Producing Collisions for PANAMA*

Vincent Rijmen**, Bart Van Rompay,
Bart Preneel and Joos Vandewalle

Katholieke Universiteit Leuven, ESAT-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{vincent.rijmen,bart.vanrompay}@esat.kuleuven.ac.be

Abstract. PANAMA is a cryptographic module that was presented at the FSE Workshop in '98 by Joan Daemen and Craig Clapp. It can serve both as a stream cipher and as a cryptographic hash function, with a hash result of 256 bits. PANAMA achieves high performance (for large amounts of data) because of its inherent parallelism. We will analyse the security of PANAMA when used as a hash function, and demonstrate an attack able to find collisions much faster than by birthday attack. The computational complexity of our current attack is 2^{82}; the required amount of memory is negligible.

1 Introduction

PANAMA is a cryptographic module that can be used in two modes, called *push* and *pull* mode. The module can be used both as a hash function and as a stream cipher. For a full description of PANAMA, we refer to [1].

In this paper we will describe a method to produce collisions for the PANAMA hash function. A collision occurs when two different messages are hashed to the same value. The PANAMA hash function maps messages of arbitrary length to a hash result of 256 bits, which means that a general birthday attack would need about 2^{128} operations fo find a colliding pair of messages. We will show that with our method a collision can be found with significantly less operations and a small amount of memory.

2 Short Description of PANAMA

The PANAMA stream/hash module has two types of internal memory: the *buffer* and the *state*. The buffer is a linear feedback shift register, containing 32 cells of 256 bits, denoted $b^0, \ldots b^{31}$. The state consists of 544 bits, divided in 17 32-bit words. Let the state of PANAMA be denoted by a 3-tuple (r, s, t), where r

* The work described in this paper has been supported in part by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 and by the Concerted Research Action (GOA) Mefisto-666.

** F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research, Flanders — Belgium.

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

denotes the first 32-bit word of the state, and s and t each denote a 256-bit (8 words) value.

In the PANAMA hash mode, the input is padded to a multiple of 256 bits, and divided into blocks of 256 bits. All internal memory bits are set to zero, and then the following steps are executed for each message block. First, the state is updated by applying the nonlinear transformation ρ (which is composed of three specific transformations $\theta \circ \pi \circ \gamma$, described in the next section):

$$(r, s, t) \leftarrow \rho(r, s, t).$$

Secondly, the message block m and b^{16}, the contents of buffer cell 16, are exored into the state. The least significant bit of word 0 is flipped. These three operations are denoted here[1] by σ:

$$(r, s, t) \leftarrow \sigma(r, s, t) = (r + 1, s + m, t + b^{16}).$$

Thirdly, the message block is fed into the buffer and the LFSR is stepped once.

When all message blocks have been processed, 33 extra iterations are performed, but now the message input to the buffer is replaced by the state (part s), and the message input of σ is replaced by the contents of buffer cell 4. The 256-bit hash result is defined as the final state (part t).

Note that the buffer content is not present in the output. This means that there are two types of collisions for PANAMA: collisions in the state only, and collisions in both the buffer and the state. The 33 iterations after the last message block has been processed have as function to make it difficult to produce collisions of the first kind. We will try to find collisions of the second kind.

We will use the following notation: A message stream is denoted by $Pn, P(n-1), \ldots, P2, P1, P0$, where $P0$ is the first message block and Pn the last. Let X be an array of 8 32-bit words X_i, then $Y = r(X) \Leftrightarrow Y_i = X_{i+2 \bmod 8}$. Throughout this paper, '+' denotes bitwise addition (exor), except when used in the indices, where it denotes addition modulo 17 (if not explicitly stated otherwise).

3 Components of PANAMA

We describe here the different components of the nonlinear state updating transformation ρ. We also make some preliminary observations which will be useful later on. The transformations operate on the 17 words of the PANAMA state, where each word consists of 32 bits. In the following the index i ($0 \leq i < 17$) denotes the position of the word in the state.

3.1 The Transformation θ

The linear transformation θ is defined as follows:

$$c = \theta(a) \Leftrightarrow c_i = a_i + a_{i+1} + a_{i+4} \tag{1}$$

[1] Our notation differs with the one in [1], where σ is included in ρ.

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

We calculated the inverse transformation θ^{-1}:

$$a_i = c_{i+1} + c_{i+2} + c_{i+5} + c_{i+9} + c_{i+10} + c_{i+11} + c_{i+12} + c_{i+14} + c_{i+16}. \qquad (2)$$

3.2 The Transformation π

The transformation π which combines cyclic word shifts with a permutation of the word positions is defined below. Here $[j]$ $(0 \leq j < 32)$ denotes the bit position in a word, the multiplication $7i$ in the index is modulo 17 like the additions, and $r(i) = i \times (i+1)/2$ is a bit rotation which should be taken modulo 32.

$$c = \pi(a) \Leftrightarrow c_i[j] = a_{7i}[j + r(i)] \qquad (3)$$

3.3 The Transformation γ

The transformation γ is the only nonlinear component. Because γ does not mix bits with different posititions in the words, it can be considered as a parallel application of 32 transformations γ_b.

Let a and $a + d$ denote two 17-bit vectors (containing one bit from each of the state words), that are input to γ_b. Let c and $c + e$ denote the corresponding outputs: $c = \gamma_b(a), c + e = \gamma_b(a + d)$. From the definition of γ, we have the following:

$$c_i = a_i + (a_{i+1} \vee (1 + a_{i+2})), \qquad (4)$$
$$c_i + e_i = a_i + d_i + ((a_{i+1} + d_{i+1}) \vee (1 + a_{i+2} + d_{i+2})). \qquad (5)$$

Using De Morgan's law, (4) can be transformed into

$$c_i = a_i + a_{i+1}a_{i+2} + a_{i+2} + 1. \qquad (6)$$

Doing the same with (5) and combining the result with (6), we get

$$e_i + d_i + d_{i+2} + d_{i+1}d_{i+2} = a_{i+1}d_{i+2} + a_{i+2}d_{i+1}. \qquad (7)$$

Table 1 below lists the solutions to this equation, in a format that is useful further on. It shows the relation between the absolute value of the input, the input difference and the output difference.

4 Message Format for Collisions

The linear updating function of the buffer imposes strict conditions on the format of colliding messages.

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Table 1. The transformation γ_b: relation between input difference (d), output difference (e) and absolute value of the input (a). An x or y in the column of a means that this bit can take two values, e.g. $(x, 1+x)$ means 'both $(0,1)$ and $(1,0)$ are possible.'

$d_i + e_i$	d_{i+1}	d_{i+2}	(a_{i+1}, a_{i+2})
0	0	0	(x, y)
0	0	1	$(1, x)$
0	1	0	$(x, 0)$
0	1	1	(x, x)
1	0	0	-
1	0	1	$(0, x)$
1	1	0	$(x, 1)$
1	1	1	$(x, 1+x)$

4.1 Collisions for the Buffer

The buffer has a linear updating function λ that is slightly more complex than in an ordinary LFSR, $b \leftarrow \lambda(b)$ is defined by:

$$\begin{aligned} b^j &\leftarrow b^{j-1} \text{ if } j \notin \{0, 25\}, \\ b^0 &\leftarrow b^{31} + m, \\ b^{25} &\leftarrow b^{24} + r(b^{31}), \end{aligned}$$

where m is the message block that is fed into the buffer.

Therefore, the simplest collision for the buffer has the following difference pattern in the message stream (remember that the rightmost message block is the first):

$$dX, 24 \text{ zero cells}, r(dX), 0, 0, 0, 0, 0, 0, dX \ . \tag{8}$$

All difference patterns for buffer collisions are composed by adding shifted versions of pattern (8).

4.2 Collisions for the State

For a difference pattern of type (8), there will be five occasions where the input difference of the state is nonzero: the first two nonzero blocks are each injected twice in the state: once when they are the current message block and once when they pass through buffer cell 16. The last nonzero block cancels out all differences in the buffer, but is injected once in the state.

We are going to use a strategy of immediate compensation in the state: every time a difference is introduced in the state, we will try to let it die out as quickly as possible. A collision can then be seen as consisting of five 'sub-collisions', where a sub-collision is defined as a collision in the state only. This is also observed by the designers of PANAMA in [1].

Since the state update function ρ is invertible, a difference dX which is introduced can only disappear under the influence of another difference dY. An intuitive choice for the colliding messages format is the following:

$$dY, dX, 23 \text{ zero cells}, r(dY), r(dX), 0, 0, 0, 0, 0, dY, dX \ . \tag{9}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Since this format is the addition of two shifted copies of (8), it will produce a collision in the buffer. The strategy of immediate compensation demands that the difference introduced in the state by the 'dX' values, is compensated by the 'dY' values. The difference propagation should be as follows:

$$\begin{array}{ccccccc} (0,0,0) & \xrightarrow{\sigma} & (0,dX,0) & \xrightarrow{\rho} & (0,dY,0) & \xrightarrow{\sigma} & (0,0,0) \\ (0,0,0) & \xrightarrow{\sigma} & (0,r(dX),0) & \xrightarrow{\rho} & (0,r(dY),0) & \xrightarrow{\sigma} & (0,0,0) \\ (0,0,0) & \xrightarrow{\sigma} & (0,0,dX) & \xrightarrow{\rho} & (0,0,dY) & \xrightarrow{\sigma} & (0,0,0) \\ (0,0,0) & \xrightarrow{\sigma} & (0,0,r(dX)) & \xrightarrow{\rho} & (0,0,r(dY)) & \xrightarrow{\sigma} & (0,0,0) \end{array} \tag{10}$$

Note that the first propagation path applies to the first and the fifth sub-collision. It turns out that due the interaction between the buffer update operation and the state update operation, there are no solutions dX, dY for (10). A proof is given in Appendix A. Therefore we will use a difference pattern of the following form:

$$dY, 0, dX, 22 \text{ zero cells}, r(dY), 0, r(dX), 0, 0, 0, 0, dY, 0, dX \ . \tag{11}$$

Table 2 gives a schematic overview of the difference values in the buffer and the state during the attack.

5 Overview of the Procedure

We will use differences where the bits of a 32-bit word are all set or all unset. This allows to denote a difference pattern in the state with 17 bits, one for each word. Furthermore, it means that the bit rotation in π has no effect on the difference values. We make this choice mainly because it is easier to think about differences of this format.

A collision will be found by combining the results of the searches for the five sub-collisions. The method we present here to find a sub-collision works for any value of the state. Note that in every sub-collision, the message input of σ is 'fresh'; it can be chosen freely. Since the message words are added to state words 1 to 8, it is easy to control that part of the state. State word 0 is exored with the constant 00000001_x so we have no direct control over its value. The buffer input of σ is added to state words 9 to 16. This input is influenced by the values of messages that have been injected in the state earlier on.

It seems difficult to make use of the fact that the buffer can be controlled: changing the buffer would require a recalculation of the current state. Our method assumes that only the message input can be controlled. Because the message input of σ influences only 8 of the 17 state words, we have not enough degrees of freedom if we vary the current message block only. Therefore, we will also have to select values for the common message block before message blocks with difference dX or $r(dX)$.

For every sub-collision, we use the following notation. The state will be denoted by the capitals A to N, the difference values by dA to dN.

$$\begin{array}{l} K \xrightarrow{\sigma 0} L \xrightarrow{\gamma} M \xrightarrow{\pi} N \xrightarrow{\theta} A \xrightarrow{\sigma 1} B \xrightarrow{\gamma} C \xrightarrow{\pi} D \xrightarrow{\theta} E \\ E \xrightarrow{\sigma 2} F \xrightarrow{\gamma} G \xrightarrow{\pi} H \xrightarrow{\theta} I \xrightarrow{\sigma 3} J \end{array} \tag{12}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Table 2. Schematic overview of the difference propagation through the buffer cells and the state. The shown values for the buffer and the state are the values *after* message block P_n has been processed. The nonzero differences in the state during subcollision i are denoted with z_i. Because of the strategy of immediate compensation, the differences in the state are zero most of the time.

n	P_n	Buffer											State		
		0	1	2	$\cdots$	16	$\cdots$	24	25	$\cdots$	30	31	r	s	t
0	0	0	0	0		0		0	0		0	0	0	0	0
1	dX	dX	0	0		0		0	0		0	0	0	dX	0
2	0	0	dX	0		0		0	0		0	0	z_1	z_1	z_1
3	dY	dY	0	dX		0		0	0		0	0	0	0	0
4	0	0	dY	0		0		0	0		0	0	0	0	0
5	0	0	0	dY		0		0	0		0	0	0	0	0
6	0	0	0	0		0		0	0		0	0	0	0	0
7	0	0	0	0		0		0	0		0	0	0	0	0
8	$r(dX)$	$r(dX)$	0	0		0		0	0		0	0	0	$r(dX)$	0
9	0	0	$r(dX)$	0		0		0	0		0	0	z_2	z_2	z_2
10	$r(dY)$	$r(dY)$	0	$r(dX)$		0		0	0		0	0	0	0	0
11	0	0	$r(dY)$	0		0		0	0		0	0	0	0	0
12	0	0	0	$r(dY)$		0		0	0		0	0	0	0	0
13	0	0	0	0		0		0	0		0	0	0	0	0
14	0	0	0	0		0		0	0		0	0	0	0	0
15	0	0	0	0		0		0	0		0	0	0	0	0
16	0	0	0	0		0		0	0		0	0	0	0	0
17	0	0	0	0		dX		0	0		0	0	0	0	dX
18	0	0	0	0		0		0	0		0	0	z_3	z_3	z_3
19	0	0	0	0		dY		0	0		0	0	0	0	0
20	0	0	0	0		0		0	0		0	0	0	0	0
21	0	0	0	0		0		0	0		0	0	0	0	0
22	0	0	0	0		0		0	0		0	0	0	0	0
23	0	0	0	0		0		0	0		0	0	0	0	0
24	0	0	0	0		$r(dX)$		0	0		0	0	0	0	$r(dX)$
25	0	0	0	0		0		dX	0		0	0	z_4	z_4	z_4
26	0	0	0	0		$r(dY)$		0	dX		0	0	0	0	0
27	0	0	0	0		0		dY	0		0	0	0	0	0
28	0	0	0	0		0		0	dY		0	0	0	0	0
29	0	0	0	0		0		0	0		0	0	0	0	0
30	0	0	0	0		0		0	0		0	0	0	0	0
31	0	0	0	0		0		0	0		dX	0	0	0	0
32	0	0	0	0		0		$r(dX)$	0		0	dX	0	0	0
33	dX	0	0	0		0		0	0		dY	0	0	dX	0
34	0	0	0	0		0		$r(dY)$	0		0	dY	z_5	z_5	z_5
35	dY	0	0	0		0		0	0		0	0	0	0	0

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

We consider in total 4 σ operations, σn denoting the σ operation with message block Pn. Message blocks $P0$ and $P2$ are equal for both messages of the collision. For sub-collisions 1, 2 and 5 block $P1$ has the 'dX difference', and block $P3$ the 'dY' difference. For the other two sub-collisions these differences are imposed by the contents of buffer cell 16 in $\sigma 1$ and $\sigma 3$. The states K to A are equal in both messages, the common block $P0$ is used to bring the state to a value that allows a sub-collision to happen. It can be seen that the absolute value of the $P3$ blocks has no importance, for every sub-collision the difference in state I must be canceled by the difference in $P3$ (or in buffer cell 16 for $\sigma 3$).

6 The Chosen Difference Format

We give here a difference pattern that is a solution of (11). We describe the propagation of the difference from state B to I for every sub-collision. As noted above the pattern is decribed by 17 bits, one bit for each word ($0 \leq i < 17$).

The difference propagation through the linear transformations π and θ does not depend on the absolute value of the state, while $\sigma 2$ doesn't change the difference at all. The difference propagation through the nonlinear transformation γ depends on the absolute value of the state and this imposes a set of conditions on the absolute value of the state at 'time' B and F. These conditions, which can be derived using the results from Table 1, must be satisfied for the sub-collisions to occur. Note again that we can work with single bits because γ can be seen as 32 parallel transformations γ_b.

Table 3 below shows the required difference propagation for sub-collision 1 (and 5). In this table state B has the difference dX in part s, as imposed by message block $P1$. State I has the difference dY in part s, which will be canceled by message block $P3$. Parts r ($i = 0$), s ($i = 1, \ldots 8$) and t ($i = 9, \ldots 16$) are clearly seperated in the table. Table 1 has been used to derive the following conditions on B and F for the difference propagation of Table 3.

$$B_5 = 1, B_6 + B_7 = 1, B_8 = 0, \tag{13}$$

$$\begin{aligned} &F_0 + F_1 = 1, F_2 = 1, F_3 = 0, F_5 = 0, F_6 = 0, F_7 = F_8, F_8 + F_9 = 1, \\ &F_{10} = 0, F_{11} = 0, F_{12} = F_{13} = F_{14}, F_{15} = 1, F_{16} = 1. \end{aligned} \tag{14}$$

For the other sub-collisions we refer to the similar Tables 4 to 5 and conditions (35) to (40) in Appendix B. All these sub-collisions are related because in every case state B contains the difference dX or $r(dX)$ in part s (imposed by the message) or part t (imposed by buffer cell 16), while state I must contain the difference dY respectively $r(dY)$ in the corresponding part of the state.

Other solutions for the difference format are possible. We have chosen one which results in the minimum number of conditions on B and F. It is not clear how to make an optimal choice for an easy solution of these conditions in the next step of the attack (section 7).

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Table 3. The required difference propagation for sub-collision 1 (and 5).

i	dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	0	0	0
1	0	0	1	1	1	0	0
2	0	0	0	0	1	1	1
3	0	0	0	0	0	1	1
4	0	0	0	1	1	0	0
5	0	1	0	0	1	1	1
6	1	1	0	0	0	1	0
7	1	1	0	1	0	0	0
8	0	0	1	1	1	1	1
9	0	0	0	1	1	1	0
10	0	0	0	0	1	1	0
11	0	0	0	0	0	1	0
12	0	0	0	1	1	1	0
13	0	0	1	1	0	0	0
14	0	0	0	1	1	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	1	1	0

7 Producing collisions

In order to generate each of the sub-collisions we need to solve the conditions for the absolute state values at 'time' B and F as given in section 6 and Appendix B. Values for state words 1 to 8 (part s) can be met by a suitable choice of the message block used at the corresponding time: $P1$ for B and $P2$ for F. To meet the values for the other state words (0 and 9 to 16, parts r and t) we need to calculate backwards to the previous message block: $P0$ for B and $P1$ for F. In this way the problem of finding a sub-collision can be solved in three steps:

1. solve a system of equations in the unknowns $L_1, \ldots L_8$, this determines message block $P0$,
2. solve a system of equations in the unknowns $B_1, \ldots B_8$, this determines message block $P1$,
3. solve a system of equations in the unknowns $F_1, \ldots F_8$, this determines message block $P2$.

The difficulty in solving these systems of equations is that when we calculate backwards to the previous message blocks, we obtain increasingly nonlinear equations because of the transformation γ. In our approach we only go back through one application of γ. To explain how we solve the problem, we will first describe the solution of the first sub-collision in some detail. Next we will summarize the procedure for the other sub-collisions and discuss the complexity of the attack.

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

7.1 Producing sub-collision 1 (and 5)

From (14) it can be seen that there are 7 conditions in the unknowns $F_1, \dots F_8$. Let $[j]$ denote the bit position, then we have the following equations for $0 \le j < 32$:

$$\begin{aligned} &F_0[j] + F_1[j] = 1, F_2[j] = 1, F_3[j] = 0, F_5[j] = 0, \\ &F_6[j] = 0, F_7[j] = F_8[j], F_8[j] + F_9[j] = 1. \end{aligned} \tag{15}$$

These conditions are easy to satisfy because $F_1, \dots F_8$ are obtained by bitwise addition of $E_1, \dots E_8$ and message block $P2$ which can be freely chosen.

In (14) we have 6 remaining conditions on $F_0, F_9, \dots F_{16}$, which can be transformed into equations in the unknowns $B_1, \dots B_8$. These have to be added to the 3 conditions we already have in these unknowns from (13),

$$B_5[j] = 1, B_6[j] + B_7[j] = 1, B_8[j] = 0, \tag{16}$$

resulting in an overdetermined system of 9 equations (and 32 bit positions). Generally such a system of equations has no solution, unless if one of the equations is dependent on the other eight. If there is no hidden structure in these equations, we can assume that we will need 2^{32} trials before we can solve this system of equations. Experimental evidence confirms this assumption.

To calculate backwards from F to B we apply the transformations σ, θ, π and γ, which lead to the following equations. Here $mb_i[j] = b^{16}_{i-9}[j]$ for $9 \le i \le 16$, $mb_0[j] = 0$ for $0 \le j \le 30$ and $mb_0[31] = 1$.

$$\begin{aligned} &F_i[j] = mb_i[j] + E_i[j] \quad (i = 0, 9 \dots 16), \\ &E_i[j] = D_i[j] + D_{i+1}[j] + D_{i+4}[j], \\ &D_i[j] = C_{7i}[j + r(i)], \\ &C_i[j] = B_i[j] + B_{i+1}[j]B_{i+2}[j] + B_{i+2}[j] + 1. \end{aligned} \tag{17}$$

Using these equations, for example the condition $F_{10}[j] = 0$ can be transformed in the following equation in the unknowns B_2, B_3 and B_4:

$$\begin{aligned} &B_2[j + 23] + B_3[j + 23]B_4[j + 23] + B_4[j + 23] \\ &+ B_9[j + 2] + B_{10}[j + 2]B_{11}[j + 2] + B_{11}[j + 2] \\ &+ B_{13}[j + 9] + B_{14}[j + 9]B_{15}[j + 9] + B_{15}[j + 9] \\ &+ mb_{10}[j] + 1 = 0. \end{aligned} \tag{18}$$

It is because of the different bit rotations in this equation (and others), that we need to solve the equations bit per bit.

In order to simplify the system of 9 equations in $B_1, \dots B_8$, we impose an extra condition $B_0[j] = 0$, which can be transformed into the equation:

$$\begin{aligned} &L_0[j] + L_1[j]L_2[j] + L_2[j] \\ &+ L_7[j + 1] + L_8[j + 1]L_9[j + 1] + L_9[j + 1] \\ &+ L_{11}[j + 10] + L_{12}[j + 10]L_{13}[j + 10] + L_{13}[j + 10] \\ &+ mb_0[j] + 1 = 0, \end{aligned} \tag{19}$$

in the unknowns L_1, L_2, L_7 and L_8. Starting from an arbitrary but specified state K and buffer, the values $L_0, L_9, \dots L_{16}$ are fixed, and condition (19) can

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

easily be satisfied, by picking random values for $L_1, \dots L_6, L_8$ and calculating L_7. This determines message block $P0$, and allows us to calculate states M, N, A and $B_0, B_9, \dots B_{16}$.

Returning to the system of equations in $B_1, \dots B_8$ we have already seen that there is a requirement on 32 bits. It turns out that there is another complication because by replacing $B_6[j]$ by $B_7[j] + 1$ (see (16)), we end up with an equation in two bit positions of the same state word B_7, of the form:

$$B_7[j+27] + B_7[j+1] = \dots,$$

which translates to two requirements, one for bitwise addition of all values obtained with even j, and one for bitwise addition of all values obtained with odd j. This can be seen as a requirement on 2 bits. Hence the probability that a solution for this system of equations can be found is $1/2^{32+2}$. We can try random values of the initial state K until this happens and at that time message blocks $P0$ and $P1$ are determined. We can then calculate states C, D, E and $F_0, F_9, \dots F_{16}$, and choose a suitable value of message block $P2$ to satisfy (15) (the conditions on $F_1, \dots F_8$).

The complexity to find sub-collision 1 (or 5) with this procedure is 2^{34}.

7.2 Producing the other sub-collisions

Sub-collision 2. In this case there are 6 conditions in the unknowns $F_1, \dots F_8$, which can easily be met by choosing message block $P2$. The 7 remaining conditions on $F_0, F_9, \dots F_{16}$, are added to the 3 existing conditions in the unknowns $B_1, \dots B_8$, resulting in an overdetermined system of 10 equations, which leads to a requirement for $2 \times 32 = 64$ bits. The system is simplified by specifying $B_0[j] = B_9[j] = 0$, which leads to 2 extra conditions in the unknowns $L_1, \dots L_8$, which are easily solved with a suitable choice of message block $P0$. The nonlinearity of the system for $B_1, \dots B_8$ imposes two more requirements to solve it: one for 2 bits (similar to the case for sub-collision 1), and another for 16 bits. This last requirement comes from an equation in the unknown B_8 of the following form:

$$B_7[j+27]B_8[j+27] + B_8[j+27] = \dots,$$

where B_7 has already been solved. It can be seen that for $B_7[j+27] = 0$ we can calculate $B_8[j+27]$, but for $B_7[j+27] = 1$ all unknowns drop from the equation. On average this happens for 16 bits and we end up with a requirement that has to be satisfied in order to get a solvable system of equations. Hence the probability that a solution can be found is $1/2^{32+32+2+16}$, and the complexity for finding this sub-collision is 2^{82}.

Sub-collision 3. There are 7 conditions in the unknowns $F_1, \dots F_8$, which can easily be met by choosing message block $P2$. The 7 remaining conditions on $F_0, F_9, \dots F_{16}$, lead to 7 equations in the unknowns $B_1, \dots B_8$. We simplify the system by specifying $B_0[j] = 0$, which added to the 3 existing conditions in

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

$B_9, \dots B_{16}$, leads to 4 equations in the unknowns $L_1, \dots L_8$, which are easy to solve with a suitable choice of message block $P0$. The nonlinear system for $B_1, \dots B_8$ can be solved with two requirements for 32 and 16 bits respectively. The 32-bit requirement comes from the fact that one equation specifies the same unknown as a set of three other equations. The complexity for finding this sub-collision is $2^{32+16} = 2^{48}$.

Sub-collision 4. This case is similar to the previous one, except there are only 4 conditions in the unknowns $F_1, \dots F_8$, and the system of 4 equations in the unknowns $L_1, \dots L_8$ imposes another requirement of 32 bits, which raises the complexity to $2^{32+16+32} = 2^{80}$.

7.3 Complexity of the attack

As seen above the complexities for finding the 5 sub-collisions are 2^{34}, 2^{82}, 2^{48}, 2^{80} and 2^{34} respectively. There is no problem in connecting the sub-collisions, as we only need an arbitrary initial state for each, which can be obtained by choosing random message blocks between the sub-collisions. So the total complexity for our collision-finding attack is determined by sub-collision 2 (which is the most difficult to find). This complexity is about 2^{82}.

We tested this attack for a reduced version of PANAMA where all words have a bitlength of 8 instead of 32 (so the hash has length 64), and this confirmed the given complexities (for the 8-bit version the complexity is about 2^{22}). Sub-collisions 1, 3 (and 5) were also tested for a 16-bit version which again confirmed the complexity.

We believe improvements to this attack are possible. First of all better methods to solve the systems of nonlinear equations can be looked for. Furthermore, our attack still has a lot of freedom. We can calculate further backwards which has the advantage that we get more unknowns, but the difficulty that we get more complex nonlinear equations.

8 Conclusion

We have presented a method for producing collisions for PANAMA. The complexity of our current attack is 2^{82}. Although this is still too high to actually find collisions for PANAMA, it is much faster than a general birthday attack which would require on the order of 2^{128} operations. Furthermore, we believe this attack can be improved, and because of the large degree of freedom we have, we think it is likely that with additional effort collisions can actually be found for PANAMA.

In order to improve the security of PANAMA, the design could be altered so that the message influences a smaller part of the state (e.g., exor only 4 message words into the state at every application of σ). Instead, more buffer cells could be used as input of σ. The consequence for the attacker is that he has a smaller degree of freedom, and therefore needs to go further backwards in the attack

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

to obtain enough unknowns, which gives him more complex equations to solve. Preferrably, the different buffer cells that would be used as input of σ, should be selected in such a way that the different sub-collisions can no longer be treated independently (cf. Table 2). Of course this would decrease the performance of the algorithm. Furthermore, there still is a fundamental problem that we have the freedom to choose message blocks, and are able to go backwards in the attack by choosing message blocks before the sub-collision. To obtain confidence in the security of the algorithm, it seems necessary to ensure that we need to consider so many message blocks for one sub-collision so that the attacks on the different sub-collisions interfere with each other, making this strategy impossible.

Acknowledgements

The authors wish to thank Lars R. Knudsen for helpful discussions.

References

1. J. Daemen and C.S.K. Clapp, "Fast hashing and stream encryption with PANAMA," *Fast Software Encryption, LNCS 1372,* S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 60–74.

A Collisions in two steps are not possible

We show here why there are no solutions for (9). In this case the 'timeline' (12) reduces to:

$$A \xrightarrow{\sigma 0} B \xrightarrow{\gamma} C \xrightarrow{\pi} D \xrightarrow{\theta} E \xrightarrow{\sigma 1} F \tag{20}$$

Here $\sigma 0$ introduces the 'dX difference', and $\sigma 1$ the 'dY difference' which must cancel the difference in state E for every sub-collision.

We start by considering the first sub-collision. We know that for any value of $P0$ and dX,

$$dB_i = 0, \text{ for } i = 0, 9, 10, 11, \ldots, 16. \tag{21}$$

Since γ does only mix nearby words it follows that

$$dC_i = 0, \text{ for } i = 9, 10, 11, \ldots 15. \tag{22}$$

Applying the definition of π gives the following:

$$dD_i = 0, \text{ for } i = 2, 4, 7, 9, 11, 14, 16. \tag{23}$$

With (2), this can be translated to the following conditions on dE_i:

$$\begin{aligned}
dE_3 + dE_4 + dE_7 + dE_{11} + dE_{12} + dE_{13} + dE_{14} + dE_{16} + dE_1 &= 0\\
dE_5 + dE_6 + dE_9 + dE_{13} + dE_{14} + dE_{15} + dE_{16} + dE_1 + dE_3 &= 0\\
dE_8 + dE_9 + dE_{12} + dE_{16} + dE_0 + dE_1 + dE_2 + dE_4 + dE_6 &= 0\\
dE_{10} + dE_{11} + dE_{14} + dE_1 + dE_2 + dE_3 + dE_4 + dE_6 + dE_8 &= 0\\
dE_{12} + dE_{13} + dE_{16} + dE_3 + dE_4 + dE_5 + dE_6 + dE_8 + dE_{10} &= 0\\
dE_{15} + dE_{16} + dE_2 + dE_6 + dE_7 + dE_8 + dE_9 + dE_{11} + dE_{13} &= 0\\
dE_0 + dE_1 + dE_4 + dE_8 + dE_9 + dE_{10} + dE_{11} + dE_{13} + dE_{15} &= 0
\end{aligned} \tag{24}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Fulfilling (9) also requires that $dF_i = 0, \forall i$. Without specifying $P1$ or dY_i, we have conditions on dE:

$$dE_i = 0, \text{ for } i = 0, 9, 10, 11, \ldots, 16. \tag{25}$$

Furthermore,

$$dE_i = dY_{i-1}, \text{ for } i = 1, 2, \ldots, 8. \tag{26}$$

Combining (24), (25) and (26), gives us the following conditions on dY:

$$\begin{aligned} dY_2 + dY_3 + dY_6 + dY_0 &= 0 \\ dY_4 + dY_5 + dY_0 + dY_2 &= 0 \\ dY_7 + dY_0 + dY_1 + dY_3 + dY_5 &= 0 \\ dY_0 + dY_1 + dY_2 + dY_3 + dY_5 + dY_7 &= 0 \\ dY_2 + dY_3 + dY_4 + dY_5 + dY_7 &= 0 \\ dY_1 + dY_5 + dY_6 + dY_7 &= 0 \\ dY_0 + dY_3 + dY_7 &= 0 \end{aligned} \tag{27}$$

This can be transformed into:

$$dY_2 = 0, dY_6 = dY_7, dY_5 = dY_1, dY_3 + dY_6 + dY_0 = 0, dY_4 + dY_1 + dY_0 = 0. \tag{28}$$

We can do the same excercise for sub-collision 3, where the same differences are injected in the state, but now via the buffer tap. We get the following:

$$dB_i = 0, \text{ for } i = 0, 1, 2, \ldots, 8 \tag{29}$$

$$dC_i = 0, \text{ for } i = 0, 1, 2, \ldots, 6 \tag{30}$$

$$dD_i = 0, \text{ for } i = 0, 3, 5, 8, 10, 13, 15. \tag{31}$$

$$\begin{aligned} dE_1 + dE_2 + dE_5 + dE_9 + dE_{10} + dE_{11} + dE_{12} + dE_{14} + dE_{16} &= 0 \\ dE_4 + dE_5 + dE_8 + dE_{12} + dE_{13} + dE_{14} + dE_{15} + dE_0 + dE_2 &= 0 \\ dE_6 + dE_7 + dE_{10} + dE_{14} + dE_{15} + dE_{16} + dE_0 + dE_2 + dE_4 &= 0 \\ dE_9 + dE_{10} + dE_{13} + dE_0 + dE_1 + dE_2 + dE_3 + dE_5 + dE_7 &= 0 \\ dE_{11} + dE_{12} + dE_{15} + dE_2 + dE_3 + dE_4 + dE_5 + dE_7 + dE_9 &= 0 \\ dE_{14} + dE_{15} + dE_1 + dE_5 + dE_6 + dE_7 + dE_8 + dE_{10} + dE_{12} &= 0 \\ dE_{16} + dE_0 + dE_3 + dE_7 + dE_8 + dE_9 + dE_{10} + dE_{12} + dE_{14} &= 0 \end{aligned} \tag{32}$$

Now we have that $dE_i = 0$, for $i = 0, \ldots, 8$ and $dE_i = dY_{i-9}$, for $i = 9, \ldots, 16$.

$$\begin{aligned} dY_0 + dY_1 + dY_2 + dY_3 + dY_5 + dY_7 &= 0 \\ dY_3 + dY_4 + dY_5 + dY_6 &= 0 \\ dY_1 + dY_5 + dY_6 + dY_7 &= 0 \\ dY_0 + dY_1 + dY_4 &= 0 \\ dY_2 + dY_3 + dY_6 + dY_0 &= 0 \\ dY_5 + dY_6 + dY_1 + dY_3 &= 0 \\ dY_7 + dY_0 + dY_1 + dY_3 + dY_5 &= 0 \end{aligned} \tag{33}$$

Combining these results with the results from sub-collision 1 (28), we get:

$$dY_0 = dY_1 = dY_2 = dY_4 = dY_5 = 0, dY_3 = dY_6 = dY_7. \tag{34}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Because sub-collison 2 and 4 will result in similar conditions on $r(dY)$, the only valid solution is $dY = 0$. It's a bit surprising that combining the results of sub-collision 1 and 3 alone does not suffice to reach this conclusion; we get 7 equations for each sub-collision, but they are not independent, e.g. 2 of the 7 equations from sub-collision 1 are redundant.

B Difference propagation for sub-collisions 2 to 4

We give here Tables 4 and 5 with the required difference propagation for sub-collisions 2, 3 and 4, as well as the corresponding conditions on the absolute value of the state at 'time' B and F. Sub-collision 1 was described in section 6, where we also discussed the relations between Tables 3 to 5.

Table 4. The required difference propagation for sub-collision 2.

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	1	0
0	0	0	0	1	0	1
0	0	0	1	1	0	0
0	1	1	1	0	1	1
1	1	0	1	1	1	0
1	1	0	0	0	1	0
0	0	0	0	1	1	1
0	0	0	1	0	1	0
0	0	1	1	1	0	1
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	1	1	1	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	0	0

The difference propagation in sub-collision 2, as specified in Table 4, leads to the following conditions for B:

$$B_3 = 1, B_4 + B_5 = 1, B_6 = 0, \tag{35}$$

and the following conditions for F:

$$\begin{aligned} &F_0 = 0, F_1 = 0, F_2 + F_3 = 1, F_3 = F_4, F_5 = 1, F_6 = 1, F_7 + F_8 = 1, \\ &F_9 = 1, F_{10} = 0, F_{12} = 0, F_{13} = 1, F_{14} = F_{15}, F_{15} + F_{16} = 1. \end{aligned} \tag{36}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag

Table 5. The required difference propagation for sub-collision 3 (left) and sub-collision 4 (right).

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	1	0
0	0	0	1	0	1	0
0	0	1	1	1	1	0
0	0	0	1	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	1	1	0	0
0	0	1	1	1	1	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	1
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
1	1	1	1	1	1	0
1	1	0	1	1	0	0
0	0	0	0	0	1	1

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	0	1	1	0
0	0	0	0	1	1	0
0	0	0	1	1	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	1	1	0	1
0	1	0	1	0	0	0
0	0	0	0	1	1	1
1	1	0	1	0	1	0
1	1	0	1	1	1	0
0	0	1	1	1	1	1
0	0	0	1	0	1	0
0	0	1	1	1	0	1

The difference propagation in sub-collision 3, as specified in Table 5 (left), leads to the following conditions for B:

$$B_{13} = 0, B_{14} + B_{15} = 1, B_{16} = 0, \tag{37}$$

and the following conditions for F:

$$\begin{gathered} F_0 = 1, F_1 + F_2 = 1, F_2 + F_3 = 1, F_4 = 0, F_5 + F_6 = 1, F_6 + F_7 = 1, F_7 = F_8, \\ F_8 = F_9, F_9 + F_{10} = 1, F_{11} = 0, F_{12} = 1, F_{13} = F_{14} = F_{15}, F_{16} = 0. \end{gathered} \tag{38}$$

The difference propagation in sub-collision 4, as specified in Table 5 (right), leads to the following conditions for B:

$$B_{11} = 0, B_{12} = B_{13}, B_{14} = 0, \tag{39}$$

and the following conditions for F:

$$\begin{gathered} F_0 = 1, F_4 = 0, F_6 = 1, F_7 = 0, F_8 + F_9 = 1, F_8 = F_{10}, \\ F_{11} = 0, F_{12} + F_{13} = 1, F_{12} = F_{14}, F_{12} = F_{15}, F_{12} = F_{16}. \end{gathered} \tag{40}$$

Appeared in *Fast Software Encryption, FSE 2001*, Lecture Notes in Computer Science 2355, M. Matsui (ed.), Springer-Verlag, pp. 37–51, 2002.
©2002 Springer-Verlag