An Improved LPN Algorithm

Éric Levieil and Pierre-Alain Fouque

École normale supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, France
{Eric.Levieil, Pierre-Alain.Fouque}@ens.fr

Abstract. HB$^+$ is a shared-key authentication protocol, proposed by Juels and Weis at Crypto 2005, using prior work of Hopper and Blum. Its very low computational cost makes it attractive for low-cost devices such as radio-frequency identification(RFID) tags. Juels and Weis gave a security proof, relying on the hardness of the "learning parity with noise" (LPN) problem. Here, we improve the previous best known algorithm proposed by Blum, Kalai, and Wasserman for solving the LPN problem. This new algorithm yields an attack for HB$^+$ in the detection-based model with work factor 2^{52}.

1 Introduction

Providing lightweight and secure cryptographic protocols for radio-frequency identification (RFID) tags is an area under quick development. The HB protocol family is one of the most promising in this field and uses very few operations and gates on the chip. The original protocol has been proposed by Hopper and Blum [8].

All protocols in the HB family rely on the computational hardness of the LPN problem.

The LPN Problem. In machine learning theory, this problem is described in the *uniform distribution model* where the algorithm only has access to a source of random samples. The LPN problem is the following:

Definition 1. (LPN Problem)
Let $\langle \cdot | \cdot \rangle$ denote the binary inner product. Let $\mathbf{s}$ be a random k-bit vector, let $\varepsilon \in]0, 1/2[$ be a constant noise parameter, let Ber_ε be the Bernoulli distribution with parameter ε (so if $\nu \leftarrow Ber_\varepsilon$ then $Pr[\nu = 1] = \varepsilon$ and $Pr[\nu = 0] = 1 - \varepsilon$), and let $A_{\mathbf{s},\varepsilon}$ be the distribution defined as

$$\left\{ \mathbf{a} \leftarrow \{0,1\}^k ; \nu \leftarrow Ber_\varepsilon : (\mathbf{a}, \langle \mathbf{s} | \mathbf{a} \rangle \oplus \nu) \right\}$$

Let $A_{\mathbf{s},\varepsilon}$ denote an oracle which outputs independent samples according to this distribution. Algorithm M is said to (q, t, m, θ)-solve the $LPN_{k,\varepsilon}$ problem if

$$Pr[\mathbf{s} \leftarrow \{0,1\}^k : M^{A_{\mathbf{s},\varepsilon}}(1^k) = \mathbf{s}] \geq \theta$$

and furthermore M runs in time at most t, memory at most m, and makes at most q queries to its oracle.

R. De Prisco and M. Yung (Eds.): SCN 2006, LNCS 4116, pp. 348–359, 2006.

This is the definition of Regev [13], and Katz *et al.* [10]. An alternative (and equivalent) definition can be found for example in [9].

In the following, we define δ as $\delta = 1 - 2\varepsilon$. This notation will be better to analyze the complexity of the algorithms. For the classical parameters $\varepsilon = 1/4$ and $1/8$, δ is equal to $1/2$ and $3/4$.

The LPN problem is an average-case version of the following problem: given a set of equations over GF(2), find a vector $\mathbf{s}$ that maximally satisfies the equations. The latter problem has been first studied as the decoding of a random linear code and has been proved to be NP-hard by Berlekamp *et al.* in [1]. It has also be shown to be hard to approximate even within a factor of two by Hastad in [7]: it is hard to find a $\mathbf{s}$ that satisfies more than half of the optimum number of equations. In the LPN problem, the instances (set of equations and values) maybe do not represent the worst case of the problem, but studies of the average-case hardness of this problem have been proposed in [8,11,2,3,13].

The HB Protocol. The Reader and the Tag share public values k, ϵ, u and r, and a k-bit secret value $\mathbf{s}$. To be authenticated by a Reader, the Tag and the Reader repeat the following round many times:

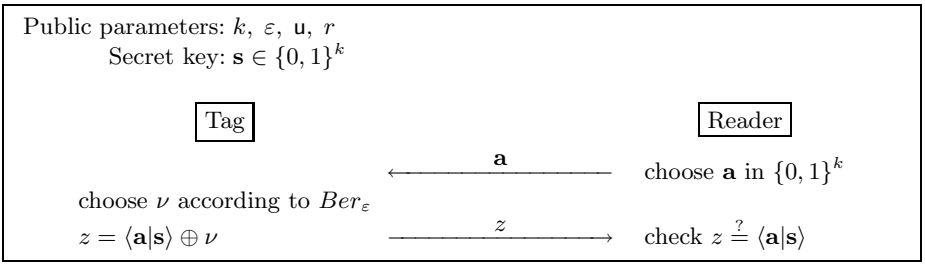

Fig. 1. Round identification of HB scheme

The round is repeated r times so that the Reader has good confidence in the answers of the Tag. To this end, the protocol has a parameter u so that if the number of errors is less than $r \cdot \mathsf{u}$, then the authentication is successful. Typical values of ε are $1/4$ or $1/8$. This value cannot be chosen too close to $1/2$, otherwise the probability of rejecting an honest Tag increases too much. If it is too close to 0, then you can find k independent equations without errors easily.

A completeness error occurs when an honest Tag is rejected. We want the probability of a completeness error, P_c to be less than 2^{-40}.

A soundness error occurs when a Tag making random answers succeeds in authenticating itself. We want the probability of a soundness error, P_s to be less than 2^{-80}.

Given ε, u, and r, we can compute the value of P_c and P_s. Let

$$g(x,y) = \left(\frac{x}{y}\right)^x \left(\frac{1-x}{1-y}\right)^{1-x}.$$

The probabilities P_c and P_s can be expressed as sums of the tail of a binomial distribution, and using Stirling's formula, we obtain:

$$P_c \sim g(\mathsf{u}, \varepsilon)^{-r} \text{ and } P_s \sim g(\mathsf{u}, 1/2)^{-r}.$$

For each ε, we compute the values of u and r such that r is as small as possible, and the above conditions on P_c and P_s are true.

We gather the result in the following table:

ε	0.01	0.05	0.125	0.25	0.4	0.49
u	0.112	0.181	0.256	0.348	0.442	0.495
r	159	249	441	1164	7622	554360

Fig. 2. Values for $r(\varepsilon)$

In this protocol, secure only against passive attackers, an adversary gets pairs of the form $(\mathbf{a}, \langle \mathbf{a}|\mathbf{s}\rangle \oplus \nu)$ and must compute $\mathbf{s}$.

There is a simple active attack against HB. Indeed, if an adversary can change the challenge, it can send several times the same value $\mathbf{a}$. Since the answer is incorrect with probability $\varepsilon \ll 1/2$, majority votes enable to recover $\langle \mathbf{a}|\mathbf{s}\rangle$. Then, k scalar products with independent $\mathbf{a}$, allow to entirely recover $\mathbf{s}$.

The HB$^+$ Protocol. Consequently, Juels and Weis in [9] have proposed HB$^+$, a protocol robust against active attacks in the detection-based model. The idea is to use a blinding factor. The Tag and the Reader now share two k-bit secret values $\mathbf{s}_1$ and $\mathbf{s}_2$. The protocol round is the following:

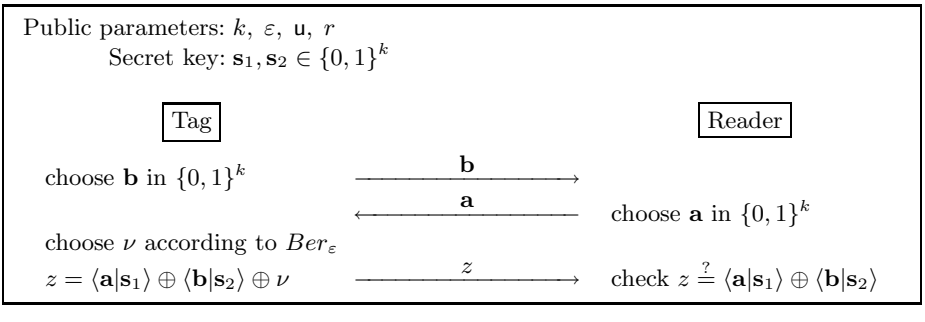

Fig. 3. Round identification of HB$^+$ scheme

The security of this protocol relies on the LPN problem for $\mathbf{s}_2$. Indeed, an active attacker can interact with the Tag in the first stage of his attack and then tries to impersonate the Tag against a Reader. The attacker can choose $\mathbf{a} = 0^k$, obtaining values of $\langle \mathbf{b}|\mathbf{s}_2\rangle \oplus \nu$. If he can solve a $LPN_{k,\varepsilon}$ problem, he can recover $\mathbf{s}_2$. Then, once $\mathbf{s}_2$ is recovered, he must recover $\mathbf{s}_1$. This can be easily done since

the attacker is now faced to a HB protocol that can be defeated by choosing the same $\mathbf{a}$ many times to know $\langle \mathbf{a} | \mathbf{s}_1 \rangle$ with high confidence.

Remark 1. The length of $\mathbf{s}_1$ is used in HB$^+$ proofs only to guarantee that the attack that consists to guess $\mathbf{s}_1$ is not efficient. But $|\mathbf{s}_1| = 80$ is sufficient to guarantee 80 bits security.

Related Works. Gilbert, Robshaw, and Sibert [5] found a man-in-the-middle attack against HB$^+$ when the adversary can interact with the Reader and the Tag during the same round. Consequently, Bringer, Chabanne and Dottax [4] proposed a new derived protocol, HB^{++}, that is resistant to a generalization of Gilbert *et al.* attack.

The security proof of HB$^+$ of Juels and Weis has also been simplified and improved by Katz and Sun Shin [10], using a recent result of Regev [13].

Our Work. The best algorithm to solve the LPN problem had been proposed by Blum, Kalai and Wasserman in [3], hereafter denoted as the BKW algorithm. The parameter security k of the HB protocols has therefore been estimated using the complexity of this algorithm. But, Blum, *et al.* only give a high-level description of the BKW algorithm and estimate the overall subexponential complexity of order $2^{O(k/\log k)}$. Juels and Weis in [9] propose practical parameters by giving an effective estimate of the query and time complexity.

However, they have not seen that the BKW algorithm could be improved. In this paper, we present in detail the BKW algorithm, analyze it precisely and give its complexity. Then, we propose an improvement for the final stage of the algorithm. Instead of throwing away almost all equations, we manage to use every one. Therefore, we need much less queries. We also use a Walsh-Hadamard transform to speed up this phase. Then, we give an heuristic improvement using Wagner's method to solve the Generalized Birthday Paradox of [14]. Finally, we compare the performance of the BKW and our algorithm. Our algorithm yields an attack in 2^{52} for the actual key-length of HB$^+$ as proposed by Juels and Weis in Crypto '05, instead of the conjectured 2^{80}.

The next section is a full analysis of BKW. In the third one, we describe and prove an improved algoritm (LF1). In the fourth, we propose an heuristic algorithm (LF2) that is more efficient in practice than LF1, as will be shown in the last section, that is focused on implementation techniques and complexity results.

2 The BKW Algorithm

The aim of this section is to describe the algorithm and the ideas behind. We also make a detailed and precise analysis of the success probability, using explicit Chernoff's bounds that are recalled in appendix A. This part is not contained in the previous papers.

2.1 Description

In the following, we denote by a and b two different parameters from $\mathbf{a}$ and $\mathbf{b}$. We use those very close notations since the first come from Blum *et al.* in [3] and the second come from [10,9].

To solve the learning parity with noise problem Blum, Kalai and Wasserman in [3] use the following idea: by picking carefully a few well-chosen vectors in a quite large set of samples and computing the xor of these vectors, we can find basis vectors, *i.e.* e_j where the jth bit is a one and all other coordinates are null. First, we have to choose a parameter a. Typical values run from 4 to 6. The main point is that we are able to find $2^a = O(k/\log k)$ vectors such that

$$\mathbf{a}_{i_1} \oplus \cdots \oplus \mathbf{a}_{i_{2^a}} = e_j. \tag{1}$$

Then, since the number (2^a) of vectors is small, the bias of the equations obtained is not too small. Consequently, if we have enough independent combinations of vectors equals to e_j, then a majority vote enables us to recover the correct value of $\mathbf{s}_j$ since $\langle \mathbf{s} | \mathbf{a}_{i_1} \oplus \cdots \oplus \mathbf{a}_{i_{2^a}} \rangle = \langle \mathbf{s} | e_j \rangle = \mathbf{s}_j$.

We set b to be $\lceil \frac{k}{a} \rceil$. From now on, we will assume for simplicity that $k = a \cdot b$. The algorithm has to search enough such independent combinations of the $\mathbf{a}_i$'s. To this end, it splits the k bits of $\mathbf{a}_i$ into a blocks of b bits. Then, according to the last b bits, the algorithm computes 2^b equivalent classes and classifies the $\mathbf{a}_i$ according to these bits. In each class, it chooses a vector at random, performs the xor with all other vectors of the same class and finally throws away this vector. Therefore, at the end of this step, in each equivalent class, the last b bits are zeroes. This procedure is called recursively beginning at the last block until the second block. Then, we keep only the equations that are of the form $\langle \mathbf{s} | e_j \rangle = \nu$. If there are enough of such equations, the majority vote says something meaningful about the value of $\mathbf{s}_j$ with high probability. By applying this algorithm for different j, we can recover all the bits of $\mathbf{s}$.

2.2 Analysis

Now, we will analyse this algorithm. To this end, we present two lemmas that will be helpful. The first lemma analyses the bias at the end of the recursion steps, while the second lemma estimates the number of elements and is useful to show an invariant of the algorithm.

Lemma 1. *If* $(\mathbf{a}_1, \nu_1), \ldots, (\mathbf{a}_n, \nu_n)$ *are the result of* n *queries to* $A_{\mathbf{s},\varepsilon}$*, then the probability that:*

$$\langle \mathbf{a}_{i_1} \oplus \ldots \oplus \mathbf{a}_{i_n} | s \rangle = \nu_{i_1} \oplus \ldots \oplus \nu_{i_n}$$

is equal to $\frac{1+\delta^n}{2}$.

This lemma is equivalent to lemma 3 of [3].

Proof. For $n = 1$, the lemma is trivially true. By induction, and using the $\mathbf{a}_i$'s independence, we have:

$$\begin{aligned} Pr[\langle \mathbf{a}_{i_1} \oplus \ldots \oplus \mathbf{a}_{i_n} | \mathbf{s} \rangle = \nu_{i_1} \oplus \ldots \oplus \nu_{i_n}] = \\ Pr[\langle \mathbf{a}_{i_1} \oplus \ldots \oplus \mathbf{a}_{i_{n-1}} | \mathbf{s} \rangle = \nu_{i_1} \oplus \ldots \oplus \nu_{i_{n-1}}] Pr[\langle \mathbf{a}_{i_n} | \mathbf{s} \rangle = \nu_{i_n}] \\ + Pr[\langle \mathbf{a}_{i_1} \oplus \ldots \oplus \mathbf{a}_{i_{n-1}} | \mathbf{s} \rangle \neq \nu_{i_1} \oplus \ldots \oplus \nu_{i_{n-1}}] Pr[\langle \mathbf{a}_{i_n} | \mathbf{s} \rangle \neq \nu_{i_n}] \\ = \frac{1}{2}(1+\delta^{n-1})\frac{1}{2}(1+\delta) + \frac{1}{2}(1-\delta^{n-1})\frac{1}{2}(1-\delta) = \frac{1}{2}(1+\delta^n). \end{aligned}$$

□

The following definition and lemma are equivalent to definition 2 and lemma 4 in [3].

Definition 2. *Let $A_{\mathbf{s},\delta,i}$ be the distribution defined as*

$$\left\{ \mathbf{a} \leftarrow \{0,1\}^{(a-i)b} \times \{0\}^{ib}; \nu \leftarrow Ber_{(1+\delta)/2} : (\mathbf{a}, \langle \mathbf{s} | \mathbf{a} \rangle \oplus \nu) \right\}$$

Also, let $A_{\mathbf{s},\delta,i}$ denote an oracle which outputs independent samples according to this distribution. We define an $(\mathbf{s}, \delta, i)$-set of size n as the result of n queries to oracle $A_{\mathbf{s},\delta,i}$.

Lemma 2. *Assume we are given an $(\mathbf{s}, \delta, i)$-set of size n. We can in time $O(n)$ construct an $(\mathbf{s}, \delta^2, i+1)$-set of size $n - 2^b$.*

Proof. Let us call $(\mathbf{a}_1, \nu_1), \ldots (\mathbf{a}_n, \nu_n)$ the elements of the $(\mathbf{s}, \delta, i)$-set. Vectors $\mathbf{a}_j$ have their last ib coordinates equal to 0. We partition them with regard to their value on the precedent b coordinates, obtaining a partition with at most 2^b classes. In each class, we pick a vector at random and add it (modulo 2) to all the others vectors in that class, and then discard it. Compiling the results for each class, and using lemma 1, we obtain a $(\mathbf{s}, \delta^2, i+1)$-set of size (at least) $n - 2^b$. □

Consequently, according to lemma 1, at the end of the algorithm, the bias of the equation 1 is $\delta^{-2^{a-1}}$ where $\delta = (1 - 2\varepsilon)$.

The next lemma give the number of combinations of equations that must xored to the vector e_j in order to have a high probability of success.

Lemma 3. *Let $A_{\mathbf{s}_i,\delta,a}$ be the distribution defined as*

$$\left\{ \nu \leftarrow Ber_{(1+\delta^{2^{a-1}})/2} : \mathbf{s}_i \oplus \nu) \right\}$$

Also, let $A_{\mathbf{s}_i,\delta,a}$ denote an oracle which outputs independent samples according to this distribution.

Then it is possible to guess the value of $\mathbf{s}_i$ with $c\delta^{-2^a}$ calls to the oracle with error probability bound by $2e^{-c/20}$.

Proof. We define that a sample $\mathbf{a}_i, \nu$ is *compatible* with the i-th bit of $\mathbf{s}$ if $\mathbf{s}_i \cdot \mathbf{a}_i = \nu$.

The idea for guessing the ith bit is to compute for $\mathbf{s}_i = 0$ and $\mathbf{s}_i = 1$, the number of compatible samples and predict that $\mathbf{s}_i = b$ according to the majority number of compatible samples.

Therefore, in order to upper bound the probability of failure of the BKW algorithm, we have to upper bound the following probability where $\mathbf{x}_i = 1 - \mathbf{s}_i$: $\Pr[\mathbf{x}_i$ has more compatible samples than $\mathbf{s}_i]$.

To this end, we upper bound the previous probability by the sum of two more easily computable probabilities.

$$\mathsf{pr}_1 = \Pr[\mathbf{x}_i \text{ is compatible with at least } \frac{1+\alpha\delta^{2^{a-1}}}{2} \cdot N \text{ samples}]$$

$$\mathsf{pr}_2 = \Pr[\mathbf{s}_i \text{ is compatible with at most } \frac{1+\alpha\delta^{2^{a-1}}}{2} \cdot N \text{ samples}]$$

If $\mathbf{s}_i$ is correct, then it is compatible with $\mathbf{a}_i, \nu$ with probability $\frac{1+\delta^{2^{a-1}}}{2}$ and otherwise with probability $1/2$. We will justify the last assertion later. Let us denote by N the number of equations $c\delta^{-2^a}$. The random variable X_j for $j = 1$ to N, is equal to 1 if $\mathbf{s}_i$ is compatible with the jth sample, and 0 otherwise.

Bounding pr_2. The expectation of X_j, $\mathbf{E}[X_j] = \Pr[X_j = 1] = \frac{1+\delta^{2^{a-1}}}{2}$ We sum these random variables and denote by X their sum, $X = \sum_{j=1}^{N} X_j$, and so $\mathbf{E}[X] = N \cdot \mathbf{E}[X_j] = N \cdot \frac{1+\delta^{2^{a-1}}}{2}$.

pr_2 is equal to $\Pr[X \leq (1+\alpha\delta^{2^{a-1}})(N/2)]$ which can be bounded using Chernoff bounds (cf. appendix A). To this end, we have $(1-\Delta)\mathbf{E}[X] = (1+\alpha\cdot\delta^{2^{a-1}})\cdot(N/2)$.

To determine Δ, we divide the right-hand side by $\mathbf{E}[X]$, and we get

$$1 - \Delta = \frac{1+\alpha\cdot\delta^{2^{a-1}}}{1+\delta^{2^{a-1}}} \approx 1 - (1-\alpha)\cdot\delta^{2^{a-1}}$$

and so $\Delta = (1-\alpha)\cdot\delta^{2^{a-1}}$.

$$\begin{aligned}\mathsf{pr}_2 &\leq e^{-(c\delta^{-2^a}/4)\cdot(1+\delta^{2^{a-1}})\cdot(1-\alpha)^2\delta^{2^{a-1}}\cdot 2} \leq e^{-(c/4)(1-\alpha)^2(1+\delta^{2^{a-1}})} \\ &\leq e^{-(c/4)(1-\alpha)^2}\end{aligned}$$

Bounding pr_1. In order to upper bound pr_1, we use the fact that for a bad guess, the expectation $\mathbf{E}[X]$ is equal to $\frac{N}{2}$, and the theorem 3 in appendix A. We have

$$\mathsf{pr}_1 \leq \Pr[X > (1+\Delta)\mu] \leq e^{-N\Delta^2/(3\cdot 2)}$$

Here, $\Delta = \alpha\cdot\delta^{2^{a-1}}$ and as $N = c\delta^{-2^a}$, then $N\Delta^2 = c\alpha^2$ and

$$\mathsf{pr}_1 \leq e^{-c\alpha^2/6}$$

$\Pr[\mathbf{x_i} \cdot \mathbf{a_i} = \nu | \mathbf{x_i} = 1 - \mathbf{s_i}] = 1/2$. It remains to justify that when $\mathbf{s}_i$ is not correct, then $\mathbf{s}_i \cdot \mathbf{a}_i = \nu$ with probability $1/2$. Let $(\mathbf{a}, \nu) \leftarrow A_{\mathbf{s},\varepsilon}$ and $\mathbf{x}_i = 1 - \mathbf{s}_i$. We want to show that $\Pr[\mathbf{x}_i \cdot \mathbf{a}_i = \nu] = 1/2$. To this end, we split the event into two incompatible events:

$$\begin{aligned}
\Pr[\mathbf{x}_i \cdot \mathbf{a}_i = \mathbf{s}_i \cdot \mathbf{a}_i] \Pr[\mathbf{s}_i \cdot \mathbf{a}_i = \nu] &+ \Pr[\mathbf{x}_i \cdot \mathbf{a}_i \neq \mathbf{s}_i \cdot \mathbf{a}_i] \Pr[\mathbf{s}_i \cdot \mathbf{a}_i \neq \nu] \\
&= \Pr[(\mathbf{x}_i \oplus \mathbf{s}_i) \cdot \mathbf{a}_i = 0] \Pr[\mathbf{s}_i \cdot \mathbf{a}_i = \nu] \\
&\quad + \Pr[(\mathbf{x}_i \oplus \mathbf{s}_i) \cdot \mathbf{a}_i \neq 0] \Pr[\mathbf{s}_i \cdot \mathbf{a}_i \neq \nu] \\
&= (1/2) \cdot (\Pr[\mathbf{s}_i \cdot \mathbf{a}_i = \nu] + \Pr[\mathbf{s}_i \cdot \mathbf{a}_i \neq \nu]) \\
&= 1/2
\end{aligned}$$

since the first equation comes from the fact that $\mathbf{a}_i$ and ν are independent and second equation as since $\mathbf{x}_i \neq \mathbf{s}_i$ and $\mathbf{a}_i$ is taken uniformly, the probability that $\mathbf{a}_i = 0$ is exactly $1/2$.

Choosing $\alpha = 3 - \sqrt{6}$ finishes the proof. □

The main ingredient of the algorithm is that with a small number of vectors, a combination of such vectors yields a basis vector. If the number required is too high, then the bias of equation (1) is too small and the number of queries becomes very large.

We are now ready to prove the theorem that gives the complexity of the BKW algorithm.

Theorem 1. *For $k = a \cdot b$, the BKW algorithm $(q = 20 \cdot \ln(4k) \cdot 2^b \cdot \delta^{-2^a}, t = O(kaq), m = kq, \theta = 1/2)$-solves the $LPN_{k,\varepsilon}$ problem .*

Proof. The original queries form a $(\mathbf{s}, \delta, 0)$-set of size q. Using lemma 2 $(a-1)$ times, we obtain a $(\mathbf{s}, \delta^{2^{a-1}}, 0)$-set of size $q-(a-1)2^b$. Keeping only the equations with one non-zero coordinate, then using lemma 3, we obtain one bit of $\mathbf{s}$ with error probability at most $1/(2k)$. Repeating this for different bits of $\mathbf{s}$, we find $\mathbf{s}$ with probability at least $1/2$. □

3 An Improved Algorithm: LF1

This algorithm is a variation of the BKW algorithm. In the BKW algorithm, the last step wastes a lot of time and queries. The idea is to deal in the last step with equations over b bits instead of one. Moreover, we will use the Walsh-Hadamard transform to quickly find the best possibility over b bits.

This algorithm does not use any heuristics. This section is devoted to prove the correctness and the performances of this algorithm. It needs lots of queries (less than BKW, though).

We now state our main theorem:

Theorem 2. *For $k = a \cdot b$, there is an algorithm that*
$q = (8b + 200) \cdot \delta^{-2^a} + (a-1)2^b, t = O(kaq), m = kq + b2^b, \theta = 1/2$
solves the $LPN_{k,\varepsilon}$ problem.

Proof. For any b-bit vector $\mathbf{x}$, we say that a sample $\mathbf{a}_i, \nu$ is $\mathbf{x}$-compatible if $\langle \mathbf{a}_i | \mathbf{x} \rangle = \nu$. The q inital queries constitues a $(\mathbf{s}, \delta, 0)$-set of size q. By iterating lemma 2, we obtain an $(\mathbf{s}, \delta^{2^{a-1}}, a)$-set S of size $q - (a-1) \cdot 2^b = N = (8b + 200)\delta^{-2^a}$.

We now try every possibility for the first b bits and choose the one that is compatible with the greatest number of examples. The naive time complexity is 2^{2b}, but using a fast Walsh-Hadamard transform reduces it to $b2^b$.

Using the same analysis as for the BKW algorithm, except that we choose $\alpha = 3/4$, we get that the probability of failure is less than

$$e^{-200/64} + 2^b e^{-(8b+200)9/96} \leq 1/(2a).$$

Repeating this a times allows us to recover all the bits of $\mathbf{s}$ with probability at least $1/2$. □

In this section, we have shown that we can lower the query complexity to $q = (a-1)2^b + (8b+200)\delta^{-2^a}$. Time and memory complexity remains comparatively small.

4 A Heuristic Algorithm: Computing All Sums: LF2

Following [14], instead of picking a vector in each class (cf. proof of 2), we could compute the sum of any couple of class elements. Unfortunately, we lose the independence that is necessary to use Chernoff bounds. However, linear relations between equations are not numerous and our implementation confirms this phenomenon has no visible effect on the success of the algorithm.

This also allows us to overcome a lack of queries: if there are only $2^{b'}$ $(b' > b/2)$ queries available, the first partitioning is made according to the last b_1 bits where $b_1 = 2b' - b - 1.5$. And for all the subsequent phases, we will have 2^b equations. Bounding the number of requests was an easy and effective defence against BKW and LF1, but it does not work against this new version.

For example, we succeed in breaking a LPN problem with $k = 66, \varepsilon = 1/4$ with 10000 queries (10 authentications) with 1 GB memory in 30 seconds.

5 Implementation

We want to do the partitioning, using only small additional memory and (almost) linear time. First, we divide our memory in $2^{b/2}$ packs of size $3.2^{b/2}$. We begin at the first equation in the first pack. Its last $b/2$ bits give the address of the pack where we send it. Here, it takes the place of an other equation, which is sent to the pack corresponding to its last $b/2$ bits, and so on. It could happen that a pack is full. In this case, the equation is lost. But few equations are lost in the process, and very few if the packs are a little bigger.

We now use an array of size $2^{b/2}$, each case being able to contain 10 equations. We put the equations of the first pack in this array, according to the values of

bits $2^{b/2}+1$ to 2^b(in reverse order). We could afford ten equations that have the same value (the average being 3).

Then we compute the xor between the first and the others for LF1, or the xor of all couples of equations in the same case for LF2, and put our new equations back to the pack.

We make an implementation in C, and make it run on a Pentium 4, with a CPU frequency of 3 GHz, and a little less than 1 GB of memory.

For $\varepsilon = 1/4$, using 1 GB of memory, our implementation breaks a LPN problem with $k = 99$ (we split the equations in four parts of sizes 24,24,27,24) with LF2, instead of a theoretic $k = 96$ with LF1. But we were able to break only a $k = 92$ with our implementation of LF1 because we need additional memory for pointers to equations. In both cases, the computing time was around 30 seconds.

5.1 Accurate Complexity

The factor $8b + 200$ in the complexity is a rough upper bound. It can often be replaced by 25. On the other hand, reading and writing in a large memory (1 GB for example) could take tens of cycle per 32-bit int. If one uses a hard drive's memory, it will be a lot worse, even if one programs very carefully to make almost only sequential access to the drive.

5.2 Performances of Our Algorithm

First, we give a comparison between BKW and LF1.

For $\varepsilon = 1/4$, we have the following results:

The value given is the maximum value for k you could hope to break with the given memory. The needed time is roughly the time for sorting the memory a times.

Memory available	BKW	LF1
1 GB	39	96
2^{52} bytes	104	225
2^{80} bytes	180	426

The following tables contains a more exhaustive study of LF1. It should be read in the following way: It takes 2^{46} bytes of memory to solve a LPN problem with $k = 256$ and $\varepsilon = 1/8$.

$\varepsilon \backslash k$	64	128	256	512	768	1024
0.01	13	19	33	56	74	98
0.05	16	25	40	67	90	118
0.125	18	29	46	77	113	131
0.25	24	34	55	89	131	150
0.4	28	45	66	106	157	174
0.49	33	55	88	130	192	208

We suggest to take a safety margin, in order to be able to resist to small improvements like LF2. We have explored a variety of other improvements, but none of them gave substantial results.

We recommend to use $k = 512$ to achieve 80 bits security for $\varepsilon = 1/4$. Choosing the value of ε depends upon a compromise between the key size and the computing time for an authentication. Using tables 2 and the above one should help. The couple $k = 768, \varepsilon = 0.05$ with $r = 249$ seems quite good.

6 Conclusion

In this article, we give a better algorithm to solve the LPN problem, thus breaking the HB$^+$ protocol with suggested size parameters. However, with a moderate increase of the key length, our attack becomes infeasible. Our algorithm gives a more precise idea of the complexity of the LPN problem.

On the other side, remark 1 allows to decrease the length of one of the secret parameters.

So, summing everything, we have shown that to achieve 80 bits security, HB$^+$ should be used with $|\mathbf{s}_1| = 80$ and $|\mathbf{s}_2| = 512$ instead of $|\mathbf{s}_1| = |\mathbf{s}_2| = 224$. The overall complexity of this protocol remains almost unchanged.

Proving algorithm LF2 is in our opinion quite difficult although feasible.

Acknowledgment. We would like to thank Louis Granboulan for various discussions and suggestions about this work.

References

1. E. R. Berlekamp, R. J. McEliece, V. Tilborg. On the Inherent Intractability of Certain Coding Problem. *IEEE Transactions on Information Theory 24*, 1978, pp. 384-386.
2. A. Blum, M. Furst, M. Kearns, and R. J. Lipton. Cryptographic Primitives Based on Hard Learning Problems. *Crypto '93*, pp. 278-291, LNCS 773, Springer-Verlag, 1994.
3. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant Learning, the Parity Problem, and the Statistical Query Problem *Journal of the ACM 50,4*, July 2003, pp. 506-519.
4. J. Bringer, H. Chabanne, and E. Dottax. HB++: A Lightweight Authentication Protocol Secure againt Some Attacks. IEEE International Conference on Pervasive Services, Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, SecPerU, 2006 Available at `http://eprint.iacr.org/2005/440`.
5. H. Gilbert, M. Robshaw, and H. Sibert. An Active Attack Against HB+ - A Provably Secure Lightweight Authentication Protocol. Available at `http://eprint.iacr.org/2005/237`.
6. O. Goldreich and L. Levin. A Hard Predicate for all one-way functions, *STOC '89*, pp. 25-32, ACM 1998.
7. J. Hastad. Some Optimal Inapproximability Results, *STOC '97*, pp. 1-10, ACM 1997.

8. N. Hopper and M. Blum. Secure Human Identification Protocols *ASIACRYPT '01*, pp. 52-66, LNCS 2248, Springer-Verlag, 2001.
9. A. Juels and S. Weis. Authenticating Pervasive Devices with Human Protocols *Crypto 2005*, pp. 293-308, LNCS 3621, Springer-Verlag, 2005. Updated version available at: `http://www.rsasecurity.com/rsalabs/staff/bios/ajuels/publications/pdfs/lpn.pdf`.
10. J. Katz and J. Sun Shin. Parallel and Concurrent Security of the HB and HB+ Protocols. *Eurocrypt '06*, pp., LNCS 4004, Springer-Verlag, 2006.
11. M. Kearns. Efficient Noise-Tolerant Learning from Statistical Queries. *J. ACM*, 45(6): 983-1006, 1998.
12. M. Mitzenmacher and E. Upfal. Probability and computing, Cambridge University Press, 2005
13. O. Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography *STOC 2005*, pp. 84-93, ACM 2005.
14. D. Wagner. A Generalized Birthday Problem *Crypto 2002*, pp. 288-303, LNCS 2442, Springer-Verlag, 2002.

A Chernoff's Bounds

We need the following Chernoff's bounds that have been proved in [12].

Theorem 3. *Let* $X_1, \ldots, X_n$ *be* n *independent Bernoulli trials such that* $\Pr(X_i) = p$. *Let* $X = \sum_{i=1}^{n} X_i$ *and* $\mu = \mathbf{E}[X] = n \cdot p$. *Then, the following Chernoff bounds hold:*

1. *for* $0 < \Delta \leq 1$, $\Pr(X \geq (1+\Delta)\mu) \leq e^{-\mu\Delta^2/3}$
2. *for* $0 < \Delta < 1$, $\Pr(X \leq (1-\Delta)\mu) \leq e^{-\mu\Delta^2/2}$