Hardware acceleration for ringSIS hash

Karthik Inbasekar,[a]

[a] *Ingonyama*

E-mail: karthik@ingonyama.com

Abstract: Ring SIS is a hash function whose preimage resistance derives its hardness from the SIS (Short Integer Solution problem). This hash cannot be used as a random oracle. In this article we discuss some salient features of this hash and write a hardware acceleration proposal.

1 Introduction to Short Integer Solution problem

Let us consider a problem of solving a system of linear equations given a matrix $A \in \mathbb{Z}_q^{n\times m}, \mathbf{t} \in \mathbb{Z}_q^n$ such that

$$A \cdot \mathbf{e} = \mathbf{t} \mod q \tag{1.1}$$

It is well known that this equation can be solved for $\mathbf{e}$ in polynomial time by Gaussian elimination in general. However, if we frame the question differently i.e given a $A \in \mathbb{Z}_q^{n\times m}$ drawn from a uniform distribution, say if we require the solution to be in a restricted domain $\mathbf{e} \in [-b, b]^m$ such that $|b| < q$, then the problem falls in an NP class of related problems.

- **SIS regime**: The Short Integer Solution (SIS) regime exists when the following conditions are met. Given $\mathbf{t} \in \mathbb{Z}_q^n$ is uniformly random, then b bounded solutions to $\mathbf{e}$ exist if

  $$(2b+1)^m >> q^n \; or \; m = \Omega\left(\frac{n\log q}{\log b}\right) \tag{1.2}$$

  Formally SIS problem is defined as $SIS(n, m, q, b)$: given a uniformly random $A \in \mathbb{Z}_q^{n\times m}$, $\mathbf{t} \in \mathbb{Z}_q^n$ to find an $\mathbf{e}$ with $||\mathbf{e}||_\infty < b$ such that (1.1) is satisfiable. The problem is called homogeneous SIS if in addition $\mathbf{t}$ is restricted to be $\mathbf{0}_n$. In general when m is large, there can be many solutions when $A, \mathbf{t}$ are uniformly random, and the hard part of the problem is to find a solution. The factors that affect the hardness are the bound b and the parameter n. In particular, smaller the bound harder the solution whereas Increasing n increases the number of constraints on $\mathbf{e}$, thus making it harder to find a satisfiable solution.

- **LWE regime**: The LWE (Learning with Errors problem is defined when the parameter regime is

  $$m << \frac{n\log q}{\log b} \tag{1.3}$$

A LWE (Learning with Errors) instance is defined as

$$\mathbf{B} = \langle A, \mathbf{s} \rangle + \mathbf{e} \tag{1.4}$$

where $\mathbf{s} \leftarrow \mathbb{Z}_q^n, A \in \mathbb{Z}_q^{m \times n}$ and the error $\mathbf{e} \in [-b, b]$ with $|b| < q/2$. The SIS and LWE problems are related as follows.

Using the right kernel $A^\perp \in \mathbb{Z}_q^{(m-n)\times m}$ [1] of the matrix A we can define a associated SIS instance $SIS(m-n, m, q, b)$

$$\mathbf{t} \equiv A^\perp \cdot \mathbf{B} = A^\perp \cdot \mathbf{e} \mod q \tag{1.6}$$

whose solution is the LWE error vector $\mathbf{e}$, which satisfies the bound $||\mathbf{e}||_\infty < b$. The parameter regime (1.3) is called a planted regime since, in (1.4) the solution to the SIS problem $SIS(m-n, m, q, b)$ given $\mathbf{t}, A^\perp$ (1.6) is "planted" in the LWE cipher text $\mathbf{B}, A$. Thus it is often common to come across the statement the "SIS problem in the planted regime is the LWE problem".

The hardness of the SIS problem stems from the difficulty of finding short vectors in cyclic Lattices. This property can be used to define a one way function, that satisfies collision resistance (asymptotic) and preimage resistance. These are desirable properties of a hash function [1]. In the following sections, we describe a specific instantiation of a hash function based on the SIS problem.

2 Ring SIS hash

In this section, we describe the instantiation of hash function called as RingSIS defined and used in Vortex [2].

Let $\mathbb{F}_q$ be a prime field and $v_i \in \mathbb{F}_q^n$ is a vector of n field elements, we will also refer to this as the "hash size" or "instance buffer size". Let b be the SIS bound such that $|b| < q$ and let d be the degree such that all polynomials are in the ring $\mathcal{R} = \frac{\mathbb{F}_q[X]}{X^d+1}$ with coefficients in $\mathbb{F}_q$ and d is a power of 2. [2].

The hash instantiation consists of the following sequence of operations

1. Sample a public key of n polynomials $(A_0(X), A_1(X), \ldots, A_{n-1}(X))$ with $A_i(X) \in \mathcal{R}$ and degree d.

2. Limb decompose each field element in the input vector in base b

$$v_i = \sum_{j=0}^{m-1} b^j \cdot l_{i,j} \tag{2.1}$$

such that $||l_{i,j}||_\infty < b$, where m is the number of limbs. Each limb is bounded by $\log_2 b$ bits of data, and thus if we have m slices, $m \sim \mathcal{O}(\log_b q)$

[1] The right kernel of the matrix A is defined as the matrix $A^\perp \in \mathbb{Z}_q^{(m-n)\times m}$ such that

$$A^\perp \cdot A = 0 \mod q \tag{1.5}$$

[2] In the implementation [3] if $d = b$ then the ring $\mathcal{R} = \frac{\mathbb{F}_q[X]}{X^d-1}$ is used.

3. Arrange the limbs into coefficients of polynomials

$$W_i(X) = \sum_{j=0}^{m-1} l_{i,j} \cdot X^j \tag{2.2}$$

4. Compute the Ring product

$$h(X) = \sum_{i=0}^{n-1} A_i(X) \cdot W_i(X) \mod X^d + 1 \tag{2.3}$$

the coefficients of $h(X)$ are the outputs of the hash function

2.1 Optimizations and suggestions for GPU computation

In the above sequence, the main computational bottleneck is step4 (2.3). In the case, $q = 1 \mod 2d$ (eg the scalar field modulus of BN254 satisfies this for all values of d that are powers of 2) we can use negative wrapped cyclic convolution [4, 5] or its variants without increasing the size of the vectors. For the above

- if the domain size is d, and if the primitive root of unity is ω_d, compute $\mu = \sqrt{w_d} = \omega_{2d}$, which is the primitive $2d$th root of unity. The coset domain is defined as

$$H \equiv \mu \cdot \{1, \omega_d, \omega_d^2, \ldots, \omega_d^{d-1}\} \tag{2.4}$$

- Let $A(X)$ be coefficient form and $\tilde{A}(X)$ be eval form representation, The $CosetNTT_H$ on the domain H can be computed using a regular NTT with the following trick

$$\begin{aligned} CosetNTT_H(A(X)) &= NTT_d(A(\mu X)) \\ CosetINTT_H(\tilde{A}(X)) &= (1, \mu^{-1}, \mu^{-2}, \ldots \mu^{-(d-1)}) \circ INTT_d(\tilde{A}(X)) \end{aligned} \tag{2.5}$$

where in the second step the $\circ$ refers to element wise product. i.e represent the coefficients of the result of the INTT as a vector and compute the element wise product.

- Step 4 is then computed in GPU by unrolling the for loop (it can be optimized further) and applying cosets as follows. However for this the PK polynomials need to be precomputed in the Coset domain. The main advantage here, is that given the precomputed polys, the rest of the computation can be integrated with existing ICICLE with a simple cosetNTT extension.

  1. Pre-compute $\tilde{A}_i(X) \equiv CosetNTT_H(A_i(X)) \forall i$
  2. Unroll the for loop and parallel compute

$$\tilde{h}_i(X) \equiv \tilde{A}_i(X) \circ CosetNTT_H(W_i(X)) \ \forall \ i \tag{2.6}$$

  3. Sum the results $\tilde{h}(X) = \sum_{i=0}^{n-1} \tilde{h}_i(X)$

4. compute inverse NTT after the sum once and output the coefficients

$$h(X) = CosetINTT_H(\tilde{h}(X)) \tag{2.7}$$

A simple flow diagram describing the whole sequence is described in fig 1. The negative wrapped convolution means that we never need to compute the extra terms (see for instance [5]) greater than degree d, since the negacyclic wrapping gives the ring reduction for free. This is because because a primitve dth root of unity satisfies the property

$$\omega_d^i \equiv -\omega_d^{i+d/2} \mod q \tag{2.8}$$

For eg, when $i = 0$ we get $\omega_d^{d/2} = \omega_{2d}^d = -1$, i.e it satisfies the relation $X^d + 1 = 0$ which means that the Ring reduction comes free. [3] In Appendix A.6 (cf fig 2) [2] the above method works for all the d , except $d = 7$.

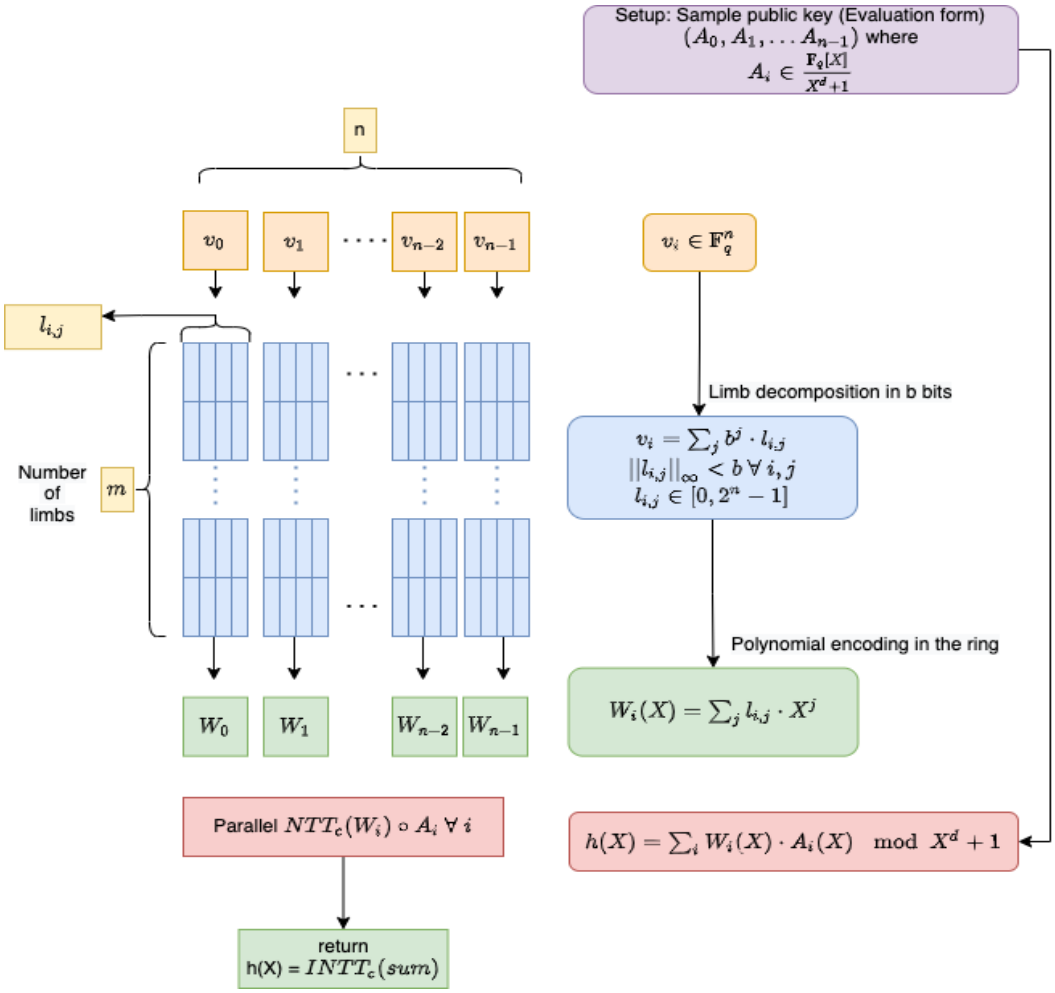

Figure 1. RingSIS function flowchart

[3] Also this means that $X^d + 1$ is reducible in $\mathbb{F}_q$, not sure how it affects security here, since the hardness depends on b, n.

Note: In the code [3] the implementation uses Montgomery reduction. It does not appear from fig 2 that memory is a bottleneck in this case.

$log_2(q)$	$log_2(\beta)$	n	BKZ attack	CPW attack
64	2	32	182.17	144.0
64	4	64	147.31	305.57
64	6	128	166.13	598.14
64	10	256	149.93	1272.31
64	16	512	136.4	2741.67
64	22	1024	160.7	5967.82
254	2	7	157.7	259.03
254	4	16	146.1	270.0
254	6	32	164.73	637.0
254	10	64	148.63	1262.46
254	16	128	135.18	2720.33
254	24	256	133.28	5921.27
254	32	512	164.03	13013.8

Figure 2. Table A.6 parameters from [2], In our discussion $b = \beta, d = n$ in the figure, for our discussion the relevant parameters are for 254 bit q.

Acknowledgments

We stand on the shoulders of giants.

References

[1] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "Swifft: A modest proposal for fft hashing." https://web.eecs.umich.edu/~cpeikert/pubs/swifft.pdf.

[2] A. Belling and A. Soleimanian, "Vortex : Building a lattice-based snark scheme with transparent setup." Cryptology ePrint Archive, Paper 2022/1633, 2022. https://eprint.iacr.org/2022/1633.

[3] GNARK, "Ringsis." https://github.com/Consensys/gnark-crypto/blob/master/ecc/bn254/fr/sis/sis.go.

[4] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography." Cryptology ePrint Archive, Paper 2016/504, 2016. https://eprint.iacr.org/2016/504.

[5] S. Kim, W. Jung, J. Park, and J. H. Ahn, *Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs*, in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, oct, 2020.