Preimage attack on Boole-n

Ivica Nikolić

University of Luxembourg

Abstract. Boole is a family of hash functions proposed for SHA-3. In this paper we present a preimage attack on Boole-n that requires $2^{\frac{9n}{16}}$ computations and negligible memory.

1 Description of Boole

Boole family of hash functions [2] is based on a stream design. Internally, Boole has a large state of 16 words[1] $\sigma_t = (R_t[0], R_t[1], \ldots, R_t[15])$ and 3 additional word accumulators denoted by l_t,x_t, and r_t(t is the time). Hashing a message in Boole is done in three phases: 1)Input phase, where the state and the accumulators are updated for each input word, 2)mixing phase, where only the state is updated depending on the accumulators, 3)output phase, where the output is produced. In this phase, the accumulators are not used.
The update of the state, referred as a *cycle*, is defined as :

$$\begin{aligned} R_{t+1}[i] &\leftarrow R_t[i+1], \text{for } i = 1 \ldots 14 \\ R_{t+1}[15] &\leftarrow f_1(R_t[12] \oplus R_t[13]) \oplus (R_t[0] \lll 1) \\ R_{t+1}[0] &\leftarrow R_t[0] \oplus f_2(R_t[2] \oplus R_t[15]) \end{aligned}$$

Note that this is an invertible function, i.e. if the new state σ_{t+1} is given, it is easy to find the previous state σ_t.
Let w_t be a message word. Then, the update of the accumulators is defined as:

$$\begin{aligned} temp &\leftarrow f_1(l_t) \oplus w_t \\ l_{t+1} &\leftarrow temp \lll 1 \\ x_{t+1} &\leftarrow x_t \oplus w_t \\ r_{t+1} &\leftarrow (r_t \oplus temp) \ggg 1 \end{aligned}$$

The update of the accumulators is also an invertible function, i.e. if we fix new accumulators $l_{t+1}, x_{t+1}, r_{t+1}$ and the input message word w_t we can find the previous accumulators l_t, x_t, r_t. Moreover, if we fix l_t, l_{t+1} (r_t, r_{t+1}) we can easily find the message word w_t. This property is often used in our preimage attack.
The message is absorbed in the input phase. Sequentially, for each message word the following is done:

1. update the accumulators

[1] Boole-224 and Boole-256 use 32-bit words. Boole-385 and Boole-512 use 64-bit words.

2. $R[3] \leftarrow R[3] \oplus l$
3. $R[13] \leftarrow R[13] \oplus r$
4. update the state (*cycle*)

The description of the mixing phase, as well as the description of f_1, f_2, has no importance in our attack. It is only relevant that this phase (functions) is invertible.

Each iteration of the output phase produces one output word. One iteration is defined as:

1. *cycle*
2. Output the word $v = R[0] \oplus R[8] \oplus R[12]$

For example, the output for Boole-256 is produced in 8 iterations.

2 Preimage attack on Boole

For finding preimage of a target hash value H^* in Boole we will use the meet-in-the-middle (MITM) attack. We will start from the initial state and by changing the input words produce a set S_1 of intermediate state values. Similarly, starting from the target hash value and going through the output and mixing phase we will produce one state. Then from this state, by taking different input words and still going backwards, we will produce a set S_2 of intermediate state values. If these two sets have at least one common element then we will have a preimage for H^*.

Note that the intermediate state in case of Boole has 16 state words and 3 accumulators, hence 19 words in total. If we apply straightforward the MITM technique to Boole-n (with words of $\frac{n}{8}$ bits), we will end up with an attack that requires $2^{\frac{19}{2} * \frac{n}{8}} = 2^{\frac{19}{16} n}$ computations, an effort higher than the simple brute force. Hence we have to reduce the size of the intermediate space for the MITM technique. This is done by fixing the state words $R[3], R[4], \ldots, R[12]$ to zero[2]. Then the elements of the sets S_1 and S_2 will differ only in $R[0], R[1], R[2], R[13], R[14], R[15], l, x, r$ (9 words in total) and the MITM attack will require $2^{\frac{9}{2} * \frac{n}{8}} = 2^{\frac{9n}{16}}$ computations. Further we will show how to fix the words $R[3], \ldots, R[12]$ in forward and backward directions. We will fix the words for $t = 10$.

2.1 Fixing $R[3], R[4], \ldots, R[12]$ forwards

From the description of the accumulator update function we can see that it is easy to find a message input word w such that the value of the $r-$accumulator after the update will be any prefixed.

From the description of the input phase is follows that $R_{10}[3] = R_9[4] = \ldots = R_1[12] = R_0[13] \oplus r_1$. Hence, we can control the value of $R_{10}[3]$ by defining the value of r_1. Similarly, for $R_{10}[4]$ we get that $R_{10}[4] = R_9[5] = \ldots = R_2[12] = R_1[13] \oplus r_2$. Again, we can control this value with r_2. The same reasoning can be applied for fixing the values of $R_{10}[5], \ldots, R_{10}[12]$. Note that we can not control the values of more than these 10 words because when we fix the value of r_t with the input word w_t, it means that we have fixed the value of l_t also. But when the value of l_t is added to $R[3]$ (which is fixed to zero) it will produce a new value not necessary equal to zero.

[2] Actually, any values can be taken.

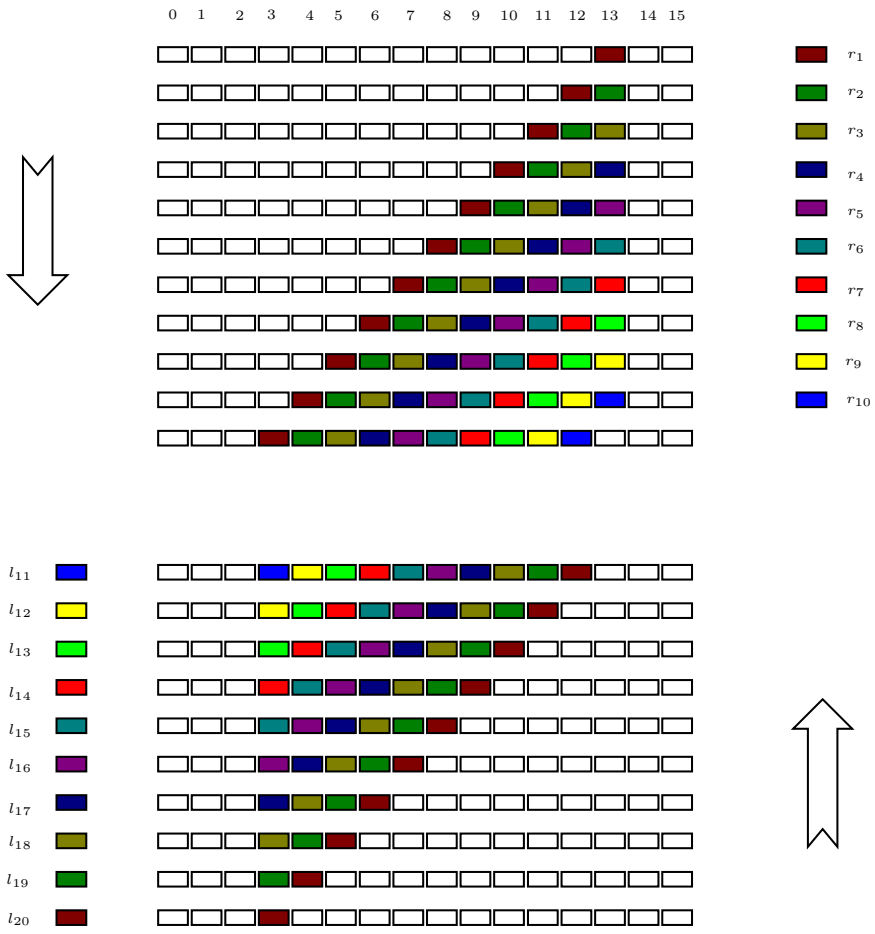

Figure 1: Meet-in-the-middle attack on Boole.

2.2 Fixing $R[3], R[4], \ldots, R[12]$ backwards

Before explaining how to fix these values when going backwards, let us first deal with reversing the final output phase.
In each *cycle* of this phase one output word is produced. Hence, the digest is produced in 8 cycles[3]. The output word v_t is defined as $v_t = R_t[0] \oplus R_t[8] \oplus R_t[12]$. Let $H^* = (h_0, \ldots, h_7)$ is the target hash value. We have to construct a state $\sigma_t = (R_t[0], \ldots, R_t[15])$ such that $h_0 = v_t, h_1 = v_{t+1}, \ldots, h_7 = v_{t+7}$. We put any values in $R_t[0], R_t[9], R_t[10], \ldots, R_t[15]$.

[3] For Boole-224 and Boole-384, the output is produced in 7 and 6 cycles, respectively.

Then, we find $R_t[8]$ from the equation $R_t[8] = R_t[0] \oplus R_t[12] \oplus h_0$. Obviously we get that $v_t = h_0$. After the *cycle* update we find the value of $R_t[1]$ from the equation $R_t[1] = R_{t+1}[0] = R_{t+1}[8] \oplus R_{t+1}[12] \oplus h_1$. The values for $R_t[2], \ldots, R_t[7]$ can be found similarly. This way we have defined the rest of the word in the state σ_t. Therefore, we have inverted the output phase for a state that produces the required target hash value.
Let us fix the accumulators to any values. Then, inverting the mixing phase is trivial because the length of the preimage is fixed[4] and the accumulators are fixed.
After we have obtained **one** valid state σ_{16} that produces the target, from this state, still by going backward, we can start building the different intermediate states for the meet-in-the-middle attack. We will fix the values of $R_{10}[3], R_{10}[4], \ldots, R_{10}[12]$ to zero by controlling the values of the $l-$accumulators. A similar technique as in forward direction is used. From the definition of the updating function we get that $R_{10}[12] = R_{11}[11] = \ldots = R_{18}[4] = R_{19} \oplus l_{20}$. Therefore if we take $l_{20} = R_{19}$ we will get $R_{10}[12]$. Similarly, for $R_{10}[11]$ we have $R_{10}[11] = R_{11}[10] = \ldots = R_{17}[4] = R_{18}[3] \oplus l_{19}$. Again, we take $l_{19} = R_{18}[3]$ and obtain $R_{10}[11] = 0$. The same method can be used to fix the variables $R_{10}[10], \ldots, R_{10}[3]$.

2.3 Meet-in-the-middle

We have obtained two sets S_1 and S_2 of intermediate state values with fixed to zero registers $R[3], \ldots, R[12]$. Hence the attack space for MITM attack is only 9 words (each word is $n/8$ bits). Therefore, if both S_1 and S_2 have at least $2^{\frac{9n}{16}}$ different elements, then with high probability these sets have one common element and therefore a preimage for the target hash value can be found.

2.4 Complexity and memory requirements

If in our attack we use the classical MITM approach than we will need $2^{\frac{9n}{16}}$ computations and memory. Yet, it is possible to launch this attack with negligible memory. This is done by using the memoryless MITM attack described in [1]. We will try to describe in short how this attack works. The memoryless version, similarly like the memoryless version of the collision search attack [3], uses the cycle finding algorithm. Recall that this algorithm deals only with one function. In the MITM attack we have two functions: 1)$f(x)$ - forwards from the initial value, and 2)$g(x)$ - backwards from the target hash value. Hence, in order to launch the memoryless birthday attack, a switching function $h(x)$ is introduced. It is defined as:

$$h(x) = \begin{cases} f(x), \text{if } cr(x) = 0 \\ g(x), \text{if } cr(x) = 1 \end{cases}$$

[4] For Boole-224 and Boole-256 it is 512 bits (16 rounds: 8 in forward direction and 8 in backward). For Boole-384 and Boole-512 it is 1024 bits.

where $cr(x)$ is some random function that outputs 0 and 1 with probability 1/2. Then, by straightforward application of the memoryless collision search algorithm, we can easily find a collision among the sets S_1 and S_2 and therefore a preimage for the hash value. Now, let us specify the functions $f(x)$ and $g(x)$ for our case. Let x in $f(x)$ (forward direction) be the first 9 input words. Hence, $f(x)$ means we simply start from different initial value. Then, we use the technique described to fix $R[3], \ldots, R[12]$ in the following 10 iterations. Similarly, let x in $g(x)$ (backward direction) be the last 9 input words. So, when we go backwards, after the output and mixing phases, we go additional 9 iterations and we assume that the input words are words from x. Then, after we have obtained some state, we use our technique to fix $R[3], \ldots, R[12]$ in backward direction. Note that this way, our preimage has length of $9 + 9 + 10 + 10 = 38$ words. The attack requires only $2^{\frac{9n}{16}}$ computations and negligible memory.

References

1. H. Morita, K. Ohta, S. Miyaguchi: A switching closure test to analyze cryptosystems Advances in Cryptology CRYPTO 1991, LNCS 576, Springer-Verlag, 1992, p. 183-193.
2. G. G. Rose:Design and Primitive Specification for Boole. `http://seer-grog.net/BoolePaper.pdf`
3. J.-J. Quisquater and J.-P. Delescaille:How easy is collision search. New results and applications to DES. Advances in Cryptology - CRYPTO 1989, LNCS 435, G. Brassard, Ed., Springer-Verlag, 1990, pp. 408413.