Circuit friendly, post-quantum dynamic accumulators from RingSIS with logarithmic prover time

Endre Abraham
Wigner research centre for Physics

July 29, 2021

Abstract

Mainstream hash functions such as SHA or BLAKE while generally efficient in their implementations, are not suitable for zero-knowledge boolean or arithmetic circuits due to their reliance on CPU designs. As a candidate hash function that uses only on trivial arithmetics which can be generalized to zeroknowledge circuits, the Ajtai lattice SIS-hasher has been proposed. In this paper we review Micciancio's R-SIS generalization and argue about it's circuit complexity, then we show how this R-SIS hasher can be used as a universal dynamic hash accumulator that has constant-time update and revocation complexity, and can be run on 16-bit hardware as well as smart contracts.

1 Introduction

Cryptographic accumulators are not a new gadget in literature but after a short uptake from the blockchain and zeroknowledge community recently, they lack intense research. Their most intuitive usecase is as a constant-size and private alternative to Merkle trees where one does not need to leak *logn* elements of the tree to prove membership of a particular element.

Recent focus by the cryptocurrency community on accumulators highlight privacy concerns around UTXO sets and their membership proofs. Recursive Zk-Snarks which show a promising scaling feature are hard to implement on top of Merkle trees, especially with hash functions designed for performance on consumer electronics and no abstract algebraic circuit friendlyness in mind.

The nominal work of Ajtai[1] which is considered as the start of lattice based cryptography research shows a toy hasher which is later generalized by Micciancio and Regev[2]. This "SIS-hasher" achieves collision resistance, and quantum safety using only modular addition and multiplication. Later Elaine Shi et al.[3] constructed a two-input version and an authenticated data structure using it's

accumulation properties. This research focuses on such properties while also making the initial construction more efficient, memory friendly and we select ideal parameters for hardware verification and smart-contract implementations. First we review the starting SIS-problem and the induced collision-resistant hash, make a two-input instance that reduces to the original problem[3], then we generalize this version into a Ring-SIS instance and argue about its security and efficiency. Later we show how this more practical version is suitable for quantum-safe membership and non-membership proofs.

1.1 Hash accumulators

Benaloh and de Mare defined hash accumulators as such:
Let $h(x)$ be a length-regular one-way hash function that accepts two inputs, and quasi-commutative. Formally:

1. $h : X_n \times Y_n \to Z_n$
2. $h(x_n, y_n) \in O(poly(n))$
3. $Pr[h(x, y) = h(x', y') < \epsilon]$ for any $y' \in Y$ and a negligibly small ϵ
4. $\forall x \in X, y_1, y_2 \in Y : h(h(x, y_1), y_2) = h(h(x, y_2), y_1)$

The *accumulator* value for a set $y_1, \ldots, y_m$ is thus $h(h(h(\ldots h(x, y_1), y_2) \ldots y_m)$ and a corresponding *witness* for the inclusion of y_i is $w = h(h(h(\ldots h(x, y_1), y_{i-1}), y_{i+1}) \ldots y_m)$ (the accumulator value of all elements except the subject). Then the verification for the inclusion of y_i is simply $h(w, y_i) \oplus x$ where $\oplus$ is the binary-xor operation. We see that hash accumulators are constant size and anonymous in the meaning that a prover does not have to provide any elements of the original set to produce a valid membership proof.

We call a hash accumulator *dynamic* if the addition/removal into/from the original set happens in constant time. Also, we call a hash accumulator *universal* if it can also produce a succint non-membership proof in addition. Benaloh and de Mare showed in their work how modular exponentiation $h(x, y) = x^n \pmod{()n}$ satisfies the one-way and quasi-commutativity properties and later RSA-accumulators have been proposed. However RSA accumulators have an impractical limitation that they can only accumulate prime numbers (otherwise it's possible to forge membership proofs) which requires a hash-to-prime solution. RSA accumulators belong to the class of "Unknown order group" accumulators. Other constructions use Bilinear maps or Merkle trees which we don't address in this work.

1.2 Lattices and discrete gaussians

A lattice is the set of all possible linear combinations of a finite real basis with integer coefficients:

$$c \in \mathbb{Z}^n, \mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^m, \Lambda = \{\mathbf{Bc} = \sum_{i\in[n]} c_i \mathbf{b}_i\}$$

The dimension of a lattice is m and its rank is n. In case $n = m$ we call the lattice full-rank. For cryptographic applications we mostly use full-rank lattices. The dual of a lattice is $\{x \in \mathbb{R}^n : \forall v \in \Lambda, \langle x, v\rangle \in \mathbb{Z}\}$ and the bidual is itself. The shortest nonzero vector in the lattice is called the minimum distance: $\lambda_1(\Lambda) = min_{0\neq x\in\Lambda}||x||$. A measurable set $\mathcal{R}(\Lambda) \subset \mathbb{R}^m$ satisfying that $\cup_{\lambda\in\Lambda}\mathcal{R}(\Lambda) + \lambda = \mathbb{R}^m$ is called a fundamental region. The fundamental parallelepiped is a fundamental region, namely the generating set $\mathbf{G} \in \mathbb{R}^m$ where $\sum_{i\in[n]} \mathbf{g}_i a_i \in \mathbb{Z}$. All fundamental region's volume is equal to $det(\mathbf{B})$ (on full-rank lattices). We use the notation $\mathcal{B}$ for the closed Euclidean ball of radius 1 around the origin: $\mathcal{B} = \{\mathbf{w} \in \mathbb{R}^n : ||\mathbf{w}|| \leq 1\}$. Besides being (so far) resistant to quantum attacks (not restricting to Shor's and Grover's algorithm) their significance in cryptography is stronger, as they offer better asymptotic complexity (most lattice algorithms only involve linear operations on 16bit integers), and it's the ideal platform group for today's most exotic cryptosystems like FHE, IO or (H)IBE.
The connection between gaussian distributions and maps to the lattice's fundamental parallelepiped allows us to prove security easier than by trying to use uniform distribution (for e.g. key-generation) directly. We recall the properties of discrete gaussian sampling over lattices and other related definitions based on the works of Regev and Micciancio[2]:
For all $\mathbf{c}, \mathbf{x}, s > 0$, let:

$$\rho_{s,\mathbf{c}}(\mathbf{x}) = e^{-\pi||(\mathbf{x}-\mathbf{c})/s||^2}$$

be the Gaussian function with center $\mathbf{c}$ and scale s. Then the continuous gaussian distribution is: $\frac{\rho_{s,\mathbf{c}}(x)}{\mathbf{s}^n}$. This distributi on is generalized for lattices as: $\frac{\rho_{s,\mathbf{c}}(x)}{\rho_{s,\mathbf{c}}(\Lambda)}$. As the scale parameter gets larger, the statistical distance between the continuous and discrete variants of the gaussian narrows and the threshold for s where the distance from $\mathbf{c}$ is (almost) $\frac{s^2 n}{2\pi}$ is called the smoothing parameter:

$$\epsilon > 0 \in \mathbb{R}, \eta_\epsilon(\Lambda) = \rho_{1/s}(\Lambda^* \setminus \{0\}) \leq \epsilon$$

where Λ^* is the lattice's dual. This smoothing parameter was key to the results of Micciancio and Regev[2] as it proved that by using discrete Gaussian distributions beyond this parameter, reducing the results into the fundamental parallelepiped resulted in a statistically close to uniform distribution over $\mathbb{Z}_q^n$. From Peikert's lemma[4] we know a lower bound on η:

$$\eta_\epsilon(\Lambda) \leq \frac{\sqrt{log(2n/(1+\frac{1}{\epsilon}))/\pi}}{\lambda_1^\infty(\lambda^*)}$$

where $\lambda_1^{\infty}(\lambda^*)$ is the minimum distance of the lattice's dual using the infinity norm.
We recap some problems on full-rank and regular lattices which are assumed no to be in BQP as no quantum algorithm currently exists that can solve an instance in polynomial time.

1. Shortest Vector Problem: *With a lattice basis* $\mathbf{B}$ *and* $l \in \mathbb{R}$*, determine whether* $\lambda_1(\mathbf{B}) \leq l$
2. Closest Vector Problem: *With basis* $\mathbf{B}$*,* $t \in \mathbb{R}^n$*,* $d \in \mathbb{R}$*, and security parameter* k *(polinomial in* n*) determine whether* $dist(t, \Lambda(B)) > kd$
3. Covering Radius Problem: *With* $d \in \mathbb{R}^n$ *determine whether* $max_{x \in \mathbb{R}^n}\{dist(x, \Lambda(B)\} \leq d$
4. Shortest Indepentend Vectors Problem: *Search for vectors* $\mathbf{S} \subset \mathbf{B}$ *for basis* $\mathbf{B}$ *that satisfies* $||\mathbf{S}|| < k \cdot min\{r : dim(span(\Lambda \cap r\mathcal{B})) \geq n\}$
5. Shortest Integer Solution: This is a worst-case generalization of (approximate) SVP. *For* $m = poly(n), q \geq \beta\sqrt{n}\omega(\sqrt{logn})$ *find* $\mathbf{x} \in \mathbb{Z}^m$ *such that* $\mathbf{Az} = \mathbf{0} \wedge ||x|| < \beta$
6. (Average-case) RingSIS: For a ring R, $q \geq 2, l \geq 1$ and a public $a_1, a_2, \ldots, a_l \in R_q$, find a nontrivial vector $e_1, e_2, \ldots, e_l \in R_{\{-1,0,1\}}$ such that $a_1e_1 + a_2e_2, \ldots, a_le_l = 0 \in R_q$
7. Learning with errors[5]: *For an arbitrary* $\mathbf{s} \in \mathbb{Z}_q^n$*, the decisional LWE asks to differentiate a "sample"* $(\mathbf{a}, as + e \mod q)$ *from a completely uniform random sample where* $a \xleftarrow{\$} \mathbb{Z}_q^n$ *and* $e \xleftarrow{D_\sigma} \mathbb{Z}_q$. LWE is proven to be as hard as SIVP when $q \geq 2\sqrt{n}/\sigma$ and an instance of $LWE(n, m, q)$ has equivalent security of an instance of $SIS(m, m - n, q)$[6]
8. Ring-Learning with errors: *Let* $\Phi(x)$ *be a cyclotomic polynomial and* $\mathbb{G} = \mathbb{Z}_q[x]/\Phi(x)$ *be a quotient polynomial ring. For an arbitrary* $s(x) \in \mathbb{G}, a(x) \xleftarrow{\$} \mathbb{G}, e(x) \xleftarrow{D_\sigma} \mathbb{G}$*, differentiate the sample* $(a(x), a(x) \cdot s(x) + e(x) \mod q)$ *from a uniformly selected one.*

1.3 Note on Zero-knowledge circuits

Zero-knowledge protocols allow a verifier to ascertain the satisfiability of a computation, without the prover giving out any information about the statement besides the single bit of satisfiability. Arithmetic circuits are an algebraic model of computation necessary for such zero-knowledge protocols, which are usually organized into a Directed Acyclic Graph (DAG) over $\mathbb{Z}_q$, where the vertices serve (usually) as addition or multiplication gates, and 0-degree vertices serving as inputs or constants. Zero-Knowledge protocols prove, or argue about the satisfiability of such algebraic (or in some cases Boolean) circuit in a cryptographically binding (to the NP language) and hiding way (which implies they often use commitment schemes).

2 R-SIS hasher with small parameters

We review the construction of Micciancio[7] for a a collision-resistant RingSIS hasher on top of irreducible polynomial rings, and show ideal parameter selections for efficient accumulation usecases.

The idea pivots around the idea of Cyclic Lattices, which generates the lattice basis matrix from the cyclic rotation of a uniform random column vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)^T \in Z_q^n$:

$$\mathbf{B} = \begin{pmatrix} a_1 & a_n & \cdots & a_3 & a_2 \\ a_2 & a_1 & \cdots & a_4 & a_3 \\ a_3 & a_2 & \cdots & a_5 & a_4 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_n & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_n \\ a_n & a_{n-1} & \cdots & a_2 & a_1 \end{pmatrix} \tag{1}$$

It is easy to see that such structure for the lattice basis is convenient for storing the lattice, but the main takeaway in terms of RingSIS is that the set of all such integer matrices form a commutative ring isomorphic to $\mathbb{Z}_q[x]/x^n - 1$[7].

While the above hasher is already one-way, it's not collision-resistant, due to the trivial reducibility of the quotient polynomial. Thankfully, if we change the quotient to be the irreducible (and easy to store) $x^n + 1$, the only change needed for the generation of the lattice basis is to replace the "cyclic shift matrix" (shifting the columns of the identity matrix) into the negacyclic form:

$$\mathbf{B} = \begin{pmatrix} a_1 & -a_n & \cdots & -a_3 & -a_2 \\ a_2 & a_1 & \cdots & -a_4 & -a_3 \\ a_3 & a_2 & \cdots & -a_5 & -a_4 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & -a_n & -a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & -a_1 & -a_n \\ a_n & a_{n-1} & \cdots & a_2 & a_1 \end{pmatrix} \tag{2}$$

With this negacyclic form, we can efficiently sample an instance of RingSIS as having only a single column vector $\mathbf{a}$:
For input $\mathbf{e} \in R_{\{0,1\}}$, $hash_{ringsis} = a_1e_1 + a_2a_2, \ldots, a_ne_n \pmod q$

To understand why a single polynomial multiplication (which you can fasten up using the Number-Theoretic-Transform) implies a RingSIS instance, check that $rot(\mathbf{a}) \cdot rot(\mathbf{e}) = rot(rot(\mathbf{a} \cdot \mathbf{e}))$ where $rot(\mathbf{x})$ means the multiplication of $\mathbf{x} \in R_q^n$ with the n-th power of the negacyclic matrix as seen above.

2.1 Accumulation with R-SIS

The trivial (and naive) method for accumulation of values using ringsis uses it's highly linear structure:

Let the verifier state be $y = \sum_{i=0}^{n} hash_{ringsis}(x_i)$. The proof π for an element with index k is then:

$$\pi = \sum_{i=0, i \neq k}^{n} hash_{ringsis}(x_i) \tag{3}$$

While this naive construction suffices for some usecases where there is an external form of authentication (not everyone can accumulate values), the lack of a trapdoor results in no cryptographic soundness as anyone can create a proof for arbitrary values. Existing hash accumulators that do involve a trapdoor however, has a prover time linear in the number of elements. To mitigate this, we use the generalized hash tree technique by Shi et. al[3] which we describe below. This way we obtain a sound dynamic accumulator that has $O(1)$ complexity on the verifier state, even for revocation, and proofs can be constructed in $O(logn)$.

To organize our hasher into a generalized hash tree, we first modify it into a two-input version that involves two cyclic generators $\mathbf{l}$ and $\mathbf{r}$:

$$hash_{ringsis}(\mathbf{x_l}, \mathbf{x_r}) = rot(\mathbf{l}) \cdot rot(\mathbf{x_l}) + rot(\mathbf{r}) \cdot rot(\mathbf{x_r}) \tag{4}$$

Unfortunately, the domain and the range of $hash_{ringsis}$ differ, so in this form it's not enough for accumulating subsequent invocations. To address this discrepancy, we use a projection function $f\,[n]^m \to R_q^n$ which simply re-embeds the input vector into R_q by it's binary representation. Let $x \in [n]^m, b = \lceil logq \rceil, m = bk$. Then to $\mathbf{y} = f(\mathbf{x})$ is computed as such:

$$y_i = \sum_{j=0}^{b-1} x_{ib} +_j 2^j \pmod{q} \tag{5}$$

We see that f is linear: $f(\mathbf{a} + \mathbf{b}) = f(\mathbf{a}) + f(\mathbf{b})$ With this knowledge, the function we define for labeling our hash tree:[3]

Let T_C be a full balanced binary tree. For a node w it's label is computed as

$$L(w) = \sum_{v \in range(w)} c_v \cdot g_{v-w}(\mathbf{1}) \tag{6}$$

, where $g_0(\mathbf{1}) = f(rot(\mathbf{l} \cdot \mathbf{1}))$ and $g_1(\mathbf{1}) = f(rot(\mathbf{r} \cdot \mathbf{1})$ (we invoke the left or right negacyclic generator based on the encountered bit of the radix-2 representation of the label index). For example, to compute the label of leaf 2 (010) and 3 (011) in a full tree of 8 leaves:

$$\begin{aligned} \lambda(2) &= f(rot(\mathbf{l}) \cdot f(rot(\mathbf{r}) \cdot f(rot(\mathbf{l} \cdot \mathbf{1})))) \\ \lambda(3) &= f(rot(\mathbf{l}) \cdot f(rot(\mathbf{r}) \cdot f(rot(\mathbf{r} \cdot \mathbf{1})))) \end{aligned} \tag{7}$$

For the prover's witness, with the linear nature of our original $hash_{ringsis}$ and f, it's trivial that their composition is also linear. This means that the occurence of a set of elements can be expressed as a linear combination of their

labels $\lambda(x)$ and updating the tree can happen by adding 1 to the coefficient of the particular label in the linear combination, and collapse the generalized hash tree. The complexity of this operation is $O(logDlog^2n)$ where D is the depth of our full binary tree.

As for the verifier state, we can follow the naive method presented in the beginning of this section. To produce the accumulator value we set $acc = \sum_i^k L(x_i)$. To add or revoke an element x_l we simply do $acc = acc + L(x_l)$ or use $-L(x_i)$ to revoke the same element.

As our hash function is purely algebraic (even linear), we see that it is significantly more compatible in circuit complexity to express as an algebraic circuit for Zero-knowledge proofs than keccak or SHA2 for which the algebraic representation results in an unpractically deep circuit depth because of ARX operations and round constants.

3 Implementation

We implemented our construction in C[8] and benchmarked it on an i7-1165G7 CPU with 16Gb ram, using an x64 linux machine. Trapdoor generation used the *getrandom* call for random numbers. We compared our benchmarks to an SHA512 Merkle tree implementation using identical input data. We used the following parameters for our implementation: $n = 1024, q = 59393$

	Tree generation	Witness generation	Verification	Revocation
SHA512 Merkle	2.9084s	0.000003s	0.000007s	2.89611865s
Our work	1.4045s	0.000001s	0.000001s	0.00000192s
Advantage	107%	200%	600%	150839413%

4 Discussions

We presented a RingSIS instantiation of the original Generalized Hash Tree concept of Shi et. al[3] and implementation benchmarks for standard (membership proof) usage. Our circuit complexity with relation to ZkStarks, expressed as an AIR[9] is 3971 (1024 constants, 1024 multiplication and 1023 addition) for a single R-SIS hash and $logDlog^2n$ times this value to construct a membership tree from scratch, which is extremely efficient for an AIR compared to RSA and Merkle-based accumulators. Further directions of this research would be to construct partial trapdoor informations for different leaves, resulting in a scoped authenticated accumulation instead of a global one. There is also a limitation in our construction such that it only handles fixed size-input which could be mitigated by using small, chained RSIS instances and breaking linearity in-between similar to SWIFFTX[10]. Special thanks to Alex Vlasov for the initial implementation of a previous version of this work and Zoltan Lovas for the Merkle tree benchmarking code.

References

[1] M. Ajtai, "Generating hard instances of the short basis problem," in *Automata, Languages and Programming* (J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds.), (Berlin, Heidelberg), pp. 1–9, Springer Berlin Heidelberg, 1999.

[2] D. Micciancio and O. Regev, "Worst-case to average-case reductions based on gaussian measures," vol. 37, pp. 372– 381, 11 2004.

[3] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *Advances in Cryptology – EUROCRYPT 2013* (T. Johansson and P. Q. Nguyen, eds.), (Berlin, Heidelberg), pp. 353–370, Springer Berlin Heidelberg, 2013.

[4] C. Peikert, "Limits on the hardness of lattice problems in lp norms," *Computational Complexity*, vol. 17, pp. 300–351, 05 2008.

[5] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, 01 2009.

[6] D. Micciancio and C. Peikert, "Hardness of sis and lwe with small parameters." Cryptology ePrint Archive, Report 2013/069, 2013. `https://eprint.iacr.org/2013/069`.

[7] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Comput. Complex.*, vol. 16, p. 365–411, Dec. 2007.

[8] "https://github.com/silur/raha."

[9] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity." Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[10] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "Swifftx : A proposal for the sha-3 standard," 2008.