Noname manuscript No.
(will be inserted by the editor)

Linear Propagation in Efficient Guess-and-Determine Attacks

Maria Eichlseder · Florian Mendel · Tomislav Nad · Vincent Rijmen · Martin Schläffer

Received: date / Accepted: date

Abstract The most successful attacks on cryptographic hash functions are based on differential cryptanalysis, where the main problem is to find a differential characteristic. Finding a differential characteristic is equivalent to solving a system of nonlinear equations. Solving these equations is usually done by a guess-and-determine approach. Recently, automated tools performing a guess-and-determine approach based on the concept of generalized conditions have been used to attack many hash functions. The core part of such tools is the propagation of information. In this paper, we propose a new approach to propagate information for affine functions and compare it to the approach used in recent hash function attacks. We apply our method to the linear functions σ_i and Σ_i used in SHA-2 and to the linear layer of SHA-3. We show that our approach performs much better than the previously used methods.

Keywords hash functions · differential cryptanalysis · automated tool · guess-and-determine · linear functions · SHA-2 · SHA-3

CR Subject Classification Cryptography

1 Introduction

Finding collisions or preimages for hash functions is a special case of the general problem of solving nonlinear equations. One of the general approaches for solving nonlinear equations is the guess-and-determine approach. Many attacks on cryptographic hash functions can be described as guess-and-determine attacks. Depending on the function, attack setting and other properties, the system of equations can be simplified such that a guess-and-determine approach can be successful. The basic idea of a guess-and-determine approach is to perform a guessing of certain bits before determining others. For instance, *message modification* and

M. Eichlseder · F. Mendel · T. Nad · M. Schläffer
IAIK, Graz University of Technology, Austria

V. Rijmen
ESAT/COSIC, KU Leuven and iMinds, Belgium

advanced message modification [14,15] and all their successors and variants perform a guessing of certain bits before determining others.

With the selection of the AES, interest in cryptanalysis using algebraic methods re-awakened, due to the simple algebraic structure of this cipher [3,4]. After the initial optimism, it became clear that although it is possible to construct relatively simple sets of equations, one still needs efficient equation solving strategies. Several attempts were made in that direction, one of the most promising being Multiple Right-Hand Side equations [12,13].

The most successful attacks on hash functions are based on differential cryptanalysis [2]. The original description of differential cryptanalysis assumes that a good characteristic is given. However, it is often not feasible to find characteristics by hand, in particular for hash functions. Cryptographers now use (semi-)automated tools to find differential characteristics. Based on the concept of generalized conditions, De Cannière and Rechberger provided the first tool to find automated differential characteristics for SHA-1 based on a guess-and-determine approach [5]. Recently, this work was improved and extended, leading to attacks on several hash functions [8–11]. A similar approach is followed by Leurent [6,7]. The core part of these tools is the propagation of constraints on the variables.

In this paper, we propose a method to store and propagate linear relations of variables efficiently. Since most cryptographic algorithms consist of large linear parts, we show that storing linear relations on more than a single bit leads to significantly better results.

In the following, we start with an introduction to guess-and-determine attacks in differential cryptanalysis in Section 2 and review previous methods and their limitations in Section 3. We continue with the main part of this paper, the linear propagation of linear information in Section 4 and evaluate it in Section 5. Finally, we conclude in Section 6.

2 Guess-and-Determine Attacks

On a high level, a guess-and-determine attack can be described as a repetition of two steps until all unknown variables have been determined: first, guess the value of some unknowns; second, determine the value of as many unknowns as possible. The second step can be slightly generalized. Instead of outputting only the values of new unknowns, it can also output simplified equations between remaining unknowns or partial information (constraints) on the remaining unknowns. It can also announce that for the currently guessed values, there is no solution to the system, signaling that some of the guesses need to be changed. Therefore, we will refer here to the second step using the more general term *propagation of information.*

A successful guess-and-determine attack employs a strategy to convert complex and dense equations into a form that is more amenable to analyze. The following elements need to be considered in such a strategy.

Choice of intermediate variables: By introducing additional intermediate variables, it becomes possible to reduce the algebraic degree and/or improve the sparsity of the equations. Of course, every newly introduced unknown also introduces a new equation. Hence, there is a trade-off between the number of equations and the simplicity of each equation.

Choice of information to store: Storing all information on each of the intermediate variables would require too much effort to keep all information up-to-date and consistent. Therefore, we store only a part of the information, and recreate the rest if we need it.

Propagation of information: Every time a variable is guessed, we need to check whether new information on other variables can be determined. There is a trade-off between the effort we spend in this step and simply guessing more bits.

Guessing strategy: We need a guessing strategy which can efficiently use the new information generated by the propagation of information introduced by previous guesses.

3 Bitsliced Propagation of Information

In efficient guess-and-determine attacks, we propagate information by solving equations. The complexity and difficulty of this equation solving step depends on several factors as mentioned in the previous section. In the case of cryptographic primitives, these equations are usually large, complex and hard to solve. Therefore, we split these equations into easier parts which can be solved more efficiently.

Recent guess-and-determine attacks on hash functions split the equations into small bitslices and store information on intermediate variables using generalized conditions [5, 8–11]. The subproblem of propagating information within bitslices is small enough to be solved efficiently using exhaustive search.

3.1 Generalized Conditions

In classical differential cryptanalysis, only one bit of information is stored for each pair of bits (x_j, x_j^*): the difference $\Delta x_j = x_j \oplus x_j^*$. Inspired by signed-bit differences [14], De Cannière and Rechberger introduced generalized conditions [5], where all information on one pair of bits (x_j, x_j^*) is taken into account.

Definition 1 Let (x_j, x_j^*) be a pair of bits. The *generalized condition* $\nabla(x_j, {x_j}^*)$ is a subset of all pairs $\{(0,0),(0,1),(1,0),(1,1)\}$ of (x_j, x_j^*).

In total, we get 16 possible generalized conditions. We use the same notation as in [5] (see Table 1). To specify differences and conditions on word level, we group generalized conditions as follows. Let $x, x^* \in \{0,1\}^n$ and $c_j = \nabla(x_j, {x_j}^*)$. Then, the notation $\nabla(x, x^*) = [c_{n-1} \dots c_0]$ provides a compact specification of all n generalized conditions of the pair (x, x^*). Let $\nabla(x, x^*) = [c_{n-1} \dots c_0]$. Then

$$|\nabla(x, x^*)| = \prod_{i=0}^{n-1} |c_i|$$

denotes the number of pairs fulfilling all generalized conditions in $\nabla(x, x^*)$.

3.2 Bitsliced Propagation

In the bitslice approach, information is propagated only between bits of a bitslice. A bitslice is defined as follows.

Definition 2 Let $f : \{0,1\}^m \rightarrow \{0,1\}^n$ be a function with $y = f(x)$. One output bit y_j depends on $|I|$ inputs with $I \subseteq \{0,\dots,m-1\}$. Then the set $\mathcal{B}_j = \{y_j, x_{i_0}, \dots, x_{i_{|I|}}\}$ is called a *bitslice.*

In a guess-and-determine attack, new conditions are imposed on bits of a bitslice. These new conditions affect at least the bits in the same bitslice. Hence, the generalized conditions of a bitslice need to be updated. This propagation of new information is done as described in Algorithm 1.

Algorithm 1 Propagation

```
Input: $\nabla(x_i, x_i{}^*)$
Output: Updated conditions on bits in all bitslice pairs ($\mathcal{B}_j$,$\mathcal{B}_j^*$) containing $x_i$, $x_i{}^*$
  for each pair of bitslices ($\mathcal{B}_j$,$\mathcal{B}_j^*$) containing $x_i$ and $x_i{}^*$ do
    Test all possibilities allowed by the generalized conditions
    Remove cases which are not possible any more
```

Algorithm 1 provides an optimal propagation for bitwise Boolean functions, where each bitslice is independent of each other. Examples are the bitwise Boolean functions IF and MAJ used in many hash function designs of the MD4-family. Furthermore, it has been shown in recent attacks on the MD4-family of hash functions [5,8–11] that the bitslice approach also works quite well for the modular addition. However, it has significant shortcomings when applied to more complex (linear) functions as shown in the next section.

3.3 Example for Bad Propagation

The shortcomings of the bitsliced approach can be illustrated best by means of an example. We consider the linear Σ_0 function which is used in the hash function SHA-2.

Example 1 Let $\Sigma_0 : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$ be a linear function with

$$\Sigma_0(x) = (x \ggg 2) + (x \ggg 13) + (x \ggg 22) = y, \tag{1}$$

which is equivalent to $y_j = x_{j+2} + x_{j+13} + x_{j+22}$, for $j = 0,\dots,31$. Hence, Σ_0 defines 32 bitslices $\mathcal{B}_j = \{y_j, x_{j+2}, x_{j+13}, x_{j+22}\}$. Note that the additions in the indices are modulo 32. Assume that we start with the following information:

$$\begin{aligned} \nabla(x, x^*) &= \texttt{[??????????????????-??????????-??]} \\ \nabla(y, y^*) &= \texttt{[--------------------------------]}. \end{aligned}$$

By filling in the information for bitslice $\mathcal{B}_0$ we get

$$\begin{aligned} \nabla(x_2, x_2{}^*) \quad &= \texttt{[-]} \\ \nabla(x_{13}, x_{13}{}^*) &= \texttt{[-]} \\ \nabla(x_{22}, x_{22}{}^*) &= \texttt{[?]} \\ \nabla(y_0, y_0{}^*) \quad &= \texttt{[-]}. \end{aligned}$$

We can then propagate information according to Algorithm 1 by exhaustively searching over all possible pairs defined by the input generalized conditions, to check which output pairs are possible (see Table 1). It follows that $\nabla(x_{22}, {x_{22}}^*) =$ `[-]`. Note that in all the other cases no information is propagated and applying the algorithm to all other indices gives the following propagated information:

$$\begin{aligned}\nabla(x, x^*) &= \texttt{[?????????-????????-??????????-??]}\\ \nabla(y, y^*) &= \texttt{[--------------------------------]}.\end{aligned}$$

However, for the invertible linear function Σ_0 we know that $\nabla(x_j, {x_j}^*) =$ `[-]` must hold for all j.

In this case, the bitslice approach of propagating information performs very poor. Therefore, we present in the following section a different approach which produces optimal results for affine functions and linear conditions.

4 Linear Propagation

The constraints defined by generalized conditions and the function f can also be expressed as a system of equations involving the input and output bits as variables (see Table 1). Propagating conditions then corresponds to manipulating this system in order to bring it to a more useful form. If all involved equations are linear, methods like elementary row and column operations can be applied to simplify the system. Unfortunately, it is not possible to translate all generalized conditions into linear equations. Furthermore, hash functions contain nonlinear building blocks such as modular additions or other nonlinear Boolean functions which cannot be expressed as linear equations.

However, large parts of cryptographic algorithms consist of affine (linear) functions. Furthermore, most generalized conditions are linear (only `7`, `B`, `D`, and `E` are nonlinear). In the following, we show a general method to efficiently extract linear information on the variables and propagate this information.

4.1 Affine Functions

Let $f : \{0,1\}^m \to \{0,1\}^n$ be an affine function. Such a function can be described by a matrix $L \in \{0,1\}^{n\times(m+n)}$, where

$$y = f(x) \Leftrightarrow L \begin{bmatrix} x \\ y \end{bmatrix} = d$$

and $d = f(0)$. When considering input pairs x and x^*, the same affine function is applied to both inputs. Denoting the vector $[x \; y]^T$ by z and $[x^* \; y^*]^T$ by z^* respectively, the whole system of linear equations can then be written as follows:

$$\begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} \cdot \begin{bmatrix} z \\ z^* \end{bmatrix} = \begin{bmatrix} d \\ d \end{bmatrix} . \qquad (2)$$

Table 1 Notation and equations of the 16 generalized conditions. For linear conditions, matrix coefficients C_{ij}, C^*_{ij} are given. The elements in rows C_i and $C_i{}^*$ are zero except for the given positions.

$\nabla(z_j, z_j{}^*)$	$(1,1)$	$(0,1)$	$(1,0)$	$(0,0)$	Equations	C_{ij}	C^*_{ij}	b_i
`#`	–	–	–	–	contradiction	0	0	1
`0`	–	–	–	×	$z_j = 0$ $z^*_j = 0$	1 0	0 1	0 0
`u`	–	–	×	–	$z_j = 1$ $z^*_j = 0$	1 0	0 1	1 0
`3`	–	–	×	×	$z^*_j = 0$	0	1	0
`n`	–	×	–	–	$z_j = 0$ $z^*_j = 1$	1 0	0 1	0 1
`5`	–	×	–	×	$z_j = 0$	1	0	0
`x`	–	×	×	–	$z_j + z^*_j = 1$	1	1	1
`7`	–	×	×	×	$z_j z^*_j = 0$			
`1`	×	–	–	–	$z_j = 1$ $z^*_j = 1$	1 0	0 1	1 1
`-`	×	–	–	×	$z_j + z^*_j = 0$	1	1	0
`A`	×	–	×	–	$z_j = 1$	1	0	1
`B`	×	–	×	×	$z_j z^*_j + z^*_j = 0$			
`C`	×	×	–	–	$z^*_j = 1$	0	1	1
`D`	×	×	–	×	$z_j z^*_j + z_j = 0$			
`E`	×	×	×	–	$z_j z^*_j + z_j + z^*_j = 1$			
`?`	×	×	×	×	no constraints	0	0	0

4.2 Linear Conditions

Definition 3 A linear generalized condition $\nabla_l(z_j, z_j{}^*)$ is a generalized condition that is an affine space.

A set of linear generalized conditions on several bits can be expressed as

$$\begin{bmatrix} C \; C^* \end{bmatrix} \begin{bmatrix} z \\ z^* \end{bmatrix} = b\,, \tag{3}$$

where $C, C^* \in \{0,1\}^{c\times(m+n)}$ are binary matrices with c linear equations on $m+n$ bits $z, z^* \in \{0,1\}^{m+n}$ and $b \in \{0,1\}^c$. Table 1 shows how the matrices C and C^* are constructed from the linear generalized conditions.

4.3 Linear Propagation of Information

In order to propagate linear information, we perform the following three steps:

1. Construct the combination of (2) and (3).
2. Use Gauss-Jordan elimination to create sparse equations (rows).
3. Convert equations only on z_j and z^*_j back to generalized conditions.

We combine the two systems (2) and (3) such that the resulting matrices $A, A^* \in \{0,1\}^{(2n+c)\times(m+n)}$ represent the columns corresponding to z and z^* respectively:

$$\begin{bmatrix} L & 0 \\ 0 & L \\ C & C^* \end{bmatrix} \begin{bmatrix} z \\ z^* \end{bmatrix} = \begin{bmatrix} A \; A^* \end{bmatrix} \begin{bmatrix} z \\ z^* \end{bmatrix} = \begin{bmatrix} d \\ d \\ b \end{bmatrix}. \tag{4}$$

To propagate linear information we simply perform Gauss-Jordan elimination and get a matrix in reduced row-echelon form. If the system is inconsistent, we know that the generalized conditions at the input and output of the affine function contradict each other. If the system is consistent, we can extract new information in form of generalized conditions. Since a generalized condition consists of equations involving only z_j and z_j^*, we get this information by a linear combination of at most two rows.

5 Comparison of Propagation Methods

In this section, we compare the different information propagation methods. We show that in the majority of cases the linear propagation described in Section 4.3 performs much better than the bitslice approach described in Section 3.2. For the rare cases where the bitslice approach performs better, a combination of both approaches results in the best propagation performance. To compare and evaluate the different propagation methods, we need to measure how well they propagate. Propagation corresponds to narrowing down the solution space, or gaining information about the solution.

Let $\nabla(z, z^*)'$ denote the generalized conditions obtained by propagating $\nabla(z, z^*)$ by means of propagation method M. Then we take as figure of merit for M:

$$I_M(z) = \log_2 \frac{|\nabla(z, z^*)|}{|\nabla(z, z^*)'|}.$$

If $\nabla(z, z^*)$ is a contradiction, then $|\nabla(z, z^*)| = |\nabla(z, z^*)'| = 0$ and we set $I_M(z) = 0$. If $|\nabla(z, z^*)'| = 0$ but $|\nabla(z, z^*)| \neq 0$, then $I_M(z)$ is undefined, which we denote by $I_M(z) = \#$. To compare the two propagation methods and measure the gain of the linear method (L) over the bitslice method (B) for one specific condition $\nabla(z, z^*)$, we compute the difference of the two methods' figures of merit:

$$I_{\text{diff}}(z) = I_{\text{B}}(z) - I_{\text{L}}(z).$$

If $I_{\text{L}}(z) = \#$ but $I_{\text{B}}(z) \neq \#$, the linear method detects the contradiction but the bitslice method does not. In this case, we set $I_{\text{diff}}(z) = \#_L$. If $I_{\text{L}}(z) \neq \#$ but $I_{\text{B}}(z) = \#$ we set $I_{\text{diff}}(z) = \#_B$. If both are $\#$, we set $I_{\text{diff}}(z) = 0$.

5.1 Applications

We evaluate the propagation methods for different functions f by computing $I_{\text{diff}}(z)$ for a large number of randomly drawn samples $\nabla(z, z^*)$. Choosing random generalized conditions at the input and output of one of the functions results in impossible characteristics with a high probability. Since such cases are less likely to occur in a guess-and-determine attack, we have performed a search for differential characteristics in SHA-2 and Keccak (SHA-3) and extracted random samples from this search. The results show the empiric distribution function of the random variable $I_{\text{diff}}(z)$.

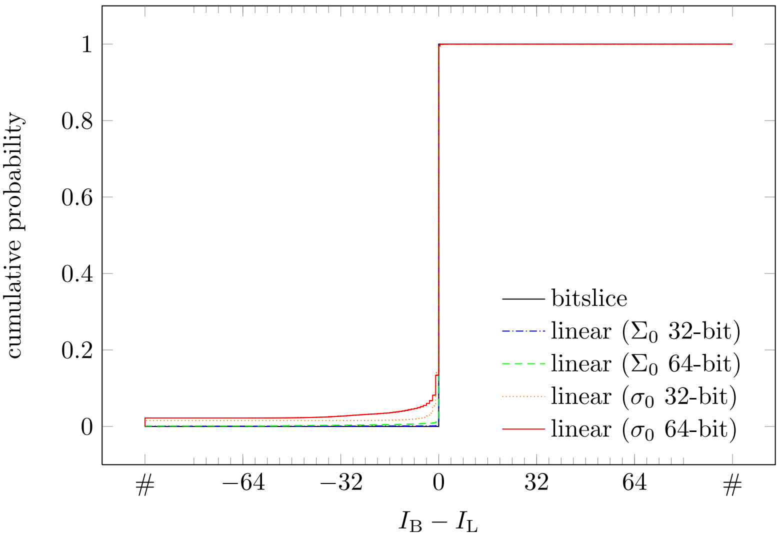

Fig. 1 Comparison of propagation methods for 32-bit and 64-bit Σ_0 and σ_0. Higher values correspond to better propagation.

5.1.1 Linear Functions of SHA-2.

Figure 1 shows the results for the 32-bit (SHA-256) and 64-bit (SHA-512) linear functions Σ_0 and σ_0 of SHA-2. The figure shows a comparison for samples, which commonly occur in a search for differential characteristics in SHA-2. The linear propagation performs significantly better than the bitslice approach for σ_0 which is used in the message expansion of SHA-2. Since Σ_0 is used in the state update transformation where other functions influence the propagation as well, the gain is limited.

5.1.2 Linear Layer of Keccak.

Finally, we apply the different propagation methods to the linear layer used in Keccak [1] which is significantly larger than the linear functions used in SHA-2. In more detail, the linear layer in Keccak updates 25 lanes, each of length w. Since in Keccak lane lengths of 8, 16, 32 and 64 bits are defined, we can compare the propagation methods for different sizes of the linear layer. Using Keccak we can show, that for larger affine functions the linear propagation method tends to perform better. Note that for SHA-3 only a length of 64 bits is defined.

Figure 2 shows the comparison of the bitslice approach with the linear approach for different lane lengths in Keccak. Most notably, for $w = 64$, the linear approach performs better in more than 97% of the samples, and significantly more contradictions are detected by the linear approach.

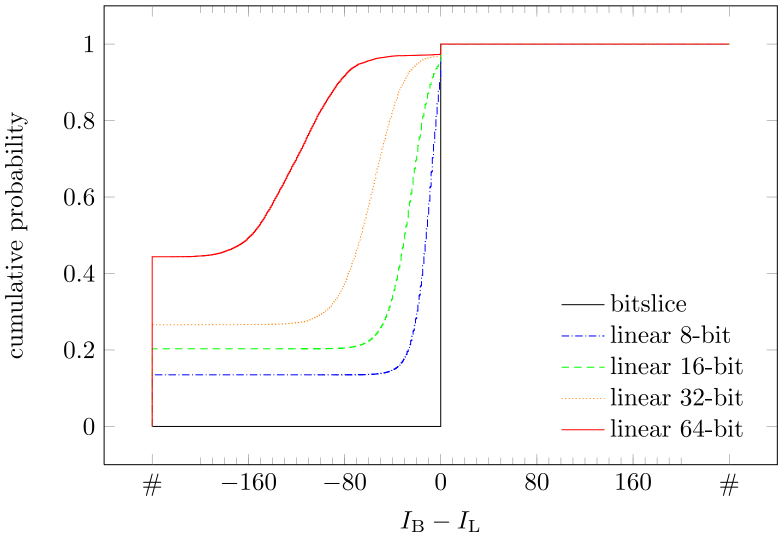

Fig. 2 Comparison of propagation methods for different lane lengths ($w = \{8, 16, 32, 64\}$) of the whole linear layer used in Keccak. Higher values correspond to better propagation.

6 Conclusions

In recent years, many differential attacks on hash functions are using automated tools to find differential characteristics. These tools are performing a guess-and-determine attack to solve the resulting nonlinear equations. The basic idea of a guess-and-determine attack is to perform a guessing of certain bits before determining others. The core part of such tools is the propagation of (new) information. Usually, a trade-off between the performance of the propagation and the amount of propagated information is made. Hence, information is lost and a better propagation method can significantly improve these tools.

We investigated how the propagation for certain functions can be improved. We focused on affine functions, since most cryptographic algorithms consist of large linear parts. We showed that the approach used in the recent attacks has some significant drawbacks. As a solution to this problem, we proposed a linear information propagation method and showed that it performs significantly better.

In our approach, we store linear relations on more than a single bit. Furthermore, instead of propagating information only within a single bitslice, we propagate information wordwise by considering the whole system of linear equations derived from the affine function and generalized conditions. Using an efficient algorithm to extract new information from this system, we achieve optimal results for linear functions and linear generalized conditions. Furthermore, we gave a detailed description of the approach and compared it to the approach used in the recent hash function attacks. For the comparison, we applied our method to the linear functions Σ_0 and σ_0 used in SHA-2 and to the linear layer of Keccak (SHA-3). We have shown that our method performs much better than previously published methods, making it possible to apply the recent techniques to more complex functions like SHA-3.

Acknowledgments

Part of this work was done while Florian Mendel was with KU Leuven. The work has been supported by the Austrian Science Fund (FWF), project P21936-N23 and TRP 251-N23; and by the Research Fund KU Leuven, OT/08/027.

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference. Submission to NIST (Round 3) (2011). URL `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`
2. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology **4**(1), 3–72 (1991). DOI 10.1007/BF00630563
3. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: B. Preneel (ed.) EUROCRYPT, *LNCS*, vol. 1807, pp. 392–407. Springer (2000). DOI 10.1007/3-540-45539-6_27
4. Courtois, N., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: E. Biham (ed.) EUROCRYPT, *LNCS*, vol. 2656, pp. 345–359. Springer (2003). DOI 10.1007/3-540-39200-9_21
5. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: X. Lai, K. Chen (eds.) ASIACRYPT, *LNCS*, vol. 4284, pp. 1–20. Springer (2006). DOI 10.1007/11935230_1
6. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: X. Wang, K. Sako (eds.) ASIACRYPT, *LNCS*, vol. 7658, pp. 226–243. Springer (2012). DOI 10.1007/978-3-642-34961-4_15
7. Leurent, G.: Construction of Differential Characteristics in ARX Designs – Application to Skein. Cryptology ePrint Archive, Report 2012/668 (2012). URL `http://eprint.iacr.org/`
8. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential Attacks on Reduced RIPEMD-160. In: D. Gollmann, F.C. Freiling (eds.) ISC, *LNCS*, vol. 7483, pp. 23–38. Springer (2012). DOI 10.1007/978-3-642-33383-5_2
9. Mendel, F., Nad, T., Schläffer, M.: Cryptanalysis of Round-Reduced HAS-160. In: H. Kim (ed.) ICISC, *LNCS*, vol. 7259, pp. 33–47. Springer (2011). DOI 10.1007/978-3-642-31912-9_3
10. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: D.H. Lee, X. Wang (eds.) ASIACRYPT, *LNCS*, vol. 7073, pp. 288–307. Springer (2011). DOI 10.1007/978-3-642-25385-0_16
11. Mendel, F., Nad, T., Schläffer, M.: Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. In: A. Canteaut (ed.) FSE, *LNCS*, vol. 7549, pp. 226–243. Springer (2012). DOI 10.1007/978-3-642-34047-5_14
12. Raddum, H., Semaev, I.: Solving Multiple Right Hand Sides linear equations. Des. Codes Cryptography **49**(1-3), 147–160 (2008). DOI 10.1007/s10623-008-9180-z
13. Semaev, I.: Sparse Algebraic Equations over Finite Fields. SIAM J. Comput. **39**(2), 388–409 (2009). DOI 10.1137/070700371
14. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: V. Shoup (ed.) CRYPTO, *LNCS*, vol. 3621, pp. 17–36. Springer (2005). DOI 10.1007/11535218_2
15. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: R. Cramer (ed.) EUROCRYPT, *LNCS*, vol. 3494, pp. 19–35. Springer (2005). DOI 10.1007/11426639_2