Attack against Hawk report

Eirik D. Skjerve

March 7, 2025

1 Notation and background

1.1 Notation

$\mathbf{v}$ denotes a column vector
$\mathbf{B}$ denotes a matrix

1.2 Hawk

A Hawk secret key denoted $\mathbf{B}$ is a $2n \times 2n$ matrix consisting of n "shifts" of polynomials f, g, F and G which satisfy the NTRU-equation $fG - Fg = 1 \mod \mathbb{Z}[X]/(X^n + 1)$.
Let $\mathsf{vec}(f)$ denote the column vector of coefficients of f, such that

$$\mathsf{vec}(f) = \begin{bmatrix} f[0] \\ f[1] \\ \cdots \\ f[n-1] \end{bmatrix}$$

Explicitly, the secret key is then

$$\mathbf{B} = \begin{bmatrix} \mathsf{vec}(f) & \mathsf{vec}(fX) & \mathsf{vec}(fX^2) & \cdots & \mathsf{vec}(fX^{n-1}) & \mathsf{vec}(F) & \mathsf{vec}(FX) & \mathsf{vec}(FX^2) & \cdots & \mathsf{vec}(FX^{n-1}) \\ \mathsf{vec}(g) & \mathsf{vec}(gX) & \mathsf{vec}(gX^2) & \cdots & \mathsf{vec}(gX^{n-1}) & \mathsf{vec}(G) & \mathsf{vec}(GX) & \mathsf{vec}(GX^2) & \cdots & \mathsf{vec}(GX^{n-1}) \end{bmatrix}$$

A Hawk public key denoted $\mathbf{Q}$ is the matrix $\mathbf{Q} = \mathbf{B}^T\mathbf{B}$.

A Hawk signature $\mathbf{s}$ is $\mathbf{s} = \frac{1}{2}(\mathbf{h} - \mathbf{w})$ where $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$ where $\mathbf{s}$ is sampled according to the discrete Gaussian Distribution. In the experiments we run a version of Hawk signature generation that directly returns $\mathbf{w}$ instead of $\mathbf{s} = \frac{1}{2}(\mathbf{h} - \mathbf{w})$, as attacker/receiver can easily recompute $\mathbf{w}$ from $\mathbf{s}$ given message.

2 Attack

2.1 Overview of method

Consider the Discrete Gaussian Distribution as described in [1]. We use our implementation of Hawk to sample many points from the practical distribution based on tables. Let $\widehat{\mathcal{D}}$ denote the practical discrete Gaussian distribution from sampled points. Let $\hat{\mu}$, $\hat{\sigma}^2$ be the expectation and variance of $\widehat{\mathcal{D}}$. Assume we sample t points from $\widehat{\mathcal{D}}$ as $X = \{x_1, x_2, ..., x_t\}$. We estimate $\hat{\mu}$ and $\hat{\sigma}^2$ simply as $\hat{\mu} = \frac{1}{t}\sum_{i=1}^{t} x_i$ and $\hat{\sigma}^2 = \frac{1}{t}\sum_{i=1}^{t}(x_i - \hat{\mu})^2$. For simplicity, we can also assume $\hat{\mu} = 0$ as claimed in [1]. To simplify later computations we also normalize our samples by computing $Z = \{z_1, z_2, ..., z_t\} = \{\frac{x_1}{\hat{\sigma}}, \frac{x_2}{\hat{\sigma}}, ..., \frac{x_t}{\hat{\sigma}}\}$ such that $\mathbb{V}[z_i] = 1$. Now, denote by $\mu_4 = \mathbb{E}[z_i^4] = \frac{1}{t}\sum_{i=1}^{n} z_i^4$. Assume observed signatures on the form $\mathbf{c} = \mathbf{C}\mathbf{z}$ where $\mathbf{C}$ is orthonormal (columns are unit vectors and pairwise orthogonal). By rewriting the terms from the original HPP paper [3] for this new, normalized, distribution $\widehat{\mathcal{D}}$, we have that

$$mom_{4,\mathbf{C}}(\mathbf{u}) = 3\|\mathbf{u}\|^4 + (\mu_4 - 3)\sum_{i=1}^{n}\langle c_i, \mathbf{u}\rangle^4$$

and

$$\nabla mom_{4,\mathbf{C}}(\mathbf{u}) = 12\|\mathbf{u}\|^2\mathbf{u} + 4(\mu_4 - 3)\sum_{i=1}^{n}\big\langle c_i, \mathbf{u}\big\rangle^3 c_i$$

where $\mathbf{u}$ is a vector on the unit sphere of $\mathbb{R}^{2n}$, i.e. $\|\mathbf{u}\| = 1$. This means that if the difference $(\mu_4 - 3)$ is big enough, one might be able to employ the same minimization technique as in the original attack to reveal a column of $\mathbf{C}$ because $\langle c_i, \mathbf{u}\rangle^4 = 1$ if $\mathbf{u} = \pm c_i$ since $\langle c_i, c_i\rangle^4 = 1$. Note that if $(\mu_4 - 3) < 0$ we have the same case as in the original attack, where minimization of the entire term entails maximization of $\sum_{i=1}^{n}\langle c_i, \mathbf{u}\rangle^4$, which gives us a row of $\pm\mathbf{C}$. If $(\mu_4 - 3) > 0$, we need to maximize the entire term $3\|\mathbf{u}\|^4 + \sum_{i=1}^{n}\langle c_i, \mathbf{u}\rangle^4$, which is achieved by doing a gradient *ascent* instead of a gradient *descent*.

2.2 Covariance matrix and hypercube transformation

In the original HPP attack one has to estimate the matrix $\mathbf{G} \approx \mathbf{V}^t\mathbf{V}$. For Hawk, the signatures are on the form $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$. Then we would need to compute $\mathbf{G} = \mathbf{B}^{-1}\mathbf{B}^{-t}$ (the HPP paper uses row vectors while Hawk use columns vectors). However, the public key $\mathbf{Q} = \mathbf{B}^T\mathbf{B}$, enables us to skip this step. Recall that in the original attack one has to take Cholesky decomposition (or an equivalent decomposition) of the inverse of the covariance matrix such that $\mathbf{G}^{-1} = \mathbf{L}\mathbf{L}^T$. For $\mathbf{G} = \mathbf{B}^{-1}\mathbf{B}^{-T}$, the inverse, $\mathbf{G}^{-1} = \mathbf{B}^t\mathbf{B} = \mathbf{Q}$. Therefore, we can simply take the Cholesky decomposition of $\mathbf{Q}$ to get $\mathbf{L}$ such that $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$. By multiplying our samples $\mathbf{w}$ by $\mathbf{L}^T$ on the left, we have transformed our samples to the hidden hypercube as in the original attack.
By taking $\mathbf{C} = \mathbf{L}^T\mathbf{B}^{-1}$, we have that

$$\mathbf{C}^t\mathbf{C} = (\mathbf{L}^T\mathbf{B}^{-1})^T(\mathbf{L}^T\mathbf{B}^{-1}) = \mathbf{B}^{-T}\mathbf{L}\mathbf{L}^T\mathbf{B}^{-1} = \mathbf{B}^{-T}\mathbf{Q}\mathbf{B}^{-1} = \mathbf{B}^{-T}\mathbf{B}^T\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}$$

and

$$\mathbf{C}\mathbf{C}^T = (\mathbf{L}^T\mathbf{B}^{-1})(\mathbf{L}^T\mathbf{B}^{-1})^T = \mathbf{L}^T\mathbf{B}^{-1}\mathbf{B}^{-T}\mathbf{L} = \mathbf{L}^T\mathbf{Q}^{-1}\mathbf{L} = \mathbf{L}^T(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{L} = \mathbf{L}^T\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{L} = \mathbf{I}$$

Thus $\mathbf{C}$ is an orthonormal matrix. Since $\mathbf{w}$ is distributed according to $\widehat{\mathcal{D}}$ over $\mathcal{P}(\mathbf{B^{-1}})$, by taking $\mathbf{c} = \mathbf{L}^T\mathbf{w}$ we have $\mathbf{c} = \mathbf{L}^T\mathbf{B}^{-1}\mathbf{x} = \mathbf{C}\mathbf{x}$, $\mathbf{c}$ is distributed according to $\widehat{\mathcal{D}}$ over $\mathcal{P}(\mathbf{C})$.

We summarize this step of the attack against Hawk in the following algorithm

Algorithm 1 Hawk Hypercube Transformation

Require: Samples $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$ and public key $\mathbf{Q}$
1: Compute $\mathbf{L}$ s.t. $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$ ▷ by Cholesky decomposition
2: Compute $\mathbf{c} = \mathbf{L}^T\mathbf{w}$
3: **return** $\mathbf{c}$ and $\mathbf{L}^{-T}$

2.3 Gradient search overview

Now, given samples $\mathbf{c} \in \mathcal{P}(\mathbf{C})$, we run a gradient search to minimize or maximize the fourth moment of one-dimensional projections onto $\mathbf{u}$, as in the original attack. In addition to multiplying the samples by $\mathbf{L}^t$, we also divide them by the scalar σ, to normalize the samples so each sample in the vector $\mathbf{c}$ has variance 1. This also aligns with our theoretical analysis of $mom_{4,\mathbf{C}}(\mathbf{u})$. Even though the distribution $\widehat{\mathcal{D}}$ is discrete, the shape of the samples $\mathbf{c} \in \mathcal{P}(\mathbf{C})$ will still have a somewhat spherical shape. The hope is that the areas in the directions of the columns of $\pm\mathbf{C}$ will deviate just enough from a perfect spherical shape to give us a clear global minima/maxima in the gradient landscape.

We evaluate the functions $mom_{4,\mathbf{C}}(\mathbf{u})$ as $\mathbb{E}[\langle \mathbf{c}, \mathbf{u} \rangle^4]$ and $\nabla mom_{4,\mathbf{C}}(\mathbf{u})$ as $\mathbb{E}[\nabla \langle \mathbf{c}, \mathbf{u} \rangle^4] = 4\mathbb{E}[\langle \mathbf{c}, \mathbf{u} \rangle^3 \mathbf{c}]$. These can be evaluated by precomputed samples $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_t\}$, or be evaluated continuously by generating one and one signature sample since we have access to the signature generation algorithm. This approach is much slower since one has to do one matrix transformation by $\mathbf{L}^T$ for each sample, but removes the need for much memory.

Below, both gradient descent and gradient ascent is described as algorithms. Note that the only difference is in what direction to move and the condition for termination, on lines 4 and 6 respectively.

Algorithm 2 Gradient descent on $\mathcal{P}(\mathbf{C})$

Require: Samples on the form $\mathbf{c} = \mathbf{C}\mathbf{x}$, descent parameter δ

1: $\mathbf{u} \leftarrow$ random vector on unit sphere in $\mathbb{R}^{2n}$
2: **loop**
3: $\quad \mathbf{g} \leftarrow \nabla mom_{4,\mathbf{C}}(\mathbf{w})$
4: $\quad \mathbf{u}_{new} = \mathbf{u} - \delta \mathbf{g}$
5: $\quad$ normalize $\mathbf{u}_{new}$ as $\frac{\mathbf{u}_{new}}{\|\mathbf{u}_{new}\|}$
6: $\quad$ **if** $mom_{4,\mathbf{C}}(\mathbf{u}_{new}) \geq mom_{4,\mathbf{C}(\mathbf{u})}$ **then**
7: $\quad\quad$ **return u**
8: $\quad$ **else**
9: $\quad\quad \mathbf{u} \leftarrow \mathbf{u}_{new}$
10: $\quad\quad$ go to step 3
11: $\quad$ **end if**
12: **end loop**

Algorithm 3 Gradient ascent on $\mathcal{P}(\mathbf{C})$

Require: Samples on the form $\mathbf{c} = \mathbf{C}\mathbf{x}$, ascent parameter δ

1: $\mathbf{u} \leftarrow$ random vector on unit sphere in $\mathbb{R}^{2n}$
2: **loop**
3: $\quad \mathbf{g} \leftarrow \nabla mom_{4,\mathbf{C}}(\mathbf{w})$
4: $\quad \mathbf{u}_{new} = \mathbf{u} + \delta\mathbf{g}$
5: $\quad$ normalize $\mathbf{u}_{new}$ as $\frac{\mathbf{u}_{new}}{\|\mathbf{u}_{new}\|}$
6: $\quad$ **if** $mom_{4,\mathbf{C}}(\mathbf{u}_{new}) \leq mom_{4,\mathbf{C}(\mathbf{u})}$ **then**
7: $\quad\quad$ **return u**
8: $\quad$ **else**
9: $\quad\quad \mathbf{u} \leftarrow \mathbf{u}_{new}$
10: $\quad\quad$ go to step 3
11: $\quad$ **end if**
12: **end loop**

2.4 Practical method

Below is a basic description of the approach.

Algorithm 4 Proposed basic version of attack

1: Collect signatures $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$
2: Using public key $\mathbf{Q}$, find $\mathbf{L}$ s.t. $\mathbf{Q} = \mathbf{L}\mathbf{L}^t$
3: Transform samples s.t. $\mathbf{c} = \mathbf{L}^t\mathbf{w}$
4: Find columns of $\pm\mathbf{C}$ by doing gradient search over $\mathcal{P}(\mathbf{C})$
5: Multiply columns of $\pm\mathbf{C}$ by $\mathbf{L}^{-t}$ on the left and round the result to get columns in $\pm\mathbf{B}^{-1}$

After having transformed the samples $\mathbf{w} \in \mathcal{P}(\mathbf{B^{-1}})$ to $\mathbf{c} \in \mathcal{P}(\mathbf{C})$ we want to recover columns of $\pm\mathbf{C}$ and transform them back to columns of $\pm\mathbf{B}^{-1}$ by multiplying by $\mathbf{L}^{-1}$ on the left. Due to the special structure of Hawk private keys, finding one column of $\mathbf{B}$ automatically gives n columns. Unfortunately, since revealing a single column of $\mathbf{B}^{-1}$ reveals a "shift" of either the two polynomials G and g *or* F and f, this is not enough to disclose the entire matrix. If samples were on the form $\mathbf{w} = \mathbf{B}\mathbf{x}$, a single column would reveal f and g, and one could simply reconstruct F and G by solving the NTRU-equation as in the key generation step of Hawk. Nevertheless, if one finds two columns of $\mathbf{B}^{-1}$, it is easy to check if they are shifts of each other. If they are not, one has found shifts of all four polynomials in the secret key, and by trying all combinations of shifts, of which there are $4n^2$ (accounting for negative and

positive sign), one can easily verify if a candidate $\mathbf{B}'$ is valid by checking if $\mathbf{B}'^T\mathbf{B}' = \mathbf{Q}$. If so, one is able to forge signatures, and the attack is done.

In experiments we have access to the correct secret key. After each gradient ascent and/or descent returns a possible solution $\mathbf{u}$, we multiply it to the left as $\mathbf{b}' = \mathbf{L}^{-T}\mathbf{u}$ where $\mathbf{b}'$ is a possible column of $\pm\mathbf{B}^{-1}$ as $\mathbf{u}$ is a possible column of $\pm\mathbf{C} = \mathbf{L}^T \pm \mathbf{B}^{-1}$. Since we have the correct secret key, we simply check directly if $\mathbf{b}' \in \pm\mathbf{B}^{-1}$. In a real word attack however, one would, as described above, have to compute candidate $\mathbf{B}'$ and check if $\mathbf{B}'^T\mathbf{B}' = \mathbf{Q}$.

Having access to the correct $\mathbf{B}^{-1}$ we can do measures on how close a proposed solution $\mathbf{b}'$ is to one of the columns of $\mathbf{B}^{-1}$. We can for example take the difference in length of $|\mathbf{b}'|$ and each vector in $|\mathbf{B}^{-1}|$, i.e. $\mathsf{diff}_{\mathsf{min}} = \mathsf{min}\{|||\mathbf{b}'|-|\mathbf{b}_i|| : \mathbf{b}_i \in \pm\mathbf{B}^{-1}\}$. A low value for this (this depends on the Hawk parameter n) would indicate a good guess, whereas a higher value indicates greater difference between the vectors, and thus a bad guess. One can also take the average of the difference of the entries in the vectors, i.e. $\mathsf{diff}_{\mathsf{min}} = \mathsf{min}\{\frac{1}{2n}|\mathbf{b}' - \mathbf{b}_i|:\mathbf{b}_i \in \pm\mathbf{B}^{-1}\}$. Alternatively, one can count entrywise how many entries out of $2n$ that matches between $\mathbf{b}'$ and $\mathbf{b}_i$.

A concern about the approach of attacking Hawk is the gradient search, as described in Algorithm 2 and 3. In the original HPP attack, they used a standard gradient descent with a constant stepsize/descent parameter δ. This worked fine for attack on GGH and NTRU since the global minima (of which there are $2n$) of the gradient landscape was much more clear. In the gradient landscape of Hawk however, there is much more noise, and the global minima is not as clear, as it depends on the discrepancy $(\mu_4 - 3)$, which is quite small. Generally, a constant stepsize δ entails a tradeoff between slow convergence and the risk of "overshooting" the correct global extremum. I have therefore researched alternative methods of gradient search, and found the method of ADAM-optimizer [2]. This method, which is mostly used in machine learning and other optimization processes, dynamically changes the stepsize δ iteratively though the gradient search, and they claim that their method works well for noisy gradient landscapes with high parameters/dimensions.

I chose to implement my gradient ascent and gradient descent algorithms using the ADAM-optimizer method. Generally the method changes δ to be greater in flat areas of the landscape, and δ to be lower in steep areas of the landscape, as it continuously calculates some information "around" the current point in the search. This method is more involved and sophisticated than the standard gradient search, but I thought this was a good idea to implement it, since finding a good δ for the standard gradient search was difficult.

Below is a description of the gradient search using ADAM-optimizer in our setting.

Algorithm 5 gradient descent ADAM

Require: signatures $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$

Require: Stepsize δ, decay rates β_1, β_2

1: $\mathbf{u} \leftarrow$ random vector on the unit sphere of $\mathbb{R}^{2n}$
2: $\mathbf{m}_0 \leftarrow 0$ ▷ Vector of 0's
3: $\mathbf{v}_0 \leftarrow 0$ ▷ Vector of 0's
4: $t \leftarrow 0$ ▷ Timestep
5: **loop**
6: $t \leftarrow t + 1$
7: $\mathbf{g}_t \leftarrow \nabla mom_{4,\mathbf{C}}(\mathbf{w})$
8: $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t$
9: $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t^2$
10: $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$
11: $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$
12: $\mathbf{u}_{new} \leftarrow \mathbf{u} - \delta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t + \epsilon})$ ▷ Flip the $-$ sign for gradient ascent
13: normalize $\mathbf{u}_{new}$ as $\frac{\mathbf{u}_{new}}{\|\mathbf{u}_{new}\|}$
14: **if** $mom_{4,\mathbf{C}}(\mathbf{u}_{new}) \geq mom_{4,\mathbf{C}}(\mathbf{u})$ **then** ▷ Flip the $\geq$ sign for gradient ascent
15: **return u**
16: **else**
17: $\mathbf{u} \leftarrow \mathbf{u}_{new}$
18: go to step 6
19: **end if**
20: **end loop**

The ϵ in step 12 of the algorithm is inserted to not divide by 0. Proposed value for $\epsilon = 10^{-8}$, and proposed good values for δ, β_1 and β_2 are $0.001, 0.9$ and 0.999 respectively. In this method, the stepsize then depends on $\delta, \mathbf{m}_t$ and $\mathbf{v}_t$.

Below is a more detailed version of the attack.

Algorithm 6 Proposed version of attack with measuring

Require: Hawk parameter n, number of samples t, Hawk key pair $\mathbf{B}$, $\mathbf{Q}$

1: Collect t signatures $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$ ▷ Optional - can also generate signatures continuously
2: Using public key $\mathbf{Q}$, compute $\mathbf{L}$ s.t. $\mathbf{Q} = \mathbf{L}\mathbf{L}^t$
3: Transform samples s.t. $\mathbf{c} = \mathbf{L}^t\mathbf{w}$
4: **loop**
5: Candidate $\mathbf{z}' \leftarrow$ gradient search ADAM ▷ Do both ascent and descent
6: Candidate $\mathbf{b}' \leftarrow \lfloor \mathbf{L}^{-T}\mathbf{z}' \rceil$ ▷ Entrywise rounding to nearest integer
7: Check if $\mathbf{b}' \in \mathbf{B}^{-1}$
8: Measure $\mathsf{diff}_{\mathsf{min}}(\mathbf{b}', \mathbf{B}^{-1})$
9: **end loop**

The loop from line 4 can be run several times to get a new random starting point $\mathbf{u}$ for the gradient search. Line 6 rounds the candidate *after* multiplying with $\mathbf{L}^{-T}$ to avoid rounding errors. Line 8 can also give which column of $\mathbf{B}^{-1}$ $\mathbf{b}'$ is closest to, so one can compare the vectors side by side.

2.5 Results

The method has not proven to work, as no correct key has been found in any of the runs. It seems that regardless of number of signatures (above a certain point, e.g. one million), the method cannot give candidate solutions with better comparison than random guessing. Random guessing in this case is assuming one knows what type of distribution the columns of a secret key is. One knows the distribution that f, g follows, but F and G depends on f and g.

For reference, I ran a test using the closeness measure $\mathsf{diff}_{\mathsf{min}} = \mathsf{min}\{\|\,|\mathbf{b}'| - |\mathbf{b}_i|\,\| : \mathbf{b}_i \in \pm\mathbf{B}^{-1}\}$ evaluation by fixing one private key ($\mathbf{B}^{-1}$), and generating random keys $\mathbf{B}'^{-1}$ (which will serve as a random guess), to check if the attack on average gives better guesses than random guessing. The following table shows results of this. This is the result of comparing a key with 100 random keys, and the result of 100 random starting points for the gradient search (both ascent and descent).

Table 1: Closeness measure for Hawk attack

Type	$\mathsf{diff}_{\mathsf{min}}$	$\mathsf{diff}_{\mathsf{max}}$	**Avg** $\mathsf{diff}_{\mathsf{min}}$	**Avg** $\mathsf{diff}_{\mathsf{max}}$
Key comparison degree 32	6.25	15.81	7.74	11.22
Attack on Hawk 32 (1m samples)	7.14	16.24	8.50	12.87
Key comparison degree 64	10.77	26.51	13.96	22.18
Attack on Hawk 64 (1m samples)	13.49	26.00	16.25	22.59
Key comparison degree 128	25.33	47.26	30.00	41.60
Attack on Hawk 128(1m samples)	24.27	46.91	29.61	46.91
Key comparison degree 256	56.33	82.34	61.02	75.02
Attack on Hawk 256 (10m samples)	something	something		

In the following table values for μ_4 from samples are described

Table 2: Values for μ_4

Hawk degree	μ_4
256	value
512	value
1024	value

References

[1] Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, and Wessel van Woerden. Hawk. Technical report, NXP Semiconductors, Centrum Wiskunde & Informatica, Mathematical Institute at Leiden University, NCC Group, PQShield, Institut de Mathématiques de Bordeaux, September 2024.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[3] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures, 2009.