collects linear equations on the variables, typically by using high-order differentials in a chosen plaintext attack. After obtaining sufficiently many equations, the unknown variables are recovered by solving the resultant linear equation system. The efficiency of the attack depends on the algebraic degree of b in terms of the plaintext, but also on the number of allocated variables which is determined by the number of unknown coefficients in the ANF representation of b in terms of the ciphertext.

Although our results were obtained using the well-known interpolation attack, its straightforward application does not seem to threaten the security of LowMC. Therefore, we had to develop new techniques such as using carefully chosen plaintext structures which allow to efficiently derive the linear system of equations. However, our main new contribution is described next by considering two variants of the interpolation attack.

In the original variant of the interpolation attack over $GF(2)$ (which we refer to as variant 1), the attacker views the ANF of some intermediate encryption bit b as an initially unknown polynomial $F_K(C)$ in the ciphertext bits $C = c_1, \ldots, c_n$, where $K = x_1, \ldots, x_\kappa$ is the unknown (fixed) secret key. In a dual approach to the interpolation attack, which we refer to as variant 2 (used, for example, in [8]), the attacker interpolates the full polynomial $F(K, C)$ by considering each monomial in the key bits $x_1, \ldots, x_\kappa$ with a non-zero coefficient as a separate (linearized) variable. For example, consider the polynomial

$$F(c_1, c_2, x_1, x_2, x_3) = c_1c_2x_1 + c_1c_2x_2 + c_1x_1 + c_1x_2 + c_2x_1 + x_1x_2 + x_3 + 1.$$

We can write

$$F_{(x_1,x_2,x_3)}(c_1, c_2) = \alpha_1c_1c_2 + \alpha_2c_1 + \alpha_3c_2 + \alpha_4,$$

and thus in the first variant we have 4 variables: $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. In this variant, the actual representation of the variables in terms of the key is not considered. In the dual variant, we write

$$F(c_1, c_2, x_1, x_2, x_3) = x_1x_2(1) + x_1(c_1c_2 + c_1 + c_2) + x_2(c_1c_2 + c_1) + x_3(1) + 1,$$

and we have 4 variables: x_1x_2, x_1, x_2, x_3.

The advantage of variant 2 over the first variant is that it directly recovers the secret key, and furthermore, in some cases it may result in a smaller number of variables in the equation system. At the same time, in order to derive the actual equation system the attacker has to evaluate the polynomial F for each ciphertext. This process is less efficient in variant 2, since each evaluation of $F(K, C)$ is expensive (it requires evaluating all the complex ciphertext expressions that are multiplied with the variables), whereas in variant 1 each evaluation of $F_K(C)$ is relatively simple (it requires evaluating simple monomials in the ciphertext). Therefore, the choice of which variant to use in order to optimize the attack depends on the underlying cryptosystem.

Our main idea is to combine the two dual variants of interpolations attacks: we first derive the equation system efficiently using the original variant of [4]. Then, we transform a carefully chosen variable subset to variables which are linearized monomials in the key bits, as in variant 2. This results in a mixed variable set that is smaller than the variable sets of each variant. Consequently, we obtain an attack which is more efficient than each one of the two variants.

In our example above, we can express $\alpha_1 = x_1 + x_2$, $\alpha_2 = x_1 + x_2$ and $\alpha_3 = x_1$, resulting in only 3 variables: x_1, x_2, α_4. Obviously, our toy example merely demonstrates the idea at a very high level, and the actual choice of which variables to transform as well as the analysis of the resultant algorithm are more involved.

The paper is organized as follows. In Sect. 2 we give some preliminaries, while in Sect. 3 we give a brief description of LowMC. Our basic attack on 9-round LowMC with an 80-bit key is described in Sect. 4, while our generic framework for optimized interpolation attacks is described in Sect. 5. In Sects. 6 and 7 we apply our optimized attack to LowMC with 80 and 128-bit keys, respectively. Finally, we conclude the paper in Sect. 8.

2 Preliminaries

In this section, we describe preliminaries that are used in the rest of the paper.

2.1 Boolean Algebra

For a finite set S, denote by $|S|$ its size. Given a vector $u = (u_1, \ldots, u_n) \in GF(2^n)$, let $wt(u)$ denote its Hamming weight.

Any function F from $GF(2^n)$ to $GF(2)$ can be described as a multivariate polynomial, whose algebraic normal form (ANF) is unique and given as

$F(x_1,\ldots,x_n) = \sum_{u=(u_1,\ldots,u_n)\in GF(2^n)} \alpha_u M_u$, where $\alpha_u \in \{0,1\}$ is the coefficient of the monomial $M_u = \prod_{i=1}^{n} x_i^{u_i}$, and the sum is over $GF(2)$. The algebraic degree of the function F is defined as $deg(F) \triangleq max\{wt(u)|\alpha_u \neq 0\}$. Therefore, a function F with a degree bounded by $d \leq n$ can be described using $\sum_{i=0}^{d} \binom{n}{i}$ coefficients. To simplify our notations, we define $\binom{n}{\leq d} \triangleq \sum_{i=0}^{d} \binom{n}{i}$.

The ANF coefficient α_u of F can be interpolated by summing (over $GF(2)$) over $2^{wt(u)}$ evaluations of F: define the set of inputs S to contain all the $2^{wt(u)}$ n-bit vectors whose bits set to 1 is a subset of the bits set to 1 in $u_1,\ldots,u_n$. More formally, let $S = \{x = (x_1,\ldots,x_n)|\bar{u} \wedge x = 0\}$ (where $\bar{u}$ is bitwise NOT applied to u, and $\wedge$ is bitwise AND), then $\alpha_u = \sum_{(x_1,\ldots,x_n)\in S} F(x_1,\ldots,x_n)$. Note that this implies that a function F with a degree bounded by $d \leq n$ can be fully interpolated given its evaluations on the set of $\binom{n}{\leq d}$ inputs whose Hamming weight is at most d, namely $\{x = (x_1,\ldots,x_n)|wt(x) \leq d\}$.

Given the truth table of an arbitrary function F (as a bit vector of 2^n entries), the ANF of F can be represented as a bit vector of 2^n entries, corresponding to its 2^n coefficients α_u. This ANF representation can be efficiently computed using the *Moebius transform*, which is an FFT-like algorithm. The Moebius transform performs n iterations on its input vector (the truth table of F), where in each iteration, half of the array entries are XORed into the other half. In total, its complexity is about $n \cdot 2^n$ bit operations. For more details on the Moebius transform, refer to [5].

2.2 High-Order Differential Cryptanalysis and Interpolation Attacks

In this section, we give a brief summary of high-order differential cryptanalysis and interpolation attacks.

High-Order Differential Cryptanalysis. High-order differential cryptanalysis was introduced in [7] as an algebraic attack that is particularly efficient against ciphers of low algebraic degree. The basic variant of high-order differential cryptanalysis over $GF(2)$ considers some target bit b (which can be either a ciphertext or an intermediate encryption value) and analyzes its ANF representation in terms of the plaintext P, denoted by $F_K(P)$ (where K is the unknown secret key). Given that $deg(F_K(P)) \leq dg$ independently of K for dg (relatively) small, then the attacker chooses an arbitrary linear subspace S of dimension $dg + 1$, and evaluates the cipher (in a chosen plaintext attack) over its 2^{dg+1} inputs. Since every differentiation reduces the algebraic degree of the target bit by 1 and $deg(F_K(P)) \leq dg$, the value of the high-order differential over S for the target bit b (namely, the sum of evaluations of b over $GF(2)$) is equal to zero (refer to [7] for details). High-order differential properties may be used in key recovery attacks, depending on the specification of the cipher (refer to [6]). However, such key recovery methods are not part of the framework described in this section.

Interpolation Attacks. The interpolation attack was introduced in 1997 by Jakobsen and Knudsen as an algebraic attack on block ciphers [4]. The attack is closely related to high-order differential cryptanalysis[1] and (similarly to high-order differential cryptanalysis) is particularly efficient against block ciphers whose round function is of low algebraic degree. The interpolation attack has several variants, and can be applied over a general finite field, exploiting known or chosen plaintexts. Here, we give a high-level description of the chosen plaintext interpolation attack over $GF(2)$, as this is the variant we apply to LowMC.

The attack considers some intermediate encryption target bit b of the block cipher, whose ANF representation can be expressed from the decryption side in terms of the ciphertext and key as $F(C, K)$. The key K is viewed as an unknown constant, and thus we can write $F_K(C) = F_K(c_1, \ldots, c_n) = \sum\limits_{u=(u_1,\ldots,u_n)\in GF(2^n)} \alpha_u M_u$, where $\alpha_u \in \{0,1\}$ is the coefficient of the monomial $M_u = \prod\limits_{i=1}^{n} c_i^{u_i}$. Therefore, the coefficients α_u of $F_K(C)$ generally depend on the secret key and are unknown in advance. The goal of the interpolation attack is to recover (interpolate) the unknown coefficients of $F_K(C)$, and then use various ad-hoc techniques (which are not part of the framework described in this section) in order to recover the actual secret key.

In order to deduce the unknown coefficients of $F_K(C)$, they are considered as variables (i.e., linearized), and recovered by solving a linear equation system. For the purpose of constructing the equation system, the attacker assumes that the algebraic degree dg of the bit b in terms of the bits of the plaintext is relatively small, which allows to use high-order differential cryptanalysis (as described above). More specifically, a high-order differential property is devised by encrypting a subspace S of plaintexts of dimension $dg + 1$, and performing high-order differentiation with respect to this subspace, whose outcome is zero on the bit b.

When expressed in terms of the ciphertexts $C_1, \ldots, C_{2^{dg+1}}$ (obtained by encrypting the plaintexts of S), this gives the equation $\sum\limits_{t=1}^{2^{d+1}} F_K(C_t) = 0$. For each ciphertext C_t, $F_K(C_t)$ is merely a linear expression in the variables α_u (the coefficient of α_u in this expression is easily deduced by evaluating M_u on C_t), and thus the subspace S gives rise to one linear equation in the variables α_u. In order to solve for the unknown variables α_u, the attacker considers several such subspaces, each giving one equation. In total, the number of equations (and

[1] In fact, some of its variants directly exploit high-order differential properties, as we describe next.

subspaces considered) needs to be roughly equal to the number of the unknown α_u variables, assuming the equations are sufficiently "random".

From the high-level description above, it is easy to conclude that the data and time complexities of the attack depend on the value of the degree dg and the number of unknown variables α_u. Therefore, in order to mount efficient interpolation attacks, the attacker tries to minimize these parameters, as we demonstrate in our attacks on LowMC.

2.3 Model of Computation

Since an exhaustive key search attack (which evaluates the LowMC encryption function) and our attacks use different bitwise operations, comparing these attacks cannot be done simply by counting the number of encryption function evaluations. Instead, we compare the complexity of straight-line implementations of the algorithms, counting the number of bit operations (such as XOR, AND, OR) on pairs of bits. This computation model ignores operations such as moving a bit from one position to another (which only requires renaming variables in straight-line programs). As calculated in Sect. 3, the straight-line implementation of one encryption function evaluation of LowMC requires about 2^{19} bit operations. Consequently, a straight-line implementation of exhaustive search for 80-bit and 128-bit keys requires about 2^{99} and 2^{147} bit operations, respectively, and these are quantities of reference for our attacks.

3 Description of LowMC

LowMC is a collection of SP-network instances, proposed at Eurocrypt 2015 [1] by Albrecht et al. The specification defined two specific instance families which are analyzed in this paper, both having a block size of $n = 256$ bits, and are characterized by their key size κ, which is either 80 or 128 bits. In this paper, we refer to these instance families as LowMC-80 and LowMC-128. The encryption function of LowMC applies a sequence of rounds to the plaintext, where each round contains a (bitwise) round-key addition layer, an Sbox layer, and an affine layer (over $GF(2)$). LowMC was designed with distinct features (as detailed in the pseudocode below): it has a linear key schedule and its affine layers are selected at random, where each selection defines a separate instance of the family. The Sbox layer of LowMC is composed of 3-bit Sboxes with degree 2 over $GF(2)$ (the actual specification of the Sboxes is irrelevant for our analysis and is omitted from this paper). Furthermore, the Sbox layers are only partial, namely, in each Sbox layer, only $3m < n$ bits go through an Sbox (where m is a parameter), while the rest of the $n - 3m$ bits remain unchanged.

Each family instance of LowMC is also defined with a data limit lim, which determines the maximal (recommended) data complexity before changing the key. In other words, the cipher is guaranteed to offer security according to its key size as long as the adversary cannot obtain more than 2^{lim} plaintext-ciphertext pairs. The parameters of the two instance families are given in Table 2.

Table 2. LowMC instance families

Instance Family	key size κ	Block Size n	Sboxes m	Data lim	Rounds r
LowMC-80	80	256	49	64	11
LowMC-128	128	256	63	128	12

The pseudocode of the encryption function (taken from [1]) is given below.

```
ciphertext = encrypt (plaintext,key)
  //initial whitening
  state = plaintext + MultiplyWithGF2Matrix(KMatrix(0),key)
  for (i = 1 to r)
    //m computations of 3-bit Sbox, n-3m bits remain the same
    state = Sboxlayer (state)
    //affine layer
    state = MultiplyWithGF2Matrix(LMatrix(i),state)
    state = state + Constants(i)
    //generate round key and add to the state
    state = state + MultiplyWithGF2Matrix(KMatrix(i),state)
  end
  ciphertext = state
```

The matrices $LMatrix(i)$ are chosen at random from all invertible binary $n \times n$ matrices, while the matrices $KMatrix(i)$ are chosen independently and uniformly at random from all binary $n \times \kappa$ matrices of rank $min(n, \kappa)$. The constants $Constants(i)$ are chosen independently and uniformly at random from all binary vectors of length n.

In this paper, we denote the 256-bit state at the input to the i'th key addition layer by X_{i-1} (e.g., the plaintext is denoted X_0), the input to the i'th Sbox layer by Y_{i-1} and the input to the i'th affine layer by Z_{i-1}. We refer to the $3m$ bits of the state that go through Sboxes in the Sbox layer as the S-part, while the remaining $n - 3m$ bits are referred to as the I-part. Given a state W, denote by $W|SP$ and $W|IP$ the S-part and I-parts of the state, respectively (e.g., $Y_5|IP$ is the I-part of the input state to the 6'th Sbox layer).

It is common practice in cryptanalysis of block ciphers to exchange the order of the final two affine operations over $GF(2)$ (namely, the keyless affine transformation and key addition). This allows the attacker to "peel off" the last affine transformation at a negligible cost by working with an equivalent last-round key (obtained by an affine transformation on the original last-round key). For the sake of simplicity, we assume in the following that we have already "peeled off" the last affine transformation of the cipher. Therefore, the final states of the last round r are denoted by X_{r-1}, Y_{r-1}, Z_{r-1} and Y_r, which denotes the ciphertext (after "peeling off" the final affine transformation).

Each affine layer of LowMC involves multiplication of the 256 state with a 256×256 matrix. This multiplication requires roughly 2^{16} bit operations, and therefore a single encryption of LowMC (that contains more than 8 rounds) requires more than $2^{16} \cdot 8 = 2^{19}$ bit operations (as already noted in Sect. 2.3).

4 A Basic 9-Round Attack on LowMC-80

In this section we describe our basic interpolation attack on 9-round LowMC, which is given first without optimizations for the sake of clarity. We begin by considering the elements that are required for the attack.

4.1 The High-Order Differential Property

We construct the high-order differential property used in the interpolation attack. A similar property was described by the LowMC designers [1], but we reiterate it here for the sake of completeness.

The algebraic degree of a single round of LowMC-80 over $GF(2)$ is 2, and therefore the algebraic degree of any bit at the input to the 6'th Sbox layer of LowMC-80, Y_5, in the input bits, X_0, is at most 32. Moreover, as the bits of the I-part of LowMC do not go through Sboxes in the first round, then the degree at the input to the 7'th Sbox layer, Y_6, in the bits of the I-part, $X_0|IP$, (given that the input bits of the S-part, $X_0|SP$, are constant) is at most 32. Furthermore, since the bits of the I-part of the 7'th Sbox layer do not go through an Sbox, the degree of any bit of $Z_6|IP$ in the input bits of the I-part, $X_0|IP$, is at most 32 (given that $X_0|SP$ is constant).

The last property implies that the value of a 33-order differential over any 33-dimensional subspace selected from $X_0|IP$, (keeping $X_0|SP$ constant) is zero for any bit of $Z_6|IP$. Moreover, as we selected a subspace whose bits do not go through an Sbox in the first round, the value of a 32-order differential for any bit of $Z_6|IP$ over any 32-dimensional subspace from $X_0|IP$, is a constant (independent of the key). This observation implies that we can select several 32-dimensional subspaces, and compute in a preprocessing phase the constants obtained by summing (over $GF(2)$) over a target bit of $Z_6|IP$ (for an arbitrary fixed value of the key). Each such constant (derived from a 32-dimensional subspace) gives one bit of information that we will exploit as the constant value of an equation in the interpolation attack.

4.2 Bounding the Number of Variables

In the interpolation attack on 9-round LowMC-80, we select a target bit from $Z_6|IP$ and denote its ANF representation in the 256-bit ciphertext (obtained after inverting the final affine transformation) and 80-bit key by $F(C, K)$. We consider K as an unknown constant, and write $F_K(C) = F_K(c_1, \ldots, c_{256}) = \sum\limits_{u=(u_1,\ldots,u_{256}) \in GF(2^{256})} \alpha_u M_u$, where $\alpha_u \in \{0, 1\}$ is the coefficient of the monomial $M_u = \prod\limits_{i=1}^{256} c_i^{u_i}$. As the complexity of the attack depends on the number of variables α_u, it is important to estimate their number with good accuracy. An initial estimation can be made by observing that the algebraic degree of the (inverse) round of LowMC-80 is 2,[2] and thus $deg(F_K(C)) \leq 4$. This implies that $\alpha_u = 0$

[2] The algebraic degree of any invertible 3-bit Sbox is (at most) 2.

in case $wt(u) > 4$, and therefore the number of unknown variables is upper bounded by $\binom{256}{\leq 4} \approx 2^{27}$.

The initial upper bound on the number of variables can be significantly improved by considering the specific round function of LowMC-80. For this purpose, it will be convenient to use additional notation to describe the variables α_u according to the degree of M_u, by defining the set of variables U_i for a positive integer i as $U_i = \{\alpha_u$ that is not identically zero as a function of the key$|wt(u) = i \bigwedge u \in GF(2^{256})\}$. We have already seen that U_i is empty for $i > 4$ (as these variables are identically zero independently of the key), and we now derive tighter bounds on $|U_i|$ for $i \leq 4$. Thus, we analyze the symbolic representation of the state variables in the decryption direction, starting from the ciphertext Y_9, up to Z_6, as polynomials in the ciphertext bits $c_1, \ldots, c_{256}$.

The ciphertext Y_9 contains 256 bits of $c_1, \ldots, c_{256}$, while in order to compute Z_8 we merely add (unknown) constants to these bits (recall that we "peeled off" the last affine layer). Then, the inverse Sbox layer is applied to Z_8 to obtain the state Y_8. Each 3-bit Sbox may contribute (up to) 3 quadratic monomials to Y_8, and 6 monomials in total, e.g., an Sbox corresponding to ciphertext bits c_1, c_2, c_3 may contribute the monomials $c_1, c_2, c_3, c_1c_2, c_1c_3, c_2c_3$. Note that these monomials may appear in the ANF of different bits of Y_8 with different unknown coefficients (e.g., c_1x_1 and c_1x_2 may appear in the ANF of two different bits of Y_8). However, in interpolation attacks, we consider the ANF of the target bit, in which the coefficient α_u of every monomial M_u in the ciphertext is linearized and considered as a single variable. Therefore, the important quantity is the number of possibilities to create the monomials M_u (for this reason, the monomial c_1 is counted only once even if it appears in the ANF of different bits of Y_8 with different unknown coefficients).

Since there are 49 Sboxes, the total number of monomials M_u in the ANF of the state bits of Y_8 is bounded by $|U_2| \leq 3 \cdot 49 = 147$, $|U_1| \leq 256$ (which is the trivial bound) and $|U_i| = 0$ for $i \geq 3$. As the affine and key addition mappings do not influence the number of monomials M_u, this bound applies also to X_8 and Z_7.

Next, the inverse Sbox layer is applied to Z_7 to obtain the state Y_7, for which we already know that $|U_i| = 0$ for $i > 4$. Since the Sbox layer is of degree 2, a trivial upper bound on the number of variables α_u in Y_7 is obtained by multiplying the $147 + 256 = 403$ monomials in unordered pairs, giving $|\bigcup_{i=1}^{4} U_i| \leq \binom{403}{2} + 403 < 2^{16.5}$. Since the key addition and affine layers do not influence the number of monomials, the upper bound of $2^{16.5}$ also applies to X_7 and Z_6, and it is much smaller than our initial bound of about 2^{27}.

We denote the set of variables $\bigcup_{i=1}^{4} U_i$ by U, and note that the explicit set $\{u|\alpha_u \in U\}$ (which gives the relevant monomials M_u) can be easily derived during preprocessing (which involves a more explicit computation of the monomial set $\{M_u|\alpha_u \in U\}$, whose size is bounded above).

4.3 Obtaining the Data

After deducing that the number of variables in the system of equations is $|U| \approx 2^{16.5}$, we conclude that we need to differentiate over about $2^{16.5}$ 32-dimensional subspaces in order to obtain sufficiently many equations to solve the system. A trivial way to do this is to select about $2^{16.5}$ arbitrary linearly independent 32-dimensional subspaces from the $256 - 3 \cdot 49 = 109$ bits of $X_0|IP$. This results in an attack with data complexity of $2^{32+16.5} = 2^{48.5}$, and is rather wasteful. A more efficient approach (which was previously used in various papers such as [2]), is to select a large 37-dimensional subspace S from $X_0|IP$, containing $\binom{37}{32} > 2^{18}$ linearly independent 32-dimensional subspaces, which should suffice for the attack (assuming that the constructed system of equations is sufficiently random). The subspaces are indexed according to $37 - 32 = 5$ constant indexes that are set to zero in S.

4.4 The Basic Interpolation Attack

We now describe a basic interpolation attack on 9-round LowMC-80. We note that this attack is incomplete, as it only computes the $|U|$ variables α_u using $e \approx |U|$ equations, without recovering the actual secret key. The details of this final step will be given in the optimized attack in Sect. 5.2. For the sake of convenience, we describe the attack in two phases: the preprocessing phase (which is independent of the data and secret key) and online phase. However, we take into account both phases in the total complexity evaluation.

Assume we selected a target bit b from $Z_6|IP$, a subspace S of dimension 37 from $X_0|IP$, and $e \approx |U|$ 32-dimensional subspaces $S_1, \ldots, S_e$ in S. The detailed attack is described below.

Preprocessing:

1. Compute an e-bit array of free coefficients for $e \approx |U|$ equations, denoted by $\boldsymbol{a_0}$: evaluate b on the subset of inputs of S (with the key set to zero), and obtain a bit array of size 2^{37}. Finally, calculate the free coefficients by summing on b for the e 32-dimensional subspaces $S_1, \ldots, S_e$ in S, and store the result in $\boldsymbol{a_0}$.
2. Calculate the $|U|$ vectors $\{u|\alpha_u \in U\}$: This can be done by first calculating the 403 monomials M_u past the first Sbox layer, and multiplying them in pairs (as described in Sect. 4.2).

Online:

1. Ask for the encryptions of the 2^{37} plaintexts in S and store the ciphertexts in a table.

2. Allocate a $2^{37} \times |U|$ matrix A, where row $A[t]$ is a bit array that represents the evaluation $F_K(C_t)$ (namely, $\sum\limits_{\{u|\alpha_u \in U\}} \alpha_u M_u(C_t)$).
3. For each ciphertext C_t, calculate $A[t]$ by evaluating $F_K(C_t)$:
   (a) For each $\{u|\alpha_u \in U\}$, evaluate the monomial $M_u(C_t)$ (the coefficient of α_u) and set the corresponding bit entry in $A[t]$ according to the result.
4. Allocate an $e \times |U|$ matrix E over $GF(2)$, representing the equation system on U.
5. For each 32-dimensional subspace S_j in S, namely $S_1, \ldots, S_e$ (that match the subspaces considered in preprocessing Step 1):
   (a) Populate the row (equation) $E[j]$ by summing over the 2^{32} rows of A corresponding to S_j.
6. Solve the equation system $E\boldsymbol{x} = \boldsymbol{a_0}$, where $\boldsymbol{x}$ represents the vector of variables of U and $\boldsymbol{a_0}$ is the vector of free coefficients calculated in preprocessing Step 1.

The data complexity of the attack is 2^{37} chosen plaintexts. The total time complexity of the attack is about 2^{65} bit operations, dominated by online Step 5 (for each of the e subspaces, we sum over 2^{32} bit vectors of size $|U|$, requiring about $e \cdot 2^{32} \cdot |U| \approx 2^{65}$ bit operations). The memory complexity of the attack is about $2^{37} \cdot |U| \approx 2^{53.5}$ bits, dominated by the storage of the matrix A in online Step 2.

We note that in the complexity evaluation of the attack we ignore indexing issues that arise (for example) in Step 3.a (that maps between a variable $\alpha_u \in U$ and its corresponding column index in $A[t]$), and in Step 5 (that maps between a subspace S_j in S and the corresponding 5 constant indexes of S). The reason that we can ignore these mappings in the complexity evaluation is that they are independent of the secret key and data, and therefore, they can be precomputed and integrated into the straight-line implementation of the program.

5 The Optimized Interpolation Attack

In this section, we introduce three optimizations of the basic 9-round attack above. The first optimization reorders the steps of the algorithm in order to reduce the memory complexity, while the second optimization further exploits the structure of chosen plaintexts to reduce the time complexity of the attack. Finally the third optimization is based on a novel technique in interpolation attacks, and allows to (further) reduce the data and time complexities. We first describe informally how to apply the optimizations to the basic 9-round attack on LowMC-80 above, and then devise a more formal and generic framework that can be applied to other LowMC variants.

The first two optimizations focus on online steps 2–5, which compute the equation system E from the 2^{37} ciphertexts. First, we reduce the memory complexity by noticing that we do not need to allocate the matrix A. Instead, we work column-wise and focus on a single column $A[*][\ell]$ at a time, corresponding to some $\{u|\alpha_u \in U\}$. We evaluate $M_u(C_t)$ for all ciphertexts (which gives an array of 2^{37} bits, $\boldsymbol{a_\ell}$) and then populate the corresponding column $E[*][\ell]$ by summing over the 32-dimensional subspaces $S_1, \ldots, S_e$ on $\boldsymbol{a_\ell}$.

Next, we reduce the time complexity by optimizing the summation process: given a bit array $\boldsymbol{a_\ell}$ of 2^{37} entries, the goal is to sum over many 32-dimensional subspaces (indexed according to 5 bits which are set to zero). This can be done efficiently using the Moebius transform (refer to Sect. 2.1). For this purpose, we can view $\boldsymbol{a_\ell}$ as evaluating a 37-variable polynomial over $GF(2)$, and the summation over a 32-dimensional subspace of $\boldsymbol{a_\ell}$ is equal to the coefficient of its corresponding 32-degree monomial. All these coefficients are computed by the Moebius transform in about $37 \cdot 2^{37}$ bit operations. We stress that the reason that we can use the Moebius transform in this case is purely combinatorial and is due to the way that we selected the structure of subspaces for the interpolation attack. Indeed, there does not seem to be any obvious algebraic interpretation to $\boldsymbol{a_\ell}$ when viewed as a polynomial.

Finally, we optimize the data complexity (and further reduce the time complexity): In order to achieve this, examine the polynomial $F(K, C)$ (as a function of both the key and ciphertext) for the target bit b selected in $Z_6|IP$. Due to the linear key schedule of LowMC, this polynomial is of degree 4, similarly to $F_K(C)$ (in which the key is treated as a constant). We consider a variable $\alpha_u \in U$ and analyze its ANF in terms of the 80 key bit variables. Since α_u is multiplied with M_u in $F(K, C)$, then $deg(\alpha_u) + deg(M_u) \leq 4$, implying that if $deg(M_u) \geq 2$, then $deg(\alpha_u) \leq 2$. This simple observation is borrowed from cube attacks [2] and can be used to significantly reduce the number of variables U, as described next.

Consider all the variables in $U_2 \bigcup U_3 \bigcup U_4$, and recall that their number was upper-bounded in Sect. 4.2 by roughly $2^{16.5}$. However, since all of these variables are polynomials of degree (at most) 2 in the 80 key bits, they reside in a linear subspace of monomials of dimension $\binom{80}{2} + 80 = 3240$. This implies that we can significantly reduce the total number of variables from $\approx 2^{16.5}$ to $3240 + 256 = 3496 < 2^{12}$ (including the 256 variables of U_1) by considering linear relations between the variables $U_2 \bigcup U_3 \bigcup U_4$. An immediate consequence of the reduction of variables is that we need less equations to solve the equation system, and therefore, we require less subspaces (or data) to obtain these equations. More specifically, a subspace of dimension 35 contains $\binom{35}{32} = 6545 > 2^{12}$ subspaces of dimension 32, which should suffice for the attack.

Assuming that we interpolate the variables of $U_2 \bigcup U_3 \bigcup U_4$ in terms of the key and recover their values, then the key itself should be very easy to deduce, as the variables of U_3 are merely key bits.

We note that while the idea above exploits the linear key schedule of LowMC, the technique is general and can be applied to block ciphers with arbitrary key schedules. In this case, it would consider each round key as independent. This increases the number of variables in the (linearized) key, but not necessarily by a significant factor. For example, if LowMC-80 had a non-linear key schedule, the optimization above would interpolate $U_2 \bigcup U_3 \bigcup U_4$ in terms of $\binom{80}{2} + 80 = 3240$ monomials in the key of round 9, and only 80 additional linear monomials and $3 \cdot 49 = 294$ quadratic monomials in the key of round 8 that are created by the inverse Sbox layer of round 8 (we can assume that the key of round 8 is added right after the 8'th Sbox layer, as the key addition and affine layer are interchangeable).

5.1 Transformation of Variables

In this section, we begin to describe our generic framework for interpolation attacks on LowMC by formalizing the last optimization described above.

Given an instance of LowMC with a 256-bit block, a key size of κ, and m Sboxes per layer, we assume that we want to interpolate a target bit b through the final r_1 rounds of the cipher. We first describe in a more generic way how to calculate the initial set of variables U, and bound its size. As in the 9-round attack, the number of monomials in the 256 ciphertext bits at Y_{r-1} (after inverting the final Sbox layer) is bounded by $256+3m$. The target bit b is a polynomial of degree 2^{r_1-1} in the state Y_{r-1}, and thus it contains at most $\binom{256+3m}{\leq 2^{r_1-1}}$ monomials. Therefore, the set of monomials with (apriori) unknown coefficients can be computed by multiplying the $256+3m$ monomials in unordered tuples (with no repetition) of size up to 2^{r_1-1}. Thus,

$$|U| \leq \binom{256+3m}{\leq 2^{r_1-1}},$$

and this set can be computed with $|U|$ multiplications of tuples. Note again that this bound is generally better than the trivial bound of $|U| \leq \binom{256}{\leq 2^{r_1}}$, which is obtained due to the fact that b is a polynomial of degree 2^{r_1} in the 256 ciphertext bits.

We consider the target bit b as a polynomial in both the ciphertext and the key, namely, $F(K,C) = F(x_1,\ldots,x_\kappa,c_1,\ldots,c_{256}) = \sum\limits_{u=(u_1,\ldots,u_n)\in GF(2^n)} \alpha_u M_u$, where $M_u = \prod\limits_{i=1}^{n} c_i^{u_i}$ and $\alpha_u(x_1,\ldots,x_\kappa)$ is a polynomial from $GF(2^\kappa)$ to $GF(2)$. We partition the variables of $|U|$ into subsets according to the degree of their monomials in the ciphertext, which is bounded by $deg(F_K(C)) = 2^{r_1}$. Denote $d = 2^{r_1}$ and write $U = \bigcup\limits_{i=1}^{d} U_i$, where $U_i = \{\alpha_u \in U | deg(M_u) = i\}$. Due to the linear key schedule of LowMC, we have $deg(F(K,C)) = deg(F_K(C)) = d$, and therefore $deg(\alpha_u) + deg(M_u) \leq d$. This allows us to transform the variable set U into a smaller variable set, considering internal linear relations due to the fact that $deg(\alpha_u) \leq d - deg(M_u)$. We stress again that the variable transformation technique can be applied to block ciphers with arbitrary key schedules by considering each round key as independent.

We choose an integral *splitting index* $1 \leq sp \leq d{+}1$, and write $U = U' \bigcup U''$, where $U' = \bigcup_{i=1}^{sp-1} U_i$ and $U'' = \bigcup_{i=sp}^{d} U_i$. The observation above implies that the algebraic degree of the variables in U'' (in terms of the key) is bounded by $d - sp$, namely, $deg(\alpha_u) \leq d - sp$, for each $\alpha_u \in U''$. Therefore, we can interpolate each variable of U'' in terms of the key, and express it as $\alpha_u = \sum_{\{v=(v_1,\ldots,v_\kappa)|wt(v)\leq d-sp\}} \beta_u M_v$, where $\beta_v \in \{0,1\}$ is the coefficient of the monomial $M_v = \prod_{i=1}^{\kappa} x_i^{v_i}$. Note that the coefficients β_v are independent of the key and can be computed during preprocessing. This interpolation transforms the set of variables U'' into the set of variables V, which are low degree monomials in the key bits $V = \{M_v = \prod_{i=1}^{\kappa} x_i^{v_i} | v = (v_1, \ldots, v_\kappa) \wedge wt(v) \leq d - sp\}$. Similarly to the partition of U, we partition the variables of V into subsets according to the degree of their monomials in the key, namely $V_i = \{M_v \in V | deg(M_v) = i\}$. In addition, we define $V_{\leq i} = \bigcup_{j=1}^{i} V_i$. Note that $\alpha_u \in U_i$ is a linear combination of variables in $V_{\leq(d-i)}$.

Recall that our initial set of variables is expressed as $U = U' \bigcup U''$, where $U' = \bigcup_{i=1}^{sp-1} U_i$ and $U'' = \bigcup_{i=sp}^{d} U_i$. This set of variables is transformed via interpolation into a new set of variables $W = U' \bigcup V$.

We compute bounds on sizes of the variables sets as follows:

$$|U'| \leq \binom{256}{\leq sp-1}, \quad |V| \leq \binom{\kappa}{\leq d - sp},$$

$$|W| = |U'| + |V| \leq \binom{256}{\leq sp-1} + \binom{\kappa}{\leq d - sp}.$$

The Variable Transformation Algorithm. We now describe the algorithm which interpolates a variable $\alpha_u \in U_i$ in terms of the variable set $V_{\leq(d-i)}$. For the sake of efficiency, the algorithm is performed in two phases, where in the first phase, we evaluate the polynomial α_u in terms of the key for all relevant keys of low Hamming weight and store the results. Note that each evaluation of α_u requires summing on 2^i evaluations of the target bit b. In the second phase, we use the evaluations to interpolate α_u in terms of $V_{\leq(d-i)}$.

1. Allocate a bit array $\boldsymbol{a_1}$ of size $|V_{\leq(d-i)}|$ for the evaluations of α_u.
2. Evaluate α_u for each key with Hamming weight at most $d - i$. Namely, for each key in the set $\{K | wt(K) \leq d - i\}$:

```
    (a) Evaluate $F(K, C)$ (the target bit) on the subset of $2^i$ inputs (with
        the fixed key $K$) $\{K, C|\bar{u} \wedge C = 0\}$, sum the result over $GF(2)$, and
        store it in $\boldsymbol{a_1}$.
3. Allocate a bit array $\boldsymbol{a_2}$ of size $|V_{\leq(d-i)}|$ for interpolation of $\alpha_u$ in terms
   of $V_{\leq(d-i)}$.
4. For each $M_v \in V_{\leq(d-i)}$ (with index $\ell$), the coefficient $\beta_v$ of $M_v$ in $\alpha_u$ is
   calculated as follows:
    (a) Sum the $2^{wt(v)}$ values of $\boldsymbol{a_1}$ calculated for the subset of keys $\{K|\bar{v} \wedge
        K = 0\}$, and store the result in $\boldsymbol{a_2}[\ell]$.
```

The total number of evaluations of b in Step 2 is $2^i \cdot |V_{\leq(d-i)}|$, each requiring $r_1 \cdot 2^{16}$ bit operations. Therefore, the total complexity of this step is $r_1 \cdot 2^{16+i} \cdot |V_{\leq(d-i)}|$. Step 4 requires less than $|V_{\leq(d-i)}| \cdot 2^{d-i}$ bit operations. In total, the interpolation of $\alpha_u \in U_i$ requires $|V_{\leq(d-i)}| \cdot (r_1 \cdot 2^{16+i} + 2^{d-i})$ bit operations.

Since $U'' = \bigcup\limits_{i=sp}^{d} U_i$, we can write the complexity of interpolating all the variables as $\sum\limits_{i=sp}^{d} |U_i| \cdot |V_{\leq(d-i)}| \cdot (r_1 \cdot 2^{16+i} + 2^{d-i})$. A simple way to bound this complexity is

$$|U''| \cdot |V| \cdot (r_1 \cdot 2^{16+d} + 2^{d-sp}) \approx |U''| \cdot |V| \cdot r_1 \cdot 2^{16+d}.$$

In some cases, we can obtain a refined bound by writing the complexity as

$$|U_{sp}| \cdot |V_{\leq(d-sp)}| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + \sum_{i=sp+1}^{d} |U_i| \cdot |V_{\leq(d-i)}| \cdot (r_1 \cdot 2^{16+i} + 2^{d-i}) \leq$$

$$|U_{sp}| \cdot |V_{\leq(d-sp)}| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + |U''| \cdot |V_{\leq(d-sp-1)}| \cdot (r_1 \cdot 2^{16+d} + 2^{d-sp+1}) \approx$$

$$|U_{sp}| \cdot |V| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + |U''| \cdot |V_{\leq(d-sp-1)}| \cdot r_1 \cdot 2^{16+d}.$$

Note that the bound is potentially better than the trivial one of $|U''| \cdot |V| \cdot r_1 \cdot 2^{16+d}$ as $|U_{sp}| \leq \binom{256}{sp}$, which may be smaller than $|U''|$. Moreover $|V_{\leq(d-sp-1)}| \leq \binom{\kappa}{\leq d-sp-1}$, which is smaller than $|V|$.

Transformation of Equations. After computing the transformation of variables from U'' to V, we need to apply the actual transformation to every equation over U that we calculated. Namely, we are interested in transforming an equation over the variable set $U = U' \bigcup U''$, into an equation over variable set $W = U' \bigcup V$. Obviously, the coefficients of the variables of U' remain the same, and we need to apply the transformation for every variable $\alpha_u \in U''$.

The complexity of transforming a single variable $\alpha_u \in U_i$ in a single equation is simply equal to its number of coefficients over V, namely $|V_{\leq(d-i)}|$. Therefore, the complexity of transforming all the variables $\alpha_u \in U''$ in an equation is $\sum\limits_{i=sp}^{d} |U_i| \cdot |V_{\leq(d-i)}|$. A simple upper bound on this complexity is

$$|U''| \cdot |V|.$$

Similarly to the variable transformation algorithm, a refined upper bound can be calculated as

$$|U_{sp}| \cdot |V| + |U''| \cdot |V_{\leq(d-sp-1)}|.$$

In total, if we transform e equations, the complexity calculations above are multiplied by e.

Finally, we observe that the splitting index determines the complexity of the variable and equation transformation algorithms. Furthermore, the splitting index also determines $|W|$, which in turn determines the number of equations e. In general, we will choose sp in order to minimize $|W|$, which in turn minimizes the data and time complexity of the attack.

5.2 Details of the Optimized Interpolation Attack

Given an instance of LowMC with a 256-bit block, a key size of κ, and m Sboxes per layer, we interpolate a target bit b through the final r_1 rounds of the cipher. Let U, U', U'', V and W be as defined above, and let $e \approx |W|$ denote the number of equations. Assume S is a sufficiently large subspace of plaintexts, such that it contains e smaller subspaces $S_1, \ldots, S_e$ whose high-order differential on b is a constant value (independent of the key).

The preprocessing phase of the optimized attack in described below.

Preprocessing:

1. Compute an e-bit array of free coefficients for $e \approx |U'|$ equations, denoted by $\boldsymbol{a_0}$: evaluate b on the subset of inputs (plaintexts) of S (with the key set to zero), and obtain a bit array of size $|S|$. Then, calculate the free coefficients by applying the Moebius transform to the bit array, and copy the values of sums over $S_1, \ldots, S_e$ to $\boldsymbol{a_0}$.
2. Calculate the $|U|$ vectors $\{u|\alpha_u \in U\}$: This is done by first calculating the $256 + 3m$ monomials past the first Sbox layer, and multiplying them in unordered tuples (with no repetition) of size up to 2^{r_1-1}(as described in Sect. 5.1).

Step 1 involves $|S|$ evaluations of the encryption scheme and one application of the Moebius transform on a vector of size S. Altogether, it requires $|S| \cdot 2^{19} + \log(|S|) \cdot |S| \approx |S| \cdot 2^{19}$ bit operations (as $\log(|S|) \ll 2^{19}$). Step 2 requires

$|U|$ monomial multiplications, each monomial can be represented with a 256-bit array, and therefore this step requires $2^8 \cdot |U|$ bit operations.

A summary of the complexity analysis of the preprocessing phase is as follows.

Step 1: $2^{19} \cdot |S|$
Step 2: $2^8 \cdot |U|$

In terms of memory, Step 1 requires $|S|$ bits, while Step 2 requires $2^8 \cdot |U|$ bits.

Online:

1. Ask for the encryptions of the plaintexts in S and store the ciphertexts in a table.
2. Allocate a bit vector of size $|S|$ for the storage of the vectors $\boldsymbol{a_\ell}$ (the ℓ'th column of the matrix A in the basic attack).
3. Allocate an $e \times |W|$ matrix E over $GF(2)$, representing the (reduced) equation system on W. The matrix is vertically decomposed into two smaller matrices: E_1 of size $e \times |U'|$ and E_2 of size $e \times |V|$.
4. For each $\{M_u | \alpha_u \in U\}$ with an index ℓ:
   (a) For each ciphertext C_t, calculate $\boldsymbol{a_\ell[t]}$ by evaluating $M_u(C_t)$.
   (b) Use the Moebius transform to sum over all subspaces of $\boldsymbol{a_\ell}$.
   (c) If $\alpha_u \in U'$, populate column ℓ of E_1: For each subspace S_j in S, namely $S_1, \ldots, S_e$, obtain its corresponding sum from $\boldsymbol{a_\ell}$ and copy it to $E_1[j][\ell]$.
   (d) Otherwise, $\alpha_u \in U''$:
      i. Given that $\alpha_u \in U_i$, interpolate the coefficients of $V_{\leq(d-i)}$ in α_u as described in Sect. 5.1.
      ii. For each subspace S_j in S, obtain its corresponding boolean sum from $\boldsymbol{a_\ell}$ (the coefficient of α_u over U). If the sum is 1, then add (over $GF(2)$) the interpolated coefficients into their indexes in $E_2[j]$ (as described in Sect. 5.1).
5. Solve the equation system $E\boldsymbol{x} = \boldsymbol{a_0}$, where $\boldsymbol{x}$ represents the vector of variables of $W = U' \bigcup V$ and $\boldsymbol{a_0}$ is the vector of free coefficients calculated in preprocessing Step 1.
6. Deduce the κ-bit secret key, which is simply given by the monomials V_1 (namely, the monomials of degree 1 in V).

The complexity of Step 1 is $|S|$ encryptions, or $|S| \cdot 2^{19}$ bit operations. In Step 4, we iterate over $|U|$ monomials, where for each one we first evaluate $M_u(C_t)$ for each ciphertext in Step 4.a. Each such evaluation can be performed with d bit operations (as $deg(M_u) \leq d$), and thus monomial evaluations require about $d \cdot |S| \cdot |U|$ bit operations. Next, we apply the Moebius transform in Step 4.b, requiring about $\log(|S|) \cdot |S|$ bit operation, and therefore the complexity of all the transforms is about $\log(|S|) \cdot |S| \cdot |U|$. The complexity of interpolating all the variables in Step 4.d.i, is bounded in Sect. 5.1 by $|U''| \cdot |V| \cdot r_1 \cdot 2^{16+d}$. The complexity of Step 4.d.ii (over all $\alpha_u \in U''$) is bounded in Sect. 5.1 by $e \cdot |U''| \cdot |V| \approx |W| \cdot |U''| \cdot |V|$.

The complexity of Step 5 is $|W|^3$ bit operations using Gaussian elimination. A summary of the complexity analysis of the online phase is as follows. Since we generally do not have a good bound for $|U''|$, we simply replace it with $|U|$ (as $|U''| \leq |U|$), and further assume that $e \approx |W|$.

Step 1: $|S| \cdot 2^{19}$
Step 2: $|S|$
Step 3: $|W| \cdot |W|$
Step 4.a: $d \cdot |S| \cdot |U|$
Step 4.b: $\log(|S|) \cdot |S| \cdot |U|$
Step 4.c: $|U'| \cdot |W|$
Step 4.d.i: $|U| \cdot |V| \cdot r_1 \cdot 2^{16+d}$
Step 4.d.ii: $|W| \cdot |U| \cdot |V|$
Step 5: $|W|^3$
Step 6: negligible

Alternatively, we can use the refined complexity bounds for steps 4.d.i and 4.d.ii, as calculated in Sect. 5.1.

Step 4.d.i: $|U_{sp}| \cdot |V| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + |U| \cdot |V_{\leq(d-sp-1)}| \cdot r_1 \cdot 2^{16+d}$
Step 4.d.ii: $|W| \cdot (|U_{sp}| \cdot |V| + |U| \cdot |V_{\leq(d-sp-1)}|)$

The total data complexity of the algorithm is $|S|$ chosen plaintexts. The total time complexity is dominated by steps 4 and 5, as calculated above. The memory complexity is potentially dominated by a few steps: the storage of variables in preprocessing that requires $2^8 \cdot |U|$ bits, the storage of ciphertexts in Step 1 that requires $2^8 \cdot |S|$ bits, and the storage of E in Step 3 that requires $|W| \cdot |W|$ bits.

6 Optimized Interpolation Attacks on LowMC-80

In this section we apply the optimized interpolation attack on LowMC-80, for which $\kappa = 80$ and $m = 49$.

6.1 A 9-Round Attack

As in the basic attack described in Sect. 4.4, we select the target bit b in $Z_6|IP$, using subspaces of dimension 32 to obtain the equations. We interpolate through $r_1 = 2$ rounds, implying that $d = 2^{r_1} = 4$. Therefore $|U| = \binom{256+3m}{\leq 2^{r_1-1}} = \binom{403}{\leq 2} \approx 2^{16.5}$.

As described at the beginning of Sect. 5, we use $sp = 2$. We compute the size of the relevant variable sets $|U'| \leq \binom{256}{\leq sp-1} = \binom{256}{\leq 1} \approx 2^8$, $|V| \leq \binom{\kappa}{\leq d-sp} = \binom{80}{\leq 2} < 2^{12}$, $|W| = |U'| + |V| < 2^{12}$.

We choose a subspace S of dimension 35 from $X_0|IP$, containing $\binom{35}{32} > 2^{12} > |W|$ 32-dimensional subspaces, which should suffice for the attack.

In terms of time complexity, the analysis of the critical steps of the attack is as follows:

Step 4.a: $d \cdot |S| \cdot |U| \approx 4 \cdot 2^{35} \cdot 2^{16.5} = 2^{53.5}$
Step 4.b: $\log(|S|) \cdot |S| \cdot |U| \approx 35 \cdot 2^{35} \cdot 2^{16.5} = 2^{56.5}$
Step 4.c: $|U'| \cdot |W| \approx 2^8 \cdot 2^{12} = 2^{20}$
Step 4.d.i: $|U| \cdot |V| \cdot r_1 \cdot 2^{16+d} \approx 2^{16.5} \cdot 2^{12} \cdot 2 \cdot 2^{20} = 2^{49.5}$
Step 4.d.ii: $|W| \cdot |U| \cdot |V| \approx 2^{12} \cdot 2^{16.5} \cdot 2^{12} = 2^{40.5}$
Step 5: $|W|^3 \approx 2^{12 \cdot 3} = 2^{36}$

In total, the time complexity of the optimized 9-round attack is about 2^{57} bit operations (or $2^{57-19} = 2^{38}$ encryptions), mostly dominated by Step 4.b. The data complexity is 2^{35} chosen plaintexts. The memory complexity is dominated by the storage of ciphertexts in Step 1, and is about $|S| \cdot 2^8 = 2^{43}$ bits.

We note that while the improvement of the optimized attack compared to the basic one is rather moderate for the 9-round attack, the effect of our optimizations is more pronounced in the attacks described next, as the reduction in the number of variables becomes more significant (a comparison for the attack on full LowMC-128 is at the end of Sect. 7.2).

6.2 A 10-Round Attack

Similarly to the 9-round attack, in order to attack 10 rounds of LowMC-80, we select the target bit b in $Z_6|IP$, using subspaces of dimension 32 to obtain the equations. We interpolate through $r_1 = 3$ rounds, implying that $d = 2^{r_1} = 8$. Therefore $|U| = \binom{256+3m}{\leq 2^{r_1-1}} = \binom{403}{\leq 4} < 2^{30.5}$.

In this attack we use $sp = 4$, and compute the size of the relevant variable sets $|U'| \leq \binom{256}{\leq sp-1} = \binom{256}{\leq 3} \approx 2^{21.5}$, $|V| \leq \binom{\kappa}{\leq d-sp} = \binom{80}{\leq 4} < 2^{21}$, $|W| = |U'| + |V| < 2^{22.5}$. We use the refined analysis for steps 4.d.i and 4.d.ii, and thus we also calculate $|U_{sp}| = |U_4| = \binom{256}{4} < 2^{27.5}$ and $|V_{\leq (d-sp-1)}| = \binom{80}{\leq 3} < 2^{16.5}$.

We choose a subspace S of dimension 39 from $X_0|IP$, containing $\binom{39}{32} > 2^{23} > |W|$ 32-dimensional subspaces.

In terms of time complexity, the analysis of the critical steps of the attack is as follows (using the refined analysis for steps 4.d.i and 4.d.ii):

Step 4.a: $d \cdot |S| \cdot |U| \approx 8 \cdot 2^{39} \cdot 2^{30.5} = 2^{72.5}$
Step 4.b: $\log(|S|) \cdot |S| \cdot |U| \approx 39 \cdot 2^{39} \cdot 2^{30.5} \approx 2^{75}$
Step 4.c: $|U'| \cdot |W| \approx 2^{21.5} \cdot 2^{22.5} = 2^{44}$
Step 4.d.i: $|U_{sp}| \cdot |V| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + |U| \cdot |V_{\leq (d-sp-1)}| \cdot r_1 \cdot 2^{16+d} \approx 2^{27.5} \cdot 2^{21} \cdot (3 \cdot 2^{20} + 2^4) + 2^{30.5} \cdot 2^{16.5} \cdot 3 \cdot 2^{24} \approx 2^{70} + 2^{72.5} \approx 2^{73}$
Step 4.d.ii: $|W| \cdot (|U_{sp}| \cdot |V| + |U| \cdot |V_{\leq (d-sp-1)}|) \approx 2^{22.5} \cdot (2^{27.5} \cdot 2^{21} + 2^{30.5} \cdot 2^{16.5}) \approx 2^{22.5} \cdot (2^{48.5} + 2^{47}) \approx 2^{71.5}$
Step 5: $|W|^3 \approx 2^{22.5 \cdot 3} = 2^{67.5}$

In total, the time complexity of the optimized 10-round attack is about 2^{76} bit operations (or 2^{57} encryptions), mostly dominated by Step 4.b. The data complexity is 2^{39} chosen plaintexts. The memory complexity is dominated by the storage of ciphertexts in Step 1, and is about $2^8 \cdot |S| = 2^{47}$ bits (note that the storage of E requires $2^{22.5 \cdot 2} = 2^{45}$ bits).

6.3 An Attack on Full LowMC-80 for Weak Instances

The 9 and 10-round attacks described above can be extended by an additional round with negligible cost for a subset of weak instances containing a fraction of about 2^{-38} of all instances. In particular, this implies that about 2^{-38} of the instances of full 11-round LowMC-80 can be attacked significantly faster than exhaustive search.

Consider the 10-round attack: as shown above, we can construct an efficient high-order differential property for any choice of target bit of $Z_6|IP$, and also for any linear combination of the bits of $Z_6|IP$. When considering interpolation from the decryption side on a full 11-round instance, we can efficiently interpolate the polynomial $F_K(C)$ for any bit of $Z_7|IP$, or any linear combination of the bits of $Z_7|IP$. Assume that there exists a linear dependency between the 109 bits of $Z_6|IP$ and the 109 bits of $Z_7|IP$. In this case, the linear combination in terms of $Z_6|IP$ does not go through an Sbox in round 8. Therefore, it is possible to extend the high-order differential property on this linear combination by another round with essentially no extra cost, and choose the target bit for interpolation to be the corresponding linear combination on the bits of $Z_7|IP$. The existence of this linear dependency is determined by the affine layer of round 7 (the transformation between Z_6 and X_7), and assuming that random invertible matrices behave roughly the same (with respect to the event considered) as random matrices, the probability of this event is about $2^{109+109-256} = 2^{-38}$ (over the choice of the 7'th affine layer).

We note that there exists an additional subset of weak instances of about the same size since the described attacks can also be mounted using chosen ciphertexts (where interpolation is performed on the decrypted plaintexts). In this case, the weakness of a given instance is determined by the choice of the third affine layer.

7 Optimized Interpolation Attacks on LowMC-128

In this section we apply the optimized interpolation attack on LowMC-128, for which $\kappa = 128$ and $m = 63$.

7.1 An 11-Round Attack and Weak Instances of LowMC-128

We describe our attack on 11-round LowMC-128 and then extend it to full LowMC-128 for weak instances. We select the target bit b in $Z_7|IP$, and interpolate through $r_1 = 3$ rounds, implying that $d = 2^{r_1} = 8$. Therefore $|U| = \binom{256+3m}{\leq 2^{r_1-1}} = \binom{445}{\leq 4} < 2^{31}$.

In this attack we use $sp = 4$, and compute the size of the relevant variable sets $|U'| \leq \binom{256}{\leq sp-1} = \binom{256}{\leq 3} \approx 2^{21.5}$, $|V| \leq \binom{\kappa}{\leq d-sp} = \binom{128}{\leq 4} \approx 2^{23.5}$, $|W| = |U'| + |V| \approx 2^{24}$.

For the high-order differential property, we use subspaces of dimension $2^6 = 64$ whose bits are not multiplied together in the first round. The outcome of such a high-order differential is a constant (independent of the key) for $1+6=7$ rounds, and this property can be extended beyond the 8'th Sbox layer when selecting the target bit from $Z_7|IP$.

Since $|W| \approx 2^{24}$, we require roughly the same number of 64-dimensional subspaces to construct the equation system and mount the attack. Therefore, we take a larger subspace of dimension 70, containing $\binom{70}{64} > 2^{24} \approx |W|$ 64-dimensional subspaces. As $X_0|IP$ contains only 67 bits, we choose the subspace from these 67 bits and additional 3 bits in $X_0|SP$, contained in 1 active Sbox. Since the active Sbox is non-linear, we guess the 3 linear key expressions that are added to its input, which allow us to construct the required $\approx 2^{24}$ 64-dimensional subspaces from a 70-dimensional subspace after the first Sbox layer.

The guess of the 3 key bits can be avoided by selecting the $70 - 64 = 6$ constant bits of the 64-dimensional subspaces from the 67 bits of $X_0|IP$ in the 70-dimensional subspace. This restriction keeps the selected Sbox fully active in all subspaces, and thus the linear subspace after the first Sbox layer (at Z_0) is independent of the key bits. The number of such restricted 64-dimensional subspaces is $\binom{67}{6} > 2^{24} \approx |W|$, and hence they should suffice for the attack.

Finally, we notice that the Moebius transforms (Step 4.b) can be optimized due to the way that we chose the subspaces in S, as for all of them, 3 specific bits of $X_0|SP$ are active. In order to exploit this, we perform the Moebius transform on a 2^{70} bit vector in two phases: in the first phase, we partition the 2^{70} big subspace into 2^{67} 3-dimensional subspaces according to the 67 bits of $X_0|IP$, and sum on all of them in time 2^{70}, obtaining a vector of size 2^{67}. In the second phase, we perform the Moebius transform on the 2^{67} vectors computed in the first phase. Therefore, the complexity of a single Moebius transform is reduced from $70 \cdot 2^{70} \approx 2^{76}$ to $2^{70} + 67 \cdot 2^{67} \approx 2^{73}$. The complexity of online Step 4.b now becomes $|U| \cdot 2^{73} \approx 2^{104}$ bit operations.

The time complexity analysis of the critical steps of the attack is as follows:

Step 4.a: $d \cdot |S| \cdot |U| \approx 8 \cdot 2^{70} \cdot 2^{31} = 2^{104}$
Step 4.b: 2^{104} (as noted above)
Step 4.c: $|U'| \cdot |W| \approx 2^{21.5} \cdot 2^{24} = 2^{45.5}$
Step 4.d.i: $|U| \cdot |V| \cdot r_1 \cdot 2^{16+d} \approx 2^{31} \cdot 2^{23.5} \cdot 3 \cdot 2^{24} \approx 2^{80.5}$
Step 4.d.ii: $|W| \cdot |U| \cdot |V| \approx 2^{24} \cdot 2^{31} \cdot 2^{23.5} = 2^{78.5}$
Step 5: $|W|^3 \approx 2^{24 \cdot 3} = 2^{72}$

In total, the time complexity of the attack is about 2^{105} bit operations, dominated by steps 4.a and 4.b. The data complexity is 2^{70} chosen plaintexts.

The memory complexity is dominated by the storage of ciphertexts in Step 1, and is about $|S| \cdot 2^8 = 2^{78}$ bits.

Extending the Attack to Full LowMC-128 for Weak Instances. Similarly to the attacks on LowMC-80, the 11-round attack on LowMC-128 can be extended by an additional round with no increase in complexity for a subset of weak instances. However, the fraction of these instances is much smaller, as the I-part of LowMC-128 contains only 67 bits, and is smaller than the one of LowMC-80. A similar analysis to the one of Sect. 6.3 shows that the fraction of such weak instances for LowMC-128 is roughly $2^{67+67-256} = 2^{-122}$. As noted in the Introduction, this attack does not violate the formal security claims of the LowMC designers.

7.2 An Attack on Full LowMC-128

We now describe our attack on full (12-round) LowMC-128. This attack is more marginal than the previous attacks, and we have to use essentially all of our previously described optimizations, as well as new ones in order to obtain an attack which is faster than exhaustive search.

In order to attack 12 rounds of LowMC-128, we extend the interpolation of the 11-round attack past another round, interpolating $Z_7|IP$ through $r_1 = 4$ Sbox layers, and hence $d = 2^4 = 16$, $|U| = \binom{256+3m}{\leq 2^{r_1-1}} = \binom{445}{\leq 8} \approx 2^{55}$.

In this attack we use $sp = 8$, and compute the size of the relevant variable sets $|U'| \leq \binom{256}{\leq sp-1} = \binom{256}{\leq 7} \approx 2^{43.5}$, $|V| \leq \binom{\kappa}{\leq d-sp} = \binom{128}{\leq 8} \approx 2^{40.5}$, $|W| = |U'| + |V| \approx 2^{44}$. We use the refined analysis for steps 4.d.i and 4.d.ii, and thus we also calculate $|U_{sp}| = |U_8| = \binom{256}{8} < 2^{48.5}$ and $|V_{\leq(d-sp-1)}| = \binom{128}{\leq 7} < 2^{36.5}$.

The High-Order Differential Property. We can try to mount the attack with high-order differentials on subspaces of dimension 64 for the target bit in $Z_7|IP$, but this results in an attack which is at best very marginally faster than exhaustive search. The main new optimization introduced in this attack is the use of reduced subspaces of dimension 60. Obviously, the result of a high-order differentiation over such a subspace is not a constant, but (as we show next) its algebraic degree in the key bits is bounded by 8. Consequently, the resultant function (polynomial) of each high-order differentiation can be expressed in terms of our reduced variable set $V = |V_{\leq(8)}|$. This polynomial can be interpolated during preprocessing and does not contribute additional variables to the equation system.

We select a big subspace S of dimension 73 that contains all the 67 bits of $X_0|IP$ and 6 additional bits of 2 active Sboxes in $X_0|SP$, and (similarly to the 11-round attack) define the 60-dimensional subspaces according to their $73-60 = 13$ constant bits in $X_0|IP$. The number of such subspaces is $\binom{67}{13} > 2^{44} \approx |W|$, and therefore they should suffice for the attack.

In order to show that the result of a high-order differentiation of the target bit in $Z_7|IP$ over a selected 60-dimensional is of degree 8 in the key bits, consider the state Z_0 obtained after the first Sbox layer. The algebraic degree of the target bit b (selected from $Z_7|IP$) in Z_0 is bounded by $2^6 = 64$. As the linear subspace undergoes a one-to-one transformation in the first Sbox layer (through the fully active 2 Sboxes), it remains a linear subspace in Z_0. Therefore, the algebraic degree of the high-order differentiation in the bits of Z_0 and the key is upper-bounded by $64 - 60 = 4$. Since each bit of Z_0 is a polynomial in the key of degree (at most) 2, the algebraic degree of the high-order differentiation in the bits of the key is upper-bounded by $4 \cdot 2 = 8$, as claimed.

The Preprocessing Phase. The main change in this attack compared to the one of Sect. 5.2 is in preprocessing Step 1, where in addition to interpolating the $e \approx |W|$ free coefficients, we interpolate the $e \cdot |V| \approx |W| \cdot |V|$ coefficients of V (since we selected 60-dimensional subspaces instead of 64-dimensional subspaces). The modified preprocessing step is described below. It is similar to the variable transformation algorithm of Sect. 5.1, interpolating first over the plaintexts and then over the keys. Note that the matrix E of linear equations is allocated and initialized already at this stage.

```
1. Allocate an $e \times |W|$ matrix $E$ over $GF(2)$, representing the (reduced)
   equation system on $W$. The matrix is vertically decomposed into two
   smaller matrices: $E_1$ of size $e \times |U'|$ and $E_2$ of size $e \times |V|$.
2. Allocated an $e \cdot |V|$ evaluation matrix $EV$.
3. Allocate a $|S| = 2^{73}$ bit array $\boldsymbol{a_1}$ for the evaluations of the target bit $b$.
4. For each key in the set $\{K | wt(K) \leq 8\}$ (with index $\ell$):
   (a) Evaluate $b$ (the target bit) on the set $S$ of $2^{73}$ inputs (with the fixed
       key $K$) and store the result in $\boldsymbol{a_1}$.
   (b) Apply the Moebius transform on $\boldsymbol{a_1}$.
   (c) Populate column $\ell$ of $EV$: For each subspace $S_j$ in $S$, namely
       $S_1, \ldots, S_e$, obtain its corresponding sum from $\boldsymbol{a_1}$ and copy it to
       $E_1[j][\ell]$.
5. For each equation $1, \ldots, e$ (with index $j$):
   (a) For each $M_v \in V_{\leq 8} = V$ (with index $\ell$):
         i Sum the $2^{wt(\bar{v})}$ values of $EV[j]$ calculated for the subset of keys
           $\{K | \bar{v} \wedge K = 0\}$, and store the result in $E_2[j][\ell]$.
```

We first note that similarly to the 11-round attack, the complexity of the Moebius transform can be optimized (due to the way that we selected the subspaces) in a 2-step process from $73 \cdot 2^{73}$ to $2^{73} + 67 \cdot 2^{67} \approx 2^{74}$.

We analyze the complexity of the computationally heavy steps 4 and 5. The complexity of Step 4.a (for all $\{K | wt(K) \leq 8\}$) is $|V| \cdot |S| \cdot 2^{19} \approx 2^{40.5} \cdot 2^{73} \cdot 2^{19} = 2^{132.5}$. The complexity of Step 4.b (using the optimized Moebius transform) is

$|V| \cdot 2^{74} \approx 2^{114.5}$. The complexity of Step 4.c is $e \cdot |V| \approx |W| \cdot |V| \approx 2^{44} \cdot 2^{40.5} = 2^{84.5}$. The complexity of Step 5.a.i is bounded by $e \cdot |V| \cdot 2^8 \approx 2^{44} \cdot 2^{40.5} \cdot 2^8 = 2^{92.5}$. In total, Step 4.a dominates the time complexity, which is about $2^{132.5}$ bit operations.

Analysis of the Full Attack. In terms of time complexity, the analysis of the critical steps of the online attack is as follows (using the optimized Moebius transform and the refined analysis for steps 4.d.i and 4.d.ii):

Step 4.a: $d \cdot |S| \cdot |U| \approx 16 \cdot 2^{73} \cdot 2^{55} = 2^{132}$
Step 4.b: $|U| \cdot 2^{74} \approx 2^{129}$
Step 4.c: $|U'| \cdot |W| \approx 2^{43.5} \cdot 2^{44} = 2^{87.5}$
Step 4.d.i: $|U_{sp}| \cdot |V| \cdot (r_1 \cdot 2^{16+sp} + 2^{d-sp}) + |U| \cdot |V_{\leq(d-sp-1)}| \cdot r_1 \cdot 2^{16+d} \approx 2^{48.5} \cdot 2^{40.5} \cdot (4 \cdot 2^{24} + 2^8) + 2^{55} \cdot 2^{36.5} \cdot 4 \cdot 2^{32} \approx 2^{115} + 2^{125.5} \approx 2^{125.5}$
Step 4.d.ii: $|W| \cdot (|U_{sp}| \cdot |V| + |U| \cdot |V_{\leq(d-sp-1)}|) \approx 2^{44} \cdot (2^{48.5} \cdot 2^{40.5} + 2^{55} \cdot 2^{36.5}) \approx 2^{44} \cdot (2^{89} + 2^{91.5}) \approx 2^{136}$
Step 5: $|W|^3 \approx 2^{44 \cdot 3} = 2^{132}$

The online phase complexity is about 2^{136} dominated by[3] Step 4.d.ii. The total complexity of the attack is less than 2^{137} bit operations, which is about $2^{128+19-137} = 2^{10}$ times faster than exhaustive search (including the preprocessing phase, whose complexity is about $2^{132.5}$). The data complexity of the attack is 2^{73} chosen plaintexts. The memory complexity is dominated by the storage of E, whose size is about $|W| \cdot |W| \approx 2^{88}$ bits.

Note that without the variable transformation, merely Step 5 (Gaussian elimination) would require about $2^{55 \cdot 3} = 2^{165}$ bit operations, which is much slower than exhaustive search.[4]

8 Conclusions

In this paper, we introduced new techniques for interpolation attacks, including a new variable transformation algorithm that can lead to savings in their data and time complexities. We applied the optimized interpolation attack to LowMC, and refuted the claims of the designers regarding the security level of both the 80 and 128-bit key variants. As a future work item, it will be interesting to optimize our techniques further and apply them to additional block ciphers.

[3] We note that the analysis of Step 4.d.ii can be refined further, and its actual complexity is lower by a factor between 2 and 4. Moreover, the actual algorithm of this step can be optimized, but we do not consider such low-level optimizations here for the sake of simplicity.

[4] Solving the equation system remains slower than exhaustive search even when using more advanced algorithms which are based on Strassen's algorithm [9], requiring about $2^{55 \cdot 2.8} = 2^{154}$ bit operations. While there are known algorithms that perform better in theory, most of them are very complex and inefficient in practice.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (2015)
2. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
3. Hell, M., Johansson, T., Maximov, A., Meier, W.: The grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
4. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 28–40. Springer, Heidelberg (1997)
5. Joux, A.: Algorithmic Cryptanalysis, 1st edn. Chapman & Hall/CRC, Boca Raton (2009)
6. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1994)
7. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography. SLSECS, vol. 276, pp. 227–233. Springer, Heidelberg (1994)
8. Shimoyama, T., Moriai, S., Kaneko, T.: Improving the higher order differential attack and cryptanalysis of the KN cipher. In: Okamoto, E., Davida, G., Mambo, M. (eds.) Information Security. LNCS, vol. 1396, pp. 32–42. Springer, Heidelberg (1997)
9. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik **13**, 354–356 (1969)