A Review of Damgård's CA Hash Function and Cryptanalysis

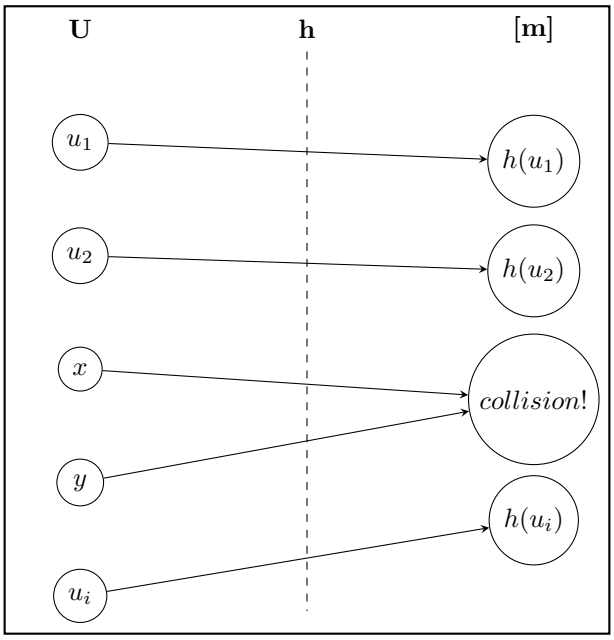
Caleb Alexander

1 Hash Function's and Their Properties

Much of the explanation contained in this section is taken, in some cases near-directly, from Mikkel Thorup's paper *High Speed Hashing for Integers and Strings* [1]. For a better idea of hash functions, a review of the aforementioned paper is recommended to the reader.

Consider some function $h : U \to [m]$, where U is a universe of so-called *keys* and $[m]$ is the set $\{0, 1, 2, \ldots, m-1\}$ with $m \in \mathbb{N}$. We say that h is a *hash function* if it achieves a suitable level of randomness in it's assignment of U to $[m]$. In practice, h is never truly random for both practical and theoretical reasons. To achieve an appropriate level of randomness, we use a *seed* of randomness in our chosen function. This seed should allow for our function to approximate a uniform random assignment of our domain to our co-domain.

It is most-often that our domain is larger, in many cases **much** larger, than our co-domain. In such cases, our function will have collisions. For some choice of h, and $x, y \in U$ we say that x and y collide, if $h(x) = h(y)$.



The challenge of finding a good hash function, can often be described as the challenge of finding a hash function which minimizes collisions. We will describe five hash function properties below, the last of which will be integral to our later review of Damgård's Hash function.

Universality is a property that a hash function can have if for our choice of h, we can say that:

$$Pr[h(x) = h(y)] \leq \frac{1}{m}.$$

In english, we say that for any $x, y \in U$, uniformly chosen, the probability that x and y collide is less than or equal to 1 over the cardinality of our co-domain. A hash function can be $c-$approximately universal if,

$$Pr[h(x) = h(y)] \leq \frac{c}{m}.$$

Strong Universality is a property which considers paired events. We consider h to be strongly universal if for distinct $x, y \in U$ and $q, r \in [m]$,

$$Pr[h(x) = q \ \ \wedge \ \ h(y) = r] \leq \frac{1}{m^2}.$$

See that strong universality implies universality.

Pre-Image Resistance considers how strong a hash function is against inversion. We say that h is pre-image resistant if for a given $y \in [m]$, it is computationally *infeasible* to find some $x \in U$ so that $h(x) = y$. Generally, we consider a problem computationally infeasible if the computation is both theoretically np-hard **and** no instance of a solution has been found in-practice. In many ways this is a *soft* definition, since we cannot prove that there will be no instance of a solution found in practice.

2nd Pre-Image Resistance is a stronger variation of pre-image resistance. We consider h to be 2nd pre-image resistant if for a given $x \in U$ it is computationally infeasible to find some y, so that $h(x) = h(y)$.

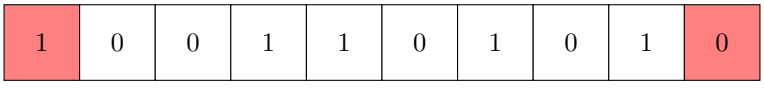
Collision Resistance, sometimes called collision-free, is one of the strongest properties that a hash function can have. We consider h to be collision resistant if it is computationally infeasible to find any $x, y \in U$, so that $h(x) = h(y)$. It is worth noting that collision resistance implies 2nd pre-image resistance which also implies pre-image resistance.

2 Damgård's Hash Function

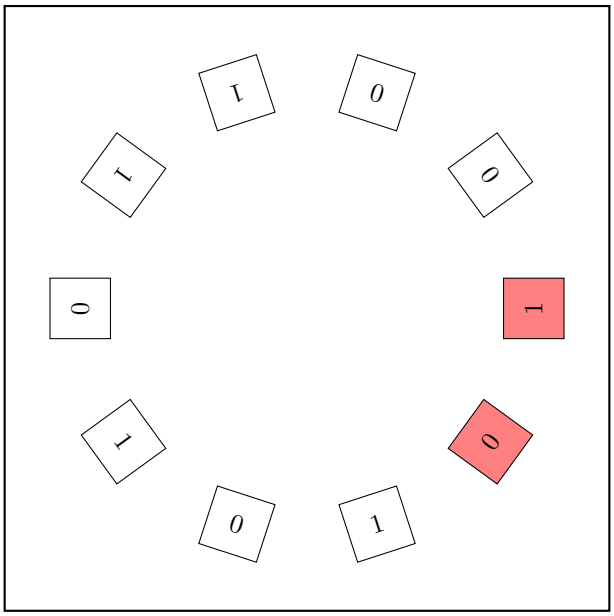
In Damgård's *A Design Principle for Hash Functions* [2], Damgård actually proposes three concrete hash functions. The function which we will focus on is based off of Wolfram's PRG which is known to be a one-dimensional cellular automata. Thus we will first give some background on one-dimensional cellular automata.

2.1 Cellular Automata Construction

A one dimensional cellular automata can be represented as a pair (B, r), where B is a starting "board" and r is a local rule which describes the evolution of a cellular automata locally.



In the board shown above, we color the ends red to denote that they are identified as "neighbors". Thus, this board can be most accurately described as a circle.



For simplicity's sake we will default to showing the board's in a "flat" style. A rule in this context is a function applied to each bit of our array locally, such that the output of the function gives each bit's next value. An example of what our rule could look like, taken from [2], is:

$$r(x_i) = x_{i-1} \oplus (x_i \vee x_{i+1}).$$

Thus for our example board, the application of this rule would look as shown below.

1	0	0	1	1	0	1	0	1	0
1	1	1	1	0	0	1	0	1	0

In general, when constructing a particular rule for our CA we consider the rule to be uniform for every bit and we define it in terms of a *diameter*. The diameter, in this context, is purely the distance surrounding our central bit such that our rule will consider those bits in the rule function. In the choice of r above, the diameter that we consider is 1 since we consider only the bits directly to the left and right of our central bit. If the diameter were 2, our function could look like:

$$r(x_i) = x_{i-2} \oplus x_{i-1} \oplus (x_i \vee x_{i+1} \vee x_{i+2}).$$

In abstract, there is no reason that the diameter we consider for each bit be uniform (and therefore the chosen r), nor is there any reason that if we consider two bits to the right of our chosen bit we **have** to consider two bits to the left as well. However, we often make the simplifications for ease of computation and communication. It is worth noting that the main benefit of CA such as this for most tasks is their massive potential for parallelization.

The natural next question is how we use such a construction as a number generator. Wolfram's used it as follows,

1. Choose $x \in \{0,1\}^n$, where n is the length of our board.
2. Compute $r(x_i)$ for all $i \in \{0, 1, \ldots, n-1\}$. (i.e. evolve our board).
3. Take the first pseudo random bit to be our new x_0.
4. Repeat the process as many times as needed for as many bits as needed.

2.2 The Function Proper

Now we define the function proper. To ensure good cryptographic qualities, Damgård mentions that two concessions must be made. First, we pick two numbers $c, d \in \mathbb{N}$ so that $c < d$. Then we define,

$$f_0(x) = b_c(x), b_{c+1}(x), \ldots, b_d(x),$$

where any $b_i(x)$ is said to be the *ith* bit returned by our PRG. Because the diameter of Wolfram's PRG is so low (only 1), it is obvious that it will take a significant amount of time for changes to propogate through our CA. To ensure good diffusion, we should choose d to be *at least* as large as the lenght of our board. This way we can ensure that every bit will have affected every other bit.

The next concession we make is in the size of our input and output bitstrings. Damgård points out that for certain choices of x such as the 0 or 1 constant bitstrings, that applications of r are constant.

1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

In general, there may be many bit strings which eventually return a constant bit. To reduce the chance of this happening, we consider only a subset of $\{0,1\}^n$ as valid starting boards. In particular, we let z be a randomly chosen constant bit string in $\{0,1\}^r$, with $r < n$. And we consider our input, x, to be a given $n-r$ length bit string. Finally our, function will consider $x||z$ (concatenation). Thus, we can finish out definition:

$$f(x) = f_0(x||z).$$

Consider the following example where we let $c = 10$ and $d = 20$, with randomly chosen x and z given below.

$x =$

0	1	0	1	1

$z =$

0	1	0	0	1

Then our evolution, starting at time step $c = 10$ and ending at time step $d = 20$ will be.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b
0	1	1	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0	1	0
0	1	0	1	0	1	0	1	1	1	0
0	1	0	1	0	1	0	1	0	0	0
1	1	0	1	0	1	0	1	1	0	1
1	0	0	1	0	1	0	1	0	0	1
1	1	1	1	0	1	0	1	1	1	1
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	1	0	1	1	0	0
0	0	0	1	1	0	0	1	0	1	0
1	0	1	1	0	1	1	1	0	1	1

Table 1: The evolution of our chosen bit string, starting at step c and ending at step d

.

Thus we have that,

$$f(x) = f_0(x||z) = 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1.$$

2.3 Desired Properties and Proof

At the time of Damgård's proposed functions, statistical methods had repeatedly shown good security properties for Wolfram's PRG. However, a proof of security against polynomial-time attackers had not yet been found. Thus, the function could not yet claim collision resistance. Below is given a claim and proof taken directly from Damgård's paper [2], which provides good evidence for collision resistance **but does not** prove the property. In later sections we will show the disproof of collision resistance.

Claim 1. If $g^v(x) = g^v(y) = z$, and $2v$ consecutive bits of x and y are equal, then $x = y$.

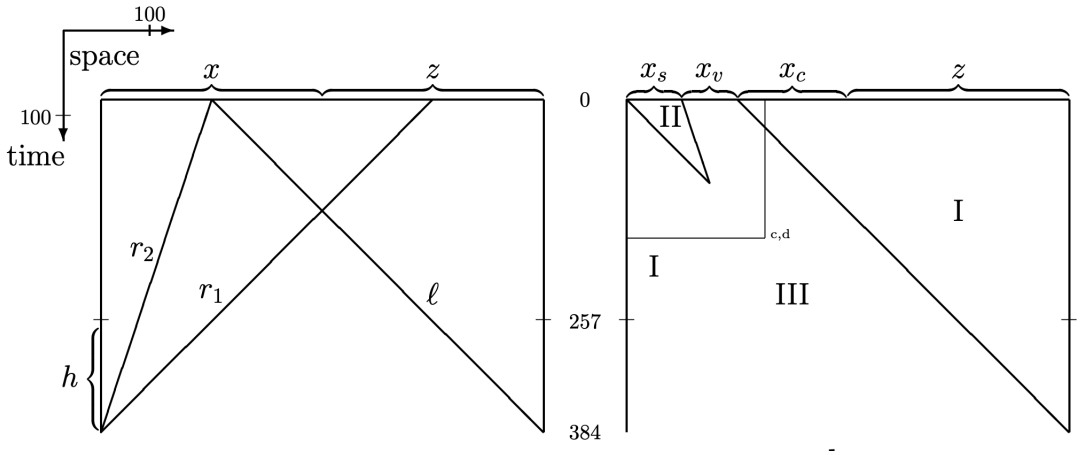
Proof. Let j be the index of the position immediately to the right of the $2v$ bits we know to be equal. Each bit of t depends on at most $2v+1$ bits of x(or y). Let l be chosen such that z_l is a function of bits $j, \ldots, j+2v$ of x(or y). Now, since inverting x_j will invert z_l, our assumptions imply that $x_j = y_j$. We can now "slide" the same argument one position to the right □

This provides evidence that a good choice of z, and sufficiently large r provides a good defense against certain collisions.

3 Cryptanalysis

Five years after Damgård published his paper which introduced the aforementioned hashing scheme, authors Joan Daemen, René Govaerts and Joos Vandewalle published the paper title *A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton* [3]. As is clear from the title, this paper includes a detailed cryptanalysis of the function which we have described previously, and proves conclusively that the function is **not** collision resistant.

The cryptanalysis of Damgård's function relies on two key observations. The first has to do with which bits actually affect our output bit. Notice that after t timesteps, the only bits which have affected our output a_0^t (the superscript denotes timestep and subscript is position), are of the form a_{k-t}^{t-k}. More simply, we can say that after t timesteps, the original value of bits which are t distance away will have affected a_0^t exactly once. Bits which are $t-1$ distance away will have affected a_0^t exactly twice and so on. The figures below, taken from [3], show a geometric interpretation of this concept.



Notice that we can think of the bits which have affected our output bit (or any particular bit), as the area of a triangle.

However, for any particular timestep, a bit can only affect the two bits which are directly two its left and right. Thus, if a bit does not change value at some time-step, it is effectively killing the propagation of changes through the array. Hence, we can say that for **most** bit strings, after t timesteps, the area of bits which will have affected the value of a_0^t will be significantly less than the area which one would expect. In particular, for the figure above, we can say that if r_1 is the line representing the triangle which would be expected, then we can say that in general the triangle of *meaningful* effect will be formed with r_2; which is clearly much steeper.

The second key observation of our cryptanalysis is that a particular pattern has predictable behavior in our CA. This pattern is any bit string which contains a sub string that is an alternating repetition (010101...). In this case, notice that the left edge of our pattern will shift one-bit to the right every timestep and the right edge shifts, on average, once to the right every three timesteps. Therefore, this pattern will expand across our array, though the expansion will be slow. From this observation, the authors of [3] show that, regardless of z, if we let x be the concatenation of three bit strings, x_s, x_v, x_c, such that x_s is the 64-bit

alternating patters, x_v is 62-bits, and x_c is 130-bits; then any sub string such that x_s and x_c are identical will give the same output. Thus, the authors have found 2^{130} bit strings such that each has 2^{62} collisions.

4 Conclusion

As expounded on by Luca Mariot in [4], instances such as this demonstrate the danger of using statistical, especially non-cryptographic, analysis methods when examining cryptographic schemes. Statistical testing showed that Wolfram's PRG, and thus Damgård's hash function, ought to be very secure. However, clearly such methods are insufficient.

For sufficiently secure hashing schemes, properties such as the five described in the first section ought to be proven (subject to the goals of the scheme) and information theoretical analysis should be performed.

References

[1] Mikkel Thorup. High speed hashing for integers and strings. *CoRR*, abs/1504.06804, 2015.

[2] Ivan Bjerre Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 416–427, New York, NY, 1990. Springer New York.

[3] Joan Daemen, René Govaerts, and Joos Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of damgård's one-way function based on a cellular automaton. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 82–96, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[4] Luca Mariot. Insights gained after a decade of cellular automata-based cryptography. In Maximilien Gadouleau and Alonso Castillo-Ramirez, editors, *Cellular Automata and Discrete Complex Systems*, pages 35–54, Cham, 2024. Springer Nature Switzerland.