Provably Secure FFT Hashing
(+ comments on “probably secure” hash functions)

Vadim Lyubashevsky
(University of California, San Diego)

Daniele Micciancio

Chris Peikert
(MIT)

Alon Rosen
(Harvard University)
Our Hash Function
A (Very) High Level Description

- Key: 3 random polynomials
- Input: 3 polynomials with small coefficients
- Function: compute sum of products
- All arithmetic performed modulo $p$ and $\beta^{n+1}$
  ($\beta$ is the indeterminate in the polynomials)
- Function is very efficient, parallelizable, and provably collision-resistant.
Efficiency and Security

Efficiency:
- Input has $b$ bits
- $O(b \log(b))$ time to compute the hash

Security (2 modes of the function):
- “Bulk mode”
  - Large output
  - Finding collisions at least as hard as solving a certain lattice problem in the \textit{worst case}.
- “Nano mode”
  - Small output
  - Same structure as the bulk mode
  - Finding collisions equivalent to solving a certain (different) lattice problem in the average case
Diffusion and Confusion
Diffusion and Confusion

- For Diffusion, we use the Fast Fourier Transform
  - Idea already appeared in [S91,S92,SV93]
- For Confusion, simply use linear combinations
- By using results in [M02,PR06,LM06], we can build a provably secure compression function.
Performing the Compression
(Step 0, Entering Input)

- Compressing a string of length $mn$ ($m=3$)
- Each $x_{i,j}$ is in $\{0,\ldots,d\}$
- So domain is of size $(d+1)^{mn}$  $(d+1)^{3n}$
- All operations performed in the field $\mathbb{Z}_p$ $(p>>d)$
Performing the Compression (Step 1, Diffusion)

- Step 1: multiply $x_{i,j}$ by $w^{j-1}$
  - (Just a trick to do multiplication modulo $\beta^{n+1}$)
  - $w$ is an element in $\mathbb{Z}_p^*$ of order $2n$
  - Thus, $w^2$ is a primitive $n^{th}$ root of unity in $\mathbb{Z}_p^*$
Performing the Compression (Step 2, Diffusion)

- **Step 2**: Compute the Fast Fourier Transform of each grouping
  - Use \( w^2 \) as the primitive \( n^{th} \) root of unity in \( \mathbb{Z}_p^* \)
  - \( y_{i,j} = \sum_{1 \leq k \leq n} (x_{i,j}w^{j-1})w^{2j(k-1)} \)
Performing the Compression (Step 3, Confusion)

- **Step 3:** Multiply $y_{i,j}$ by $a_{i,j}$
  - The $a_{i,j}$ are uniformly random in $\mathbb{Z}_p$
  - They are the hash function key
Performing the Compression (Step 4, Confusion)

- **Step 4:** $z_j = \sum_{1 \leq i \leq n} a_{i,j} y_{i,j}$
- **Output size:** $p^n$
Equivalent Hash Function

- **Input:** \( x_1,...,x_m \) in \( Z_p[\beta]/<\beta^{n+1}> \) (\( m=3 \))
  - Each coefficient of \( x_i \) is in \( \{0,...,d\} \)
- **Hash key:** \( a_1,...,a_m \) in \( Z_p[\beta]/<\beta^{n+1}> \)
- **Output:** \( z = a_1x_1 + ... + a_mx_m \)

- This function is completely equivalent security-wise to the one presented and it’s much easier to understand.
Security Guarantee

- **Input:** \( x_1, \ldots, x_m \) in \( \mathbb{Z}_p[\beta] / \langle \beta^{n+1} \rangle \) (m=3)
  - Each coefficient of \( x_i \) is in \( \{0, \ldots, d\} \)
- **Hash key:** \( a_1, \ldots, a_m \) in \( \mathbb{Z}_p[\beta] / \langle \beta^{n+1} \rangle \)
- **Output:** \( z = a_1 x_1 + \ldots + a_m x_m \)

- **Theorem [M02,PR06,LM06]:**
  - For appropriate values of \( p,n,d,m \), finding a collision for random \( a_1, \ldots, a_m \) implies solving the approximate Shortest Vector Problem for all lattices in a certain class.
The Function in Practice
(“Bulk Mode”)  

- Can build a compression function whose security is based on a worst-case problem  
- It’s efficient, but ... the output is big.  
- Sample parameters and security:  
  - Domain: \( \approx 65,000 \text{ bits} \)  
  - Range: \( \approx 28,000 \text{ bits} \)  
  - Security: Finding collisions implies approximating Shortest Vector to within factor \( \approx 2^{32} \) in any \( 1024 \)-dimensional lattice in a certain class of lattices.  
- Could be used to hash large files, but impractical for other purposes
Why such a large range?

- Recall the hash function:
- Input: $x_1, \ldots, x_m$ in $\mathbb{Z}_p[\beta]/\langle \beta^n + 1 \rangle$
  - Each coefficient of $x_i$ is in $\{0, \ldots, d\}$
  - Domain is of size $(d+1)^{mn}$ (mn lg(d+1) bits)
- Hash key: $a_1, \ldots, a_m$ in $\mathbb{Z}_p[\beta]/\langle \beta^n + 1 \rangle$
- Output: $z = a_1x_1 + \ldots + a_mx_m$
  - Range is of size $p^n$ (n lg(p) bits)
- In the proof of security, $p$ has to be large
Making the Range Smaller

- Making the range smaller:
  - Make p smaller
  - Still the same structure as provably secure function
  - Lose proof of security, but finding collisions still seems to be hard

- By lowering p, can get:
  - Domain=1024 bits, Range=513 bits
  - Finding collisions is equivalent to a certain average-case (no longer worst-case) lattice problem
Equivalent Lattice Problem

- Let \( \mathbf{a} = (a_1, \ldots, a_n) \) be a random vector \((0 \leq a_i < p)\). Define \( \text{Rot}(\mathbf{a}) \) as:

\[
\begin{array}{cccccc}
  a_1 & a_2 & a_3 & \cdots & a_n \\
-a_n & a_1 & a_2 & \cdots & a_{n-1} \\
-a_{n-1} & -a_n & a_1 & \cdots & a_{n-2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-a_2 & -a_3 & -a_4 & \cdots & -a_1 \\
\end{array}
\]
Equivalent Lattice Problem

Lattice generated by the rows of matrix $\mathbf{B}$

Problem: find vector in lattice with small inf. norm

\[
\mathbf{B} = \begin{bmatrix}
\text{Rot}(g_1) & \text{Rot}(g_2) & \cdots & \text{Rot}(g_{m-1}) \\
\end{bmatrix}
\]
Equivalent Lattice Problem

- Hardness of SVP for previous lattice depends on what Rot(g_i) is.
  - If Rot(g_i) is as we defined it, then finding collisions in the hash function is equivalent to finding a vector in the lattice with inf. norm \( \leq d \)

- Note: If Rot(g_i) is a random matrix, then we get a version of a well-studied (and believed to be hard) problem
  - Great for security ... but we don’t know how to make efficient hash function equivalent to the hardness of that problem

- To get equivalency to an efficient hash function, Rot(g_i) needs to have some “algebraic structure”.
Algebraic Structure of $\mathbf{B}$

- The lattice generated by $\mathbf{B}$ has a lot of "algebraic" structure.
- The structure does not seem to be useful for standard lattice algorithms (e.g. LLL).
- But other attacks exploiting the structure may be possible (for example, defining Rot(a) slightly differently makes the SVP problem very easy).
- But the fact that we have a proof that works for larger values of $p$ gives some evidence that the algebraic structure is not exploitable for smaller $p$’s as well.
Sample Parameters for Hash Function

- **Input**: $x_1, \ldots, x_m$ in $\mathbb{Z}_p[\beta]/<\beta^n+1>$
  - Each coefficient of $x_i$ is in $\{0, \ldots, d\}$
- **Hash key**: $a_1, \ldots, a_m$ in $\mathbb{Z}_p[\beta]/<\beta^n+1>$
- **Output**: $z = a_1x_1 + \ldots + a_mx_m$

- $n=64$, $m=8$, $d=3$, $p=257$
- Domain=1024 bits, Range=513 bits
- Takes $\approx 15$ times longer than SHA-256 (we’re in the initial stages of implementation)
Conclusion

- Presented an approach for using FFT to construct efficient, provably collision-resistant hash functions.

- Using this approach:
  - Constructed an efficient hash function, which may be useful for hashing large files, whose security is based on a worst-case problem.
  - Constructed an efficient hash function whose security is based on an average-case lattice problem.
Comments on Probably Secure Hash Functions

- LASH-k (from this workshop)
  - k = output length (e.g. k=160, 256, 384, 512)

- We can break compression function for e.g. k=232, 368, 1056, 2096, 10248, ...

- “Lunch-time” attack ... literally