**Squirrels Essay II**

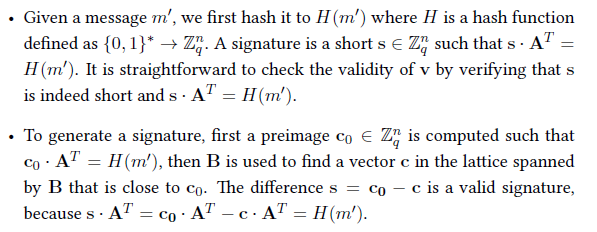
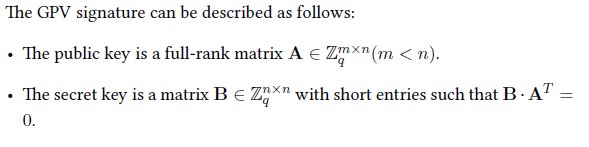
Names: Daymon Wu, Jaraad Hussain, Leonardo Yeung, Al Fahim

In this essay we will explore how certain encryption methods, GPV schemes, show the vulnerabilities in older versions and previous branches of squirrel. We will also introduce co-cycle lattices and show how their role is important to find the right balance between randomness and structure for a secure communication. We will also explain how the Hermite Normal Form is used to quickly check lattice membership, a critical step in verifying digital signatures. The essay will also go one to explain the sparsity in secret vectors as well as the runtime and performance of Squirrel.

Leonardo Yeung

GPV framework

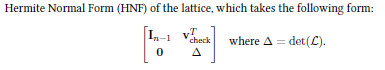
The pioneering GGH signature scheme laid the groundwork by mapping messages onto a lattice. However, it faced significant setbacks, particularly the leakage of the secret basis during verification. The GPV scheme addresses this by randomizing lattice point selection, ensuring signature security through a discrete Gaussian distribution.



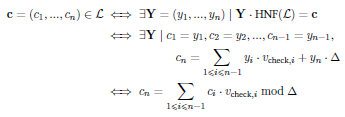
Co-cyclic lattice

Co-cyclic lattices, characterized by a cyclic quotient group Zn/L, offer an approximately 85% density, striking a balance between randomness and structure that is ideal for cryptographic applications. These lattices support the security of Squirrels by contributing to the intractability of the underlying hard problems.

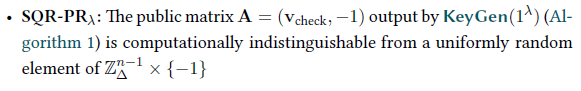
Hermite Normal Form in Cryptography

The Hermite Normal Form is leveraged in Squirrels to expedite the lattice membership checks, critical to verifying digital signatures. It transforms a lattice basis into an upper triangular matrix, simplifying the membership verification to a comparison of vector components.

Check if the point c is in the lattice



Regularity of the keygen output



**Generation of the first vectors:**

The process starts by creating n - 1 vectors. These vectors are produced in such a way that their Gram-Schmidt norms (a way of measuring vector length after making vectors orthogonal to each other) are between two values, g-min and g-max. To ensure the last Gram-Schmidt vector is within desired bounds, the algorithm controls a value δ. This value is calculated based on the product of the norms of the previously generated vectors and the desired determinant, denoted as Δ, of the lattice. The determinant represents the volume of the parallelotope formed by the lattice basis vectors. The algorithm adjusts δ at each step by a factor depending on the sequence position to keep it small in absolute value. For each vector v needed, the algorithm samples a candidate vector in two parts:

* vB: This part is uniformly distributed among the vectors of (the inverse of the basis matrix) with a norm bound between b-low and b-up.
* vB again: This part follows a Gaussian distribution within the basis matrix B with a standard deviation set by a formula involving g-max.

The sampled vector v is then rounded to an integral vector. This rounding can introduce an error in the length of the Gram-Schmidt vector. The vectors are accepted if their Gram-Schmidt norm after rounding is within the range [ g-min, g-max ], which is expected to happen with at least 90% probability. The algorithm sets initial bounds b-low = g-min and b-up = g-max. It also adjusts these bounds at each step based on the drift δ to control the determinant of the lattice. At each step, the algorithm checks the determinant distance and adjusts b-up and b-low to correct any drift.

**Computation of the last secret vector:**

Following the generation of the initial n -1 vectors in our lattice basis, we face the task of determining the final vector, v-last, which plays a pivotal role in the integrity of our cryptographic lattice. The vector v-last is distinctive because it not only completes the lattice but does so while maintaining the overall lattice determinant Δ and conforming to the pre-established Gram-Schmidt norm bounds, g-min and g-max. To achieve this, we delve into the underlying mathematics of matrices. We leverage the concept that the determinant of a matrix B can be expanded into minors, which are determinants of smaller matrices obtained by removing one row and one column from B. The last vector is intricately tied to these minors, as it is constructed to ensure that the lattice determinant remains fixed at Δ. We select a subset of minors that are co-prime—numbers that have a greatest common divisor of one with each other—and employ Euclid's extended algorithm to find specific coefficients c-i that, when combined with these minors, sum up to one. This relationship is then scaled up by the determinant Δ to find our last vector, v-last. For efficiency, we focus on calculating just the last 4 minors, as this proves to be sufficient—the probability of these minors being co-prime is somewhat high (about 42%). This selective computation considerably speeds up the key generation process without compromising security. Once we have v-last, we further reduce the matrix to ensure that the coefficients are small for efficiency. This is done is 3 steps:

* The coefficients are reduced using the ComputeReducedGCD algorithm to ensure their absolute values are lower than a certain threshold. This is important to keep the lattice basis as small and efficient as possible.
* A further reduction is done by constructing a matrix with these coefficients and performing an LLL reduction, a process that helps to find a shorter and nearly orthogonal lattice basis.
* Finally, the Babai Nearest Plane algorithm is applied to the coefficients to further reduce the vector.

**Public Key Derivation:**

This part of the process is about transforming the complete secret basis of a lattice, denoted as B, into a form that can be shared publicly without compromising security. This transformation is done through computing what's called the row Hermite Normal Form (HNF) of B. The HNF is a specific way of representing a matrix that simplifies many types of computations, including those needed for cryptographic purposes. The row HNF of a matrix is a unique matrix that is row-equivalent to the original matrix B, meaning they are related by elementary row operations. The row HNF has certain properties that make it desirable for cryptographic use; for instance, it can verify that the lattice is co-cyclic. A co-cyclic lattice is one whose basis vectors generate a cyclical structure, which is an important property for certain cryptographic schemes. To compute the row HNF efficiently, the Pernet-Stein algorithm is typically used. However, any algorithm that computes the row HNF will arrive at the same result because the row HNF of a matrix is unique. The process of computing the row HNF and verifying the co-cyclic property of the lattice is implemented in a function called ComputePK. While the algorithm explicitly checks for the co-cyclic nature of the lattice, this step is technically not necessary if certain conditions are met. Specifically, if the determinant of the lattice is square-free (a number that is not divisible by the square of any prime) when choosing the public parameters (as detailed in an appendix of the paper), the lattice will automatically be co-cyclic.

On Sparsity - Daymon

To understand the sparsity level of secret vectors we have to know how to attack sparse secrets. In my last essay, I discussed a method for attacking sparse secrets. This is a simple breakdown of the steps: First, randomly guess where the zeros might be. Then, create a new lattice called L' by combining the original lattice L and removing positions not common to both sets. Next, using the BKZ algorithm we organize the lattice basis for easier identification of shorter vectors. Then applying the Gram-Schmidt process to find the norm of the Gram-Schmidt vector from the basis vectors. If the vector's norm is smaller than 4/3 times the norm of the Gram-Schmidt vector, include it in the list. Lastly, use Babai’s nearest plane algorithm to find the closest lattice point within L' for each vector. Check the length of the lifted vector in L' and compare it to gmax (specific to the problem's needs). If the length is shorter than gmax, consider it as a potential lattice vector.

Understanding the sparsity level of secret vectors is important for finding out how many elements in a vector are zero. A sparse secret vector indicates that most of its elements are zero, while a non-sparse vector implies a higher number of non-zero elements. This information holds significance for a lot of reasons.

One key aspect is checking the security of the system. By identifying patterns in the numbers and selecting where zeros occur, potential vulnerabilities can be recognized, contributing to a more secure system.

Also evaluating sparsity can enhance computational efficiency. If a significant portion of the vector consists of zeros, it opens up the opportunity to employ computational shortcuts and optimize the speed of calculations.

Understanding the relevance of each element in the vector also aids in error prevention and correction. Knowledge of which numbers matter allows for more effective correction of mistakes that may arise when using secret codes, strengthening the reliability of the encoded information. The sparsity level influences the selection of appropriate settings for cryptographic parameters. Parameters such as key lengths and security levels are essential for encryption strength, and awareness of vector sparsity helps in making informed decisions to maintain the desired level of security.

Assessing the sparsity level of secret vectors is also essential for enhancing system security, optimizing computational efficiency, preventing errors, and making informed decisions about cryptographic parameters.

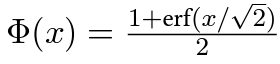
To do this first we apply the Gaussian Mode and picture the zeros in our list as dots on a graph, checking how they're distributed. The Gaussian model is centered at zero, so the values decrease symmetrically as you move away from zero. The width or spread of the distribution is managed by the standard deviation (σ) sigma.



This is the Gaussian model.

In a continuous setting, the expression -gmin/√n signifies that, on average, the size of vectors is represented by gmin. Here, gmin stands for the minimum desired norm for the vectors. The term "continuous setting" means that the mathematical formula is constructed to function with any real numbers, without gaps in between, ensuring a smooth and consistent application across the entire range of values.

Next we have the formula for calculating probability of zeros.



This is also the formula for the standard normal CDF (Cumulative Distribution Function). We calculate the chance that each number on the list is zero with the cumulative distribution formula. Then to calculate the probability of each individual vector we use this formula where vi represents the i-th coefficient or element of a vector.

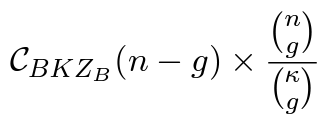


For the rest of our secret vectors, we count how many zeros we expect based on the probability where K represents a constant or variable and N represents the dimension or size of vector space.



There is always a plan to attack these vectors, and we want to understand how hard it is for someone to do it. The main goal is to figure out how resistant our system is to potential attacks.

So using the variables we found in the probability formulas, we plug them into this formula:



* C represents a constant factor.
* B represents the size of vector or lattice structure.
* K represents a constant or variable.
* Z represents a parameter related to lattice structure.
* N represents the dimension or size of vector space.
* G represents minimum distance between lattice points.

This kind of formula is often used in cryptography to represent a measure of complexity..

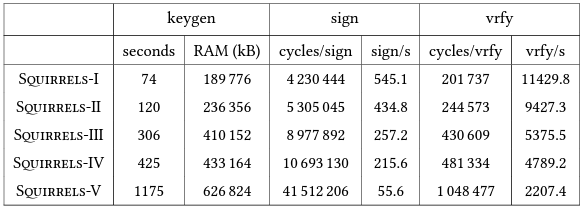
We want to pick the best settings for our system. This means finding the right parameters (like how spread out the zeros are) to make sure our system stays secure and works well.

Performance - Al Fahim

Cryptographic implementations play a crucial role in ensuring the security of communication and data integrity. This report provides a detailed performance evaluation of the Squirrels cryptographic implementation, focusing on key generation, signing, and verification procedures. The assessment was conducted on two distinct hardware platforms: a 2018 Lenovo Y530 laptop featuring an Intel Core i5-8300H processor and a 2022 Lenovo Thinkpad P14s equipped with a Ryzen Pro 75850U processor.

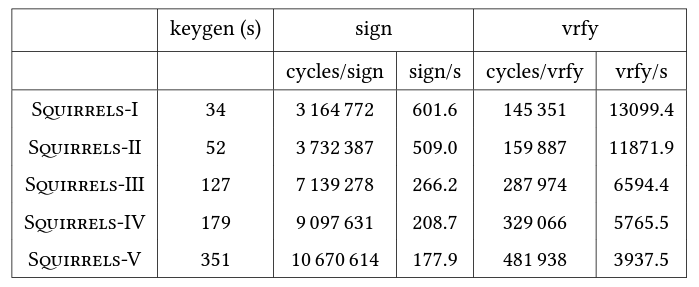
The reference implementation, written exclusively in portable C, supports all five security levels and includes adaptations using a compilation flag. For key generation involving significant integer and matrix computations, dynamically linked libraries such as GMP, Flint, and fplll are utilized.

The NIST x64 Reference Target evaluation employs GMP, Flint, and fplll as dynamically linked libraries. Key generation involves computations using Flint structures, encompassing matrix determinants and HNF computations. The verification procedure ensures that variables fit within specified ranges, maintaining computational efficiency.



The evaluation on the Lenovo Y530 laptop, featuring an Intel Core i5-8300H processor with 8 CPU threads at 2.3GHz, 32GB of physical RAM, and running Manjaro 22.1, provides insights into the Squirrels implementation's behavior.

The results reveal execution times, RAM usage, and cycles per operation for key generation, signing, and verification procedures. Notably, Squirrels-I through Sqirrels-V exhibit varying performance characteristics, offering a comprehensive view of the implementation's behavior on the specified hardware.



The assessment on the Lenovo Thinkpad P14s, equipped with a Ryzen Pro 75850U processor featuring 16 CPU threads at 3GHz (boost disabled) and running Manjaro 22.1, further extends the evaluation scope.

Performance metrics for key generation, signing, and verification operations on the x64 AMD platform illustrate how the Squirrels implementation adapts to different hardware configurations.

The performance evaluation offers valuable insights into the Squirrels cryptographic implementation's efficiency across diverse hardware platforms. The results, encompassing execution times, memory usage, and cycles per operation, serve as a valuable reference for assessing the practical utility of Squirrels in real-world cryptographic scenarios. Understanding the strengths and considerations of the Squirrels implementation is essential for informed decisions on its deployment and integration into secure communication systems.  
Intel platform performs better at better security levels than AMD whilst they are better at mid levels.

To sum it up, we looked at how certain encryption methods, like GPV schemes, act in squirrel. We introduced co-cycle lattices, showing their role in balancing randomness and structure for secure communication. We also explained how the Hermite Normal Form helps quickly check if something belongs to a group, crucial for verifying digital signatures. We talked about sparse secret vectors and touched on how squirrel perform in terms of time and efficiency. In a nutshell(squirrel), we've learned more about what works and what needs improvement in secure communication methods.