Software Speed Records for Lattice-Based Signatures

Tim Güneysu[1] Tobias Oder[1] Thomas Pöppelmann[1]
Peter Schwabe[2]

[1]Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany

[2]Digital Security Group, Radboud University Nijmegen, The Netherlands

June 5, 2013
PQCrypto 2013, Limoges, France

Outline

- Motivation
- Introduction of implemented signature scheme
- Optimizations and implementation techniques for high speed
  - Fast polynomial multiplication
  - Vector instructions (AVX)
- Results and comparison
- Outlook

Motivation

- Lattices are post-quantum
- Recent proposals have practical parameters
- How fast is ideal lattice-based cryptography on modern CPUs?
- How to speed-up computation?

The [GLP'12] Signature Scheme

- Introduced at CHES'12 by Tim Güneysu, Vadim Lyubashevsky, Thomas Pöppelmann [GLP'12][1]
- Optimized version of [Lyu12][2]
- Based on ideal lattices (lattices with additional algebraic structure)
- Practical parameters and adapted to needs of embedded hardware (no Gaussian sampling)

[1] Practical lattice-based cryptography: A signature scheme for embedded systems, Tim Güneysu, Vadim Lyubashevsky, Thomas Pöppelmann, CHES 2012

[2] Lattice signatures without trapdoors, Vadim Lyubashevsky, Eurocrypt 2012

Notation

- n is a power of 2
- p is a prime congruent to 1 modulo $2n$ (necessary for efficiency)
- $\mathcal{R}$ is the ring $\mathbb{Z}_p[x]/\langle f = x^n + 1\rangle$
- $\mathcal{R}_k$ subset of $\mathcal{R}$ with coefficients $[-k, k]$.

Hardness Assumptions

Standard lattice hardness assumption:

Definition (**Decisional Ring-LWE**)

Given $(\mathbf{a}_1, \mathbf{t}_1), ..., (\mathbf{a}_m, \mathbf{t}_m) \in \mathcal{R} \times \mathcal{R}$. Decide whether $\mathbf{t}_i = \mathbf{a}_i\mathbf{s} + \mathbf{e}_i$ where $\mathbf{s}, \mathbf{e}_1, ..., \mathbf{e}_m \leftarrow D_\sigma$ and $\mathbf{a} \xleftarrow{\$} \mathcal{R}$ or uniformly random from $\mathcal{R} \times \mathcal{R}$ (D_σ denotes a Gaussian distribution).

Hardness Assumptions

Standard lattice hardness assumption:

Definition (**Decisional Ring-LWE**)

Given $(\mathbf{a}_1, \mathbf{t}_1), ..., (\mathbf{a}_m, \mathbf{t}_m) \in \mathcal{R} \times \mathcal{R}$. Decide whether $\mathbf{t}_i = \mathbf{a}_i\mathbf{s} + \mathbf{e}_i$ where $\mathbf{s}, \mathbf{e}_1, ..., \mathbf{e}_m \leftarrow D_\sigma$ and $\mathbf{a} \xleftarrow{\$} \mathcal{R}$ or uniformly random from $\mathcal{R} \times \mathcal{R}$ (D_σ denotes a Gaussian distribution).

More "aggressive" hardness assumption:

Definition (**Decisional Compact Knapsack Problem**)

Given $(\mathbf{a}, \mathbf{t}) \in \mathcal{R} \times \mathcal{R}$. Decide whether$(\mathbf{a}, \mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2)$ where $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$ and $\mathbf{a} \xleftarrow{\$} \mathcal{R}$ or uniformly random from $\mathcal{R} \times \mathcal{R}$.

Simple Signatures Scheme

crypto_sign_keypair()

Comments:

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Comments:

Secret key is two random polynomials $\mathbf{s}_1, \mathbf{s}_2$ with $-1/0/1$ coefficients

Simple Signatures Scheme

crypto_sign_keypair()
Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$
Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

Comments:

Extracting the secret key from the public key is exactly solving the Search Compact Knapsack problem (SCK) and $\mathbf{a}$ is a global constant

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign($\mu, \mathbf{s}_1, \mathbf{s}_2$)

Comments:

The message to be signed is μ

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign($\mu, \mathbf{s}_1, \mathbf{s}_2$)

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$

Comments:

Pick two "masking values" $\mathbf{y}_1, \mathbf{y}_2$ from $\mathcal{R}_k$ with coefficients somewhat smaller than p

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign($\mu, \mathbf{s}_1, \mathbf{s}_2$)

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$

2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$

Comments:

Bind "nonce" $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ to the message by hashing. Use 160 bit hash output and transform it into a polynomial $\mathbf{c}$ with only 32 coefficients being either -1 or 1

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign($\mu, \mathbf{s}_1, \mathbf{s}_2$)

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$

Comments:

Compute $\mathbf{z}_1$ and $\mathbf{z}_2$ by multiplying the sparse and small polynomial $\mathbf{c}$ by the small (but not sparse) secret key $\mathbf{s}$. Add the masking values

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1

Comments:

Rejection sampling step testing whether $\mathbf{z}_1, \mathbf{z}_2$ in the range $\mathcal{R}_{k-32}$

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1
5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

Comments:

Signature is $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1
5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

crypto_sign_open$(\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{t})$

Comments:

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathrm{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1
5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

crypto_sign_open$(\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{t})$

1: Accept iff
$\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{R}_{k-32}$

Comments:

Check size of coefficients to prevent attack

Simple Signatures Scheme

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathsf{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1
5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

crypto_sign_open$(\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{t})$

1: Accept iff
$\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{R}_{k-32}$ and
$\mathbf{c} = \mathsf{H}(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$

Comments:

Correctness:

$$\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} = \mathbf{a}\mathbf{s}_1\mathbf{c} + \mathbf{a}\mathbf{y}_1 + \mathbf{s}_2\mathbf{c} + \mathbf{y}_2 - \mathbf{a}\mathbf{s}_1\mathbf{c} - \mathbf{s}_2\mathbf{c} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$$

Parameters

Implemented parameter set:

- $n = 512, p = 8383489$ (23-bit), $k = 2^{14}$
- $|sig| = 8950$ bits
- $|sk| = 1620$ bits
- $|pk| = 11800$ bits
- Parameter **a** is chosen as global constant and not included into public key size
- Security: ~~100~~ 80 bits (new Crypto 2013 result)
- Parameters are chosen with the root-Hermite factor methodology
  - Solve underlying lattice problem (80 bits)
  - Find preimage in the hash function (output size of 160 bits) - quantum computer
- On average 7 attempts needed (rejection sampling)

Most Expensive Operations

crypto_sign_keypair()

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1$

Verification Key: $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

crypto_sign$(\mu, \mathbf{s}_1, \mathbf{s}_2)$

1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
2: $\mathbf{c} \leftarrow \mathrm{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}$, goto step 1
5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

crypto_sign_open$(\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{t})$

1: Accept iff
$\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{R}_{k-32}$ and
$\mathbf{c} = \mathrm{H}(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$

- Sampling: Sampling of $2n = 1024$ uniformly random values from range $[-k, k]$
- Polynomial multiplication: Multiplication of 512×512-coefficient polynomials (sparse and non-sparse)

High Level Optimizations

High Level Optimization - Random Polynomials

Uniformly random sampling of $\mathbf{y}_1$ and $\mathbf{y}_2$ from $\mathcal{R}_k$:

- Generate random bytes using Salsa20 stream cipher (`/dev/urandom` just for the seed)
- We need rejection sampling to achieve uniform distribution
  - For one random polynomial sample $n + 16$ unsigned 32-bit integers
  - Discard coefficient r_i if $r_i \geq (2k+1) \cdot \lfloor 2^{32}/(2k+1) \rfloor$
  - Compute $c_i = (r_i \mod (2k+1)) - k$
  - Probability to discard a coefficient is $4/2^{32} = 2^{-30}$

High Level Optimization - Polynomial Multiplication

Computation of $\mathbf{ay}_1, \mathbf{s}_1\mathbf{c}, \mathbf{s}_2\mathbf{c}, \mathbf{az}_1, \mathbf{tc}$:

- Expensive - polynomials have 512 coefficients
- Schoolbook multiplier has complexity $O(n^2)$ and requires $n^2 = 262144$ multiplications
- Number Theoretic Transform (NTT) has complexity $O(n \log n)$
- NTT is simplified an FFT in $\mathbb{Z}_p$
- For $\mathbf{a}, \mathbf{b}, \mathbf{d} \in \mathcal{R}$ the multiplication $\mathbf{d} = \mathbf{a} \cdot \mathbf{b}$ corresponds to the negative wrapped convolution (modulus $x^n + 1$)

High Level Optimization - Polynomial Multiplication

Theorem (Wrapped Convolution)

Let ω be a primitive n-th root of unity in $\mathbb{Z}_p$ and $\psi^2 = \omega$.

1. *Let d be the negative wrapped convolution of a and b. Let $\bar{a}, \bar{b}$ and $\bar{d}$ be defined as $(a_0, \psi a_1, ..., \psi^{n-1} a_{n-1})$, $(b_0, \psi b_1, ..., \psi^{n-1} b_{n-1})$, and $(d_0, \psi d_1, ..., \psi^{n-1} d_{n-1})$. Then $\bar{d} = NTT_w^{-1}(NTT_w(\bar{a}) \circ NTT_w(\bar{b}))$.*

Advantages:

- Reduction by $x^n + 1$ for free and no zero padding
- Store constants (e.g., a) in NTT representation
- Only $\frac{1}{2} n \log n$ multiplications for one NTT
- Constant time

Disadvantage:

- Storage for powers of $\omega, \psi, \omega^{-1}, \psi^{-1}$

Low Level Optimizations

Low Level Optimization - SIMD/AVX

Modern processors are equipped with powerful vector engines:

- Allows Single Instruction Multiple-Data (SIMD) operations
- Supported by Advanced Vector Extensions (AVX) extending the x86 instruction set
- Included in Intel Sandy Bridge, Intel Ivy Bridge, and AMD Bulldozer

Low Level Optimization - SIMD/AVX

Modern processors are equipped with powerful vector engines:

- Allows Single Instruction Multiple-Data (SIMD) operations
- Supported by Advanced Vector Extensions (AVX) extending the x86 instruction set
- Included in Intel Sandy Bridge, Intel Ivy Bridge, and AMD Bulldozer

General idea:

- Represent 512-coefficient polynomial as array of 512 double-precision floating-point values
- 4 double-precision floats fit into the 256-bit-wide AVX `ymm` vector registers
- Perform operations on four coefficients at the same time (e.g., addition, reduction)
- For example AVX supports one double-precision-vector multiplication and one addition every cycle

Low Level Optimization - Modular Reduction and Addition

Parallel modular reduction:

- Modular reduction mod p extremely important
- Multiply x by inverse $c = x \cdot \overline{p^{-1}}$ (`vmulpd`)
- Round c, multiply c by p and then subtract c from x (`vroundpd`/`vsubpd`)
- Work on four coefficients in parallel
- *Lazy reduction:* Reduce only when necessary (p has 23 bits, mantissa has 53-bit)

Parallel addition:

- Just requires 256 vector loads, 128 vector additions or subtractions, and 128 vector stores

Low Level Optimization - Optimizing the NTT

- NTT is most speed-critical operation
- Adapted standard fast iterative algorithm
- $\log_2 n = 9$ levels of operations
- Additions (and subtractions) are a bottleneck
- Merging of levels to reduce load and stores

Results

Results

Cycle counts for Intel Ivy Bridge:

Operation	**Cycles**	**Op/s (@2 GHz)**
crypto_sign_keypair	31140	64226
crypto_sign	634988	3149
crypto_sign_open	45036	44408
ntt	4484	446030
poly_mul	16096	124254
poly_mul_a	11044	181093
poly_setrandom_maxk	10824	184774
poly_setrandom_max1	5464	366032

- Signing attempt takes 85384 cycles
- On average 7 attempts: $7 \cdot 85384 = 597688$ cycles + overhead
- NTT is even advantageous for sparse multiplication
- Timing variation is independent of secret data - protection against timing attacks

Comparison - ECC/RSA

Note that the implementation is now in eBACS benchmarking framework (`lattisigns512`).

Software	**Security**	**Cycles/s**		**Sizes**	
Our work	80 bits	**sign:**	634988	**pk:**	1536
		verify:	45036	**sk:**	256
				sig:	1184
ed25519	128 bits	**sign:**	67564	**pk:**	32
		verify:	209328	**sk:**	64
				sig:	64
ronald2048 (RSA-2048)	112 bits	**sign:**	5768360	**pk:**	256
		verify:	77032	**sk:**	2048
				sig:	256

Comparison - PQ

Software	Security	Cycles/s		Sizes	
Our work	80 bits	**sign:**	634988	**pk:**	1536
		verify:	45036	**sk:**	256
				sig:	1184
XMSS	82 bits	**sign:**	7261100	**pk:**	912
		verify:	556600	**sk:**	19
				sig:	2451
mqqsig160	80 bits	**sign:**	1996	**pk:**	206112
		verify:	33220	**sk:**	401
				sig:	20
rainbow binary16242020	80 bits	**sign:**	29364	**pk:**	102912
		verify:	17900	**sk:**	94384
				sig:	40

Lessons learned and outlook

Lessons learned:

- (Ideal) lattice-based cryptography is fast and parallelizable
- Results are applicable to other schemes (e.g., LWE-encryption)
- For high speed do not rely on high level libraries (e.g., NTL)

Lessons learned and outlook

Lessons learned:

- (Ideal) lattice-based cryptography is fast and parallelizable
- Results are applicable to other schemes (e.g., LWE-encryption)
- For high speed do not rely on high level libraries (e.g., NTL)

Outlook:

- *Lattice Signatures and Bimodal Gaussians*, Leo Ducas and Alain Durmus and Tancrede Lepoint and Vadim Lyubashevsky, Crypto 2013, to appear
  - Signature size of 5600 bit providing 128 bit security
  - Analysis suggests only 80-bit security for implemented scheme ([GLP'12] claim was 100)
- Different architectures, e.g., GPU or ARM with NEON
- Application of results to other schemes (e.g., LWE-encryption, homomorphic, IBE)

Thank you for your attention!
Any questions?

Results are online:
`http://cryptojedi.org/crypto/index.shtml#lattisigns`

Backup

Security Proof

Sketch of the security proof:

- Choose invalid public key $(a, t = as_1' + s_2')$ with $s_{1,2}'$ in R_k' with sufficiently large k'. Invalid key is indistinguishable from valid key.
- Deliver signatures by programming the ROM (without knowing the private key)
- When adversary produces forgery (z_1, z_2, c), we can produce a second forgery (z_1', z_2', c) by the Forking Lemma
- Therefore it holds that $H(az_1 + z_2 - tc, m) = H(az_1' + z_2' - tc, m)$. This allows us to extract small u_1, u_2 such that $au_1 + u_2 = 0$ which allows us to solve the DCK problem