# Easing collision finding in cryptographic primitives with genetic algorithms

Julio César Hernández, Pedro Isasi and Arturo Ribagorda
Security Group & Artificial Intelligence Group, Computer Science Dpt.
Carlos III University, 28911 Leganés Madrid Spain
{jcesar,arturo}@inf.uc3m.es, isasi@ia.uc3m.es

**Abstract - The finding of collisions (i.e. different inputs that map to the same output) in cryptographic primitives (hash functions or block ciphers) is an extremely difficult task. It generally requires hundreds or thousands of hours of a talented cryptanalyst. Even in this case, results are not always guaranteed. In this work, we will present a new method for easing the find of collisions, based in the use of genetic algorithms. Our method automatically seeks for correlations between the input and the output bits that can be used for producing pseudocollisions (i.e. collisions of parts of the output). These pseudocollisions are then useful for creating a full output collision. These ideas are shown to work over a version of the block cipher TEA reduced to one round.**

## I. INTRODUCTION

A cipher [1] is an algorithm characterised by a tuple ($\Pi$, $\Gamma$, K, $\epsilon$, $\Delta$) that verifies:

1. $\Pi$ is the finite set of plaintexts
2. $\Gamma$ is the finite set of ciphertexts
3. K is the finite set of possible keys
4. For every $\kappa$ in K there is a ciphering function $\varepsilon_\kappa$ in $\epsilon$ and a deciphering function $\delta_\kappa$ in $\Delta$ that verifies $\delta_\kappa(\varepsilon_\kappa(\pi))=\pi$ for every plaintext $\pi$ de $\Pi$.

The fundamental property here is number 4, which implies that the original plaintext can be recovered if the ciphering key is known. This knowledge of the ciphering key must not only be sufficient for recovering the text, but also necessary. Any robust cipher must not allow the recovery of plaintext without the knowledge of the key used.

Block ciphers are a special class of ciphers, characterised for operating over bit blocks instead of over single bits as the stream ciphers do. This block length (typically 64, 128 or 256 bits) and the key length used to cipher (typically 64 or 128 bits) are fixed and fundamental for the block cipher's strength.

An ideal block cipher must behave as a random mapping and, if we fix any given key, as a random permutation. One of the additional properties that are needed for block ciphers and in hash functions (functions that can take an input of arbitrary length and generate an output of a fixed length), is to be collision resistant. Essentially, this means that finding inputs $x \neq x'$ such as $f(x)=f(x')$ is computationally infeasible. This does not mean collisions do not exist (they trivially exist if the input space is greater than the output space), only that they must be extremely difficult to find.

If a block cipher or a hash function can be proved not to be collision-resistant, it will immediately be disregarded for any cryptographic use. The security of many cryptographic protocols, notably the digital signing of documents and some micropayments protocols, is totally reliant on this property.

However, even if the primitives have a relatively poor security or are simplified (i.e. the number of rounds is reduced, etc.), finding collisions is usually quite hard. For helping with this, we present a new technique that is based on genetic algorithms. We will also show it is useful in simplifying the task of finding collisions in a reduced round version of the block cipher TEA.

## II. THE TEA ALGORITHM

TEA stands for Tiny Encryption Algorithm. It is the name of a block cipher invented by David Wheeler and Roger Needham, members of the Computer Security Laboratory of Cambridge University. It was presented in the 1994 Fast Software Encryption Workshop [2].

TEA is a very fast block cipher that does not use predefined tables or s-boxes and does not need much initialisation time. It is a Feistel type algorithm, thus named because it divides its input in two halves and operates over them individually in each round and interchanges them at the end of every round. It works over 64 bit blocks and uses keys of 128 bits, which are large enough for today's security standards.

As they authors say, it has a security (with 8, 16 or 32 rounds) at least comparable with DES (then the Data Encryption Standard) and it is quite faster. However, it seems that the block cipher TEA has some additional advantages: it is very robust (only one *academic* attack [3], that is, a theoretical attack with little or none practical implications) in more than 7 years of existence. Additionally, it is very portable, efficient and simple as its compact code below shows:

0-7803-7282-4/02/$10.00 ©2002 IEEE

```
void code(long* v, long* w, long* k)
{unsigned long y=v[0],z=v[1], sum=0,
 delta=0x9e3779b9, n=8 ;
 while (n-->0)
{sum += delta ;
 y += (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
 z += (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;
 }
w[0]=y ; w[1]=z ;
}
```

## III. A General Procedure for Finding Collisions

The fundamental idea is to select a subset of the output (generally 8 or 16 consecutive bits) and then search for the subsets of the input space that have a greater *influence* over (or a higher correlation with) these observed output bits. This knowledge can be used to produce a pseudo collision over them, that is, to determine, if possible, a subset of the input that always generates the same output over the observed bits. After this, we must select other subsets of the output bits (normally disjoint with the previous) and repeat the search for input subsets that produce a pseudo collision on them.

Repeating this procedure, we will eventually cover all the output bits. Then, we would have found a full-output collision if the previously computed input subsets have a common intersection with cardinality greater than one.

Before implementing this general technique, it is clear that we have to choose a representation of the input subsets, and also a way to measure the influence of these input subsets over the observed output bits. Our proposed answer to these two questions is detailed in the next section.

## IV. Finding Collisions with Genetic Algorithms

In this preliminary implementation, we propose the use of binary bitmasks to represent input subsets. Such bitmasks will be used to perform a logical AND with all randomly generated input vectors, effectively fixing some of the input vector values to zero. In this way, we will be able of representing any subset of the input which members are characterised by the fixing of some of their bits to zero. Although this first representation is quite simple and limited, it is also very convenient (these bitmasks have a straightforward codification as individuals in the population of our genetic algorithm) and produces very interesting results.

The *influence* of a given input subset over the observed output can be measured as a deviation between two probability distributions. One is the observed probability distribution over the output bits. The other is the uniform probability distribution one should expect of the output of a random mapping. This difference between two probability distributions can be measured in many ways. But a simple and convenient procedure for reflecting it in a single value is to use a chi-square test. So our first proposal for the fitness function of the genetic algorithm is a chi-square statistic that measures the difference between the observed probability distribution and the theoretical (uniform) one. Obviously, we will try to maximise this difference.

However, this deviation cannot increase indefinitely. It will reach a maximum when an input subset produces an output distribution that is as far as possible from the uniform. This degenerate distribution occurs when all possible output values collapse in a single one, that is, when having a pseudocollision of the output. This is exactly what we are looking for.

By repeating this procedure over all the output bits we will eventually be able of finding a full output collision.

## V. Genetic Algorithm Implementation

We are trying to find which fixing of the input bits (performed by ANDing input vectors with the appropriate bitmask) helps in the process of fixing the output value, thus producing a pseudocollision of the observed bits.

But how to decide which bits to fix in the input is a very complex problem. It is, essentially, a search in a huge space with $2^{\Pi+K}$ elements. For TEA this space has $2^{192}$ possible values, so an exhaustive search is completely infeasible. For finding good bitmasks in those huge spaces we propose the use of genetic algorithms, where individuals will codify bitmasks. These bitmasks will be used to perform a logical AND with every random input, fixing some of the input bits to zero. It is important to mention that this fixing of some bits in the input must be reasonable in length to allow for enough different inputs for producing statistically significant deviations from uniformity in the output.

We decided to start observing the distribution of the 16 more significant bits of the first output word of TEA (that is w[0]>>16 using the notation of the TEA code shown before, where the operator >> designates a right shift).

These 16 bits can be interpreted as the binary representation of the integers between 0 and $2^{16}$-1=65535, and their distribution should uniformingly take all these values. The distribution of the computed statistic should follow a chi-square distribution with 65536-1=65535 degrees of freedom. The values for different percentiles of this distribution are shown in Table I:

TABLE I

VALUES OF THE CHI-SQUARE DISTRIBUTION WITH 65536 D.O.F. FOR DIFFERENT PERCENTILS

| p-value | 0.5 | 0.90 | 0.95 | 0.99 |
|---|---|---|---|---|
| $X^2$ | 65534.3 | 65999.3 | 66131.6 | 66380.1 |

Our objective is to find bitmasks for the TEA input (both the input block of 64 bits and the key block of 128 bits) that provoke a value in the chi-square statistic as high as possible.

We also need to know what to search for when finding a bitmask that produces a collapse (or pseudocollision) of the output, or analogously, how to decide which bitmask is better if the are two that produce a collapse of the output. In this case, we must prefer the heavier bitmask, the one that has a higher number of ones. A zero value in a given position of the bitmask implies that every input vector, after the AND process with the bitmask, will have a zero in this position, so more nonzero values in the bitmask allow for more input diversity because represent more cardinality in the input set. This is preferable because in general, the bigger the input subsets, the better. This will significantly increase the probability of having a common intersection with cardinality greater than one, that is, the probability of being useful for finding a full collision. Once the maximum possible deviation is reached, this preference towards bitmasks with greater weight will be reflected in the fitness function of our genetic algorithm.

Once we have found a bitmask that produces a collapse on w[0]>>16 and has a high weight, we proceed to search for a different bitmask producing a collapse in w[0]&0xFFFF and as heavy as possible. Repeating this for w[1]>>16 and w[1]&0xFFFF we would have four bitmasks $\mathbf{m_1}$, $\mathbf{m_2}$, $\mathbf{m_3}$, $\mathbf{m_4}$ such as $\mathbf{m} = \mathbf{m_1}\&\mathbf{m_2}\&\mathbf{m_3}\&\mathbf{m_4}$ will produce a full output collision providing $w(\mathbf{m})\geq 1$.

There are some problems with this approximation. The harder one is related with the uncertainty over the weight of the resulting mask. Only we would succeed in finding a full collision if this weight were greater than zero (thus allowing at least two different inputs that map to the same output). Using this approach, we do not have a direct way of being sure of this weight. The only thing we can do is trying to find the heaviest possible bitmasks for every output bits, in order to increase the probability of ones after the AND process.

The second one is related with the difficulty of finding bitmasks that produce pseudocollisions. In some cases, this would not be possible at all, or would be extremely difficult. When this happens, only partial solutions may be obtained and no direct way of finding a full output collision is got.

However, the application of this procedure can always, if not find a full collision, at least reduce significantly the search space for it thus making its finding much easier. This is exactly the case with TEA. The results are presented below.

## VI. RESULTS

We have used the implementation of the genetic algorithm of William M. Spears, from the Navy Center for Applied Research. After a number of preliminary tests, we determined that a 0.95 probability of *crossover*, a 0.02 *mutation* probability and a population size of 100 individuals were adequate for our problem and we decide to fix them to these values.

Every bitmask (codified as individuals in the genetic algorithm population) was evaluated by performing an AND with $2^{16+3=19}$ inputs generated at random, different for every individual and every generation. This makes convergence much harder, but drastically improves the generality of the results obtained, because it makes overfitting nearly impossible.

The maximum value for the chi-square statistic under these assumptions will occur when the observed distribution is as far from uniformity as possible, that is, when only one of the 65536 possible values really occurs. The *fitness* value in this degenerated case will be

$$\chi^2 = \sum_{i=0}^{65536} (o_i - 8)^2 / 8 =$$

$$= \sum_{i=0}^{65535} (8)^2 / 8 + (2^{19} - 8)^2 / 8 = 34359214080$$

We started our search seeking for the heavier bitmask able of producing a collapse on w[0]>>16. The fitness function we used was

$$\begin{aligned} &If \quad \chi^2 = 34359214080 \\ &then \quad fitness = \chi^2 / 10^6 + weight^3 \\ &else \quad fitness = \chi^2 / 10^6 \end{aligned}$$

The preference for heavier bitmasks is reflected in the fitness function for improving the probabilities of heavier masks after the AND operation. The normalisation of the chi-square statistic is optional.

Using this approach, we managed to obtain the following bitmask that fixes the 16 more significant bits of w[0] to a value of 1001111000110111:

{00000000000000000111101101111111,
00000000000000000000101111110111,
00000000000000000011111111111111,
00000000000000000011111111111111,
11111011111111101111111111011111,
11111111010101111011101111101111}

This bitmask has length 192 bits (192=64+128) and weight 106. This implies that we can generate up to $2^{106}$ different inputs to the block cipher TEA1 that collapse over the observed output. This makes the bitmask useful and applicable.

This procedure can also be used to produce a collapse of the output of w[0]&0xFFFF to the value 0111100110111001 by using the following bitmask, which has a weight of 108.

{11111011111110110000000000000000,
11111011111000000000000000000000,
01111111101111010000000000000000,
11111111101111100000000000000000,
11111111111111111111011110111111,
01110111111110110111110111111011}

But obtaining a mask for producing a pseudocollision on any part of the second output word of TEA (w[1]>>16 or w[1]&0xFFFF) appears to be much harder. In fact, after many executions of the genetic algorithm we were unable of getting any of them. Obviously this is not impossible, only computationally harder.

When this happens, we have to change our objective from the initial ambitious aim of directly finding full collisions to a more realistic attempt of easing or simplifying in some way this search for collisions. We must decide if we have made any progress by finding the masks shown above. All depends on a very simple fact: the weight of the AND of these masks. Let the value of this weight be w. This means that we have found a subset of the input of size $2^w$ in which every element produce a pseudocollision of all w[0] (it produces a collision of both w[0]>>16 and w[0]&0xFFFF). The important question here is if we can prove that there is at least one element in this subset that produces a full output collision. If this can be done, we would have made some progress reducing the search space (w<192) for finding a collision.

For this, all we need is to be sure that w>32. This is because, when applying the bitmask to the TEA input, we can consider TEA1 as a function from $2^w$ to $2^{32}$ (the masks effectively fixes the first 32 output bits, the first output word w[0]) and only the last 32 (w[1]) change. If only 32 bits change, then there are only $2^{32}$ possible different outputs and if the input subset has more than $2^{32}$ elements we can be sure we will find a collision.

This is our case. The mask produced by the AND of the two masks shown before is

{00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
11111011111111101111011110011111,
01110111010100110011100111101011}

and has weight 47, which is bigger than 32. This means that there will be at least $2^{47}$-$2^{32}$ full collisions in our input subset.

After these results, finding collisions now is no longer a computationally infeasible task. In fact, it is quite trivial. Furthermore, the calculations made in the past paragraph are quite pessimistic because we suppose that the bitmask that fixes all bits of w[0] has no effect over the distribution of the bits of w[1]. If this were not the case, finding collisions would be even easier.

Unfortunately, this is not the case for TEA. Both the distribution of w[1]>>16 and the distribution of w[1]&0xFFFF are not affected by the bitmask that produces a pseudocollision of half of the TEA output, as shown in the table below

Now that we have extracted all the benefits our technique can give, it is time to use the classical ways of searching for collisions, with the advantage a reduced-sized problem. We will use another technique based in what is called the *birthday paradox*. It basically consists in generating outputs and storing them so that every output obtained is compared with all the previously generated. In this way, there is a probability greater than 0.5 of finding a collision after $2^{t/2}$ operations if $2^t$ is the size of the output space. Applied to our case, we can expect to find a collision after only $2^{16}=2^{32/2}$ operations, when initially TEA would have required $2^{32}=2^{64/2}$ operations. Now the problem is at our hands.

## VII. CONCLUSIONS

This is a preliminary work with the use of genetic algorithms in the finding of collisions. In this paper, we have presented a new technique for generating collisions. Then we have proven it is useful at simplifying the initially computationally infeasible task of finding collisions in a reduced round version of a robust cryptographic primitive such as TEA.

Furthermore, the results obtained in this paper have other implications. A closer look at the bitmask that fixes all the 32 bits of w[0] reveals that the distribution of its 47 non-zero bits is far from random. All the 1's are at positions that only affect the last part of the input key, namely k[2] and k[3]. This points out that the influence of these input bits over the first half of the input is close to none. We can reach this conclusion even without knowing the code of TEA, which confirms this, and in a completely automated way. The fact that the output of w[1] is much harder to fix, also points out that it is, in some way, better encrypted. This is also true, and can be deduced from the code, but it is not trivial at all.

In this way, this technique could be used to verify the strength or find weaknesses of new cryptographic primitives like block ciphers or hash functions.

## VIII. Future Work

Some improvements and extensions to this preliminary work are possible, and some of them are being developed now or will be tested in a near future. For example, the fixing of some of the input bits to zero, although it has proved their usefulness and has produced interesting results, can be easily generalised. We can use a ternary ([0,1,*] for example) representation of the individuals instead of the more common binary representation. It is conceivable that a cryptographic algorithm can be more sensible to the fixing of some of its input bits to zero than to the fixing of these bits to zero and only using a ternary representation we will be able of discovering this.

Another point that probably deserves more research is the election of the output bits we will analyse to discover a deviation from the expected distribution. Although we have motivated and justified the election of the 10 rightmost bits of the first output word we have to acknowledge that this election is somehow arbitrary. An interesting possibility would be to codify in the genetic algorithm which bits to observe, so this could also evolve towards the bits that suffer a greater influence by a given input fixing.

## IX. References


[1] Douglas R. Stinson, *Cryptography, Theory and Practice*, CRC Press, 1995.

[2] D. Wheeler, R. Needham, *TEA, A Tiny Encryption Algorithm*, en Proceedings of the 1995 Fast Software Encryption Workshop. pp. 97-110 Springer-Verlag. 1995

[3] John Kelsey, Bruce Schneier, David Wagner, *Related-Key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2 and TEA*, Proceedings of the ICICS'97 Conference, pp. 233-246, Springer-Verlag, 1997.