Ayin, A Collision-Free Function for Low-Cost RFID Systems

No Author Given

No Institute Given

Abstract. In this work we propose Ayin, a new, simple and efficient function based in the Bellare-Micciancio construction that is also collision-free. All these characteristics make it suitable even in very constrained computational environments, such as low-cost RFID tags or sensor networks. We provide a detailed security analysis together with evidence (supported by a hardware implementation analysis) that shows that the use of Ayin is realistic for these extremely constrained devices due to its small footprint and good speed.

1 Introduction

There is a common misunderstanding among designers of cryptographic protocols for RFID systems: most of them simply assume that hash functions could be easily implemented in such environments and, accordingly, they do extensive usage of them in their protocols. However, as clearly pointed out by [11], this is not the case and many of these protocols run the risk of never being of any use unless researchers do find a way of designing new cryptographic tools for environments with such scarce computational resources.

Although many interesting new proposals are blossoming lately (see e.g. [6]), classical cryptographic constructions generally lie well beyond the computational reach of very low-cost RFID tags, which typically have about 5K-10K logic gates and can dedicate at most about 3K Gate Equivalents (GE) to security tasks.

This is the main reason behind our attempt of designing a very fast and simple function that exhibits a good degree of collision-freeness but nothing more. As affirmed in [14], "among the many properties a cryptographic hash function is expected to have, collision resistance seems to be the hardest to achieve." This is indeed the main property we expect to offer: a function suitable for being used into applications and protocols where collision resistance is needed.

Our work is heavily inspired by the presentation of Adi Shamir at RFIDSec'07 and later at the CRYPTO'07 and FSE'08 Conferences [23]. What Shamir proposed was a new efficient primitive named SQUASH, and loosely based on Rabin's variant of the RSA function for achieving one-wayness. This is in many cases sufficient for simple authentication protocols. Unfortunately, no current implementations and no efficiency analysis have been published yet. Furthermore, SQUASH is clearly not designed to be collision free, and it is in fact extremely easy to create collisions for it.

In this work, we propose a function that should present a high degree of collision-freeness, and that could be of higher practical interest because, as collision-freeness implies one-wayness, it could be used in every application where SQUASH is used, and also in protocols or applications where collision-freeness is required.

The rest of the paper is organized as follows: In Section 1.1 we present some related work while in Section 1.2 the main contribution of the paper is quickly introduced. Section 2 presents the main proposal in more detail. This is followed by Section 3 where an analysis of the security of the Bellare-Micciancio construction, together with an in-deep explanation of how the underlying f function was obtained by means of Genetic Programming and a preliminary analysis of its security. Hardware results are introduced in Section 4, with a Gate Equivalent count for the different proposals. In Section 5 we present some conclusions and guidelines for future work. Finally, in the Appendix the reader can find the main components of the Ayin code in C.

1.1 Related Work

There are not many proposals of new cryptographic primitives for very constrained environments. Apart from the previously cited work of Shamir [23], the most significant contributions to this field are a family of lightweight DES-based ciphers [17] and the ultralightweight PRESENT block cipher [6], inspired by the AES finalist SERPENT. Other interesting works in this area are the PRNG LAMED [21] (1566 GE, 17.2 Kbps) and those issued from the eSTREAM project, notably on the hardware oriented Profile 2, such as Grain [13] (1714 GE, 100 Kbps) and Trivium (3000 GE, 100 Kbps) [8], which were especially designed for very constrained architectures [12]. The security of these stream ciphers, however, has been questioned in the light of some recent attacks (e.g. [3, 16].)

Additional works on the implementation of the AES on RFID systems were carried out in [9, 10], but results are still discouraging, as both the footprint needed (around 3400 GE) and the throughput achieved (12 Kpbs) are still far from those required in most of the intended applications.

Finally, in [26] we find a recent work on the implementation of a new hash function with a reduced number of gates. Although this proposal seems to be light enough to fit in a low-cost RFID tag, its security remains a complete open question.

1.2 Contribution

In this paper, we propose a very lightweight collision-free function suitable for low-cost RFID systems. The scheme follows the incremental Bellare-Micciancio construction [2], using in its core a specially-designed highly nonlinear function obtained by means of a novel genetic programming technique.

We provide sufficient evidence supporting that this scheme is adequate even for very low-cost RFID tags. For this we discuss a detailed hardware implementation analysis which shows that different versions of Ayin could be implemented

INITIALIZATION
$\texttt{State}[i] = E_i, i = 1, \ldots, L$
$\texttt{counter} = 0$

ALGORITHM AYIN(M)
For each 128-bits block $m = [m_1, m_2, m_3, m_4]$ of M do
For $i = 1, \ldots, 4$
$\texttt{State} = \texttt{State} \oplus f_i(\texttt{counter} + (i-1),\ m_i)$

with $f_i(\texttt{counter},\ m_i) = f(c_i + \texttt{counter},\ m_i)$

Fig. 1. Overall structure of Ayin algorithm.

using from 1.8 to 2.7K GE. We propose three versions of Ayin with different output lengths varying from 96 to 160 bits and study their security properties, mainly by using the latest results presented in the literature in [4]. We believe this might be an interesting step in the way towards a truly lightweight hash function, something that seems far from easy in these very constrained environments.

2 Ayin Description

A description of the algorithm is provided in Fig. 1. The algorithm maintains an internal state of L 32-bits words, with L varying from 3 (96 bits) to 5 (160 bits). The initialization phase simply consists in assigning to the initial state the values of Euler's gamma constant (E_i), following a "nothing up my sleeve" policy.

A message M is iteratively processed in blocks of 128 bits, seen as 4 32-bits words. For each block, the internal state is updated 4 times by using 4 different functions f_i. These functions perform essentially the same computation (termed f) but each time operating on different inputs. The general behavior of the f function could be compared with that of a PRNG, which is used with different constant values c_i to obtain essentially four different functions f_1, f_2, f_3 and f_4, which are then used following a Bellare-Micciancio construction. These four constants c_i are derived from the value of π in hexadecimal. Additionally, and trying to follow some of the guidelines for hash function design recommended in [5], we have included a counter as an element of the first argument for f. If we were to strictly follow the recommendations in [5], this counter should be processed as a separate argument, but this will clearly have a negative impact on the overall speed of the primitive, so we included it in this way trying to get a good compromise in terms of security and efficiency.

As Ayin is intended to be used only in very specific environments, such as protocols for Reader to Tag authentication, key exchange/agreement, etc. where messages should be highly formated and with a fixed, predefined length, we do

not find necessary to add a padding phase. If for any particular reason padding is needed, classical solutions such as MD-strengthening [7] could be applied.

The source code in C of the various Ayin versions is provided in the Appendix. Some more in-deep analysis can be found in Section 3.

3 Analysis

We will firstly analyze the security of the Bellare-Micciancio scheme, mostly by recalling results already presented in the literature [2, 24, 4], but putting an special interest in their application to our particular proposal. After that, we will focus on the design and cryptographic properties of the f-function at the core of Ayin.

3.1 On the security of the Bellare-Micciancio construction

The Bellare-Micciancio scheme basically consists in compressing the message $(m_1, m_2, \ldots, m_n)$ into the value $f_1(m_1) \odot f_2(m_2) \odot \cdots \odot f_n(m_n)$ for some operation $\odot$ and some fixed n. The security of this construction has been extensively studied. First, by the authors themselves, who were unconvinced of the security of the XOR version ($\odot = \oplus$) that we use in Ayin, and presented an attack [2] that, from their point of view, would make its use unwise. Then, Wagner [24] established additional security bounds for this construction. These bounds were further refined by Bernstein [4]. He basically pointed out that the security of the XOR construction could be acceptable when some extra measures were taken (used only for processing short messages), and in fact proposed the first cryptographic primitive that followed the Bellare-Micciancio construction, namely the Rumba20 collision-free function. This used at its core the Salsa20 function, with four different diagonal constants for creating four different functions f_1, f_2, f_3 and f_4. Only one attack against a reduced-round version of Rumba20 [1] has been published so far, and this attack does not take advantage of the Bellare-Micciancio construction, and is not able to successfully attack Rumba20 with more than three rounds. This, to some extend, shows that breaking the scheme, provided some care is put in its use, is far from a trivial task.

Specifically, in [4] the author studies the case of the 4-xor compression function, arriving to provide more general bounds of its security, taking into account not only the computational time of an attack but also its machine cost. Precisely, the function $(m_1, m_2, m_3, m_4) \rightarrow f_1(m_1) \oplus f_2(m_2) \oplus f_3(m_3) \oplus f_4(m_4)$, where $f_i : \mathbb{Z}_2^I \rightarrow \mathbb{Z}_2^O$ has an attack that needs, for each $c \in [0, \frac{2}{9}O]$ and for each $t \in [\frac{c}{2}, O - 4c]$ a time of 2^t on a circuit of size 2^c, having a success probability of around 2^{t+4c-O}. If $t > 2c$ then the best attack consists in using the parallel-collision-search circuit of van Oorschot and Wiener [20] which takes time 2^t on a circuit of size 2^c and has a success probability of around $2^{2t+2c-O}$.

The main reason why Bellare and Micciancio rejected the proposal used here (with $\oplus$ as the joining operator) is because for very large values of the number n above it is relatively easy to find linear dependencies between the

Table 1. Security results for Ayin

Function	Machine Cost (Number of processors)	Price-performance Ratio
Ayin-96	$2^{21.33}$	2^{32}
Ayin-128	$2^{28.44}$	$2^{42.66}$
Ayin-160	$2^{35.55}$	$2^{53.33}$

values $f_1(m_1) - f_1(m'_1), f_2(m_2) - f_2(m'_2), \ldots, f_n(m_n) - f_n(m'_n)$ that make a collision trivial to obtain. This problem is quite straightforward to address by simply fixing n to a relatively small value. In this work, as in [4], we set $n = 4$ as this makes that a linear dependency between the four different components has a very low probability of only 2^{4-O}. Thus, these concerns do not have a significative impact on the security of Ayin with output length O.

On the other hand, when the attacker is able to get around $2^{\frac{2}{9}O}$ processors to parallelize collision-search, an attack as in [4] has a limiting price-performance ratio of around $2^{\frac{O}{3}}$. This means that for the more efficient version of Ayin (96 bits), a collision could be found by distributing a relatively light work of around 2^{32} operations on 2.642.246 processors. Whether finding and being able to use more than 2.6 million processors to attack a compression function used in low-cost RFID tags could be or not a real possibility, we can not advocate the use of Ayin-96 except in applications where this attack is, for some reason, impractical or uneconomical.

The amount of work and the number of processors needed for successfully attacking Ayin-128 and Ayin-160 is, on the other hand, well beyond the usual security margins needed for RFID protocols, as reflected in Table 2.

3.2 On the security of the f-function

The methodology devised to obtain the f function at the core of Ayin is based on the use of Genetic Programming (GP). GP is a stochastic population-based search method invented in 1992 by John R. Koza [15]. This technique evolves computer programs instead of just particular solutions to a specific problem as, for example, is the case in Genetic Algorithms. As highly nonlinear functions such as f should be designed and implemented as computers programs, the use of GP in the problem is justified.

In our experiments, we have used the lil-gp library [27]. In the following we briefly describe the main GP parameters that have to be set to our particular problem.

- Function Set. The function set is the building block of the individuals we will obtain. We decided to include only very efficient operations which are also very easy to implement in hardware: **rrot** (right rotation), **xor** (addition mod 2), **and** (bitwise and), **or** (bitwise or), and **not** (bitwise not). The **sum** (sum mod 2^{32}) operator is also necessary, in order to avoid linearity. We did not

include multiplication because the multiplication of two 32-bit values could be a very costly operator, depending of the particular architectures. The use of right rotations (by a small number of bits), rather than the more widely used left rotations is due to the lower latency they induce. This has some impact on the throughput on different platforms, specially when rotation is combined with modular additions as is indeed the case with some of our best individuals.
It is interesting to note that, due to the operators used in the design of Ayin, all of which take constant computation time independently of their input, this compression function is immune, by construction, to timing cryptanalysis.

- Terminal Set. The terminals will be represented by two 32-bit unsigned integers (a_0, a_1). We also included Ephemeral Random Constants (ERCs), which are constant values (in our problem, 32-bit random values) that the GP environment uses to try to generate better individuals.
- Fitness Function. A function $F : 2^m \rightarrow 2^n$ is said to have $PC(t)$, the Propagation Criterion [22] of degree t, if

$$\forall x, y | H(x, y) \leq t, H(F(x), F(y)) \approx B(n, \frac{1}{2})$$

that is, if complementing t or less bits of its input produces that every output bit changes with a probability of around $\frac{1}{2}$.
We will use this Propagation Criterion to evaluate the nonlinearity of the individuals, in particular that of order 4, $PC(4)$.
To measure the proximity of the distribution of the computed Hamming distances to the sought theoretical binomial $B(n, \frac{1}{2})$ a χ^2 goodness-of-fit test statistic is employed.
Concretely, the proposed fitness function is computed in the following way:

$$Fitness = \frac{10^6}{\chi^2}$$

It was necessary to amplify the fitness function (multiplying by 10^6) because the initial values of the χ^2 statistic were extremely high, making the fitness negligible at the beginning of the evolution process.
In more detail, the fitness of each individual is calculated as follows: we use the Mersenne Twister generator [19] to randomly generate the pair (a_0, a_1). The output O_0 for this input is stored. Then, we randomly flip between one and four bits of this two 32-bit input, and we obtain a new output O_1. Now, we store the Hamming distance between those two output values $H(O_0, O_1)$. This process is repeated a number of times ($2^{12} = 4096$ was experimentally proved to be enough), and each time a Hamming chi-square statistic is obtained.
- Tree Size Limitations. The depth and/or the number of nodes of the individuals should be limited. We tried both limiting the depth and not limiting the number of nodes, and vice versa. The best results were consistently obtained by using the latter option.

```
=== BEST-OF-RUN ===
              generation: 1187
                   nodes: 12
                   depth: 5
                    hits: 160126
TOP INDIVIDUAL:
-- #1 --
                    hits: 160126
             raw fitness: 177257.1474
    standardized fitness: 177257.1474
        adjusted fitness: 177257.1474
TREE:
 (xor (rrot_8 (sum (rrot_3 (sum a0 a1)) a1))
      (rrot_3 (sum a0 a1)))
```

Fig. 2. Best individual found.

We allowed our individuals, in different runs, to use up to $10, 15, 20, 25$ and finally 30 nodes for trying to ensure a high degree of propagation and robustness, without exceeding the processing and temporal requirements of a low-cost RFID tag.

Results showed that more than 15 nodes were not necessary, as the genetic programming algorithm arrived in many cases to produce nearly optimal individuals, that is, functions where the fitness (as described above) was indistinguishable of that of a random function.

When the parameters were adjusted, we ran 20 experiments with different seeds for generating the initial population, with a population size of 500 individuals, a crossover probability of 0.8, a reproduction probability of 0.2, and an ending condition of reaching 2000 generations. These parameters were experimentally found to be adequate for our purposes.

The best individual (shown in LISP-like notation in Fig. 2) produced by following the approach described above (and with only 12 nodes) has an Avalanche Effect of 16.0126 (being 16.00 the optimal value) and presents a χ^2 goodness-of-fit test statistic of 5.641521 for a probability distribution with 32 degrees of freedom implying that, with probability 0.999999945288 the computed Hamming distances come from a Binomial distribution B(32,1/2).

Some additional tests were carried out to verify the randomness properties of the f function. We used the batteries ENT [25] and Marsaglia's tests [18] to check the statistical properties of the output. Even in cases of output produced by a very low-entropy input (such as a counter), it successfully passed all the tests.

4 Hardware Results

In this section we will present our hardware results on the three different versions of Ayin (namely Ayin-96, Ayin-128 and Ayin-160) that we envisage.

These results were obtained by using a XFAB xi10, 1.0 μm technology synthesizer.

Table 2. Hardware results for Ayin

Function	Gate Equivalents	Throughput@100kHz
Ayin-96	1877 GE	800kBHz
Ayin-128	2216 GE	800kBHz
Ayin-160	2697 GE	800kBHz

The throughput is measured at 100kHz, a very common frequency in these devices. Surprisingly enough, the synthesizer is able to find an even better solution if maximum throughput is required, reaching a peak at 1GBHz@12.5MHz with 2002 GE, for Ayin-96.

5 Conclusions and Future Work

Based on the ideas that lead to the design of SQUASH, we show that not only the basic minimum for authentication protocols (namely one-wayness) but also the stronger property of collision freeness is possible even in the very restrained computational environments of RFID tags and sensor networks.

However, for obtaining collision-freeness we should pay a slightly higher price that if we were only focusing on one-wayness, as due to the use of the Bellare-Micciancio scheme larger outputs are required to attain a given security level (with respect to other, classical constructions). Attackers against the BM construction can take advantage of the parallelism inherent to the scheme, although we have shown in Section 3.1 that even in this case successfully attacking Ayin with more than 96 output bits is well beyond the usual security limits assumed in low-cost RFID environments. On the other hand, the resulting primitive has a much wider application scope.

Additionally, we prove that the additional cost of having collision resistance is completely affordable by presenting a detailed description of Ayin, our proposed algorithm, and a rigorous hardware analysis. The only drawback of our approach is a slightly increase in the length of the messages exchanged with respect to those needed in other environments, so perhaps this solution could not be suitable for environments where the cost of communications is very high, such as in certain sensor networks. We do not claim, on the other hand, that the proposed primitive has all the requisites normally associated with a cryptographic hash function: for example it doesn't provide with a PRNG in the usual way nor hides its input well. Although efforts in this direction will be interesting, we believe that any

additional cryptographic property added to that of Ayin will clearly increment its cost and made it less suitable for the intended applications.

References

1. Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier and Christian Rechberger. New Features of Latin Dances. Analysis of Salsa, ChaCha and Rumba. Proceedings of the Fast Software Encryption Workshop 2008, pages 467-485.
2. Mihir Bellare, Daniele Micciancio, A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, Advances in cryptology: EUROCRYPT 97, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, Berlin, 1997.
3. C. Berbain, H. Gilbert, and A. Maximov, Cryptanalysis of grain, `http://www.ecrypt.eu.org/stream/`.
4. Daniel J. Bernstein. What output size resists collisions in a XOR of independent expansions?. ECRYPT Workshop on Hash Functions, 2007
5. Eli Biham, Orr Dunkelman. A Framework for Iterative Hash Functions-HAIFA. Report of the Technion's Computer Science Department, 2007 availiable at `http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2007/CS/CS-2007-15`
6. A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher," Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 07), LNCS 4727, Springer, 2007, pp. 450-466
7. I. Damgaard. A Design Principle for Hash Functions. In Advances in Cryptology - CRYPTO '89 Proceedings, LNCS Vol. 435, G. Brassard, ed, Springer-Verlag, 1989, pp. 416-427
8. De Canniere Ch., Preneel B. Trivium Specifications eSTREAM proposal at `http://www.ecrypt.eu.org/stream/triviump2.html`
9. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, Strong authentication for RFID systems using the AES algorithm, in Proc. of CHES04, ser. LNCS, vol. 3156, 2004, pp. 357370.
10. M. Feldhofer, J.Wolkerstorfer, and V. Rijmen, AES implementation on a grain of sand, in Proc. of IEEE on IS, vol. 152, 2005, pp. 1320.
11. M. Feldhofer, C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. In Proceedings of First International Workshop on Information Security (IS06), volume 4277 of Lecture Notes in Computer Science, pages 371-381, Springer, 2006.
12. T. Good, W. Chelton, and M. Benaissa, Review of stream cipher candidates from a low resource hardware perspective `http://www.ecrypt.eu.org/stream/hw.html`
13. Hell M., Johansson T., Meier W. Grain a stream cipher for constrained environments eSTREAM proposal at `http://www.ecrypt.eu.org/stream/grainp2.html`
14. L. R. Knudsen, C. Rechberger, and S. S. Thomsen. The Grindahl Hash Functions. In A. Biryukov, editor, Fast Software Encryption 2007, Proceedings, volume 4593 of Lecture Notes in Computer Science, pages 39-57. Springer, 2007.
15. Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992
16. O. Kucuk, Slide resynchronization attack on the initialization of grain 1.0, `http://www.ecrypt.eu.org/stream/`.

17. G. Leander, C Paar, A. Poschmann, and K Schramm A Family of Lightweight Block Ciphers Based on DES Suited for RFID Applications. In A. Biryukov, editor, Proc. Fast Software Encryption (FSE 07), LNCS 4593, Springer-Verlag, 2007, pp. 196-210
18. G. Marsaglia and W.W. Tsang. Some difficult-to-pass tests of randomness. Journal of Statistical Software, Volume 7, Issue 3:33-51, 2002.
19. M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)
20. Paul C. van Oorschot, Michael Wiener, Parallel collision search with cryptanalytic applications, Journal of Cryptology 12 (1999), 128.
21. Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, Arturo Ribagorda, LAMED A PRNG for EPC Class- 1 Generation-2 RFID Specification, Computer Standards & Interfaces (2007), doi: 10.1016/j.csi.2007.11.013
22. Preneel, B., W. Van Leekwijck, L. Van Linden, R. Govaerts and J. Vandewalle. 1990. Propagation Characteristics of Boolean Functions. Advances in Cryptology – EUROCRYPT '90. pages 161-173.
23. Adi Shamir. SQUASH: A New One-way Hash Function With Provable Security Properties for Highly Constrained Devices such as RFID Tags. In Proceedings of FSE 2008, to appear.
24. David Wagner. A generalized birthday problem. In Moti Yung, editor, CRYPTO vol. 2442 of LNCS, pages 288-303, 2002.
25. J. Walker. ENT Randomness Test. `http://www.fourmilab.ch/random/` 1998.
26. K. Yksel, J.P. Kaps, and B. Sunar. Universal hash functions for emerging ultra-low-power networks. In Proc. of CNDS'04, 2004.
27. IlliGAL Lilgp Genetic Programming Library `http://garage.cse.msu.edu/software/lil-gp/`

A Ayin C source code

This is the C source code of Ayin.
The parameter L varies from 3 to 5 for the three different versions of Ayin-96, 128 and 160.

```
#define RROT32(v, n) ((unsigned) ((v) >> (n)) | ((v) << (32 - (n))))
#define L 3
int f (unsigned int n, unsigned int i, unsigned int o[L])
{
int j, k;
unsigned int c, d, r;
         c=n; d=i;
         for(k=0;k<L;k++)
         {
            for(j=0;j<4;j++)
            {
              r=RROT32(RROT32(c+d,3)+d,8)^RROT32(c+d,3);
              c=d; d=r;
             }
```

```
            o[k]^=r;
          }
}
int Ayin (unsigned int *m, unsigned int o[L], int counter)
{
        int i;
        for (i=0;i<sizeof(m)/4;i=+4)
        /* Constants based in Pi */
        {
                        f(0x243f6a88 + counter, m[i] ,o);
                        f(0x85a308d3 + counter + 1, m[i+1] , o);
                        f(0x13198a2e + counter + 2, m[i+2] , o);
                        f(0x03707344 + counter + 3, m[i+3] , o);
        }
}
int Init_hash(unsigned int o[L])
{ unsigned int EG[5];
  int i;
 /* Set O to a given initial state value */
 /* Constants based in Euler's Gamma */
 EG={0x93c467e3, 0x7db0c7a4, 0xd1be3f81, 0x0152cb56, 0xa1cecc3a};
 for(i=0;i<L;i++) o[i]=EG[i];
}
```