Documentation of the repository of the paper "Reduction from sparse LPN to LPN, Dual Attack 3.0"

April 21, 2024

Contents

1 Overview of the repository

References to Proposition, Figure or Model point to the eprint version of the article uploaded on December 4th:

`https://eprint.iacr.org/archive/2023/1852/1701452846.pdf`

Summary of each folder

- "Verify_Poisson_Model": A program to show that the poisson Model 1 is valid, it reproduces a figure close to Figure 2. It contains in particular parts of doubleRLPN implemented in C++. Documented in Section 2.
- "Lattice_Prediction": A program to show that we can predict the distribution of the score function of dual attacks in lattices. It essentially reproduces Figure 3 and Figure 4. Documented in Section 3.
- "Complexity_Claim": A program to verify the complexity claims relative to doubleRLP. It contains in particular a dataset with the optimized asymptotic parameters of doubleRLPN to decode at the relative Gilbert-Varshamov distance. Documented in Section 4.

Dependencies

For "Verify_Poisson_Model"

- gcc/g++, available at `https://gcc.gnu.org`. Tested with version 13.1 but an older version with support for C++20 should suffice.

For "Verify_Poisson_Model" and "Complexity_Claim"

- python3, available at `https://www.python.org/downloads/`. Tested with version 3.11.3. Modules needed:
  - Python 3 standard Library
  - NumPy (Tested with version 1.24.3)
  - Scipy (Tested with version 1.10.1)
  - Matplotlib (Tested with version 3.7.1)

For "Lattice_Prediction"

- Jupyter notebook, available at `https://jupyter.org/`. Tested with version 6.5.4.
- SageMath, available at `https://www.sagemath.org/`. Tested with version 10.0.
- unzip.

Everything was tested on a 64 bit Arch-Linux distribution.

2 Verification of the Poisson Model

In folder

```
Verify_Poisson_Model/
```

The goal here is to verify the Poisson Model which is used to bound the expected number of false candidates in Proposition 5, namely the quantity

$$\mathbb{E}\left(\left|\{\mathbf{x} \in \mathbb{F}_2^{k_{\text{aux}}} \setminus \{\mathbf{e}_{\mathscr{P}}\mathbf{G}_{\text{aux}}\} \;:\; f_{\widehat{\mathbf{y},\widetilde{\mathscr{H}},\mathbf{G}_{\text{aux}}}}(\mathbf{x}) \geq T\}\right|\right).$$

The goal is to show that the expected number of false candidates is the same experimentally and by supposing that the Poisson Model true.

Remark: This section does not exactly reproduce Figure 2 of the article. The latter was generated in the case where the set $\mathscr{H}$ of LPN samples is a random subset of $\widetilde{\mathscr{H}}$ of size N. While here we focus on the framework of Proposition 5, that is when $\mathscr{H} = \widetilde{\mathscr{H}}$, which is much simpler and shows in the same manner that the Poisson Model is valid.

Overview of the folder

- "doubleRLPN": contains parts of doubleRLPN implemented in C++. Allow to compute the expected number of false candidates in doubleRLPN. Documented in Section 2.1.
- "Poisson_Model": computes the expected number of false candidates under the Poisson Model. Documented in Section 2.2.
- "Plot": Gather in a plot the number of false candidates given by doubleRLPN and the Poisson Model.

2.1 Number of false candidates in doubleRLPN

In folder

```
Verify_Poisson_Model/doubleRLPN/
```

What it does

Gives an empirical value for the expected number of false candidates in each iteration of doubleRLPN for different values of threshold T. More precisely: given the parameters of the algorithm $w, t_{\text{aux}}, k_{\text{aux}}, s, k, n, t$ and N_{iter} it runs a number N_{iter} of times the following procedure:

- Take $\mathcal{C}$ and $\mathcal{C}_{\text{aux}}$ uniformly at random in $[n, k]$ and $[s, k_{\text{aux}}]$ respectively by choosing two generator matrices $\mathbf{G}$ and $\mathbf{G}_{\text{aux}}$ uniformly at random among matrices of $\mathbb{F}_2^{k \times n}$ of rank k and matrices of $\mathbb{F}_2^{k_{\text{aux}} \times s}$ of rank k_{aux}. Compute $\mathbf{y} = \mathbf{c} + \mathbf{e}$ where $\mathbf{c}$ and $\mathbf{e}$ are taken uniformly at random in $\mathcal{C}$ and $\{\mathbf{x} \in \mathbb{F}_2^n : |\mathbf{x}| = t\}$ respectively. Choose a set $\mathscr{P} \subset [\![1, n]\!]$ of size s uniformly at random among sets such that $\mathcal{C}_{\mathscr{P}}$ is of dimension s. Define $\mathscr{N}$ as $[\![1, n]\!] \setminus \mathscr{P}$.
- Compute the set of false candidates

  $$\{\mathbf{x} \in \mathbb{F}_2^{k_{\text{aux}}} \setminus \{\mathbf{e}_{\mathscr{P}} \mathbf{G}_{\text{aux}}\} : \widehat{f_{\mathbf{y}, \widetilde{\mathscr{H}}, \mathbf{G}_{\text{aux}}}}(\mathbf{x}) \geq T\}$$

  where for $\mathbf{x} \in \mathbb{F}_2^{k_{\text{aux}}}$,

  $$\widehat{f_{\mathbf{y}, \widetilde{\mathscr{H}}, \mathbf{G}_{\text{aux}}}}(\mathbf{x}) = \sum_{(\mathbf{h}, \mathbf{m}_{\text{aux}}) \in \widetilde{\mathscr{H}}} (-1)^{\langle \mathbf{y}, \mathbf{h} \rangle - \langle \mathbf{x}, \mathbf{m}_{\text{aux}} \rangle}$$

  and

  $$\widetilde{\mathscr{H}} = \{(\mathbf{h}, \mathbf{m}_{\text{aux}}) \in \mathcal{C}^{\perp} \times \mathcal{C}_{\text{aux}} : |\mathbf{h}_{\mathscr{N}}| = w \text{ and } |\mathbf{h}_{\mathscr{P}} + \mathbf{m}_{\text{aux}} \mathbf{G}_{\text{aux}}| = t_{\text{aux}}\}.$$

It outputs a file containing, for different values of T, the experimental average (computed over the N_{iter} iterations) number of false candidates.

How to run

– python3 doubleRLPN.py w t_{aux} k_{aux} s k n t N_{iter}

Example :

– python3 doubleRLPN.py 5 2 20 28 30 60 8 100

N_{iter} is advised to be more than 1000 if possible to get the most accurate estimation as possible.

Typical output

An output file in

data/doubleRLPN_w_t_{aux}_k_{aux}_s_k_n_N_{iter}.csv

of the format

T_1 ,y_{T_1}
T_2 ,y_{T_2}
...

where y_{T_i} is the average number of false candidates for the threshold T_i.

2.2 Number of false candidates under the Poisson Model

In folder

Verify_Poisson_Model/Poisson_Model

What it does

Gives an estimate of the expected number of false candidates under the Poisson Model. More precisely, similarly to Lemma 5 we can show that the expected number of false candidates can be rewritten as

$$\mathbb{E}_{\mathcal{C},\mathcal{C}_{\text{aux}}}\left(\left|\{\mathbf{x} \in \mathbb{F}_2^{k_{\text{aux}}} \setminus \{\mathbf{e}_{\mathscr{P}}\mathbf{G}_{\text{aux}}\} \;:\; \widehat{f_{\mathbf{y},\widetilde{\mathscr{H}},\mathbf{G}_{\text{aux}}}}(\mathbf{x}) \geq T\}\right|\right) = \left(2^{k_{\text{aux}}}-1\right)\mathbb{P}_{\mathcal{C},\mathcal{C}_{\text{aux}},\mathbf{x}}\left(\widehat{f_{\mathbf{y},\widetilde{\mathscr{H}},\mathbf{G}_{\text{aux}}}}(\mathbf{x}) \geq T\right)$$

where $\mathcal{C}$ and $\mathcal{C}_{\text{aux}}$ uniformly at random in $[n,k]$ and $[s,k_{\text{aux}}]$ respectively and $\mathbf{x}$ is taken uniformly at random in $\mathbb{F}_2^{k_{\text{aux}}} \setminus \{\mathbf{e}_{\mathscr{P}}\mathbf{G}_{\text{aux}}\}$. Using Lemma 1 and Proposition 4 we have that

$$\widehat{f_{\mathbf{y},\widetilde{\mathscr{H}},\mathbf{G}_{\text{aux}}}} = \frac{1}{2^{k-k_{\text{aux}}}}\sum_{i=0}^{n-s}\sum_{j=0}^{s} N_{i,j} K_w^{(n-s)}(i)\, K_{t_{\text{aux}}}^{(s)}(j)\,.$$

Then, under the Poisson model (replacing $N_{i,j}$ by a compound Poisson variable) we have that

$$\mathbb{E}\left(\left|\left\{\mathbf{x} \in \mathbb{F}_2^{k_{\text{aux}}} \setminus \{\mathbf{e}_{\mathscr{P}}\mathbf{G}_{\text{aux}}\} \,:\, f_{\widehat{\mathbf{y},\widetilde{\mathscr{H}},\mathbf{G}_{\text{aux}}}}(\mathbf{x}) \geq T\right\}\right|\right) = \left(2^{k_{\text{aux}}} - 1\right)\mathbb{P}\left(Z \geq T\right) \tag{1}$$

where

$$Z = \frac{1}{2^{k-k_{\text{aux}}}} \sum_{i=0}^{n-s} \sum_{j=0}^{s} \widetilde{N_{i,j}} K_w^{(n-s)}(i)\, K_{t_{\text{aux}}}^{(s)}(j)$$

and

$$\widetilde{N_{i,j}} \sim \text{Poisson}\left(\widetilde{N_j}\frac{\binom{n-s}{i}}{2^{n-k}}\right) \text{ and } \widetilde{N_j} \sim \text{Poisson}\left(\frac{\binom{s}{j}}{2^{k_{\text{aux}}}}\right)$$

and where the variables are independent.

Given the parameters of the algorithm $w, t_{\text{aux}}, k_{\text{aux}}, s, k, n, t$ and N_{iter}, this script estimates Equation (1) by a monte-carlo method: it draws $N_{\text{iter}}\ 2^{k_{\text{aux}}}$ variables Z to heuristically estimate $\mathbb{P}(Z \geq T)$.

How to run

– python3 PoissonModel.py w t_{aux} k_{aux} s k n t N_{iter}

Example :

```
– python3 PoissonModel.py 5 2 20 28 30 60 8 100
```

N_{iter} is advised to be more than 1000 if possible to get the most accurate estimation as possible. This part is usually the longest, consider parallelization.

Typical output

An output file in

data/PoissonModel_w_t_{aux}_k_{aux}_s_k_n_N_{iter}.csv

of the format

T_1 ,y_{T_1}
T_2 ,y_{T_2}
...

where y_{T_i} is the average number of false candidates for the threshold T_i under the Poisson Model.

2.3 Plot

In folder

```
Verify_Poisson_Model/plot/
```

What it does

If the datasets

$\texttt{data/PoissonModel\_}w\texttt{\_}t_{\text{aux}}\texttt{\_}k_{\text{aux}}\texttt{\_}s\texttt{\_}k\texttt{\_}n\texttt{\_}N_{\text{iter}}\texttt{.csv}$
$\texttt{data/doubleRLPN\_}w\texttt{\_}t_{\text{aux}}\texttt{\_}k_{\text{aux}}\texttt{\_}s\texttt{\_}k\texttt{\_}n\texttt{\_}N_{\text{iter}}\texttt{.csv}$

do not exist, run the two previous programs to create them, then plot the expected number of false candidates given by these datasets.

How to run

- python3 plot.py w t_{aux} k_{aux} s k n t N_{iter}

Example :

- python3 plot.py 5 2 20 28 30 60 8 100

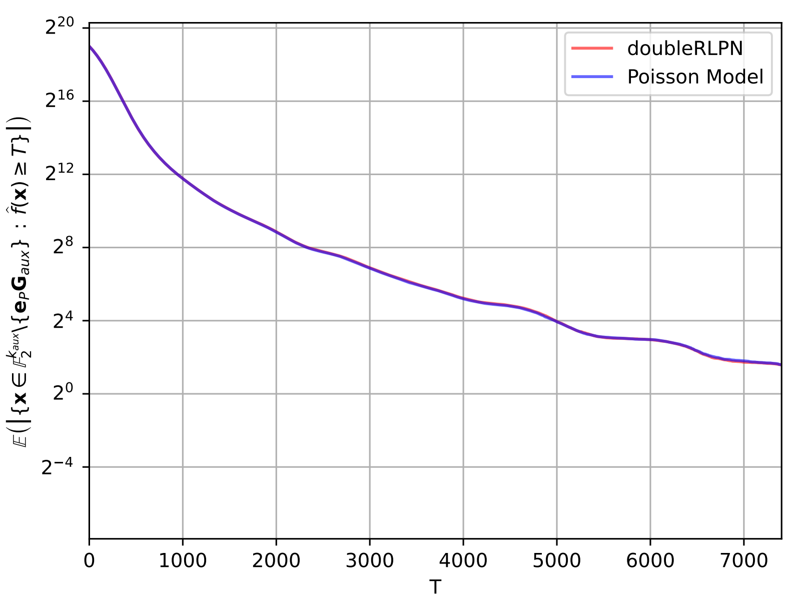
Typical output

An image in

$\texttt{plot\_}w\texttt{\_}t_{\text{aux}}\texttt{\_}k_{\text{aux}}\texttt{\_}s\texttt{\_}k\texttt{\_}n\texttt{\_}N_{\text{iter}}\texttt{.pdf}$

Example with

plot_5_2_20_28_30_60_8_100.pdf



The limit on the T axis is set to T such that the number of false candidates is equal to $\frac{300}{N_{\text{iter}}}$, this prevents the two curve from diverging from each other due to lack of data. Consider increasing N_{iter} to get information for larger T's.

3 Prediction of lattice score function

In folder

```
Lattice_Prediction/
```

Overview of the folder

- `prediction_lattices.ipynb`
  - Reproduces Figure 3 and 4 for different parameters as described in Section 8 of the article.
- `Data_DP23/`
  - is taken from `https://github.com/ludopulles/DoesDualSieveWork/tree/main/data`
- `out_nX_fftY_enumZ.txt`
  - File containing information about the lattice and short dual vectors returned by the sieve. This file was created by showing the variables "Bprime" and "dual_db" of `https://github.com/ludopulles/DoesDualSieveWork/tree/main/code/unif_score.py` with input $n = \mathbf{X}$, fft$= \mathbf{Y}$, enum$=$ $\mathbf{Z}$.

How to run

First, unzip the following compressed dataset:

```
– unzip out_n90_fft22_enum26.zip
```

then, run the notebook:

```
– jupyter notebook prediction_lattices.ipynb
```

4 Verification of complexity claims

In folder

```
Complexity_Claim/
```

The files are meant to verify the complexity claims relative to doubleRLPN.

Overview of the folder

- doubleRLPN_BJMM12.csv

  - Contains, for different code rates R, the optimized relative parameters and the associated complexity of the doubleRLPN decoder to decode at the relative Gilbert-Varshamov distance when using BJMM12 technique to compute low-weight parity-checks. These parameters are used in Proposition 9 to compute the asymptotic complexity of the algorithm. The file contains, for different rates R the values of $\sigma, R_{\text{aux}}, \nu, \omega, \tau$ along with $\lambda_1, \lambda_2, \pi_1, \pi_2$, the later 4 parameters are used in Proposition 11 to compute the complexity of computing the parity-checks using BJMM12 technique. All the parameters (even $\lambda_1, ...$) are written relatively to n. τ_{aux} is implicitly set to be equal to $\sigma h_2^{-1}\left(1 - \frac{R_{\text{aux}}}{\sigma}\right)$ and N_{aux} is implicitly set to be equal to 1. The parameters relative to the two subroutine DUMER-DECODER and SOLVE-SUBPROBLEM will be computed on the fly in the following file.

- complexity_doubleRLPN_BJMM12.py

  - Using the relative parameters contained in the parameter file, this script re-computes, using the formula in Proposition 9, the time complexity exponent ($\alpha_{\text{doubleRLPN}}$) of the doubleRLPN decoder. This script also assert that the parameters meet the constraints of Proposition 9 and Proposition 11 (executions fails if one constraint is not verified).

How to run

```
python complexity_doubleRLPN_BJMM12.py
```

Typical output

A list of complexity exponent

```
Rate: 0.01000;  Complexity: 0.00539
Rate: 0.02000;  Complexity: 0.01009
...
```