Provably Secure FFT Hashing

Vadim Lyubashevsky* Daniele Micciancio† Chris Peikert‡ Alon Rosen§

July 28, 2006

Abstract

We propose a new family of collision resistant hash functions with the distinguishing feature of being provably secure. The main technique underlying our functions is a novel use of the *Fast Fourier Transform* to achieve ideal "diffusion" properties, together with a random linear function to achieve *compression* and "confusion". Our functions admit fast implementation both in hardware and software, but are set apart from previous proposals (based on similar building blocks) in the literature by a supporting security proof: it can be formally proven that (asymptotically) finding collisions to our functions (for keys chosen uniformly at random) with non-negligible probability is at least as hard as solving certain lattice problems in the worst case. Our proposal and techniques are based on previous work by Micciancio (FOCS 2002, Computational Complexity 2007), Peikert and Rosen (TCC 2006) and Lyubashevsky and Micciancio (ICALP 2006).

1 Introduction

Provably secure collision resistant hash functions can be built based on the assumption that various problems from computational number theory (e.g., the discrete logarithm problem or factoring large integers) are hard to solve on the average. Unfortunately, all known such proposals result in hash functions with computation cost comparable to typical public key cryptographic operations, making them unattractive from a practical point of view. Intuitively, collision free hashing seems a substantially easier problem than public key encryption, and better solutions are expected. As a result, practical design of collision resistant hash functions has, so far, focused on ad-hoc constructions, similar to those employed in the design of block ciphers.

Ad-hoc hash functions (like MD5, SHA1, etc.) are not supported by security proofs. Rather, their design is usually justified based on intuitive but vague cryptographic engineering principles (like "diffusion" and "confusion" properties) and validated by intensive cryptanalytic efforts. Recently, newly discovered cryptanalytic attacks [19, 20, 3] have started casting serious doubts both on the security of these specific functions and the effectiveness of the underlying design methodology.

In this paper we propose a new family of hash functions that is very appealing and intuitive from a traditional design point of view, and, at the same time, achieves the robustness and reliability benefits of provable security. The high level structure of our functions (see Figure 1) is very simple:

*University of California, San Diego vlyubash@cs.ucsd.edu
†University of California, San Diego daniele@cs.ucsd.edu
‡Massachusetts Institute of Technology cpeikert@mit.edu
§Harvard University alon@eecs.harvard.edu

1. The input is represented as an $m \times n$ matrix $(x_{i,j})$, where $n = 2^k$ is the security parameter, m is a small constant (e.g., $m = 8$), and each $x_{i,j}$ is a small (e.g., $x_{i,j} \leq 4$ or 8) non-negative integer.

2. The input is first processed by multiplying each column by ω^{j-1} for an appropriately-chosen constant ω, and taking the *Fast Fourier Transform* of every row:

$$(y_{i,1}, \ldots, y_{i,n}) = \mathrm{FFT}(\omega^0 x_{i,1}, \ldots, \omega^{n-1} x_{i,n}).$$

   We remark that this first operation is easy to invert, and it is performed to achieve "diffusion," i.e., to mix the input bits in every row.

3. Next a random linear combination of every column

$$z_j = a_{1,j} y_{1,j} + \cdots + a_{m,j} y_{m,j}$$

   is computed. The output is the vector $(z_1, \ldots, z_n)$. This second operation compresses the input and achieves "confusion."

All arithmetic operations are performed modulo a small prime (or product of primes) p of the form $p = 2t \cdot n + 1$, so that $\mathbb{Z}_p^*$ has an element ω of order $2n$ and the n-dimensional Fourier transform can be efficiently computed over $\mathbb{Z}_p$. We remark that the modulus p is fairly small (e.g., 16 bits or so, but bigger than the input values $x_{i,j}$) and m is just a small constant (e.g., $m = 8$).

Viewed independently, the linear equations $z_j = a_{1,j} y_{1,j} + \cdots + a_{m,j} y_{m,j}$ admit many solutions, and these solutions can easily be found individually. However, the rows $(y_{i,1}, \ldots, y_{i,n})$ are constrained to be the result of applying Step 2 to vectors whose entries $x_{i,j}$ are small integers.

Perhaps surprisingly, these constraints turn out to be sufficient to guarantee *provable* cryptographic hardness (i.e., one-wayness and even collision-resistance) of the compression function. Specifically, the function admits a proof of security of a very strong type: finding collisions on the average (when the key $(a_{i,j})$ is chosen uniformly at random in $\mathbb{Z}_p$) even with very small but non-negligible probability is at least as hard as solving an underlying mathematical problem on *point lattices* in the *worst case* (over the choice of the input, and with probability exponentially close to 1 over the internal randomness of the algorithm). The proof of security follows from the observation that the above function is a special case of the *ideal lattice* functions of [9], and a variant of the *cyclic lattice* functions of [10, 13].

Related work. The idea of using the Fast Fourier Transform (FFT) as a building block for the construction of hash functions is not new. Examples of such functions are the FFT hash functions of Schnorr [14, 15, 16]. Unfortunately, most of these functions were subsequently cryptanalyzed and shown to be insecure [5, 2, 17]. The distinguishing feature of our hash function, that sets it apart from previous work, is the way FFT is used, and the resulting proof of security. Namely, while in previous work [14, 15, 16] FFT was applied to unrestricted input vectors $(x_1, \ldots, x_n) \in \mathbb{Z}_p^n$, here we require the input values x_i to belong to a small subset of $\mathbb{Z}_p$. This introduces non-linear constraints on the output values of the FFT operation, a fact that plays a fundamental role both in our theoretical proof of security as well on the impact of heuristic attacks to our function. We believe that our novel use of FFT might be of independent interest, and find other applications in cryptographic design.

Another important ingredient in the conceptual design of our function (and associated proof of security) is the use of lattices with special structure as an underlying mathematical problem. Special classes of lattices (with closely related, but somehow different structure than ours) also have been used before in practical constructions (most notably, the NTRU encryption scheme [8]), but without a mathematical proof of security.

More closely related to our work, is the theoretical study initiated by Ajtai [1] of cryptographic functions that are provably secure based on worst-case assumptions for lattice problems. Ajtai's work and subsequent improvements [7, 4, 11, 12] leading to provably secure hash functions, are based on general lattices and do not lead to very efficient implementations, mostly because of the huge key size (which grows quadratically in the security parameter.) A first step toward bridging the gap between theoretical constructions and practical functions was taken by Micciancio [10] who proposed the use of lattices with special structure (namely, cyclic lattices) and showed how they lead to cryptographic functions that are provably secure (in the strong worst-case/average-case sense of [1]) and still admit very fast implementation. In fact, the use of FFT in the implementation of these provably secure cryptographic functions was already suggested in [10]. The main limitation of the function proposed in [10] was the notion of security achieved: they are provably one-way (based on worst-case assumption on the complexity of cyclic lattices), but not (as subsequently shown in [13, 9]) collision resistant. This difficulty was finally overcome by the authors in [13, 9], where the function initially proposed in [10] is modified and generalized to achieve collision resistance.

From a theoretical point of view, the functions proposed in this paper are equivalent to and inherit all provable security features from the cyclic/ideal hash functions of [13, 9]. But differently from [10, 13, 9], the emphasis in this paper is on practical implementation issues, and the construction of instances and variants of those hash functions that enjoy very efficient implementation from a practical point of view. For a deeper understanding of the theoretical ideas undelying the proof of security of our hash functions the reader is referred to [10, 13, 9].

Organization The rest of the paper is organized as follows. In section 2 we give some mathematical background. In section 3 we give a detailed description of our hash function family. In section 4 we describe the relation between our functions and the functions of [10, 13, 9], resulting in a security proof. In sections 5 and 6 we describe implementation issues and possible instantiations of our function family to give an idea of the efficiency of our construction. Finally, in section 7 we discuss practical attacks, their (in)effectiveness and open problems.

2 Preliminaries

The preliminaries below are not needed for the reader who is only interested in the implementation of the hash function. Such a reader may safely skip to section 3.

Lattices An n-dimensional *integer lattice* is a subgroup of $\mathbb{Z}^n$ generated by linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^n$. The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is called a *basis* for the lattice, and can be compactly represented by the matrix $\mathbf{B}$ having the basis vectors as rows. The *minimum distance* of a lattice $\mathcal{L}(\mathbf{B})$, denoted $\lambda_1^\infty(\mathcal{L}(\mathbf{B}))$, is the minimum distance between any two (distinct) lattice points and equals the length of the shortest nonzero lattice vector. The minimum distance can be defined with respect to any norm, but we are mainly interested in the ℓ_∞ norm. The γ-approximate *Shortest Vector Problem*, SVP_γ is the following: given a lattice $\mathcal{L}(\mathbf{B})$, find a nonzero

vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v}\|_\infty \leq \gamma \lambda_1^\infty(\mathcal{L}(\mathbf{B}))$. The shortest vector problem in the infinity norm was shown to be NP-hard for factor up to $\gamma(n) = n^{1/\log\log n}$ by Dinur [6] and it is conjectured to be hard to approximate to a polynomial factor.

Algebra An ideal I of a ring R is an additive sub-group of R with the additional property that the product of any element in I with any element in R is in I. If $R = \mathbb{Z}[\alpha]/\langle f \rangle$ for some irreducible polynomial f of degree n, then any ideal I of R is isomorphic as an additive group to a full-rank subgroup of $\mathbb{Z}^n$. In [9], an *ideal lattice* was defined as an integer lattice that is isomorphic as an additive group to some ideal in a ring $R = \mathbb{Z}[\alpha]/\langle f \rangle$ for some f. Ideal lattices are a generalization of cyclic lattices that were introduced in [10]. The complexity of ideal lattices has not been studied rigorously, but it does not seem that any current lattice basis reduction algorithm is able to take advantage of their special structure. Thus it is reasonable to believe that finding the shortest vector in ideal lattices is not much easier than finding the shortest vector in a general lattice.

The ideal lattices that are most relevant in this work are lattices that are isomorphic as additive groups, under the obvious isomorphism, to ideals in the ring $R = \mathbb{Z}[\alpha]/\langle f \rangle$, where $f = \alpha^n + 1$ for n a power of 2. For such values of n, f is a monic irreducible polynomial over $\mathbb{Q}[\alpha]$. In fact, f is known as the $2n$th *cyclotomic polynomial*, which has as its zeros all n of the the primitive $2n$th complex roots of unity. As an aside, we mention n being a power of 2 is also a necessary condition for $\alpha^n + 1$ to be irreducible over $\mathbb{Q}[\alpha]$.

3 Algorithmic Description of the Function

We start by describing the function in manner that can be easily translated into an implementation. This description fills in many of the details in the informal overview from Section 1.

3.1 Function Parameters

Our function involves the following parameters (concrete choices for our implementation are suggested in Section 6):

- n — a positive power of 2, and the number of columns in the key and input matrices.
- m — the "width" of the function, i.e. the number of rows in the key and input matrices.
- p — a prime modulus over which the function is computed, of the form $p = 2tn + 1$ for some positive integer t. This guarantees that $\mathbb{Z}_p^*$ has an element ω of order $2n$.
- d — a bound on the entries of the input matrix, significantly smaller than p.

3.2 Modular Fast Fourier Transform

The main tool which allows our function to be computed efficiently is the *modular Fast Fourier transform* (mFFT) over the field $\mathbb{Z}_p$. The modular FFT is a variant of the "classical" FFT. Both algorithms compute the *discrete Fourier transform* (DFT) of a vector, but over different fields. While the classical FFT computes the DFT over the field $\mathbb{C}$ of complex numbers, the modular FFT computes the DFT over the field $\mathbb{Z}_p$.

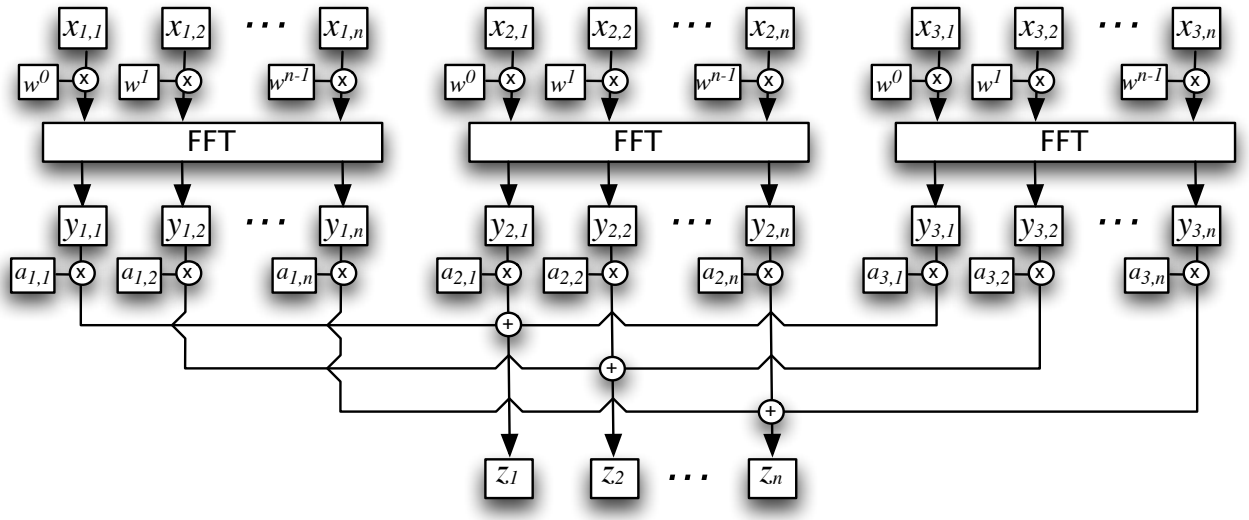

Figure 1: Compression function with $m = 3$

To operate on vectors of dimension n over $\mathbb{Z}_p$, the mFFT algorithm requires an element $\zeta \in \mathbb{Z}_p^*$ which is a *primitive nth root of unity*. An element ζ is a primitive nth root of unity if its order is n, i.e. $\zeta^n = 1$ and $\zeta^i \neq 1$ for every positive integer $i < n$. Because p is of the form $p = 2tn + 1$ and we have an element $\omega \in \mathbb{Z}_p^*$ of order $2n$, we may take $\zeta = \omega^2$.

The modular FFT performs all its operations using integer arithmetic modulo p, and requires only $0.5n \lg n$ multiplications and $n \lg n$ additions in $\mathbb{Z}_p$. The mFFT is highly parallelizable and suitable for implementation both in hardware and software. See Section 5 for more details.

3.3 The Algorithm

The algorithm for computing our compression function can be described as follows (see Figure 1 for a graphical depiction):

Key: an $m \times n$ matrix $\mathbf{A} = (a_{i,j})$. Each entry $a_{i,j}$ is chosen uniformly and independently in $\mathbb{Z}_p$.

Input: an $m \times n$ matrix $\mathbf{X} = (x_{i,j})$. Each entry $x_{i,j}$ is an arbitrary value in $D = \{0, \ldots, d\} \subset \mathbb{Z}_p$.

Function evaluation: Perform the following steps, where all arithmetic is performed in $\mathbb{Z}_p$:

1. For each i, j, let $x'_{i,j} = \omega^{j-1} \cdot x_{i,j}$.
2. For each row $i = 1, \ldots, m$, let $(y_{i,1}, \ldots, y_{i,n}) = \text{mFFT}(x'_{i,1}, \ldots, x'_{i,n})$.
3. For each column $j = 1, \ldots, n$, let $z_j = a_{1,j}y_{1,j} + \cdots + a_{m,j}y_{m,j}$.
4. Output $\mathbf{z} = (z_1, \ldots, z_n)$.

It can be seen that the algorithm maps $mn\lfloor \lg(d+1) \rfloor$ input bits into $n\lceil \lg p \rceil$ bits. Thus the compression factor depends on the relationships among m, d, and p (these relationships also have implications for the security of the function; see Section 4 for details). The key size is $mn\lceil \lg p \rceil$ bits. We refer the reader to Section 6 for concrete choices of parameters.

4 Proof of Security

In this section we relate our compression function to the theoretical results in [10, 13, 9]. In particular, we show the equivalence of the compression function described in the previous section with an algebraic function proven to be collision resistant in [9]. A reader interested only in the algorithmic description and implementation may safely skip this entire section.

4.1 The Algebraic Function

The basic operations of the function are performed in the ring $R = \mathbb{Z}_p[\alpha]/\langle f \rangle$, where $f = \alpha^n + 1$ is a monic irreducible polynomial and p is a suitably-chosen integer. The ring R is an integral domain consisting of all polynomials of degree less than n and having coefficients in $\mathbb{Z}_p$, where multiplication and addition are performed modulo p and f. As above, let $D = \{0, \ldots, d\}$. Define $S \subset R$ to be the subset of all polynomial residues in R whose coefficients are all in D. That is, $S = \{x = x_1 + x_2\alpha + \cdots + x_n\alpha^{n-1} \in R \ : \ \forall \ i, \ x_i \in D\}$. The function can then be described mathematically as follows:

Key: an m-tuple $A = (a_1, \ldots, a_m) \in R^m$. Each entry is chosen uniformly and independently in R.

Input: an arbitrary m-tuple $X = (x_1, \ldots, x_m) \in S^m$.

Function value: The function is defined as follows (all arithmetic is performed in R):

$$h_A(X) = \sum_{j=1}^{m} a_j \cdot x_j.$$

The following theorem states that if one can find collisions, with non-negligible probability, for randomly chosen keys A, then one can find the approximate shortest vector in any lattice isomorphic to an ideal in $\mathbb{Z}[\alpha]/\langle f \rangle$.

Theorem 4.1 *[9, Theorem 2] Let $p > 6dmn^{1.5} \log n$ and $\gamma = 72dmn \log^2 n$. If there is an algorithm that, for a random choice of $A \in R^m$, is able to produce $X_1 = X_2 \in S^m$ such that $h_A(X_1) = h_A(X_2)$ with non-negligible probability, then the problem SVP_γ can be solved in polynomial time for any lattice isomorphic to an ideal in $\mathbb{Z}[\alpha]/\langle f \rangle$.*

A result similar to Theorem 4.1 was shown by [13] for the related case of ideals in the ring $\mathbb{Z}[\alpha]/\langle f \rangle$ where $f = \alpha^n - 1$, and n is a prime. One may consider other choices of f as well, as done in [9]. In this work, we chose $f = \alpha^n + 1$ for the sake of convenience of implementation. In particular, it allows the dimension n to be a power of two (recall that $\alpha^n + 1$ is irreducible over $\mathbb{Q}$ if and only if n is a power of two).

4.2 Equivalence of Functions

Now we show that the above function is equivalent to the one described in Section 3 (i.e. finding collisions in one implies finding collisions in the other). This can be seen via the following observations:

- There is a correspondence between n-dimensional vectors $\mathbf{x} \in \mathbb{Z}_p^n$ and elements $x \in R$: the vector $\mathbf{x} = (x_1, \ldots, x_n)$ corresponds to the polynomial $x = x_1 + x_2\alpha + \cdots + x_n\alpha^{n-1} \in R$.

- Multiplication of polynomials $a, x \in \mathbb{Z}_p[\alpha]/\langle\alpha^n + 1\rangle$ can be implemented by evaluating a and x on all the roots of $\alpha^n + 1$ and multiplying the corresponding values. The product polynomial $a \cdot x$ can be interpolated from the result.

  The roots of $\alpha^n + 1$ are the primitive $2n$th roots of unity, which are of the form $\omega^{2j+1} = \zeta^j \cdot \omega$ for $j = 0, \ldots, n-1$. The polynomial x can be evaluated on these points by appropriately pre-multiplying and applying the mFFT to the corresponding vector $\mathbf{x}$:

$$x(\zeta^j \cdot \omega) = \sum_{i=1}^{n} x_i \cdot (\zeta^j \omega)^{i-1} = \sum_{i=1}^{n} (x_i \cdot \omega^{i-1}) \cdot \zeta^{j(i-1)} = \sum_{i=1}^{n} (x_i) \cdot \zeta^{j(i-1)} = \mathrm{mFFT}(\mathbf{x}\,')_j$$

  where $\mathbf{x}\,' = (x_1, \ldots, x_n)$ and $x_i = x_i \cdot \omega^{i-1}$ as in the algorithm from Section 3.3.

- Because mFFT computes a bijection and the key matrix $\mathbf{A}$ is completely uniform, then so is the mFFT of each column of $\mathbf{A}$. Therefore in the algorithm we view $\mathbf{A}$ as already having had mFFT applied to each of its rows.

- Finally, we note that there is no need for the algorithm to interpolate its output via an inverse mFFT: the mFFT computes a bijection and is efficiently invertible, so collision-resistance and one-wayness are preserved. In addition, the range of mFFT is equal to its domain, so omitting the inverse mFFT does not introduce any difficulties in representing the output.

5 Implementation Details

5.1 Parallelizing the Computation

Our compression function is parallelizable to a very large degree. The two main targets of parallelism are in (1) the FFT implementation and (2) the m invocations of FFT on each row of the input matrix. There are many off-the-shelf FFT libraries which are optimized for parallel architectures, and parallelizing the m invocations of FFT is elementary, because they are independent. As dual- and multi-core processors become more common in commodity hardware, we expect that optimized implementations will be able to achieve many-fold improvements in speed.

5.2 Chinese Remaindering for Larger Moduli

For the best performance of a software implementation, multiplication modulo p must be done using the on-chip registers and standard instruction set of the hardware. One way to ensure this would be to simply choose a small enough value of p. However, by doing so we would no longer know how to provide a proof of security for the function.

We instead propose choosing $p = p_1 p_2$ to be the product of two different primes (of the form $p_i = 2tn + 1$), and computing the compression function in parallel modulo p_1 and p_2 (on the same input). By the Chinese Remainder Theorem, it can be shown that the resulting compression function is equivalent to the original one; in particular, finding collisions remains hard. This idea can of course be generalized to values of p that are products of more than two primes.

Note that no explicit Chinese remainder reconstruction of the output is necessary; the outputs of the parallel evaluations may simply be concatenated. We also stress that the factorization of p need not be secret, and that the choice of p having these special properties does not affect the security of the function.

5.3 Using Classical FFT Algorithms

Our compression function can also be implemented using a "classical" FFT over the field of complex numbers, for which there is much preexisting code and specialized hardware. However, one must account for the following issues (which do not arise when using a modular FFT):

- A classical FFT requires floating-point calculations, which are inexact and tend to be slower than integer arithmetic. However, the advantages of customized hardware and parallel architecture may outweigh these concerns.
- The algorithm will additionally have to perform pre-multiplication and a classical FFT on the rows of the key vector $\mathbf{A}$, as it does with the rows of $\mathbf{X}$. If the same key is to be used multiple times, this processing can be performed as part of a setup phase, and the results stored.
- The result of the "confusion" step (which has complex entries) must be interpolated via an inverse FFT, interpreted as an integer vector, and reduced modulo p.

6 Concrete Parameters: Three Modes

The most attractive feature of our hash function family is an *asymptotic* proof of security showing that, unless an underlying mathematical problem can be solved in polynomial time in the *worst case*, finding collisions is infeasible. In particular (and differently from typical ad-hoc hash functions with are defined only for specific values of the input/output/key parameters), the security of the function can be arbitrarily increased by choosing a suitable large value of the security parameter. Of course, increasing the security parameter results in a loss of efficiency (both in terms of computation time and output size), but this loss is only polynomial (in fact, almost linear), while the complexity of the underlying problem is conjectured to be superpolynomial (or even exponential). So, the security of the function increases much faster than the computational cost of evaluating it, and attacks can always be countered (unless an efficient algorithm to solve lattice problems in the worst case is discovered) by choosing larger values of the security parameter.

Unfortunately, the current proofs of security underlying our function [10, 13, 9] are not very tight from a numerical point of view, and the security parameter must be fairly large before the asymptotics kick in and breaking the function becomes provably as hard as an underlying lattice problem for which no efficient solution is believed to exist.

For concreteness, in this section we consider some possible instantiations of our function for specific values of the security parameter. The goal here is mainly to give a sense of the potential efficiency of functions based on our design techniques. More extensive analysis of the concrete security of the functions is necessary before any of the instantiations can be used in practice.

We propose three sets of function parameters, which we call "modes," in Figure 2. The "Bulk" mode uses dimension $n = 1024$ and has a large input and output block size (64 Kbits and 28 Kbits, respectively). For this concrete choice of parameters our proof of security is based on a reasonable

	n	m	d	p	Input (bits)	Output (bits)
Bulk	1024	16	15	$\approx 2^{28}$	65536	28672
Mini	128	8	3	257	2048	1025
Nano	64	8	3	257	1024	513

Figure 2: Concrete parameter choices for three modes of our compression function.

	Bulk	Mini	Nano	SHA-1 (openssl)	SHA-1 (sha1sum)	SHA-256 (openssl)
Rate (MB/s)	0.43	0.69	0.67	155	35	42

Figure 3: Speed of three modes of our compression function (Bulk, Mini, Nano), two versions of SHA-1, and SHA-256

hardness assumption (i.e. hardness of approximating the shortest vector in a certain class of 1024-dimensional lattices to within a factor of 2^{32} *in the worst case*. This is out of reach of the current state of the art lattice algorithms.[1]) We propose the Bulk mode primarily for applications which hash very large inputs and require a high level of security. If a smaller output size is needed, the Bulk mode can be composed with a different provably secure hash function based on other number-theoretic assumptions. This allows to amortize the cost of expensive number-theory based functions over a very large document, leading to a function which is at the same time reasonably efficient and provably secure.

We also propose the "Mini" and "Nano" modes, which have smaller output sizes (about 1 Kbit and 512 bits, respectively) and have somewhat better throughput. Unfortunately, for these choices of parameters, theorem 4.1 does not apply because the value of p is too small, and thus we do not get a proof of security. It's possible to increase the value of p, but the resulting proof of security will *not* be based on reasonable hardness assumptions: that is, the security reduction will yield an approximation factor which is *easy* to achieve efficiently. Said plainly, for these choices of parameters we *do not have a proof of security*. However, we also *do not know how to find collisions in these modes of the function*. It may be that finding such collisions is hard, but we do not know how to prove it. We propose the Mini and Nano versions as targets for further study and cryptanalysis.

6.1 Performance

We implemented the three modes of the compression function described above. Our implementation was written in C, using a standard iterative modular FFT algorithm (for the Bulk mode, we used the Chinese Remainder technique described in Section 5.2). No attempt was made to optimize or parallelize the code, and we believe that there are many standard optimization techniques as well as improvements to the core FFT algorithm which will improve performance. Our implementation was compiled using `gcc` version 4.0.3 (compiler flags: `-O4`) on a Pentium M 1.6 GHz system having 2 MB of L2 cache and running Linux kernel 2.6.15.

We compared our implementation with the SHA-1 and SHA-256 hash algorithms on the same system. We used the highly-optimized implementations included in `openssl` version 0.9.8a (using

[1]We remark that 1024 is the dimension of lattice problems that can be provably solved in the worst case should our function be broken. Cryptanalysing our function seem to require the solution of lattice problems in even higher dimension.

the `openssl speed` benchmarks), as well as the `sha1sum` command-line utility on a 40 MB file. Aggregate performance numbers are presented in Figure 3. Our function appears to be about 60 times slower than SHA-256. With some modest programming effort, we increased the speed of the "Nano" version of the function by a factor of four, thus making it only 15 times slower than SHA-256. We believe that with additional programming effort, the performance of the function can be greatly improved and we are currently working on a more efficient implementation.

7 Possible Attacks

We are still in the early stages of cryptanalysis, and we only considered a few standard attacks. One possible attack is a lattice attack which was implicitly mentioned in [10]. The idea of the attack is the following: given a random m-tuple of vectors $(\mathbf{a}_1, \ldots, \mathbf{a}_m)$, create an $mn \times n$ matrix consisting of $Rot(\mathbf{a}_1), \ldots, Rot(\mathbf{a}_m)$ (see figure 4) and call the resulting matrix $\mathbf{A}$.

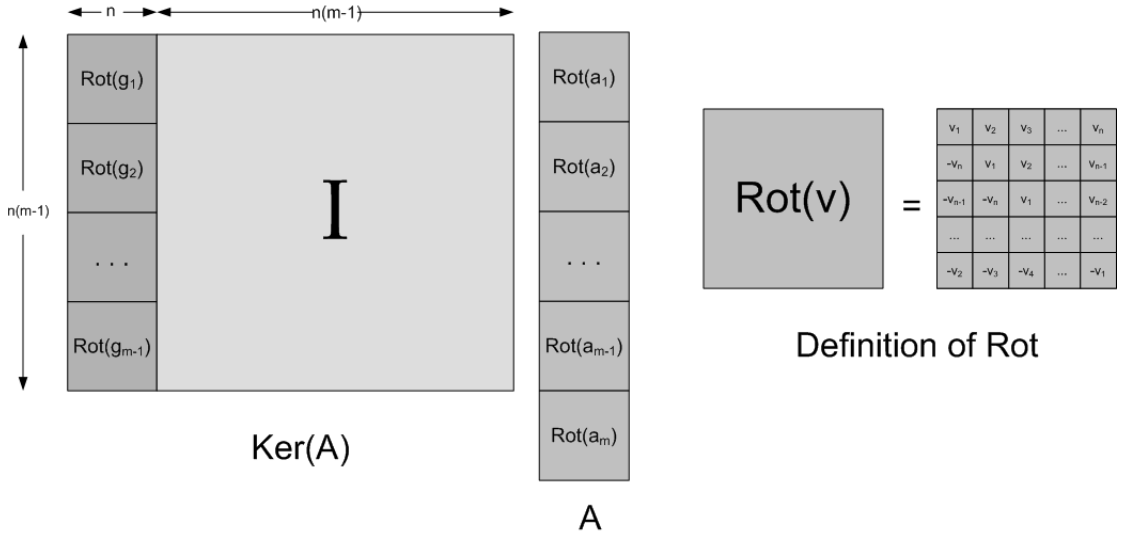

Figure 4: $\mathbf{A}$ and its kernel $Ker(\mathbf{A})$

If we are able to find a vector $\mathbf{x}$ such that $\mathbf{xA} = \mathbf{0} (\text{mod } p)$ where the absolute values of all the coefficients of $\mathbf{x}$ are at most d, then we have a collision for the hash function. To find such an $\mathbf{x}$, we create a lattice all of whose vectors are solutions to $\mathbf{xA} = \mathbf{0} (\text{mod } p)$ and attempt to find the shortest vector in this lattice. To compute the lattice, we compute the kernel of $\mathbf{A}$, which can be represented as shown in figure 4. We then transform $Ker(A)$ into a lattice basis as in Figure 5 and run LLL.

During our experiments, this attack worked only up to $n = 32$ (i.e. we were not able to find collisions even in the "Nano" instantiation of the function). One of the reasons may have been the fact that LLL and its Block-Korkin-Zolotarev (BKZ) variant return the smallest vector in the ℓ_2 norm, while we are looking for a smallest vector in the ℓ_∞ norm. But another reason for the failure of the attack is that the lattices of dimension nm are too large for LLL to find the shortest vector. A possible way to improve the attack is after finding a relatively short vector in the ℓ_2 norm, try to enumerate the short vectors around it in hope of finding a short vector in ℓ_∞.

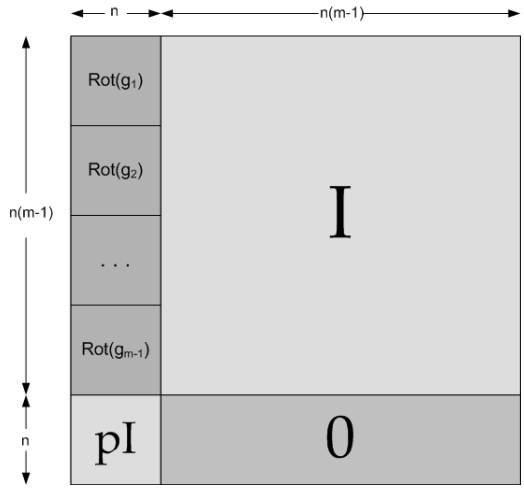

Figure 5: Basis for a lattice consisting of $\{\mathbf{x}|\mathbf{xA} = \mathbf{0}(mod\, p)\}$

Another possible attack is Wagner's generalized birthday attack [18]. We do not describe the particulars here, but the attack would take on the order of $2^{\frac{n \log p}{\log d+1}}$ time. This makes it impractical even for the "Nano" instantiation.

Essentially, any attack on our hash function is equivalent to trying to find a small $\mathbf{x}$ such that $\mathbf{xA} = 0(\text{mod } p)$ (where $\mathbf{A}$ is as in figure 4). We believe that if there is an attack that will succeed in finding collisions for the "Nano" and "Micro" versions, it should somehow use the fact that the rows of $\mathbf{A}$ consist of $Rot(a_i)$ rather than random independent vectors. Because if $\mathbf{A}$ consists of random independent vectors, then trying to find a small $\mathbf{x}$ such that $\mathbf{xA} = \mathbf{0}(\text{mod } p)$ is essentially the same as solving a version of the random subset sum problem which has been studied extensively, but eluded anything remotely close to an efficient solution. Both approaches that we have tried do not take advantage of the structure of $\mathbf{A}$, and thus it's unlikely that they will lead to finding collisions. We think that finding collisions efficiently may require a novel "algebraic" approach.

Finally, we mention that it would be interesting to analyze the security of our function (even for scaled down values of the parameters for which the proof of security does not hold) with respect to conventional cryptanalysis techniques, like linear or differential cryptanalysis. This may give further insight both into the security of our function, and the applicability of the underlying techniques in a wider cryptographic design context.

Acknowledgements

We would like to thank Ron Rivest for his kind advice and suggestions.

References

[1] M. Ajtai. Generating hard instances of lattice problems. In *STOC*, pages 99–108, 1996.

[2] T. Baritaud, H. Gilbert, and M. Girault. FFT hashing is not collision-free. In *EUROCRYPT*, pages 35–44, 1992.

[3] E. Biham, R. Chen, A. Joux, P. Carribault, W. Jalby, and C. Lemuet. Collisions of SHA-0 and reduced SHA-1. In *EUROCRYPT*, 2005.

[4] J. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In *FOCS*, pages 468–477, 1997.

[5] J. Daemen, A. Bosselaers, R. Govaerts, and J. Vandewalle. Collisions for schnorr's hash function FFT-hash presented at crypto '91. In *ASIACRYPT*.

[6] I. Dinur. Approximating SVP_∞ to within almost-polynomial factors is NP-hard. *Theor. Comput. Sci.*, 285(1):55–71, 2002.

[7] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. Technical Report TR-42, ECCC, 1996.

[8] J. Hoffstein, J. Pipher, and J. H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[9] V. Lyubashevsky and D.Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, 2006.

[10] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. *Computational Complexity*. (To appear. Preliminary version in FOCS 2002).

[11] D. Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor. *SIAM J. on Computing*, 34(1):118–169, 2004.

[12] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. on Computing*. (To appear. Preliminary version in FOCS 2004).

[13] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, 2006.

[14] C. P. Schnorr. FFT-hash, an efficient cryptographic hash function. In *Crypto Rump Session*, 1991.

[15] C. P. Schnorr. FFT–Hash II, efficient cryptographic hashing. In *EUROCRYPT*, pages 45–54, 1992.

[16] C.P. Schnorr and Serge Vaudenay. Parallel FFT-hashing. In *Fast Software Encryption*, pages 149–156, 1993.

[17] S. Vaudenay. FFT-Hash-II is not yet collision-free. In *CRYPTO*, pages 587–593, 1992.

[18] D. Wagner. A generalized birthday problem. In *CRYPTO*, pages 288–303, 2002.

[19] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis for hash functions MD4 and RIPEMD. In *EUROCRYPT*, 2005.

[20] X. Wang and H. Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, 2005.