A Variation on Knellwolf and Meier's Attack on the Knapsack Generator

Florette Martinez
florette.martinez@u-picardie.fr

Abstract

Pseudo-random generators are deterministic algorithms that take in input a random secret seed and output a flow of random-looking numbers. The Knapsack generator, presented by Rueppel and Massey in 1985 is one of the many attempt at designing a pseudo-random generator that is cryptographically secure. It is based on the subset-sum problem, a variant of the Knapsack optimization problem, which is considered computationally hard.

In 2011 Simon Knellwolf et Willi Meier found a way to go around this hard problem and exhibited a weakness of this generator. In addition to be able to distinguish the outputs from the uniform distribution, they designed an algorithm that retrieves a large portion of the secret. We present here an alternate version of the attack, with similar costs, that works on a slightly larger range of parameters and retrieves a larger portion of the secret.

1 Introduction

Pseudo-random generators are deterministic algorithms that take in input a random secret seed and output a flow of random-looking numbers. They allow applications to have access to large amount of randomness for a very low computational cost. Cryptosystems are the kind of applications that require a lot of randomness, if only to generate the secrets needed and to remain secure theses applications need good randomness, computationally indistinguishable from the uniform distribution. The Knapsack Generator, presented by Rueppel and Massey in 1985 [3], is one of the many attempt at designing a pseudo-random generator that is cryptographically secure. It is based on the subset-sum problem, a variant of the Knapsack optimization problem, which is considered computationally hard. One should note that the security of the Knapsack Generator does not reduce to the subset sum problem.

At FSE 2011 Simon Knellwolf et Willi Meier [1] found a way to go around this hard problem and exhibited a weakness that is not related to the subset sum problem. In addition to be able to distinguish the outputs from the uniform distribution, they designed an algorithm that retrieves a large portion of the secret. Their attack relies on lattice techniques.

We present here an alternate version of the attack, with similar costs, that works on a larger range of parameters and retrieves a larger portion of the secret. Our attack equally relies on lattice techniques but we manage to reduce our attack to a lattice problem in a more meaningful way.

2 Preliminaries

2.1 Norms and matrix

The natural norm in lattice is the ℓ_2 norm, denoted by $\|.\|_2$. For practical reason, we will also use the ℓ_∞ norm, denoted by $\|.\|_\infty$. For vectors $x \in \mathbb{R}^n$, the standard inequality $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$ holds.

In this paper the norm used for matrices will be the induced ℓ_p norm (right or left depending on the context). We define the right induced ℓ_p norm of a matrix $M \in \mathcal{M}_{n\times m}(\mathbb{R})$ as:

$$\|M\|_p = \max_{x \in \mathbb{R}^m \setminus 0} \frac{\|Mx\|_p}{\|x\|_p}$$

The following inequality derives from the definition of the induced norm:

$$\forall x \in \mathbb{R}^m, \|Mx\|_p \leq \|M\|_p \|x\|_p$$

2.2 Basis of lattices and fundamental domains

Definition 1 (Lattice). *Let $\mathcal{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be a family of linear independent vector of $\mathbb{R}^m$. The lattice $\mathcal{L}(\mathcal{B})$ is the additive linear subgroup generated by the integer combinations of the vectors in $\mathcal{B}$:*

$$\mathcal{L} = \{\alpha_1 \mathbf{b}_1 + \cdots + \alpha_n \mathbf{b}_n | \alpha_i \in \mathbb{Z}\}.$$

The family $\mathcal{B}$ is a basis of $\mathcal{L}$ and n is the dimension, or rank, of the lattice. The basis in not uniquely determined but the dimension is.

From now we will only consider *full rank* lattices, in other words lattices that have the same dimension as the ambient space.

If we consider the matrix M which lines are the vectors $\mathbf{b}_i$, the lattice can be rewritten as

$$\mathcal{L} = \{\boldsymbol{\alpha} M | \boldsymbol{\alpha} \in \mathbb{Z}^n\}.$$

To ensure that two basis generate the same lattice, there needs to exist a transition matrix between the two, invertible in $\mathbb{Z}$ and thus of determinant ± 1.

Definition 2 (Determinant). *Let $\mathcal{L}(\mathcal{B})$ be a lattice generated by a basis $\mathcal{B}$. We define the determinant of lattice $det(\mathcal{L})$ as the absolute value of the determinant of the basis $\mathcal{B}$.*

This determinant is unique and does not depend on the basis used to make the computation.

We can consider a more geometric approach of lattices with the fundamental domain.

Definition 3 (Fundamental domain). *The region enclosed by the basis vectors of a lattice in the euclidean space is a fundamental domain, denoted by $\Lambda_{\mathcal{B}}(\mathcal{L})$.*

The fundamental domain is tied to the basis and two different basis will led to to different fundamental domains. But, because theses two basis have the same determinant (up to the sign), the two fundamental domains will have the same volume $Vol(\mathcal{L})$(that is exactly the determinant of the lattice.)

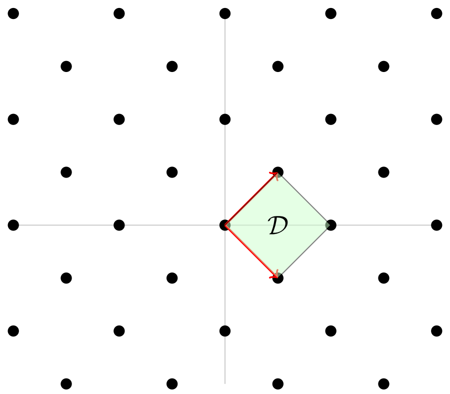

Figure 1: The fundamental domain defined by the basis (1,1),(1,-1)

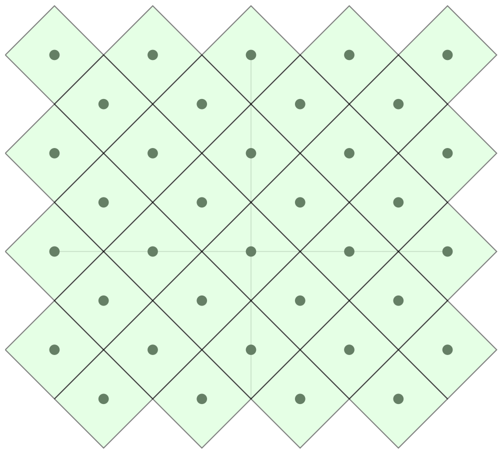

Figure 2: The space paved with fundamental domains

Gaussian Heuristic We can pave the ambient space with fundamental domains centred on the points of the lattice.

Because of that , we can approximate the number of lattice points in a convex shape C by computing $\frac{vol(C)}{det(\mathcal{L})}$. This approximation technique is called the *Gaussian heuristic*.

2.3 First minimum and hard problems

Definition 4 (First minimum)**.** *The first minimum of a lattice $\mathcal{L}$, denoted by $\lambda_1(\mathcal{L})$, is the shortest length of a non-zero vector in the lattice.*

$$\lambda_1(\mathcal{L}) = \min_{x \in \mathcal{L} \setminus 0} \|x\|_p$$

Finding this first minimum (or a shortest vector in the lattice) is a hard problem called the *Shortest Vector Problem* (SVP). In the case of the infinite norm the problem is NP-hard [4].

We can bound the first minimum using Minkowski's theorem.

Proposition 1 (Minkowski's bound on the shortest vector)**.** *Let $\mathcal{L}$ be a lattice. Then there is a $x \in \mathcal{L}\backslash 0$ with $\|x\|_\infty \leq det(\mathcal{L})^{1/n}$.*

Another classical problem in lattice-based cryptography is the following.

Definition 5 (The Closest Vector Problem)**.** *Given a lattice $\mathcal{L}$ and a target $\mathbf{t} \in \mathbb{R}^n$ usually not in the lattice,* the Closest Vector Problem (CVP) *consists in finding $\mathbf{v}$ the closest vector to $\mathbf{t}$ in $\mathcal{L}$. In other words $\mathbf{v} \in \mathcal{L}$ must satisfies :*

$$\|\mathbf{v} - \mathbf{t}\|_p = \min_{\mathbf{x}\in\mathcal{L}} \|\mathbf{x} - \mathbf{t}\|_p.$$

The design of the attack requires to solve several instances of this problem. It might be counter intuitive to design an efficient attack that requires to solve a hard problem, but the only instances of the CVP that we will encounter will be of small dimension and solvable in a reasonable time frame.

3 The Knapsack Generator and the original attack

3.1 Definition and weakness of the Knapsack Generator

Definition 6 (Subset-Sum Problem)**.** *We consider a multiset $\boldsymbol{\omega} = (\omega_0, \ldots \omega_{n-1})$ of weights and a target v, the subset-sum problem consist in finding a subset of $\boldsymbol{\omega}$ that sum to v. In other words we search for a binary vector $\mathbf{u} = (u_0, \ldots, u_{n-1})$ such that*

$$\langle \boldsymbol{\omega}, \mathbf{u} \rangle := \sum_{j=0}^{n-1} \omega_j u_j = v.$$

The problem remains hard if we consider several instances over the same multiset or the modular version of the problem.

The Knapsack Generator was presented in 1985 by Rueppel and Massey in [3]. The public parameters are the length of the multiset $n \in \mathbb{N}$, the number of truncated bits $\ell \in \mathbb{N}$ and an irreducible polynomial $P \in \mathbb{F}_2[X_0, \ldots, X_{n-1}]$. The secrets of this generator are the seed $\mathbf{u} = (u_0, \ldots, u_{n-1}) \in \{0, 1\}^n$ and the multiset $\boldsymbol{\omega} = (\omega_0, \ldots \omega_{n-1}) \in \{0, \ldots, 2^n - 1\}^n$. We present the functioning of the knapsack generator in figure 3.1.

For any $i \in \mathbb{N}$,

$$\begin{array}{ll} u_{n+i} & = P(u_i, \ldots, u_{i+n-1}) \\ \mathbf{U}_i & = (u_i, \ldots, u_{i+n-1}) \\ v_i & = \langle \boldsymbol{\omega}, \mathbf{U}_i \rangle \bmod 2^n \\ & = \sum_{j=0}^{n-1} \omega_j u_{i+j} \bmod 2^n \end{array}$$

The output s_i is given by the leading $n - \ell$ bits of v_i and we denote by δ_i the truncated value:

$$s_i = v_i // 2^\ell \text{ and } v_i = 2^\ell s_i + \delta_i$$

$$\begin{tikzcd}
\mathbf{u} \arrow[d] & & & \\
\boxed{LFSR} \arrow[d] & P \arrow[l] & \boldsymbol{\omega} \arrow[d] & \\
\mathbf{U}_0, \mathbf{U}_1, \ldots \arrow[rr] & & \boxed{\langle \boldsymbol{\omega}, . \rangle \bmod 2^n} \arrow[r] & v_0, v_1, v_2, \ldots \arrow[d] \\
& & & \boxed{//2^\ell} \arrow[d] \\
& & & s_0, s_1, s_2, \ldots
\end{tikzcd}$$

Figure 3: Design of the Knapsack Generator

At FSE 2011, S. Knellwolf and W. Meier found the main weakness of this generator and presented a first attack [1].

Weakness: The secret of the generator is made of the seed $\mathbf{u}$ –n bits– and the vector of weights $\boldsymbol{\omega}$ – n^2 bits –. It is *unbalanced*: if we decide to work with a secret key of 1024 bits it means the seed $\mathbf{u}$ will only be made of 32 bits. We can construct an attack based on an exhaustive search on $\mathbf{u}$.

If we consider $\mathbf{u}$ to be known, it means we can compute the $\mathbf{U}_i$. In this situation, the new problem we want to solve is the following.

Problem A: Given

- m binary vectors $\mathbf{U}_i$ outputted from a LFSR
- m corresponding outputs s_i

we want to retrieve an approximation of the vector $\boldsymbol{\omega}$ such that, for $0 \leq i < m$

$$(\langle \mathbf{U}_i, \boldsymbol{\omega} \rangle \bmod 2^n)//2^\ell = s_i$$

Remark 1. *The generator is linear on $\boldsymbol{\omega}$, which means that if $\boldsymbol{\omega}'$ is close to $\boldsymbol{\omega}$, then the outputs generated with $\mathbf{u}, \boldsymbol{\omega}'$ will be close to the one generated by $\mathbf{u}, \boldsymbol{\omega}$. The knowledge of $\mathbf{u}$ and $\boldsymbol{\omega}'$ is enough to predict the higher bits of the outputs of the generator run on $\mathbf{u}, \boldsymbol{\omega}$. Thus enough to distinguish between the possible $\mathbf{u}$.*

In subsection 3.2 we will briefly presented at FSE 2011 designed to solve the problem A and in section 4 we will present an improvement on the algorithm that gives solutions closer to $\boldsymbol{\omega}$.

3.2 Solving Problem A by FSE 2011 method

We consider the matrix $U \in \mathcal{M}_{n\times m}(\mathbb{Z})$ which lines are $m > n$ consecutive binary vectors $\mathbf{U}_i$ outputted from a LFSR, $\mathbf{v} = (v_0, \ldots, v_{m-1})$ the vector of the m corresponding internal states, as well as $\mathbf{s}$ the vector of the outputs and $\boldsymbol{\delta}$ the vector of truncated bits. These values satisfy

$$\begin{aligned} \boldsymbol{\omega} \times U &\equiv \mathbf{v} \bmod 2^n \\ &\equiv 2^\ell \mathbf{s} + \boldsymbol{\delta} \bmod 2^n. \end{aligned} \tag{1}$$

As the $\mathbf{U}_i$ are derived from a LFSR, the matrix U is full rank over 2^n (see [5]) and we can construct a matrix $T \in \mathcal{M}_{m\times n}(\mathbb{Z})$ such that

$$UT = I_n \bmod 2^n.$$

If we apply this matrix on equation (1) we obtain

$$\begin{aligned} \boldsymbol{\omega} &\equiv vT \bmod 2^n \\ &\equiv 2^\ell \mathbf{s}T + \boldsymbol{\delta}T \bmod 2^n. \end{aligned} \tag{2}$$

The only unknown left on the right-hand side of the equation is $\boldsymbol{\delta}T$ where $\boldsymbol{\delta}$ is small as its coordinates are bounded by 2^ℓ. If we can control the size of T we can obtain the higher bits of $\boldsymbol{\omega}$ as the higher bits of $2^\ell \mathbf{s}T$.

Reducing the size of T: We consider the lattice $\mathcal{L}_1$ defined by $\mathcal{L}_1 = \{\, x \in \mathbb{Z}^m \mid Ux \equiv 0 \bmod 2^n\}$ and construct the matrix $\hat{T}$ column by column.

For the i-th column we consider a vector T_i such that $UTi \equiv \mathbf{e}_i \bmod 2^n$ where $\mathbf{e}_i$ is the vector with a one on the i-th position and zeros elsewhere. We solve an (approximate) instance of the CVP to find $\mathbf{x}_i$ a close vector to T_i in $\mathcal{L}_1$. Then the vector $\hat{T_i} = T_i - \mathbf{x}_i$ is small and satisfies $U\hat{T_i} \equiv \mathbf{e}_i \bmod 2^n$: it becomes the i-th column of $\hat{T}$.

The value $|\boldsymbol{\omega} - 2^\ell \mathbf{s}\hat{T}|_\infty$ is bounded by $|\boldsymbol{\delta}|_\infty \|\hat{T}\|_\infty$. Bounding heuristically $\|\hat{T}\|_\infty$ appears to be challenging.

4 Solving Problem A with a new method

We reconsider equation (1)

$$\begin{aligned} \boldsymbol{\omega} \times U &\equiv \mathbf{v} \bmod 2^n \\ &\equiv 2^\ell \mathbf{s} + \boldsymbol{\delta} \bmod 2^n. \end{aligned}$$

and recall that U and $\mathbf{s}$ are *known* while $\boldsymbol{\omega}, \mathbf{v}$ and $\boldsymbol{\delta}$ are *unknown*.

4.1 The Lattice $\mathcal{L}_2$

We consider the lattice $\mathcal{L}_2$ defined by

$$\mathcal{L}_2 = \{\ \boldsymbol{\alpha} \times U \bmod 2^n \mid \boldsymbol{\alpha} \in \mathbb{Z}^n\}$$

This lattice is of interest because it is computable and contains $\mathbf{v}$. As $\mathbf{v}$ is close to the known vector $2^\ell \mathbf{s}$, we can compute the closest vector to $2^\ell \mathbf{s}$ in $\mathcal{L}_2$ but we find that it is not $\mathbf{v}$. We call this new vector $\mathbf{v}'$.

If we compute a pre image of $\mathbf{v}'$ by the matrix $U \bmod 2^n$, we obtain a vector $\boldsymbol{\omega}'$ such that

$$\boldsymbol{\omega}' \times U \equiv \mathbf{v}' \bmod 2^n.$$

and we experimentally discover that the vector $\boldsymbol{\omega}'$ is a good approximation of the vector $\boldsymbol{\omega}$.

Before trying to understand why these two vectors are close, we will quickly give a bit more information on the structure of $\mathcal{L}_2$.

The lattice $\mathcal{L}_2$ is generated by the rows of the following matrix

$$\begin{array}{c|c} & \overset{m}{\longleftrightarrow} \\ n \updownarrow & \begin{pmatrix} & U & \\ \hline 2^n & & \\ & \ddots & \\ & & 2^n \end{pmatrix} \\ m \updownarrow & \end{array}$$

But the rows of this matrix do not form a basis of $\mathcal{L}_2$ because they form a family that is not free.

We split the matrix U in two parts

$$U = \begin{pmatrix} \overset{\overset{n}{\longleftrightarrow}}{U_1} & | & \overset{\overset{m-n}{\longleftrightarrow}}{U_2} \end{pmatrix} \updownarrow n$$

It is shown in [5] that n consecutive row vectors $\mathbf{U}_i$ are always linearly independent over the integers modulo 2^n thus U_1 is invertible in $\mathbb{Z}/2^n\mathbb{Z}$. We call U_1^{-1} the inverse matrix of U_1 over $\mathbb{Z}/2^n\mathbb{Z}$ and we can compute a basis of $\mathcal{L}_2$ as the rows of the following matrix:

$$\begin{pmatrix} I_n & U_1^{-1}U_2 \\ 0 & 2^n I_{m-n} \end{pmatrix}$$

We obtain for free that the determinant of $\mathcal{L}_2$ is $(2^n)^{m-n}$.

4.2 Short vectors in $\mathcal{L}_2$ have short preimages

We only consider the case where $k < n$.

Let $\mathbf{y}$ be a short vector in $\mathcal{L}_2$ such that $\mathbf{y} \in A_k$ where :

$$A_k\colon = \{\mathbf{y} \in \mathcal{L}_2 \mid \|\mathbf{y}\|_\infty < 2^k\}.$$

We want to prove that there exist a short preimage of $\mathbf{y}$ in B_k where

$$B_k\colon = \{\mathbf{x} \in \mathbb{Z}^n \mid \|\mathbf{x}\|_\infty < \frac{2^k}{\|U\|_\infty}\}.$$

We need to consider the set C_k given by $C_k\colon = \{\mathbf{x} \times U | \mathbf{x} \in B_k\}$. As U is of full rank, the sets B_k and C_k have the same cardinality. And by definition of the induced norm, $C_k \subset A_k$.

To show that $C_k = A_k$, we only have to prove that $|B_k| > |A_k|$ (where $|.|$ denotes the cardinality of a set).

Counting the exact number of points in set B_k is easy, it is all the integers vectors which coordinates are strictly bounded by $\frac{2^k}{\|U\|_\infty}$

$$|B_k| = (2 \times \lceil \frac{2^k}{\|U\|_\infty} \rceil - 1)^n$$

The set A_k is given by $A_k = \mathcal{L}_2 \bigcup B_\infty(2^k)$ where $B_\infty(r)$ denotes the volume containing all points of $\mathbb{R}^m$ of infinite norm smaller than r. Using the Gaussian heuristic (see in subsection 2.2), we can compute an approximation of the number of points in A_k as $\frac{vol(B_\infty(2^k))}{det(\mathcal{L}_2)}$.

We obtain that

$$|A_k| \simeq \frac{(2 \times 2^k - 1)^m}{(2^n)^{m-n}}$$

Theorem 1. *Under the hypothesis that the Gaussian heuristic is a good approximation method, for all* $\mathbf{y}$ *in* $\mathcal{L}_2 \bigcup B_\infty(2^k)$ *with* k *satisfying*

$$\frac{(2 \times 2^k - 1)^m}{(2^n)^{m-n}} < (2 \times \lceil \frac{2^k}{\|U\|_\infty} \rceil - 1)^n$$

there exist $\mathbf{x} \in \mathbb{Z}^n \bigcap \mathcal{B}_\infty\left(\frac{2^k}{\|U\|_\infty}\right)$ *such that* $\mathbf{x} \times U = \mathbf{y}$

4.3 The vector $\boldsymbol{\omega} - \boldsymbol{\omega}'$ is short

We go back to the vectors $\mathbf{v}$ and $\mathbf{v}'$ where $\mathbf{v}$ denotes the internal states of the generator and $\mathbf{v}'$ the closest vector to $2^\ell \mathbf{s}$ in $\mathcal{L}_2$.

By definition of $\mathbf{s}$, we know that $\|\mathbf{v} - 2^\ell \mathbf{s}\|_\infty < 2^\ell$. As $\mathbf{v}'$ is the closest vector to $2^\ell \mathbf{s}$ in $\mathcal{L}_2$, we have $\|\mathbf{v}' - 2^\ell \mathbf{s}\|_\infty < \|\mathbf{v} - 2^\ell \mathbf{s}\|_\infty$ thus $\|\mathbf{v}' - 2^\ell \mathbf{s}\|_\infty < 2^\ell$. By the triangular inequality we obtain that $\|\mathbf{v} - \mathbf{v}'\|_\infty < 2^{\ell+1}$. As $\mathcal{L}_2$ is stable by addition, $\mathbf{v} - \mathbf{v}'$ is in the set $A_{\ell+1}$.

By theorem 1, as long as ℓ satisfies

$$\frac{(2 \times 2^{\ell+1} - 1)^m}{(2^n)^{m-n}} < (2 \times \lceil \frac{2^{\ell+1}}{\|U\|_\infty} \rceil - 1)^n,$$

there exist $\mathbf{x}$ in $B_{\ell+1}$ such that $\mathbf{x}U = \mathbf{v} - \mathbf{v}'$.

As U is invertible modulo 2^n, $\mathbf{x} \equiv \boldsymbol{\omega} - \boldsymbol{\omega}' \bmod 2^n$. The coordinates of $\boldsymbol{\omega}$ and $\boldsymbol{\omega}'$ being bounded by 2^n, it means that

$$\|\boldsymbol{\omega} - \boldsymbol{\omega}'\|_\infty < \frac{2^{\ell+1}}{\|U\|_\infty}.$$

5 The Algorithms

We first present the generator, with an option to use pre-existing secrets

Algorithm 1 The Knapsack Generator

```
function PRNG($m, \mathbf{u}, \boldsymbol{\omega}$)
    if no $\mathbf{u}$ given then
        $\mathbf{u} \leftarrow$ random $\mathbf{u} \in \{0,1\}^n$
    if no $\boldsymbol{\omega}$ given then
        $\boldsymbol{\omega} \leftarrow$ random $\boldsymbol{\omega} \in \{0,\dots,2^n-1\}^n$
    $\mathbf{s} \leftarrow [0,\dots,0] \in \mathbb{Z}^m$
    for all $i \in \{0,\dots,m-1\}$ do
        $U_i \leftarrow (u_i, \dots u_{i+n-1})$
        $v_i \leftarrow < U_i, \boldsymbol{\omega} > \bmod 2^n$
        $s_i \leftarrow v_i // 2^\ell$
        $\mathbf{s}[i] \leftarrow s_i$
    return $\mathbf{s}$
```

We have presented at the beginning of section 4 an algorithm that given the $n \times m$ matrix U and the outputs vector $\mathbf{s}$, returns an approximation of the vector $\boldsymbol{\omega}$.

Algorithm 2 Solving Problem A

```
function ProblemASolver($U, \mathbf{s}$)
    $U_1^{-1} \leftarrow Invert(U_1, \mathbb{Z}/2^n\mathbb{Z})$                ▷ (1)
    $M \leftarrow \begin{pmatrix} I_n & U_1^{-1}U_2 \\ 0 & 2^n I_{m-n} \end{pmatrix}$
    $\mathbf{v}' \leftarrow CVPSolver(M, 2^\ell \mathbf{s})$                ▷ (2)
    $\boldsymbol{\omega}' \leftarrow \mathbf{v}'[0{:}n] \times U_1^{-1}$                ▷ (3)
    return $\boldsymbol{\omega}'$
```

(1) The matrix U_1 is the $n \times n$ left-component of the matrix U and U_1^{-1} is its inverse modulo 2^n. A lot of arithmetic solvers have a built-in polynomial functions to compute these matrices.

(2) We will discuss CVP Solvers in subsection 5.2

(3) As $\boldsymbol{\omega}'$ must satisfies $\boldsymbol{\omega}' \times U \equiv \mathbf{v}' \bmod 2^n$ it implies that $\boldsymbol{\omega}' \times U_1 \equiv \mathbf{v}'[0{:}n] \bmod 2^n$ where $\mathbf{v}'[0{:}n]$ denotes the first n coordinates of $\mathbf{v}'$.

5.1 Distinguishing the correct u

Once we have obtained pairs $\mathbf{u}', \boldsymbol{\omega}'$, we need a way to distinguish between them.

If $\mathbf{u}' = \mathbf{u}$ then $\boldsymbol{\omega}'$ is close to $\boldsymbol{\omega}$ and the outputs of PRNG($\mathbf{u}, \boldsymbol{\omega}'$) will be close to the outputs of PRNG($\mathbf{u}, \boldsymbol{\omega}$).

Now, if $\mathbf{u}' \neq \mathbf{u}$, then the m first outputs of PRNG($\mathbf{u}', \boldsymbol{\omega}'$) will be close to the m first outputs of PRNG($\mathbf{u}, \boldsymbol{\omega}$), by construction. As there is no obvious statistical attack against this generator, it seems reasonable to assume that the following outputs appear random. The probability for the ten following computed outputs to have the same 4 first bits as the original outputs would be $1/2^{40}$, which is greater than size of the loop (in the case $n = 32$).

This is why we consider the following algorithm CheckU

Algorithm 3 Distinguishing the $\mathbf{u}$

```
1: function CheckU($\mathbf{s}, \mathbf{s}'$)
2:     $m \leftarrow length(\mathbf{s})$
3:     for all $i \in \{0, \ldots, m-1\}$ do
4:         $\delta \leftarrow min(\mathbf{s}[i] - \mathbf{s}'[i] \bmod 2^n, \mathbf{s}'[i] - \mathbf{s}[i] \bmod 2^n)$
5:         if $\delta > 2^{n-4}$ then
6:             return False
7:     return True
```

We now present the global algorithm to retrieve $\mathbf{u}$ and an approximation of $\boldsymbol{\omega}$. The algorithm take as input a vector $\hat{\mathbf{s}}$ containing $m+10$ consecutive outputs of the generator. The m first coordinates form the vector $\mathbf{s}$ and the 10 last will be used in the final verification.

Algorithm 4 Retrieve Approximate seed

```
1: function ApproximateSeedRetriever($\hat{\mathbf{s}}$)
2:     for all $\mathbf{u} \in \{0,1\}^n$ do
3:         for all $i \in \{1, \ldots, m-1\}$ do
4:             $u_{n+i} \leftarrow P(u_i, \ldots, u_{i+n-1})$          ▷ Where $P$ is the public retroactive polynomial
5:         $U \leftarrow \begin{pmatrix} u_0 & \ldots & u_{m-1} \\ u_1 & \ldots & u_m \\ & \ldots & \\ u_{n-1} & \ldots & u_{n+m-2} \end{pmatrix}$
6:         $\boldsymbol{\omega}' \leftarrow ProblemASolver(U, \mathbf{s})$
7:         $\mathbf{s}' \leftarrow PRNG(\mathbf{u}, \boldsymbol{\omega}', m+10)$
8:         if CheckU($\hat{\mathbf{s}}, \mathbf{s}'$) is True then
9:             return $\mathbf{u}, \boldsymbol{\omega}'$
```

5.2 About CVP solvers

All the CVP solvers we will be considering have a preprocessing phase where the basis of the lattice is reduced. The less expensive algorithm to do that is LLL [2] and it has a polynomial time complexity. We can also consider the BKZ algorithm whose time complexity and precision depend on a parameter β. If $\beta = 2$, then BKZ acts exactly as LLL and if β is the dimension of the lattice then BKZ will have an exponential time complexity.

Exact CVP Solvers There exists exact CVP solvers that have an exponential time complexity. In our cases, as the dimension of the lattices we are considering are small, we can afford to use theses solvers, there not that expensive comparatively to polynomial approximate CVP solvers.

Babai rounding algorithm The Babai rounding is an extremely simple and polynomial algorithm that retrieve an element of a lattice that is somehow close to the target. If we consider the invertible matrix M and are searching for integer vector x such that xM is close to t, then we can compute $x = \lfloor tM^{-1} \rceil$ where $\lfloor y \rceil$ denote the closest integer vector to y. This algorithm is fast but does not give a very good approximation of the closest vector. The shorter the basis is, the better is the approximation.

Babai nearest plane algorithm The Babai nearest plane algorithm is also a polynomial algorithm that find an approximation of the closest vector to t in a lattice. It is a bit more expensive than the rounding algorithm but is also more precise.

6 Experimental Results

We firstly compute the estimated and experimental minimal m needed to solve problem A and thus being able to distinguish u. For these calculations we use an approximate CVP solver : the Babai nearest plane algorithm.

ℓ	5	10	15	20	25
estimated m	38	39	42	46	59
experimental m	34	34	34	35	39

Figure 4: Minimal m need to recover u for $n = 32$

We notice that our estimations are not tight and overestimate the number of outputs needed.

As the problemASolever algorithm distinguishes between the u by computing an approximation of the vector $\boldsymbol{\omega}$, we are interested in how close this approximation is. We present in the figure 6 the average number of correct bit per weight retrieved (we recall that a weight is coded on n bits).

ℓ	5		10		15		20		25	
m	34	40	34	40	34	40	35	40	39	40
correct bits (over 32)	27	28	22	23	5	18	4	13	6	8

Figure 5: Quality of $\boldsymbol{\omega'}$ for $n = 32$

To distinguish the between the **u**s, we need to run ProblemASolver approximately 2^n times. Once **u** is known, we can run ProblemASolver one last time with a larger m to obtain a better approximation of $\boldsymbol{\omega}$.

We now compare these results with the ones of the original algorithm presented at FSE in 2011.

ℓ	5	10	15	20	25
experimental m	34	35	36	41	> 45

Figure 6: Minimal m need to recover u for $n = 32$ for FSE 2011 algorithm

ℓ	5		10		15		20
m	34	40	35	40	36	40	41
correct bits (over 32)	10	22	10	17	8	12	6

Figure 7: Quality of $\boldsymbol{\omega'}$ for $n = 32$ for FSE 2011 algorithm

References

[1] Simon Knellwolf and Willi Meier. Cryptanalysis of the knapsack generator. In *Fast Software Encryption: 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers 18*, pages 188–198. Springer, 2011.

[2] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[3] RA Rueppel and JL Massey. Knapsack as nonlinear function, ieee intern. symp. of inform. theory, 46. IEEE Press, NY, 1985.

[4] Peter van Emde Boas. Another np-complete problem and the complexity of computing short vectors in a lattice. *Tecnical Report, Department of Mathmatics, University of Amsterdam*, 1981.

[5] Joachim von zur Gathen and Igor E Shparlinski. Predicting subset sum pseudorandom generators. In *Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers 11*, pages 241–251. Springer, 2005.