Birthday type attacks to the Naccache-Stern Knapsack cryptosystem

M. Anastasiadis[1], N. Chatzis[1] and K.A. Draziotis[2]

[1]*Aristotle University of Thessaloniki*
Department of Mathematics
54124 Thessaloniki, Greece.
e-mail : maanastas@math.auth.gr, chatzisnv@math.auth.gr
[2]*Aristotle University of Thessaloniki*
Department of Informatics
P.O. Box 114, 54124 Thessaloniki, Greece.
e-mail : drazioti@csd.auth.gr

Abstract

We present a (probabilistic) birthday type attack to Naccache-Stern knapsack cryptosystem. We provide the space and time complexity of this attack alongside with the success probability. Furthermore, we applied successfully the attack for messages of low or large Hamming weight.

Keywords: Public Key Cryptography, Naccache-Stern Knapsack cryptosystem, Birthday attack.

1. Introduction-Statement of results

In the present study we introduce an attack on the Naccache-Stern Knapsack cryptosystem (NSK) [9]. It is worth pointing out that our attack is based on the birthday paradox and is feasible only for small or large Hamming weight of the message. To the best of our knowledge there is not any efficient attack on NSK cryptosystem.

The NSK cryptosystem is a public key cryptosystem, defined by the following algorithms. First, we choose a large safe prime p (i.e. $p = 2q + 1$, where q is a large prime number). Let n denote the largest positive integer such that:

$$p > \prod_{i=0}^{n} p_i,$$

where p_i is the $(i + 1)-$th prime. The message space of the system is $\mathcal{M} = \{2^n, \ldots, 2^{n+1} - 1\}$ i.e. the binary strings of $(n + 1)-$bits. The *Key Generation* algorithm generates a positive integer $s < p - 1$ (the secret key) such that $\gcd(s, p - 1) = 1$ i.e. $s \in \mathbf{Z}^*_{p-1}$. This guarantees that there exists the $s-$th root mod p of an element in $\mathbf{Z}^*_p$. We set $u_i = \sqrt[s]{p_i} \mod p \in \mathbf{Z}^*_p$. The public key is the vector $(p,n;u_0,\dots,u_n) \in \mathbf{Z}^2 \times (\mathbf{Z}_p^*)^{n+1}$ and the secret key is s. For the *encryption* of a message m with $n+1$ bits, we calculate its binary expansion $\sum_{i=0}^n 2^i m_i$ and we compute

$$c = \prod_{i=0}^n u_i^{m_i} \mod p,$$

where $m_n = 1$ as in the original paper [9]. The *decryption*, given a ciphertext c, is provided by the formula

$$m = \sum_{i=0}^n \frac{2^i}{p_i - 1} \times (\gcd(p_i, c^s \mod p) - 1).$$

Some of the recent results of this system are provided on [1, 3, 6] and some applications are indicated on [2].

From the description of the NSK scheme we see that the security is based on the Discrete Logarithm Problem (DLP). It is enough to solve $u_i^x = p_i$ in $\mathbf{Z}_p^*$ for some i. The best algorithm for computing DLP in prime fields is subexponential [5, 10]. Thus, for large p the system can not be attacked by using DLP. In fact the security relies on the following more general problem.

Definition 1.1 (Subset Product Problem). Given a list of numbers L and a target c, find a subset of L whose product is c.

The previous problem is NP-complete [4]. NSK cryptosystem is based on the modular version of Subset Product Problem.

Definition 1.2 (Multiplicative Knapsack problem). Given a prime p, an integer $c \in \mathbf{Z}_p$ and a vector $(u_0, u_1, ..., u_n) \in \mathbf{Z}_p^{n+1}$, find a binary vector $m = (m_i)_i$:

$$c = \prod_{i=0}^n u_i^{m_i} \pmod p.$$

From this problem, we can easily see how someone can apply a birthday type attack. We start by constructing two sets $I_1, I_2 \overset{R}{\longleftarrow} \{0,1,...,n\}$ with $|I_1| \approx |I_2| \approx n/2, I_1 \cap I_2 = \emptyset$ and $I_1 \cup I_2 = \{0,1,...,n\}$. Then we construct the sets

$$U_1 = \Big\{ \prod_{i \in I_1} u_i^{\varepsilon_i} \pmod p : \varepsilon_i \in \{0,1\} \Big\},$$

$$U_2 = \Big\{ c \prod_{i \in I_2} u_i^{-\varepsilon_i} \pmod p : \varepsilon_i \in \{0,1\} \Big\},$$

where c is the encryption of the message m. The two sets U_1, U_2 have a common element, say x. Then, we construct the original message m. Indeed, since there exist ε_i for $i \in I_1$ and ε_i for $i \in I_2$, such that

$$x = \prod_{i \in I_1} u_i^{\varepsilon_i} = c \prod_{i \in I_2} u_i^{-\varepsilon_i}, \tag{1}$$

then $m = \sum_{i=0}^{n} 2^i \varepsilon_i$. This attack needs $2^{n/2+1}$ elements of $\mathbf{Z}_p^*$ for storage. To compute the common element we first sort and then we use binary search. Overall, we need $O(2^{n/2} \log_2 n)$ arithmetic operations in the group $\mathbf{Z}_p^*$. Using p with at least 1320 bits, we get $n \geq 161$, so the previous attack is infeasible. The recommended value for p is 2048 bits. In that case $n = 232$.

In [7] a generalization of the system was provided. Instead of prime numbers p_i the authors used polynomials that represent elements of a (non prime) finite field. If the characteristic is 2, then the message space is string of bits. In this case the birthday attack can be applied and also our improvement.

Finally, we shall provide an improvement of this attack, but it is feasible only for low or large Hamming weights of the message. Furthermore, for $n = 232$ we managed to find messages m of Hamming weights ≤ 8 and ≥ 225. Even for such small values of Hamming weight, the classic birthday attack can not be applied.

Roadmap. This paper is organized as follows. In Section 2 we briefly present a first improvement of the birthday attack, using the Hamming weight of the message. We proceed in Section 3 by providing and analyzing a memory efficient version of our attack. Section 4 is dedicated to the implementation of our attack and we further provide some experimental results. Finally, in Section 5 we provide some concluding remarks.

2. Improved birthday attack

Let $H_m = \sum_{i=0}^{n} m_i$, be the Hamming weight of the message m, where m_i are the bits of m. Let also c be the encryption of the message m. If H_m is small (or large) enough, then we shall see that we can practically find the message m.

Instead of the sets U_1 and U_2 we used in the introduction, we now consider the sets

$$U_{1,h_1} = \Big\{ \prod_{i \in I_1} u_i^{\varepsilon_i} \pmod p : \sum_{i \in I_1} \varepsilon_i = h_1, \varepsilon_i \in \{0,1\} \Big\},$$

$$U_{2,h_2} = \Big\{ c \prod_{i \in I_2} u_i^{-\varepsilon_i} \pmod p : \sum_{i \in I_2} \varepsilon_i = h_2, \varepsilon_i \in \{0,1\} \Big\},$$

for a partition (I_1, I_2) of $\{0,1,2,...,n\}$. The numbers h_1, h_2 are fixed to be not both zero and their sum is less than or equal to H_m. We define as U_m an upper bound of the Hamming weight of the message m. We set

$$W = W(U_m) = \{(i,j) \in \mathbf{Z}^2 : 0 < i + j \leq U_m\},$$

then we let (h_1, h_2) run in W. So, if we know an upper bound for the Hamming weight of the message m, then the algorithm is correct and deterministic. Moreover, we provide the pseudocode of the attack.

Birthday attack : Attack-1

```
INPUT: The encrypted message $c$, an upper bound for the Hamming weight
$U_m$ and the public key $pk$
OUTPUT: the message $m$
```

```
1: $I_1 \leftarrow \{0,1,...,\lceil n/2 \rceil\}, I_2 \leftarrow \{\lceil n/2 \rceil + 1, ..., n\}$
2: For $(h_1, h_2) \in W$
3:     if $U_{1,h_1} \cap U_{2,h_2} \neq \emptyset$
4:         return $m$
```

After $O(U_m^2)$ iterations (in the worst case) this algorithm will terminate and will return the message m. However, the basic disadvantage of this attack is the large memory requirements (see table 1).

3. A memory efficient version of attack-1

In this Section we present a further improvement. Remark that our previous attack succeeds always, since the positions of bits in the message m are contained in the union of I_1 and I_2. Furthermore, there exist a pair $(h_1, h_2) \in W$ such that $U_{1,h_1} \cap U_{2,h_2} \neq \emptyset$.

For instance, let $m = m_0 + 2m_1 + 2^2 m_2 + \cdots + 2^n$, and say that $m_1 = m_3 = m_n = 1$ and $m_j = 0$ $(j \neq 1, 3, n)$. Then, the numbers $1, 3, n$ i.e. the positions of the 1's in the binary expansion of m always belong to the union $I_1 \cup I_2$. If $1, 3 \in I_1$ and $n \in I_2$, then for $(h_1, h_2) = (2, 1)$ our algorithm will succeed.

If the sets I_1, I_2 are smaller, say of length $b < n/2$, our attack may fail, because the positions of 1's may not be contained in $I_1 \cup I_2$. So we need to choose many pairs (I_1, I_2) until the positions of 1's in the message, will be in $I_1 \cup I_2$. We pick b to be the maximum positive integer $b \leq n/2$, that our memory constraints allow. In the next subsection we explain this specific choice. Also, instead of the set W we consider the following set,

$$J = J(U_m) = \{(i, j) \in \mathbf{Z}^2 : 0 < i + j \leq U_m, 0 \leq i - j \leq 1\}.$$

We have $|J(U_m)| = U_m$ and $J(U_m) \subset W(U_m)$. So our algorithm becomes randomized due to the choice of I_1, I_2 and the set J. Below, we present the pseudocode of our attack. First, we need a function that generates the sets I_1 and I_2.

Function generate(n, b)

```
INPUT: $n, b$ ($2b < n$)
OUTPUT: $I_1, I_2$ of equal length $b$
1: $L \leftarrow \{n\}$
2: While $|L| < 2b$ and the elements of $L$ are distinct
3:     $x \xleftarrow{R} \{0, ..., n-1\}$
4:     $L \leftarrow L \cup \{x\}$
5: $I_1$ is the set generated from the first $b-$elements of $L$
6: $I_2$ is the set generated from the next $b-$elements of $L$
7: return $I_1, I_2$
```

If $b = n/2$ and n is even, the algorithm will return two sets I_1, I_2 with $|I_1| = n/2$,

$|I_2| = n/2+1$ and $I_1 \cup I_2 = \{0,1,...,n\}$. If n is odd, then a small modification of the algorithm again provides disjoint sets with $I_1 \cup I_2 = \{0,1,...,n\}$.

The first line stores n to set L since the message space is the set of $n+1$ bits, i.e. m is of the form $m = 2^n + \cdots$. Now we provide the pseudocode of the attack under the assumption that we know an upper bound of H_m, say U_m.

Birthday attack : Attack-2

```
INPUT: The cryptographic message $c$, an upper bound $U_m$ for the Hamming
weight $H_m$, a bound $b$ and the public key $pk$.
OUTPUT: the message $m$
1: While True
2:     $(I_1, I_2) \leftarrow generate(n, b)$
3:     For $(h_1, h_2) \in J(U_m)$
4:        if $U_{1,h_1} \cap U_{2,h_2} \neq \emptyset$
5:           return $m$
```

In line 3 we consider pairs $(h_1, h_2) \in J(U_m)$. Also, we note that the *real* information about the Hamming weight is $H_m - 1$ instead of H_m, since the largest bit is always 1.

Remark 3.1. If we store the hashes of the elements of the sets U_{1,h_1} and U_{2,h_2}, then less memory is used. This also will speed up the computation of the intersection $U_{1,h_1} \cap U_{2,h_2}$. In case we get a collision we construct the original elements in $U_{1,h_1} \cap U_{2,h_2}$ and check whether they are still equal.

3.0.1. Space complexity

Suppose that $(h_1, h_2) \in J$. We have $|U_{1,h_1}| = \binom{|I_1|}{h_1}$ and $|U_{2,h_2}| = \binom{|I_2|}{h_2}$. By choosing $|I_1| = |I_2| = b$, we get $|U_{1,h_1}| = \binom{b}{h_1}$ and $|U_{2,h_2}| = \binom{b}{h_2}$. So the memory requirements to construct the sets $U_{1,h_1}, U_{2,h2}$ are

$$\binom{b}{h_1} \log_2 p \text{ and } \binom{b}{h_2} \log_2 p \text{ bits, respectively.}$$

Overall, the space bit-complexity is:

$$C_{mem}(J) = \log_2 p \max_{(h_1,h_2)\in J(U_m)} \Big\{ \binom{b}{h_1} + \binom{b}{h_2} \Big\}.$$

Since, $J \subset W$ we get $C_{mem}(J) < C_{mem}(W)$. Furthermore, $\binom{b}{k} < \left(\frac{eb}{k}\right)^k$ and $b \leq n/2$, thus $C_{mem}(J) = O\Big(\big(\frac{2be}{U_m}\big)^{\frac{U_m+1}{2}} \log_2 p\Big) = O\Big(\big(\frac{ne}{U_m}\big)^{\frac{U_m+1}{2}} \log_2 p\Big)$.

3.0.2. Time complexity

Let $(h_1, h_2) \in J(U_m)$ and M_p is the bit-complexity of the multiplication of two integers $\mod p$. For instance, the usual multiplication $\mod p$ gives $M_p = O((\log_2 p)^2)$. To construct the sets U_{1,h_1} and U_{2,h_2} (ignoring for the moment the cost for the inversion) we need

$$h_1 \binom{b}{h_1} M_p \text{ and } (h_2+1) \binom{b}{h_2} M_p \text{ bit-operations respectively.}$$

U_m	7	8	9	10	11
GBytes($C_{mem}(J)$)	1.76	3.41	39.9	76.4	745.76
GBytes($C_{mem}(W)$)	11116	151467	$1.8 \cdot 10^6$	$1.9 \cdot 10^7$	$1.87 \cdot 10^{11}$

Table 1: Assume that $n = 232$, thus p has 2048 bits. Also we set $b = n/2$. For instance, if the upper bound for the Hamming weight is 10, then we need at least 76.4Gb for the storage using Attack-2. For $U_m \geq 11$ attack-2 is infeasible, so we need to choose a smaller b.

Therefore $C_{time}(J) =$

$$M_p\Big(\sum_{(h_1,h_2)\in J(U_m)} h_1\binom{b}{h_1} + (h_2+1)\binom{b}{h_2} + \frac{n}{2}\Big) = O\Big(M_p U_m^2\big(\frac{ne}{U_m}\big)^{\frac{U_m}{2}}\Big).$$

We can assume that $M_p = O((\log_2 p)^{1+\varepsilon})$ $(0 < \varepsilon < 1)$ by using fast multiplication. Again, as in the space complexity, $C_{time}(J) < C_{time}(W)$. More precisely, we want

$$(\text{average rounds for success}) \cdot C_{time}(J) < C_{time}(W).$$

We shall study the previous inequality in Section 4, after the computation of the success probability.

The exhaustive search for a message with the same assumption $H_m \leq U_m$, has asymptotic running time:

$$O\Big(M_p U_m^2 \binom{n}{U_m}\Big) = O\Big(M_p U_m^2 \big(\frac{ne}{U_m}\big)^{U_m}\Big),$$

which is worse than $C_{time}(J)$. Also, for a practical comparison between exhaustive search and attack-2[1], see table 4.

If we do not know an upper bound U_m, we can set $U_m = \lceil n/2 \rceil + 1$. Indeed, if the hamming weight is $\leq \lceil n/2 \rceil + 1$ we apply the attack to the original message m. If $H_m > \lceil n/2 \rceil + 1$, then we apply the attack to the message m' of Lemma 3.4, which has $H_{m'} \leq \lceil n/2 \rceil + 1$.

3.1. Success probability

In this subsection we compute the success probability of attack-2, in the case where $U_m = H_m$.

Definition 3.2. Let m be a binary string of length $n+1$. We define the support of m to be the set $sup(m) = \{i \in \mathbb{Z}_{>0} : m = \sum_{i=0}^{n} x_i 2^i \textit{ with } x_i \neq 0\}$.

Definition 3.3. Let b a positive integer and integer x such that $0 \leq x \leq b$. We set $B_b(x) = \binom{b}{x}$.

Let I_1, I_2 disjoint subsets of $\mathcal{N} = \{0, 1, ..., n\}$, both of length $b \leq (n+1)/2$. Let also a set $P \subset \mathcal{N}$ with $1 < |P| < b$ and assume that $|P| = h_1 + h_2$. Then,

$$\mathbb{P} = Pr\big(I_1, I_2 \xleftarrow{R} \mathcal{N} : |I_1 \cap P| = h_1,\ |I_2 \cap P| = h_2\big) = \frac{B_b(h_1)B_b(h_2)}{B_{n+1}(h_1+h_2)}.$$

[1]The code can be found in `https://goo.gl/WLUTQ4`

To see this we work as follows. Firstly, we set $M = \mathcal{N} - P$ (M is of size $n+1-h_1-h_2$). Then, we pick I_1, I_2 from $\mathcal{N}$ without replacement (so $I_1 \cap I_2 = \emptyset$). We want the b elements of I_1 to be a selection of h_1 elements of P and $b-h_1$ elements of M. Similar for I_2. The set $\mathcal{N}$ needs to be partitioned in type $(b, b, n+1-2b)$, i.e. into parts of size b, b and $n+1-2b$.

So the probability is:

$$\mathbb{P} = \frac{\binom{h_1+h_2}{h_1,h_2}\binom{n+1-h_1-h_2}{b-h_1,b-h_2,n+1-2b}}{\binom{n+1}{b,b,n+1-2b}} = \frac{\binom{b}{h_1,b-h_1}\binom{b}{h_2,b-h_2}}{\binom{n+1}{h_1+h_2,n+1-h_1-h_2}} = \frac{B_b(h_1)B_b(h_2)}{B_{n+1}(h_1+h_2)}.$$

For the second equality, we used simple algebra and the definition of multinomial coefficient $\binom{a}{b,c,d} = \frac{a!}{b!c!d!}$ (i.e. the ways to partition a set of size $a = b+c+d$ into parts of sizes b, c, d).

Consider now a message m with Hamming weight H_m. In attack-2 the pairs $(h_1, h_2) \in J(H_m)$. Therefore only one pair (h_1, h_2) have a sum equal to H_m (this (h_1, h_2) is the successful $(h_1, h_2) \in J(H_m)$). We denote this pair as (α_1, α_2). If H_m is even, then

$$(\alpha_1, \alpha_2) = (H_m/2, H_m/2), \text{ else } (\alpha_1, \alpha_2) = (\lceil H_m/2 \rceil, \lfloor H_m/2 \rfloor).$$

If we set $P = sup(m)$, then the probability to choose I_1, I_2 such that $|I_i \cap P| = \alpha_i$ ($i = 1, 2$) is

$$Pr_s(n, b, H_m; \alpha_1, \alpha_2) = \frac{B_b(\alpha_1)B_b(\alpha_2)}{B_{n+1}(H_m)}$$

This is the success probability of attack-2[2] (although $Pr_s(n, b, h; h_1, h_2) \neq 0$, for all $h = h_1 + h_2 < H_m$, $(h_1, h_2) \in J(H_m)$, the success probability of the attack for these cases is obviously zero). Also, this formula allow us to compute the average rounds of success, $R_s \approx \frac{1}{Pr_s}$. Finally, to explain why we choose b as the maximum positive integer $\leq n/2$ that our memory constraints allow, it is enough to remark that, if we fix H_m and n, then Pr_s is non-decreasing sequence with respect to b. This is straightforward, since $\binom{b}{x}$ is non-decreasing sequence with respect to b.

Finally, if we know some bits of the message, say they are in the positions $\{j_1, ..., j_k\}$, then we choose I_1 and I_2 from the set $\{1, 2, ..., n\} - \{j_1, ..., j_k\}$. In this way we improve the success probability and the running time for the same choice of b.

3.2. The case where the Hamming weight is large

If the message m has large enough Hamming weight, say $H_m = n + 1 - \varepsilon$, where ε is a small positive integer, then again we can reduce the problem to one with small Hamming weight. Let $c = Enc(m)$ and $c' = c^{-1}u_n^2 \prod_{i=0}^{n-1} u_i$. We provide the following Lemma.

[2] $Pr_s \approx 0.66 - 0.12\,H - 0.009\,b + 0.0095\,H^2 + 0.00008\,b^2 - 0.00001168\,e^H$. This is experimental formula for $n = 232, H \geq 2$ and $50 \leq b \leq n/2$. We used the Fit function of `Wolfram Mathematica` using data from success probability.

Lemma 3.4. *The decryption of c' is $m' = 2^{n+1}+2^n-m-1$, where $H_{m'} = \varepsilon+1$.*

Proof. If $m = \sum_{i=0}^{n} 2^i m_i$, then $c = \prod_{i=0}^{n} u_i^{m_i}$. Since, $c' = c^{-1}u_n^2 \prod_{i=0}^{n-1} u_i$, and substituting c we get $c' = u_n \prod_{i=0}^{n} u_i^{1-m_i}$. So the decryption of c' is a message m' with binary digits,

$$(2-m_n, 1-m_{n-1}, ..., 1-m_0) = (1, 1-m_{n-1}, ..., 1-m_0). \qquad (2)$$

So from (2) we get, $H_{m'} = (n+1) - (H_m - 1) = \varepsilon + 1$. The result follows. $\square$

4. Experimental results

In this Section, we present some experimental results of the implementation of the attack-2 described above. In our experiments we used the computer algebra system Sagemath[3] in a machine with I3-4150 2-core 3.5GHz processor and 16Gb memory. Furthermore, in our implementation we used remark 3.1 by including md5 hash. The last row of table 2 contains the CPU time for a successful attack using the algorithm attack-2. To be more accurate, we executed 10 times the attack-2 (for the same message) and we computed the average CPU time. In all the examples as an upper bound we considered the H_m (i.e. $U_m = H_m$). Of course, if we do not know the Hamming weight and

len(p)	600			1024			2048	
U_m	8	9	10	8	9	10	7	8
Attack-2	42s	300s	7.5m	8.1m	1.1h	1.96h	1.17h	2.15h

Table 2: We used $b = n/2$, where $n = 84, 130, 232$, for *len*$(p) = 600, 1024, 2048$, respectively.

the upper bound is much larger, the success probability will not change, but the the space and time complexity will be increased. We can estimate the average time if we multiply the average CPU time per round (this can be computed experimentally and depends on the machine) with the average rounds given by theory.

Finally, we checked the validity of the inequality $R_s \cdot C_{time}(J) < C_{time}(W)$, where R_s is the number of rounds on average for attack-2, for suitable values of (n, b, H_m). The left-hand side of inequality is the overall complexity of attack-2 and the right-hand side the complexity of attack-1.

len(p)	1024			2048		
H_m	9	10	11	9	10	11
λ	$3.4 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$	$1.1 \cdot 10^{-4}$	$2.8 \cdot 10^{-5}$	$6.5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$

Table 3: We set $\lambda = R_s \cdot \frac{C_{time}(J(H_m))}{C_{time}(W(H_m))}$ and $b = n/2$.

Finally, we provide a table which compares the exhaustive search and attack-2.

[3]`www.sagemath.org`

H_m	4	5	6
μ	0.0542	0.0114	0.00360

Table 4: We set $\mu = \frac{\text{average time of attack-2}}{\text{average time of exhaustive search}}$ and $n = 84$. We executed 50 random instances for each column. For each instance we measured the time for attack-2 and exhaustive search and then we computed the average time.

5. Conclusions

In the present work we provided a memory and time efficient attack on NSK cryptosystem based on birthday paradox problem. We analyzed its success probability and it was implemented and tested successfully for messages having *low* or *large* Hamming weight for $n = 232$. To avoid the provided attacks, *balanced* messages should be chosen, i.e. with Hamming weight $\approx n/2$.

Furthermore, in [3, 8] the authors provide the small prime packing method to achieve greater information rate. Our attack can be applied only if the parameter $\gamma = 2$ (using the notation of [3]). In this case, we can do even better, since the message space is smaller. This may be a matter of future studies.

Also, our attack can be easily parallelized. We can dedicate each node to execute one inner loop of the attack (i.e. for each value of (h_1, h_2) we dedicate one node). Such an implementation can improve the time of our attack and also find messages with larger Hamming weight.

Acknowledgement. The authors are indebted to the reviewers for their helpful suggestions.

References

[1] Eric Brier, Remi Geraud and David Naccache, Exploring Naccache-Stern Knapsack Encryption. eprint 2017/421.

[2] J. Bringer, H. Chabanne and Q. Tang, An Application of the Naccache-Stern Knapsack Cryptosystem to Biometric Authentication, 2007 IEEE Workshop on Automatic Identification Advanced Technologies, p. 180–185.

[3] B. Chevallier-Mames, D.Naccache and Jacques Stern, Linear Bandwidth Naccache-Stern Encryption. SCN 2008, p.327–339.

[4] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. NY, USA 1979.

[5] D. Gordon, Discrete logarithms in $GF(p)$ using the number field sieve. SIAM J. Discrete Math. **6** (1993), no. **1**, p.124–138.

[6] G. Micheli, J. Rosenthal and R. Schnyder, Hiding the carriers in the polynomial Naccache Stern knapsack cryptosystem. TWCC2015, Paris, France.

[7] G. Micheli and M. Schiavina, A general construction for monoid-based knapsack protocols, Adv. in Math. of Communications **8 (3)**, p.343–358.

[8] G. Micheli, J. Rosenthal and R. Schnyder, An Information Rate Improvement for a Polynomial Variant of the Naccache-Stern Knapsack Cryptosystem, LNEE **358** (1996), Springer, Cham, p.173–180.

[9] D. Naccache and J. Stern, A new public key cryptosystem, Proceedings of Eurocrypt 97, LNCS **1233** (1997), Springer-Verlag, p.27–36.

[10] D. Panario, X. Gourdon and P. Flajolet, An analytic approach to smooth polynomials over finite fields. ANTS III, LNCS **1423** (1998), p.226–236.