KEYED HASH FUNCTION FROM LARGE GIRTH EXPANDER GRAPHS

MONIKA POLAK

College of Computing & Information Sciences,
Rochester Institute of Technology,
Rochester, NY 14623, USA

EUSTRAT ZHUPA

Department of Computer Science,
University of Rochester,
Rochester, NY 14627, USA

ABSTRACT. In this paper we present a keyed hash function (message authentication code MAC). Our approach uses a family of expander graphs of large girth denoted as $D(n,q)$, $n \in \mathbb{N}_{\geq 2}$ and q is a prime power. Graphs $D(n,q)$, $n \geq 2$ for arbitrary q form a family of q-regular almost Ramanujan graphs ($|\lambda_1(G_i)| \leq 2\sqrt{q}$). Expander graphs are known to have excellent mixing properties because are very dense. The girth of this family of graphs can be descibed by the formula $g_n \geq \log_q(q-1)\log_{q-1}(v_n)$, where v_n is size of graph $D(n,q)$. All requirements for a good MAC are satisfied in our method and a discussion about collisions and preimage resistance is also part of this work. The outputs very closely approximate the uniform distribution and the results we get appear to be indistinguishable from random sequences of bits, when tested by statistical test suite for random and pseudorandom number generators. Based on the tests, our graph based keyed hash functions shows good efficiency in comparison to other techniques. The number of operations per bit of input for DMAC-1 is given by the formula $\dfrac{2n+2}{N}\left(1 + \dfrac{r}{l(M)}\right)$, where r is the length of secret key S, N is block size and $l(M)$ is the number of blocks in a message.

1. INTRODUCTION

Our work is motivated by the work of Charles, Goren and Lauter, [7]. They proposed the construction of collision resistant hash function from expander graphs. The family of graphs they used were Ramanujan graphs constructed by Lubotzky,

---

Philips and Sarnak (see [19]) and Pizer's Ramanujan graphs, [24]. Hash functions from LPS require 7 field multiplications per bit of input, but the field size may need to be bigger (1024 bit prime p instead of 256 bits, for example), and the output is $4log(p)$ bits. When the hash function from Pizer's graph $G(p,l)$, for $l = 2$, requires $2log(p)$ field multiplications per bit of input, which is quite inefficient (the authors propose to use a graph of cryptographic size $p \approx 2^{256}$). The output of this hash function is $log(p)$ bits. The idea presented in [7] is very good, but collisions for this hash were found (see [27]). However, expander graphs can be used to produce keyed hash function (message authentication code). When secret parameters are involved (like colouring and initial vertex) the adversary cannot find collision with mentioned method.

We propose a construction of message authentication code based on another family of expander graphs $D(n, q)$ of large girth, [17]. The most important advantages are: the output can have arbitrary length (like in case of variants of SHA-3: SHAKE128 and SHAKE256), max message size is unlimited and the performance is very good (4 field operations per bit of input can be achieved). Another advantage of our construction is that graphs from the family $D(n, q)$ have a nice representation by vectors and incidence relations are described by system of multivariate equations, which is very easy to implement. From the other side such systems of nonlinear equations are used for multivariate cryptography that is considered to be a good candidate for a post-quantum cryptography, [11].

The basics about hash function and keyed hash function (message authentication code; MAC) can be found in [26]. A hash function accepts a message M as input and produces a fixed-size hash value $h = H(M)$. Hash functions are often used to determine whether or not data has changed. In general terms, the main goal of a hash function is to ensure data integrity.However it can be also used for authentication, to create one way password files, as a source of pseudorandom numbers (bits) or for intrusion and virus detection. A cryptographic hash function is a function that is acceptable for security applications. It means that it shall be computationally infeasible to find

i: a x that is the preimage of h for a given hash value $h = H(x)$ (one way property),

ii: two data objects $x \neq y$, for which $H(x) = H(y)$ (the collision-free property).

In addition, a good hash function has the property that the output looks like random data and even a small change in input causes big changes in the output.

It is possible to use a hash function but no encryption for a message authentication. There are a few techniques to achieve this: use hash function + encryption on the hash, compute a hash value over the concatenation of M and S (a common secret value) and append the resulting hash value to the message or use keyed hash function. There are many reasons why it is worth to use techniques that avoid encryption [13]. For example, if there is no need to keep message confident but we want to authenticate it, those techniques are faster.

Figure 1 illustrates the mechanism for message authentication using keyed hash function. In order to check the integrity of the message, the keyed hash function is applied to the message and the result is compared with the associated tag. The secret key K_s is known for the receiver and for the sender, so the sender can be easily verified. The MAC approach guarantees data integrity and authenticity.
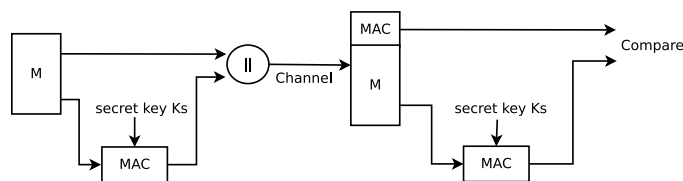
Figure 1. Message authentication using MAC

2. Background

Graphs of large girth. We define the *girth* of a graph as the length of the shortest cycle. In the analyzed context girth is a very important property of the graph. Let $\{G_i\}_{i \in \mathbb{N}}$ be a family of k-regular graphs with increasing order. Let g_i and v_i denote respectively the girth and the order of the graph G_i. A family of graphs with increasing girth is a sequence of graphs such that $g_i \leq g_j$ for $i < j$. According to the definition introduced in [2], we say that a family of k-regular graphs is a family of graphs of large girth if

$$g_i \geq \gamma \log_{k-1}(v_i)$$

for a constant γ and all $i \geq 1$.

A nice survey about graphs of large girth is presented in [3]. It is known that $\gamma = 2$ (see [6]) is the best possible constant, but there is no explicit construction of such family of graphs for which it can be obtained. This topic started in 1959 when Paul Erdős proved existence of such families with bounded degree k and $\gamma = 1/4$, without providing a construction [8]. There have been numerous investigations of the field by several authors. However, until 2017 the list of major known results is short. The list of explicit constructions is the following:

(1) the first explicit construction of such family with $\gamma = 0.44$, denoted by $X(p, q)$, where p and q are primes was introduced by G. A. Margulis in 1982 [20],
(2) generalisation of the family $X(p, q)$ proposed by M. Morgenstern [23],
(3) constructions for arbitrary k with $\gamma = 0.48$ and construction of family of 3-regular graphs with $\gamma = 0.96$ obtained by V. Imrich in 1983 [14],
(4) family of sextet graphs introduced in 1983 by Biggs and Hoare [5] (Alfred Weiss [34] proved that $\gamma = \frac{4}{3}$),
(5) second construction by G. A. Margulis in 1988 [21],
(6) constructions of cubic graphs, presented in a popular article [3],
(7) construction by Lubotzky, Phillips and Sarnak [19] (Biggs and Boshier [4] proved that $\gamma = \frac{4}{3}$ for this family of graphs),
(8) algebraic graphs $CD(n, q)$ given by the nonlinear system of equations over finite field $\mathbb{F}_q$, with $\gamma \geq \log_q(q - 1)$, [16] (Furedi [9] proved that for arbitrary prime power q: $\gamma = log_q(q-1)$),
(9) the polarity graphs of $CD(n, q)$ (see [18]) has an induced subgraph of degree $q - 1$ which is a family of graphs of large girth (it is shown in [32]).
(10) a new construction by G. Arzhantseva and A. Biswas [1] (finite 4-regular graphs with $g_i > C \log p$ for an absolute constant $C > 0$).

Expander graphs. An important property that a family of graphs must have in order to be a good candidate for a construction of a hash function is to be a

family of expander graphs. Cryptographic hash function from expander graphs were presented in works of [7, 28, 36].

Let's consider a spectrum of a graph with eigenvalues $\lambda_0 > \lambda_1 > ... > \lambda_{s-1}$. A family of k-regular graphs of increasing order $\{G_i\}_{i\in\mathbb{N}}$ is called a family of Ramanujan graphs if $|\lambda_1(G_i)| \leq 2\sqrt{k-1}$ for all i ([12], p. 452). If k is stable and $v_i \to \infty$ the limit $2\sqrt{k-1}$ is the best we can get. Ramanujan graphs are the best expanders.

3. The family of graphs $D(n,q)$

The family of graphs we use in our construction of a MAC was introduced in 1992 by Lazebnik and Ustimenko, [16]. It is denoted by $D(n,q)$, $n \in \mathbb{N}_{\geq 2}$ and q is a prime power. The similar notations that we use appeared later and can be found in [17]. To simplify, we don't use double notations for coordinates of vectors. The family $D(n,q)$ is special because of a few important properties. The first one is that this is a family of graphs of a large girth, as mentioned in Sec. 2. Secondly, this is a family of very good expander graphs that are close to Ramanujan graphs. The idea of almost Ramanujan graphs was introduced in [29]. We refer to a family of k-regular graphs as almost Ramanujan graphs if $|\lambda_1(G_i)| \leq 2\sqrt{k}$ for all i. Graphs $D(n,q)$, $n \geq 2$ for arbitrary q form a family of q-regular almost Ramanujan graphs ($|\lambda_1(G_i)| \leq 2\sqrt{q}$) and thus have excellent mixing properties.

Graphs $D(n,q)$ are bipartite with set of vertices V containing two subsets: $V = P \cup L$, where $P \cap L = \emptyset$. Originally the subset of vertices P is called a set of points and another set L is called a set of lines. Let P and L be two copies of Cartesian power $\mathbb{F}_q{}^n$, where $n \geq 2$ is a integer. Two types of brackets are used in order to distinguish points from lines. We write $(\vec{z})$ if $\vec{z} \in P$ and $[\vec{z}]$ if $\vec{z} \in L$. The set of vertices of graph $D(n,q)$ (collection of points and lines) can be considered as n-dimensional vectors over $\mathbb{F}_q$:

$$(\vec{p}) = (p_1, p_2, p_3, p_4, ..., p_n),$$
$$[\vec{l}] = [l_1, l_2, l_3, l_4, ..., l_n].$$

Coordinates of $(\vec{p})$ and $[\vec{l}]$ are elements of finite field $\mathbb{F}_q$. Because of this we have: $|P| = |L| = q^n$ and $|V| = 2q^n$. The vertex $(\vec{p})$ (point $(\vec{p})$) is incident with the vertex $[\vec{l}]$ (line $[\vec{l}]$) and we write: $(\vec{p})I[\vec{l}]$, if the following relations between their coordinates hold:

(1)
$$\begin{cases} l_2 - p_2 = l_1 p_1 \\ l_3 - p_3 = l_2 p_1 \\ l_4 - p_4 = l_1 p_2 \\ l_i - p_i = l_1 p_{i-2} \\ l_{i+1} - p_{i+1} = l_{i-1} p_1 \\ l_{i+2} - p_{i+2} = l_i p_1 \\ l_{i+3} - p_{i+3} = l_1 p_{i+1} \end{cases}$$

where $i \geq 5$. The set of edges E consists of all pairs $((\vec{p}), [\vec{l}])$ for which $(\vec{p})I[\vec{l}]$. This is a family of q-regular graphs, which means that each vertex has exactly q neighbors. $D(n,q)$ becomes disconnected for $n \geq 6$. Graphs $D(n,q)$ are edge transitive. It means that their connected components are isomorphic. A connected component of $D(n,q)$ is denoted by $CD(n,q)$. Notice that all connected components of infinite

graph $D(q)$ are q-regular trees. The length of the shortest cycle (the girth) of a graph $D(n,q)$ is given by the formula:

$$g(D(n,q)) = \begin{cases} n+5, & \text{for odd } n \\ n+4, & \text{for even } n \end{cases}$$

Graphs $D(n,q)$ were successfully used as a base for symmetric and public key multivariate cryptography (see for example: [15, 25, 31–33]), error correcting codes (see [10]) and currently we constructed a pseudorandom number generator. The related cryptosystems are very good candidates for post quantum cryptography and resistant to linearization attacks. The base of our message authentication code is a stream cipher algorithm where most of the constructions of message authentication functions are based on block ciphers.

4. Keyed hash function

Few notations are used in this work. Let M denote the message, N the number of bits per block of the M and by $l(M)$ we represent the number of blocks. So the message can be expressed as $M = m \ldots m_{l(M)}$. We consider that the message is written in alphabet that corresponds to elements of finite field $\mathbb{F}_q$ and by $l(q)$ we denote the number of bits needed to represent number q (for example UTF-8 uses number field $\mathbb{F}_{2^8}$). Calculations shall be performed in bigger number field than the number field ($\mathbb{F}_q$) that is used for the alphabet in order to achieve collection resistance property. Let denote by $\mathbb{F}_Q$ the number field used for calculations. The choice of N determines $\mathbb{F}_Q$. Any change in message shall change the hash so different input blocks m_i shall correspond to different edge colouring. To achieve it the following condition shall be satisfied

$$Q \geq 2^N.$$

It is convenient to choose $Q = p$, where p is a prime number. In such case field arithmetic is simply modulo p arithmetic.

From now on we denote by h the size of output (tag). The input message M is used as direction for walking around the graph $D(n,Q)$. We start with initial vertex $IV = \vec{v_0}$, which we consider to be a point $((\vec{v_0}) \in P)$. The next visited vertex is obtained by the formula

$$N_t(\vec{w} = (w_1, w_2, \ldots, w_n)) = [(w_1+t)^2, \underbrace{*, \ldots, *}_{n-1}],$$

$$N_t(\vec{w} = [w_1, w_2, \ldots, w_n]) = ((w_1+t)^2, \underbrace{*, \ldots, *}_{n-1}),$$

where $*$ can be uniquely calculated from equations (1) (see Example 1). Recall that $D(n,q)$ graphs are bipartite: points cannot be incident to points and lines cannot be incident to lines.

Example 1. *Let consider graph $D(6,11)$ and $\vec{w} \in P$.*

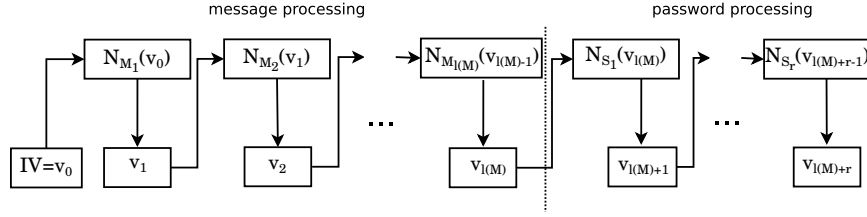$$N_3(\vec{w} = (1,8,4,2,7,0)) = [(1+3)^2, *, *, *, *, *]$$

Figure 2. General structure of the DMAC-1

Names are assigned for $*$: $[(1+3)^2, *, *, *, *, *] = [5, l_2, l_3, l_4, l_5, l_6]$. Then

(2)
$$
\begin{cases}
l_2 - 8 = 5 \cdot 1 \\
l_3 - 4 = l_2 \cdot 1 \\
l_4 - 2 = 5 \cdot 8 \\
l_5 - 7 = 5 \cdot 4 \\
l_6 - 0 = l_4 \cdot 1
\end{cases}
$$

where all operations are in finite field $\mathbb{F}_{11}$. The calculated neighbor of $(\vec{w})$ is $[5, 2, 6, 9, 5, 9]$.

We propose two approaches to calculate the keyed hash function based on this family of graphs. We named the message authentication codes DMAC, because constructions are based on family of graphs $D(n, q)$. Keyed hash functions use a secret, which is used to calculate the hash. We propose a secret key to be a pair (IV, S). IV is an initial vector of length n with coordinates from $\mathbb{F}_q$. S is a password of s characters from alphabet $\mathbb{F}_q$ such that

$$
s \leq \frac{1}{2} g(D(n, Q)).
$$

In our constructions, after all blocks m_i of a message are processed, we process a password S. The details are described in the next subsections.

4.1. **Basic construction (DMAC-1).** Fig. 2 illustrates the first type of proposed DMAC's. Steps to authenticate the message with DMAC-1:

(1) Agree secret key K_s, which is a pair (IV, S).
(2) split M in blocks m_i of length N (add padding if needed)
(3) Process the message. For $i = 0, \ldots, l(M) - 1$ do
   - Characters of the block of message m_i are concatenated to obtain a number M_i.
   - Calculate the vertex $\vec{v_{i+1}}$ which is incident to vertex $\vec{v_i}$. So, we calculate the next visited vertex by using operator $N_{M_i}(v_i)$:

$$
N_{M_i}(\vec{v_i} = (v_1^i, v_2^i, \ldots, v_n^i)) = [(v_{i \bmod n+1}^i + M_i)^2 \mod Q, \underbrace{*, \ldots, *}_{n-1}],
$$

$$
N_{M_i}(\vec{v_i} = [v_1^i, v_2^i, \ldots, v_n^i]) = ((v_{i \bmod n+1}^i + M_i)^2 \mod Q, \underbrace{*, \ldots, *}_{n-1}),
$$

   where $*$ are calculated from equations (1). We start in vertex $\vec{v_0}$ that is equal to IV.
(4) Process the password S. For a $i = l(M), \ldots, l(M) + r - 1$ do

message processing   password processing
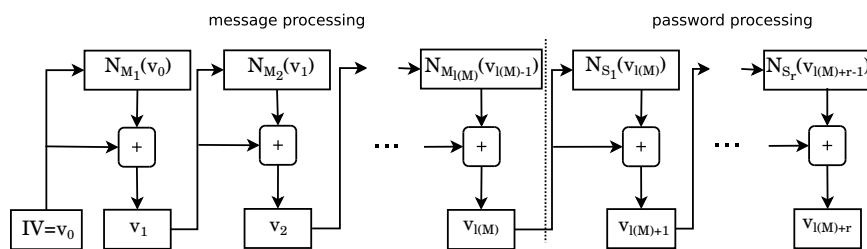

FIGURE 3. General structure of the DMAC-2

- Calculate the vertex $\vec{v_{i+1}}$ which is incident to vertex $\vec{v_i}$. So, we calculate the next visited vertex by using operator $N_{S_i}(\vec{v_i})$:

$$N_{S_i}(\vec{v_i} = (v_1^i, v_2^i, \ldots, v_n^i)) = [(v_{i \bmod n+1}^i + S_i)^2 \mod Q, \underbrace{*, \ldots, *}_{n-1}],$$

$$N_{S_i}(\vec{v_i} = [v_1^i, v_2^i, \ldots, v_n^i]) = ((v_{i \bmod n+1}^i + S_i)^2 \mod Q, \underbrace{*, \ldots, *}_{n-1}),$$

where $*$ are calculated from equations (1). We start in vertex $\vec{v}_{l(M)}$ (the last visited vertex in step 3).

4.2. **Modified construction (DMAC-2).** For $n \geq 6$ graphs become disconnected. In order to move from one component to another we can use simple modifications presented in Fig. 3. A vectors addition $\boxed{+}$ over $\mathbb{F}_Q$ is added.

In this case steps to authenticate the message with message authentication code are the same like for DMAC-1 except one additional step. Steps to authenticate the message with DMAC-2:

(1) Agree secret key K_s, which is a pair (IV, S).
(2) split M in blocks m_i of length N (add padding if needed)
(3) Process the message. For $i = 0, \ldots, l(M) - 1$ do
   - Characters of the block of message m_i are concatenated to obtain the number M_i.
   - Calculate the vertex $\vec{v_{i+1}}$ which is incident to vertex $\vec{v_i}$. So, we calculate the next visited vertex by using operator $N_{M_i}(\vec{v_i})$. We start in vertex $\vec{v_0}$ that is equal to IV.
   - Add vectors $\vec{v_i}$ and $\vec{v_{i+1}}$ over $\mathbb{F}_Q$.
(4) Process the password S. For a $i = l(M), \ldots, l(M) + r - 1$ do
   - Calculate the vertex $\vec{v_{i+1}}$ which is incident to vertex $\vec{v_i}$. So, we calculate the next visited vertex by using operator $N_{S_i}(\vec{v_i})$. We start in vertex $\vec{v}_{l(M)}$ (the last visited vertex in step 3).
   - Add vectors $\vec{v_i}$ and $\vec{v_{i+1}}$ over $\mathbb{F}_Q$.

Example 2. *(A toy example) Let's consider the following example. The alphabet is $\mathbb{F}_{29}$. We want to calculate DMAC-2 of 15 bits ($h = 15$) for a message M and a secret key K_s.*

A	B	C	D	E	F	...	Z	.	,	−
0	1	2	3	4	5	...	25	26	27	28

M :A–$BEAUTIFUL$–DAY corresponds to the vector $(0, 28, 1, 4, 0, ..., 24)$
S :.AY corresponds to the vector $(26, 0, 24)$
$IV = (5, 10, 27) = \vec{v_0}$

   In this case $l(q) = 5$ bits. We shall use n that satisfies: $h \leqslant n \cdot l(q)$. Hence $n = 3$. If we set $N = 25$ bits then each block has 5 characters and we decide to use $Q = 33554467$. We use graph $D(3, 33554467)$. We have 3 blocks $(l(M) = 3)$ total and the last block is padded:

$$m_1 = (0, 28, 1, 4, 0)$$

$$m_2 = (20, 19, 8, 5, 20)$$

$$m_3 = (11, 28, 30, 24, 0)$$

(1) $i = 0$

$$\vec{v_1} = N_{M_1}(\vec{(v_0)}) = [(v_1^0 + M_1)^2 \mod 33554467, v_2^1, v_3^1]$$
$$= [(5 + 28140)^2 \mod 33554467, v_2^1, v_3^1]$$
$$= [20388284, 1278029, 6390172]$$

because
$$\begin{cases} v_2^1 - 10 = 20388284 \cdot 5 \\ v_3^1 - 27 = v_2^1 \cdot 5 \end{cases}$$
$$\vec{v_1} := \vec{v_1} + \vec{v_0} = [20388289, 1278039, 6390199]$$

(2) $i = 1$

$$\vec{v_2} = N_{M_2}(\vec{(v_1)}) = ((v_2^1 + M_2)^2 \mod 33554467, v_2^2, v_3^2)$$
$$= ((1278039 + 20198520)^2 \mod 33554467, v_2^2, v_3^2)$$
$$= (30968786, 21891813, 29421730)$$

because
$$\begin{cases} 1278039 - v_2^2 = 20388289 \cdot 30968786 \\ 6390199 - v_3^2 = 1278039 \cdot 30968786 \end{cases}$$
$$\vec{v_2} := \vec{v_2} + \vec{v_1} = (17802608, 23169852, 2257462)$$

(3) $i = 2$

$$\vec{v_3} = N_{M_3}(\vec{(v_2)}) = [(v_3^2 + M_3)^2 \mod 33554467, v_2^3, v_3^3]$$
$$= [(2257462 + 112830240)^2 \mod 33554467, v_2^3, v_3^3]$$
$$= [14009975, 4873348, 10691714]$$

because
$$\begin{cases} v_2^3 - 23169852 = 14009975 \cdot 17802608 \\ v_3^3 - 2257462 = v_2^3 \cdot 17802608 \end{cases}$$
$$\vec{v_3} := \vec{v_3} + \vec{v_2} = [31812583, 28043200, 12949176]$$

(4) $i = 3 \ldots$
(5) $i = 4 \ldots$
(6) $i = 5 \ldots$

$$\vec{v_6} := (\vec{v_6} + \vec{v_5}) \mod 29 = h.$$

Table 1. Example n values for fixed tag size h and coding

		UTF-8	UTF-16	UTF-32
	$2^7 = 128$ bits	$2^7 = n2^3 \Rightarrow n = 2^4$	$2^7 = n2^4 \Rightarrow n = 2^3$	$2^7 = n2^5 \Rightarrow n = 2^2$
h	$2^8 = 256$ bits	$n = 2^5$	$n = 2^4$	$n = 2^3$
	$2^9 = 512$ bits	$n = 2^6$	$n = 2^5$	$n = 2^4$
	$2^{10} = 1024$ bits	$n = 2^7$	$n = 2^6$	$n = 2^5$

4.3. **Properties of DMACs.** A cryptographic hash function must work as follows: a small change in the input drastically changes the output. This is called *avalanche effect*. DMAC-1 and DMAC-2 were implemented and tested in Python. Our DMACs are the case of a high-quality keyed hash functions (see Table 3). Results presented in table are for the following parameters: graph $D(32, 257)$, $N = 32$, $h = 256$. Output of the presented keyed hash functions passed the well known Diehard tests, developed by George Marsaglia, for measuring the quality of random number generators, [22].

As defined above h is the number of bits of the output (tag). Popular size of tags are $h = 128$ bits, $h = 256$ bits, $h = 512$ bits and $h = 1024$ bits. When for most of commonly used algorithms the size of tag is fixed (for example: SHA-3-224 and SHA-3-256), in our approach the tag can have arbitrary length. The size of block length N can be chosen quite arbitrarily but it has to be much smaller than the size of the message M and $N \geq l(q)$. The longer the size of block, the more efficient the algorithm. Notice that if $l(q) = N$ then message is encoded character by character and it becomes a kind of 'string' algorithm.

The parameters of graph $D(n, Q)$ that is used depend on the block size N and the size of tag h. The paramater Q is chosen to satisfy the property the inequality $Q \geq 2^N$ and the parameter n is chosen as the smallest possible n that satisfies the inequality

$$h \leqslant n \cdot l(q).$$

The most commonly used encodings are UTF-8 ($l(q) = 8$), UTF-16 ($l(q) = 16$), UTF-32 ($l(q) = 32$). The Table 1 presents example values of n when h and alphabet $\mathbb{F}_q$ ($l(q)$) are fixed.

Example 3. *Let's consider a message M of 2000 characters writen in UTF-8 (alphabet $\mathbb{F}_{2^8}$; $l(q) = 8$) parameter S of 10 characters . We want to divide the message on blocks of 4 characters ($N = 32$ bits) and compute a tag of length $h = 512$ bits for this message.*
Parameter n can be computed from the formula $h \leq n \cdot l(q)$:

$$512 = 8n \Rightarrow 2^9 = 2^3 n \Rightarrow n = 64.$$

Then we choose a prime power Q such that

$$Q \geq 2^{32} \Rightarrow Q = 2^{32}.$$

Thus we use $D(64, 2^{32})$ graph. The length of the shortest cycle in this graph is $g(D(64, 2^{32})) = 68$ and the order of the graph is $2 \cdot (2^{32})^{64} = 2^{2048}$.

5. Collision resistance and one way property

Recall that, the family of graphs is a family of graphs of large girth and $g(D(n,q)) = 2[(n+5)/2]$. Hence there are no cycles shorter than $2[(n+5)/2]$ and therefore for numer of blocks smaller than $[(n+5)/2]$ no collisions can be find. First, we consider the following problems.

Problem 1. Find a cycle in graph $D(n,Q)$ that passes through vertex v_0 and $v_{l(M)}$.

Problem 2. Find a path between vertex v_0 and h in graph $D(n,Q)$, that contains a subpath defined by S that ends in h.

First of all we shall notice that the secret key is a pair $(IV = v_0, S)$. For an adversary that doesn't know the secret key those problems are not defined precisely. The problem of collision resistance is essentially the problem of finding a shortest cycle in the graph $D(n,q)$ (similarly as it was considered in [7] for other graphs). We have the following theorem.

Theorem 1. *Finding a collision in DMACs is a solution to Problem 1.*

Proof. If we set v_0 to be zero vector then DMAC-1 and DMAC-2 are exactly the same functions. Finding a collision in DMAC-2 cannot be easier than finding a collision in DMAC-1. Therefore, without lost of generalisation we can consider collision resistance of DMAC-1.

To compute a hash DMAC-1 we start a walk in vertex v_0 and an input $M = m_1 m_2 \ldots m_{l(M)}$ (message M) gives us directions M_i how to walk in this graph. Each vertex is Q-regular and $M_i \leq Q$ so different blocks correspond to different edges incydet to a given vertex. To find a collision we have to find two different inputs $M \neq M'$, which hash to the same output h. To calculate output h first we have to calculate vertex $v_{l(M)}$ and then using secret S we can walk to vertex that corresponds to h. If $v_{l(M)} \neq v_{l(M')}$ then S such that: $s < \frac{1}{2} g(D(n,Q))$ would lead us to different h and h' ($g(D(n,Q)$ denotes the length of the shortest cycle in graph). Hence, to find a collision we have to find two different inputs $M \neq M'$, which leads us to one vertex $v_{l(M)} = v_{l(M')}$. Two paths in graph, that start and end in the same vertices form a cycle. □

Graphs $D(n,q)$ form a family of a simple graphs. In this case $O(|V|)$ time is required to find a cycle in an $|V|$-vertex graph, that starts in a given vertex. However here $|V| = 2Q^n$ so the complexity becomes exponential. Then $O(|V|)$ is a time required requaired to find any cycle. In our case we are looking for a specific cycle that contains also $v_{l(M)}$.

The family of graphs is a family of graphs of large girth and $g(D(n,q)) = 2[(n+5)/2]$. Therefore, the problem of finding a shortest cycle in $D(n,q)$ graphs cannot be easier than the general problem of finding the shortest path in a regular graph, which is considered to be hard.

Theorem 2. *Finding a preimage of h is a solution to Problem 2.*

Proof. Because of the reason given in the proof of Theorem 1, without lost of generalisation we can consider preimage resistance of DMAC-1. If we have knowledge about S then vertex $v_{l(M)}$ can be computed. An input $M = m_1 m_2 \ldots m_{l(M)}$ message gives us directions M_i how to walk in this graph. We start a walk in initial

vertex $IV = \vec{v_0}$. The second visited vertex is defined by the operator $N_{M_1}(\vec{v_0})$ and uniquely determined from equations (1). The next visited vertex is defined by the operator $N_{M_2}(\vec{v_0})$ and uniquely determined from equations (1). We repeat the calculations until we deal with all blocks m_i. Graph $D(n, Q)$ is Q-regular and $M_i \leqslant Q$ so different M_i gives us different directions. Each M_i corresponds exactly to one edge incidence to a given vertex. There are many different paths from v_0 to $v_{l(M)}$. Find the preimage is to find the right path from v_0 to $v_{l(M)}$. Notice that v_0 is a part of a secret key. $\square$

Composition of operators N_{M_i} and N_{S_i} gives a nonlinear system of $n - 1$ cubic eqations (see Theorem 2 in [35]). Variables are: numbers M_i, s character of S and n coordinates of initial vertex v_0. There is $l(M) + n + s$ total variables in this system.

In general, solving a set of quadratic equations over a finite field is NP-hard (MQ problem) for any finite field. There is a conjecture that this is a probabilistically hard problem and Shor's algorithm cannot be used to speed it up, [11]. Solving a set of cubic equations over a finite field cannot be easier than solving the MQ problem. However, the system related to the set of equations 1 and other systems used for multivariate cryptography are not random, for a large enough parameters it is computationally infeasible to solve them (see [11]).

Brute force attack to completely break the keyed hash function (find secret K_s) may require to check $q^n q^r$ possibilities (q^n possible initial vectors and q^r possible passwords of length r), if we consider that the length of the S is known. A very efficient algorithm to find the shortest path in a graph is Dijkstra's algorithm of complexity $O(|V| \log |V| + |E|)$ and it can be adopted to find collisions. In the case of the used graphs it gives $O(2Q^n \log(2Q^n) + Q^{n+1})$ and it's not more efficient than brute force. The complexity is increased because calculations are made over bigger number field $\mathbb{F}_Q$, without changing the alphabet for IV and S.

6. TIMINGS

The number of operations per bit of input depend on block size N and on the parameter n of the graph $D(n, Q)$. The number of field operations in the system of equations 1 is $2(n-1)$ (one step of the walk in the graph $D(n, Q)$ costs $2(n-1)$ field operations).

To process one block of input with DMAC-1 we need two additions and one multiplication as specified by operator $N_t(w)$ (to calculate the first coordinate of the neighbor), one mod operation and $2(n-1)$ finite field operations. After the block is processed, we process vector S. Therefore, the number of operations per bit of input for DMAC-1 is given by the formula

$$\frac{(2n+2) \cdot l(M) + (2n+2) \cdot r}{N \cdot l(M)} = \frac{2n+2}{N}\left(1 + \frac{r}{l(M)}\right),$$

where r is the length of S, N is block size and $l(M)$ is the number of blocks, as specified above.

To process one block of input with DMAC-2 we need to add vectors over number field $\mathbb{F}_q$ which require n field additions. It gives us the following formula for the number of operations per bit of input when DMAC-2 is used

$$\frac{(2n+2+n)l(M) + (2n+2+n)r}{N \cdot l(M)} = \frac{3n+2}{N}\left(1 + \frac{r}{l(M)}\right).$$

Table 2. Summary of the Tests for DMAC-1

Original Message	Output	S	IV
The sky announced a beautiful day: the setting moon shane pale in an immense field of azure, which, towards the east, mingled itself lightly with the rosy dawn.	49 a7 df d0 58 51 6a 9d 4e 94 2d 43 2a b9 60 f2 ab 22 5a a8 18 13 20 7d f7 1 5f ad 21 3f 56 45	hint	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
The sky ...	2d 48 b6 e4 50 de 8b d2 2e b4 1b d fb 9f b6 63 a1 7b e2 ee 4 e7 b1 ed 88 25 51 ca c4 7d e3 36	hint	[149, 219, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
The sky ...	78 46 b2 50 81 c1 ba b2 c c4 e6 6c 7a 69 b4 fd c6 64 a 69 30 e0 4d 30 1e e7 9c 36 55 e d1 8a	hunt	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
Da sky ...	86 f8 16 c5 dd 100 28 d1 91 8f 48 3c ff 3a a6 e2 b1 31 23 91 17 73 64 86 be 6b 81 ad 5e 10 67 56	hint	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]

If the password S is short (not more than 10 characters) and the message, for which we want to calculate the tag, is long (which is true in general when MACs are used) then $\frac{r}{l(M)}$ is very small. Notice that the change of finite field doesn't increase much the number of operations per bit of input (except for the fact that resulting vector has to be divided   mod q).

Example 4. *Lets consider data like in Example 3. The length of M in bits is $2000 \cdot l(q) = 2000 \cdot 8$. The block size is $N = 32$ bits so the number of blocks*

Table 3. Summary of the Tests for DMAC-2

Original Message	Output	S	IV
The sky announced a beautiful day...	6f ed d1 fb 2f cf 56 fc a9 5e c8 1d 90 ec f7 4a df 42 1a 1e 3b 16 62 54 90 81 a2 a4 7e 3f 8d db	red	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
The sky ...	59 9 c2 a b1 ba 88 1b 12 e7 f3 a 65 71 87 7b 25 c4 20 57 38 6e 54 b0 b8 19 74 5b d8 33 46 d	r<u>i</u>d	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
The sky ...	d7 92 60 73 0 dd ef fa 6 3 f2 c6 9b 62 c6 58 e7 59 31 a5 2d 5e 34 67 7d a9 95 30 86 12 9a e0	red	[<u>149</u>, <u>221</u>, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]
<u>Da</u> sky ...	54 c7 8 d3 d9 fc c1 ed 57 18 9d 74 62 d2 5d 35 5a cd 15 3c b7 19 9a 3c 79 1d 4c 68 69 3b d8 6b	red	[147, 217, 2582, 2976, 1718, 1599, 27, 1083, 471, 1461, 1076, 2255, 2875, 2696, 2793, 1015, 1477, 1271, 2856, 221, 961, 2839, 1789, 1845, 1157, 622, 758, 882, 210, 1846, 3009, 410]

$l(M) = \dfrac{2000 \cdot 8}{32} = 500$. *In this case the number of field operations per bit of input is*

$$\frac{130}{32}\left(1 + \frac{10}{500}\right) \approx 4,$$

which is very efficient.

7. Conclusions

A new technique for message authentication was presented in this work. To the best of our knowledge, the family of graphs $D(n, q)$ has never been used before in this context. The algorithms here introduced, for DMAC-1 and DMAC-2 respectively, were implemented in Python and tested with different inputs. The results

of our tests and the theoretical base show that the technique we introduce is a very efficient and safe approach to compute message authentication code.

Acknowledgement. The authors would like to express their gratitude to Vasyl Ustimenko for sharing his knowledge about graphs $D(n, q)$, which made this research possible. Special thanks also to Stanislaw Radziszowski for his useful remarks.

References

[1] G. Arzhantseva and A. Biswas, Large girth graphs with bounded diameter-by-girth ratio, arXiv:1803.09229 (2018).
[2] Norman Biggs, Graphs with large girth, Ars Combinatoria, 25C (1987), 73–80 .
[3] Norman Biggs, Constructions for cubic graphs with large girth, The electronic jurnal of Combinatorics Vol. 5 (1998).
[4] N. L. Biggs and A.G Boshier, Note on the girth of Ramanujan graphs, Journal of Combinatorial Theory, Vol. 49 (1990), 190–194.
[5] N. L. Biggs and M. J. Hoare, The sextet construction for cubic graphs, Combinatorica, Vol. 3 (1983), 153–165.
[6] Béla Bollobás, Extremal Graph Theory, Dover Publications, 2004.
[7] Denis X. Charles, Eyal Z. Goren and Kristin E. Lauter, Cryptographic hash functions from expander graphs, Journal of Cryptology, Vol. 22 (2009), 93–113.
[8] Erdős, Paul, Graph Theory and Probability, Modern Birkhauser Classics, Classic Papers in Combinatorics (1987), 276–280.
[9] Z. Furedi and F. Lazebnik and A. Seress and V.A. Ustimenko and A.J. Woldar, Graphs of Prescribed Girth and Bi-Degree, Journal of Combinatorial Theory, Series B, Vol. 64 (1995), 228–239.
[10] P. Guinand and J. Lodge, Tanner type codes arising from large girth graphs, Proceedings of Canadian Workshop on Information Theory CWIT ?97, Toronto, Ontario, Canada (1997), 5–7
[11] Goubin Louis, Patarin Jacques and Yang Bo-Yin, Multivariate Cryptography, Encyclopedia of Cryptography and Security, Springer US, 2011.
[12] S. Hoory and N. Linial and A. Wigderson, Expander graphs and their applications, Bulletin of the American Mathematical Society, Vol. 43 (2006), 439–561.
[13] G. Tsudik, Message authentication with one-way hash functions, Preecedings INFOCOM '92 (1992).
[14] Imrich, Vrto, Explicit construction of graphs without small cycles, Combinatorica, Vol. 4 (1984), 53–59.
[15] M. Klisowski and V. Ustimenko , On the Comparison of Cryptographical Properties of Two Different Families of Graphs with Large Cycle Indicator, Mathematics in Computer Science, Vol. 6 (2012), 181–198.
[16] F. Lazebnik and V. A. Ustimenko, Explicit construction of graphs with an arbitrary large girth and of large size, Discrete Applied Mathematics, Vol. 60 (1995), 275–284.
[17] F. Lazebnik and V. A. Ustimenko and A. Woldar , A New Series of Dense Graphs of High Girth, Bull (New Series) of AMS, Vol. 32 (1995), 73–79.
[18] F. Lazebnik and V. A. Ustimenko and A.J. Woldar, Polarities and $2k$-cycle-free graphs, Discrete Mathematics (1999), 503–513.
[19] A. Lubotzky and R. Phillips and P. Sarnak, Ramanujan graphs, Combinatorica, Vol. 8 (1988), 261–277.
[20] Grigorij A. Margulis, Explicit constructions of graphs without short cycles and low density codes, Combinatorica, Vol. 2 (1982), 71–78.
[21] Grigorij A. Margulis, Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators, Problems of Informations Transmission, Vol. 24 (1988), 51–60.
[22] Marsaglia, Gorge, The Marsaglia Random Number CDROM, with The Diehard Battery of Tests of Randomness, produced at Florida State University under a grant from The National Science Foundation, 1985.

[23] M. Morgenstern , Existence and explicit constructions of $q+1$-regular Ramanujan graphs for every prime power q, Journal of Combinatorial Theory, Series B, Vol. 62 (1994), 44–62.

[24] A.K. Pizer, Ramanujan Graphs and Hecke Operators, Bulletin of the AMS, Vol. 23, No 1 (1990).

[25] M. Polak and U. Romańczuk and V. Ustimenko and A. Wróblewska , On the applications of Extremal Graph Theory to Coding Theory and Cryptography, Electronic Notes in Discrete Mathematics, Vol. 43 (2013), 329–342.

[26] William Stallings, Cryptography and Network Security: Principles and Practice, 3rd, Pearson Education, 2002.

[27] Jean-Pierre Tillich and Gilles Zémor, Collisions for the LPS Expander Graph Hash Function, Advances in Cryptology – EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings" (2008), 254–269.

[28] Jean-Pierre Tillich and Gilles Zémor, Hashing with SL_2, Advances in Cryptology, Crypto?94, Lecture Notes in Computer Science, Vol. 839 (1994).

[29] Vasyl Ustimenko, Coordinatisation of Trees and their Quotients, Voronoj's Impact on Modern Science, Vol. 2 (1998), 125 – 152.

[30] Vasyl Ustimenko, Maximality of affine group and hidden graph cryptosystems, J. Algebra Discrete Math. (2005), 133–150.

[31] Vasyl Ustimenko, On the extremal graph theory for directed graphs and its cryptographical applications, In: Shaska T., Huffman W.C., Joener D. and Ustimenko V., Advances in Coding Theory and Cryptography, Series on Coding and Cryptology, Vol. 3 (2007), 181–.

[32] Vasyl Ustimenko, On linguistic dynamical systems, families of graphs of large girth and cryptography, Zapiski Nauchnykh Seminarov POMI, Vol. 326 (2005), 214–234.

[33] V. Ustimenko and A. Wróblewska, On some algebraic aspects of data security in cloud computing, Proceedings of International conference: Applications of Computer Algebra, Malaga, Vol. 32 (2013), 144–147.

[34] Alfred Weiss *Girth of bipartite sextet graphs*, Combinatorica, Vol. 4 (1984), 241–245.

[35] Aneta Wroblewska, On some properties of graph based public keys, Albanian J. Math. 2 , no. 3 (2008), 229–234.

[36] Gilles Zémor, Hash functions and Cayley Graphs, Designs, Codes and Cryptography, Vol. 4 (1994), 381–394.