***K**riptografi **A**tasi **Z**arah* Digital Signature (KAZ-SIGN)

Algorithm Specifications and Supporting Documentation

(Version 1.6)

Muhammad Rezal Kamel Ariffin[1] Nur Azman Abu[2] Terry Lau Shue Chien[3]
Zahari Mahad[1] Liaw Man Cheon[4] Amir Hamzah Abd Ghafar[1]
Nurul Amiera Sakinah Abdul Jamal[1] Vaishnavi A/P Nagaraja[1]

[1]Institute for Mathematical Research, Universiti Putra Malaysia
[2]Faculty of Information & Communication Technology, Universiti Teknikal Malaysia Melaka
[3]Faculty of Computing & Informatics, Multimedia University Malaysia
[4]Antrapolation Technology Sdn. Bhd., Selangor, Malaysia

Table of Contents

Name of the proposed cryptosystem:	KAZ-SIGN
Principal submitter:	Muhammad Rezal Kamel Ariffin Institute for Mathematical Research Universiti Putra Malaysia 43400 UPM Serdang, Selangor Malaysia Email: rezal@upm.edu.my Phone: +60123766494
Auxilliary submitters:	Nor Azman Abu Terry Lau Shue Chien Zahari Mahad Liaw Man Cheon Amir Hamzah Abd Ghafar Nurul Amiera Sakinah Abdul Jamal Vaishnavi A/P Nagaraja
Inventor of the cryptosystem:	Muhammad Rezal Kamel Ariffin
Owner of the cryptosystem:	Muhammad Rezal Kamel Ariffin
Alternative point of contact:	Amir Hamzah Abd Ghafar Institute for Mathematical Research Universiti Putra Malaysia 43400 UPM Serdang, Selangor Malaysia Email: amir_hamzah@upm.edu.my Phone: +60132723347

1. INTRODUCTION

The proposed KAZ Digital Signature scheme, KAZ-SIGN (in Malay *Kriptografi Atasi Zarah* - translated literally "cryptographic techniques overcoming particles"; particles here referring to the photons) is built upon the hard mathematical problem coined as the Modular Reduction Problem (MRP). The idea revolves around the difficulty of reconstructing an unknown parameter from a given modular reducted value of that parameter. The target of the KAZ-SIGN design is to be a quantum resistant digital signature candidate with short verification keys and signatures, verifying correctly approximately 100% of the time, based on simple mathematics, having fast execution time and a potential candidate for seamless drop-in replacement in current cryptographic software and hardware ecosystems.

2. THE DESIGN IDEALISME

(i) To be based upon a problem that could be proven analytically to require exponential time to be solved;

(ii) To be able to prove analytically that the cryptosystem is indeed resistant towards quantum computers;

(iii) To utilize problems mentioned in point (i) above in its full spectrum without having to induce "weaknesses" in order for a trapdoor to be constructed;

(iv) To use "simple" mathematics in order to achieve maximum simplicity in design, such that even practitioners with limited mathematical background will be able to understand the arithmetic;

(v) Achieve 128 and 256-bit security with key length roughly equivalent to the non-quantum secure Elliptic Curve Cryptosystem (ECC);

(vi) To achieve maximum speed upon having simplicity in design and short key length;

(vii) To have a sufficiently large signature space;

(viii) The computation overhead for both signing and verification increases slightly even if the key size increases in the future;

(ix) To be able to be mounted on hardware with ease;

(x) The plaintext to signature expansion ratio is kept to a minimum.

One of our key strategy to obtain items (i) - (v) was by utilizing our defined Modular Reduction Problem (MRP). It is defined in the following section.

3. MODULAR REDUCTION PROBLEM (MRP)

Let $N = \prod_{i=1}^{j} p_i$ be a composite number and $n = \ell(N)$. Let p_k be a factor of N. Choose $\alpha \in (2^{n-1}, N)$. Compute $A \equiv \alpha \pmod{p_k}$.

The MRP is, upon given the values (A, N, p_k), one is tasked to determine $\alpha \in (2^{n-1}, N)$.

4. COMPLEXITY OF SOLVING THE MRP

Let $n_{p_k} = \ell(p_k)$ be the bit length of p_k. The complexity to obtain α is $O(2^{n-n_{p_k}})$. When deploying Grover's algorithm on a quantum computer, the complexity to obtain α is $O(2^{\frac{n-n_{p_k}}{2}})$. In other words, if $p_k \approx N^{\delta}$, for some $\delta \in (0,1)$, the complexity to obtain α is $O(N^{1-\delta})$. When deploying Grover's algorithm on a quantum computer, the complexity to obtain α is $O(N^{\frac{1-\delta}{2}})$.

5. THE HIDDEN NUMBER PROBLEM (HNP) (Boneh and Venkatesan, 2001)

Fix p and u. Let $O_{\alpha,g}(x)$ be an oracle that upon input x computes the most u significant bits of $\alpha g^x \pmod{p}$. The task is to compute the hidden number $\alpha \pmod{p}$ in expected polynomial time when one is given access to the oracle $O_{\alpha,g}(x)$. Clearly, one wishes to solve the problem with as small u as possible. Boneh and Venkatesan (2001) demonstrated that a bounded number of most significant bits of a shared secret are as hard to compute as the entire secret itself.

The initial idea of introducing the HNP is to show that finding the u most significant bits of the shared key in the Diffie-Hellman key exchange using users public key is equivalent to computing the entire shared secret key itself.

6. THE HERMANN MAY REMARKS (Herrmann and May, 2008)

We will now observe two remarks by Herrmann and May. It discusses the ability and inability to retrieve variables from a given modular multivariate linear equation. But before that we will put forward a famous theorem of Minkowski that relates the length of the shortest vector in a lattice to the determinant (see Hoffstein et al. (2008)).

Theorem 1. *In an ω-dimensional lattice, there exists a non-zero vector v with*

$$\|v\| \leq \sqrt{\omega}\, det(L)^{\frac{1}{\omega}}$$

In lattices with fixed dimension we can efficiently find a shortest vector, but for arbitrary dimensions, the problem of computing a shortest vector is known to be NP-hard under randomized reductions (see Ajtai (1998)). The LLL algorithm, however, computes in polynomial time an approximation of the shortest vector, which is sufficient for many applications.

Remark 1. *Let $f(x_1,x_2,\ldots,x_k) = a_1x_1 + a_2x_2 + \ldots + a_kx_k$ be a linear polynomial. One can hope to solve the modular linear equation $f(x_1,x_2,\ldots,x_k) \equiv 0 \pmod N$, that is to be able to find the set of solutions $(y_1,y_2,\ldots,y_k) \in \mathbb{Z}_N^k$, when the product of the unknowns are smaller than the modulus. More precisely, let X_i be upper bounds such that $|y_i| \leq X_i$ for $1,\ldots,k$. Then one can roughly expect a unique solution whenever the condition $\prod_i X_i \leq N$ holds (see Herrmann and May (2008)). It is common knowledge that under the same condition $\prod_i X_i \leq N$ the unique solution $(y_1,y_2,\ldots,y_k)$ can heuristically be recovered by computing the shortest vector in an k-dimensional lattice by the LLL algorithm. In fact, this approach lies at the heart of many cryptographic results (see Bleichenbacher and May (2006); Girault et al. (1990) and Nguyen (2004)).*

We would like to provide the reader with the conjecture and remark given in Herrmann and May (2008).

Conjecture 1. *If in turn we have $\prod_i X_i \geq N^{1+\varepsilon}$ then the linear equation $f(x_1,x_2,\ldots,x_k) = \sum_{i=1}^k a_ix_i \equiv 0 \pmod N$ usually has N^ε many solutions, which is exponential in the bit-size of N.*

Remark 2. *From Conjecture 1, there is hardly a chance to find efficient algorithms that in general improve on this bound, since one cannot even output all roots in polynomial time.*

7. THE KAZ-SIGN DIGITAL SIGNATURE ALGORITHM

7.1 Background

This section discusses the construction of the KAZ-SIGN scheme. We provide information regarding the key generation, signing and verification procedures. But first, we will put forward functions that we will utilize and the system parameters for all users.

7.2 Utilized Functions

Let $H(\cdot)$ be a hash function. Let $\phi(\cdot)$ be the usual Euler-totient function. Let $\ell(\cdot)$ be the function that outputs the bit length of a given input.

7.3 System Parameters

From the given security parameter k, determine parameter j. Next generate a list of the first j-primes larger than 2, $P = \{p_i\}_{i=1}^j$. Let $N = \prod_{i=1}^j p_i$. As an example, if $j = 43$, N is 256-bits. Let $n = \ell(N)$ be the bit length of N. Choose a random prime in $g \in \mathbb{Z}_N$ of order G_g where at most $G_g \approx N^\delta$ for a chosen value of $\delta \in (0,1)$ and $\delta \to 0$. That is, $g^{G_g} \equiv 1 \pmod N$. Choose a random prime $R \in \mathbb{Z}_{\phi(N)}$ of order G_R, where $G_R \approx \phi(N)^\varepsilon$

for $\varepsilon \to 1$. That is, choose R with a large order in $\mathbb{Z}_{\phi(N)}$. Let $n_{G_R} = \ell(G_R)$ be the bit length of G_R. Such R, has its own natural order in $Z_{\phi(G_g)}$. Let that order be denoted as G_{Rg}. We can observe the natural relation given by $R^{G_{Rg}} \equiv 1 \pmod{G_g}$ where $\phi(N) \equiv 0 \pmod{G_g}$ and $\phi(G_g) \equiv 0 \pmod{G_{Rg}}$. Let $n_{\phi(G_g)} = \ell(\phi(G_g))$ be the bit length of $\phi(G_g)$ and $n_{\phi(G_{Rg})} = \ell(\phi(G_{Rg}))$ be the bit length of $\phi(G_{Rg})$. Let q be a random k-bit prime. Let $Q = \prod_{i=1}^{25} p_i = 116431182179248680450031658440253681535$. Ensure that $\phi(\phi(G_{Rg})) < \phi(\phi(Q))$. The system parameters are $(g,k,q,Q,N,R,G_g,G_{Rg},n,n_{\phi(G_g)})$.

7.4 KAZ-SIGN Algorithms

The full algorithms of KAZ-SIGN are shown in Algorithms 1, 2, and 3.

Algorithm 1 KAZ-SIGN Key Generation Algorithm

Input: System parameters $(g,k,q,Q,N,R,G_g,G_{Rg},n,n_{\phi(G_g)})$
Output: Public verification key pair, (V_1,V_2), and private signing key, α
Choose random prime a, $\alpha \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
Compute public verification key, $V_1 \equiv \alpha \pmod{G_{Rg}q}$.
Compute secret parameter $b \equiv a^{\phi(\phi(G_{Rg}))} \pmod{\phi(G_g)}$.
Compute public verification key, $V_2 \equiv Q(\alpha^{\phi(Q)b}) \pmod{qQ}$.
Output public verification keys, (V_1,V_2), keep signing key (α,b) secret and destroy a.

Algorithm 2 KAZ-SIGN Signing Algorithm

Input: System parameters $(g,k,q,Q,N,R,G_g,G_{Rg},n,n_{\phi(G_g)})$, private signing key, (α,b), and message to be signed, $m \in \mathbb{Z}_N$.
Output: Signature, S
Let $m \in \mathbb{Z}_N$ be the message to be signed and let $h = next\,prime(H(m))$.
Choose ephemeral random prime $r \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
Compute secret parameter $\beta \equiv r^{\phi(\phi(G_{Rg}))} \pmod{\phi(G_g)}$.
Compute $S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \pmod{G_{Rg}qQ}$.
Output signature, S, and destroy (β,r).

Algorithm 3 KAZ-SIGN Verification Algorithm

Input: System parameters $(g, k, q, Q, N, R, G_g, G_{Rg}, n, n_{\phi(G_g)})$, public verification key pair, (V_1, V_2), message, m, and signature, S.

Output: Accept or reject signature

1: Compute $h = nextprime(H(m))$.
2: Compute $y \equiv (V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}Q}$ and $S_{F1} = CRT([\frac{V_2}{Q}, y], [q, G_{Rg}Q])$.
3: Compute the following procedure: Set $S_{F2} = 0$. Set $modulus = 1$.
4:     $VQ \equiv (V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}}$
5:     **for** each factor $r_i^{e_i}$ of $G_{Rg}Q$ **do**
6:         **for** $soln = 0, 1, 2, \ldots, r_i^{e_i} - 1$ **do**
7:             **if** $soln$ mod $\gcd(Q, r_i^{e_i}) \not\equiv 1 \pmod{\gcd(Q, r_i^{e_i})}$ **then next; end if**
8:             **if** $soln$ mod $\gcd(G_{Rg}, r_i^{e_i}) \not\equiv VQ$ mod $\gcd(G_{Rg}, r_i^{e_i})$ **then next; end if**
9:             **if** $soln \cdot Q$ mod $\gcd(q \cdot Q, r_i^{e_i}) \not\equiv V_2$ mod $\gcd(q \cdot Q, r_i^{e_i})$ **then next; end if**
10:             **break**
11:         **end for**
12:         $S_{F2} = CRT([S_{F2}, soln], (modulus, r_i^{e_i}))$
13:         $modulus = modulus \cdot r_i^{e_i}$
14:     **end for**
15: Compute $w_0 \equiv (S \pmod{G_{Rg}qQ}) - S$.
16: **if** $w_0 \neq 0$ **then**
17:     Reject signature $\perp$
18: **else** Continue Step 20
19: **end if**
20: Compute $w_1 \equiv (S \pmod{G_{Rg}qQ}) - S_{F1}$.
21: **if** $w_1 = 0$ **then**
22:     Reject signature $\perp$
23: **else** Continue Step 25
24: **end if**
25: Compute $w_2 \equiv S - S_{F2} \pmod{q}$.
26: **if** $w_2 = 0$ **then**
27:     Reject signature $\perp$
28: **else** Continue Step 30
29: **end if**
30: Compute $w_3 \equiv QS \pmod{qQ})$. Compute $w_4 = w_3 - V_2$.
31: **if** $w_4 \neq 0$ **then**
32:     Reject signature $\perp$
33: **else** Continue Step 35
34: **end if**
35: Compute $y_1 \equiv g^{(R^S \pmod{G_g})} \pmod{N}$.
36: Compute $y_2 \equiv g^{(R^{(V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}})} \pmod{G_g})} \pmod{N}$.
37: **if** $y_1 = y_2$ **then**
38:     accept signature
39: **else** reject signature $\perp$
40: **end if**

Steps 15, 16, 17, 18, and 19, during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 1**, steps 20, 21, 22, 23, and 24 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 2**, steps 25, 26, 27, 28, and 29 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 3**, steps 30, 31, 32, 33, and 34 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 4**.

8. THE DESIGN RATIONALE

In this section we will analyse the rationale behind the design vis-à-vis a valid signature parameter S.

8.1 Proof of correctness (Verification steps 35, 36, 37, 38, 39, and 40)

We begin by discussing the rationale behind steps 35, 36, 37, 38, 39, and 40 with relation to the verification process. Observe the following,

$$\begin{aligned}
g^{(R^S \pmod{G_g})} &\equiv g^{R^{((\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \pmod{G_{Rg}})} \pmod{G_g}} \\
&\equiv g^{R^{((\alpha^{(\phi(Q))})(h^{(\phi(qQ))}) \pmod{G_{Rg}})} \pmod{G_g}} \\
&\equiv g^{(R^{((V_1^{\phi(Q)}(h^{\phi(qQ)}) \pmod{G_{Rg}})} \pmod{G_g})} \pmod{N}
\end{aligned}$$

because $\alpha \equiv V \pmod{G_{Rg}}$, $b \equiv a^{\phi(\phi(G_{Rg}))} \equiv 1 \pmod{G_{Rg}}$ and $\beta \equiv r^{\phi(\phi(G_{Rg}))} \equiv 1 \pmod{G_{Rg}}$ since $\phi(G_g) \equiv 0 \pmod{G_{Rg}}$. As such the verification process does indeed provide an indication that the signature is indeed from an authorized sender with the private signing key α.

8.2 Proof of correctness (Verification steps 15, 26, 17, 18, and 19: KAZ-SIGN digital signature forgery detection procedure type – 1)

In order to comprehend the rationale behind steps 15, 26, 17, 18, and 19, one has to observe the following,

$$w_0 \equiv (S \pmod{G_{Rg}qQ}) - S = 0$$

because $S < G_{Rg}qQ$.

8.3 Proof of correctness (Verification steps 20, 21, 22, 23, and 24: KAZ-SIGN digital signature forgery detection procedure type – 2)

In order to comprehend the rationale behind steps 20, 21, 22, 23, and 24, one has to observe the following; obviously S_{F1} is not constructed with secret parameters (α, b). As such from

$w_1 \equiv (S \pmod{G_{Rg}qQ}) - S_{F1}$, we will have $w_1 \neq 0$.

8.4 Proof of correctness (Verification steps 25, 26, 27, 28, and 29: KAZ-SIGN digital signature forgery detection procedure type – 3)

In order to comprehend the rationale behind steps 25, 26, 27, 28, and 29, one has to observe the following; obviously S_{F2} is not constructed with secret parameters (α, b). As such from $w_2 \equiv S - S_{F2} \pmod{q}$, we will have $w_2 \neq 0$.

8.5 Proof of correctness (Verification steps 30, 31, 32, 33, and 34: KAZ-SIGN digital signature forgery detection procedure type – 4)

In order to comprehend the rationale behind steps 30, 31, 32, 33, and 34, one has to observe

$$w_3 \equiv QS \equiv Q(\alpha^{\phi(Q)b}) \pmod{qQ}.$$

Hence,

$$w_4 = w_3 - V_2 = 0.$$

8.6 Deriving forged signature identifiable by KAZ-SIGN digital signature forgery detection procedure type – 1.

An adversary utilizing a valid signature, S and resends it as follows:

$$S_{F0} \equiv S + G_{Rg}qQx \pmod{\theta G_{Rg}qQ}$$

for some random value of $x \in \mathbb{Z}$ and small value of $\theta \in \mathbb{Z}$, such that $\ell(S_{F0}) \approx \ell(S)$. That is, $\ell(S_{F0})$ is not suspicious to the verifier. It is easy to observe that S_{F0} will pass steps 35, 36, 37, 38, 38, 39, and 40. However, since

$$w_0 \equiv (S_{F0} \pmod{G_{Rg}qQ}) - S_{F0} \neq 0 \in \mathbb{Z}$$

the signature will fail KAZ-SIGN digital signature forgery detection procedure type – 1.

8.7 Deriving forged signature identifiable by KAZ-SIGN digital signature forgery detection procedure type – 2

An adversary that constructs a forged signature S as follows; compute $y \equiv (V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}Q}$ and $S = CRT([\frac{V_2}{Q}, y], [q, G_{Rg}Q])$, and then transmits it as a signature S would result in

$$w_1 \equiv (S \pmod{G_{Rg}Q}) - S_{F1} = 0.$$

It is easy to observe that S will pass steps 35, 36, 37, 38, 38, 39, and 40. However, the signature will fail KAZ-SIGN digital signature forgery detection procedure type - 2.

8.8 Deriving forged signature identifiable by KAZ-SIGN digital signature forgery detection procedure type – 3

An adversary that constructs a forged signature S as described in steps 3-14 within the verification algorithm, and then transmits it as a signature S would result in

$$w_2 \equiv S - S_{F2} \pmod q = 0.$$

It is easy to observe that S will pass steps 35, 36, 37, 38, 38, 39, and 40. However, the signature will fail KAZ-SIGN digital signature forgery detection procedure type - 3.

8.9 Deriving forged signature identifiable by KAZ-SIGN digital signature forgery detection procedure type – 4

An adversary that constructs a forged signature S without the private key α; and at the same time aspires to pass steps 35, 36, 37, 38, 38, 39, and 40 would result in having to utilize the relation

$$S \equiv (\lambda^{\phi(Q)}) \pmod{qQ}$$

where $\lambda = V_1 + G_{Rg}qt$ is a prime for some $t \in \mathbb{Z}$ and t is not the solution for the MRP$(V_1, \alpha, G_{Rg}q)$. It is clear that $\alpha \not\equiv V_1 + G_{Rg}qt \pmod{qQ}$. As such, $w_4 = w_3 - V_2 \neq 0$, where $w_3 \equiv Q(\lambda^{\phi(Q)}) \pmod{qQ}$. Thus, the signature will fail KAZ-SIGN digital signature forgery detection procedure type – 4.

8.10 Extracting α

An approach to forge the signature would be to produce either one of the following:

1. $y_{\alpha 1} \equiv \alpha \pmod{G_{Rg}qQ}$ OR
2. $y_{\alpha 2} \equiv \alpha^{\phi(Q)} \pmod{G_{Rg}qQ}$.

8.10.1 Producing $y_{\alpha 1} \equiv \alpha \pmod{G_{Rg}qQ}$

From the public parameter $V_1 \equiv \alpha \pmod{G_{Rg}q}$ and adversary can produce:

1. $\alpha \pmod{G_{Rg}q}$
2. $\alpha \pmod{G_{Rg}}$
3. $\alpha \pmod q$.

Thus, the adversary needs to obtain the corresponding parameters to execute the Chinese Remainder Theorem (CRT) to obtain $\alpha \pmod{G_{Rg}qQ}$.

1. $\alpha \pmod{Q} \rightarrow$ to execute CRT with $\alpha \pmod{G_{Rg}q}$
2. $\alpha \pmod{qQ} \rightarrow$ to execute CRT with $\alpha \pmod{G_{Rg}}$
3. $\alpha \pmod{G_{Rg}Q} \rightarrow$ to execute CRT with $\alpha \pmod{q}$.

8.10.1.1 To obtain $\alpha \pmod{Q}$

To obtain $\alpha \pmod{Q}$, the adversary will utilize equation S. Observe that

$$S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \equiv 1 \not\equiv \alpha \pmod{Q}.$$

Thus, this option is not viable.

8.10.1.2 To obtain $\alpha \pmod{qQ}$

To obtain $\alpha \pmod{qQ}$, the adversary will utilize equation S. Observe that

$$S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \equiv \alpha^{\phi(Q)b} \not\equiv \alpha \pmod{qQ}.$$

Thus, this option is not viable.

8.10.1.3 To obtain $\alpha \pmod{G_{Rg}Q}$

To obtain $\alpha \pmod{G_{Rg}Q}$, the adversary will utilize equation S. Observe that

$$S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \not\equiv \alpha \pmod{G_{Rg}Q}.$$

Thus, this option is not viable.

8.10.2 Producing $y_{\alpha 2} \equiv \alpha^{\phi(Q)} \pmod{G_{Rg}qQ}$

To obtain $y_{\alpha 2}$, one begins with,

$$z_1 \equiv V_1^{\phi(Q)} \equiv \alpha^{\phi(Q)} \pmod{q}.$$

Then, one needs to produce the parameter $\alpha^{\phi(Q)} \pmod{G_{Rg}Q}$. However,

$$z_2 \equiv S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \not\equiv \alpha^{\phi(Q)} \pmod{G_{Rg}Q}.$$

Thus, with the available parameters (S, V_1), one is unable to produce $y_{\alpha 2}$.

Another direction would be to observe

$$z_2 \equiv G_{Rg}V_2 \equiv G_{Rg}Q\alpha^{\phi(Q)b} \not\equiv \alpha^{\phi(Q)} \pmod{G_{Rg}Q}.$$

This direction, with the available (V_2, S) would also not be able to produce $y_{\alpha 2}$.

8.11 Modular linear equation of S

In this direction we analyze

$$S \equiv (\alpha^{(\phi(Q)b)})(h^{(\phi(qQ)\beta)}) \pmod{G_{Rg}qQ}$$

Let

1. $X_1 \equiv \alpha^{\phi(Q)b} \pmod{G_{Rg}qQ}$
2. $X_2 \equiv h^{\phi(qQ)\beta} \pmod{G_{Rg}qQ}$

Moving forward we have,

$$X_1X_2 - S \equiv 0 \pmod{G_{Rg}qQ} \tag{1}$$

Let $\hat{X}_1$ be the upper bound for X_1 and $\hat{X}_2$ be the upper bound for X_2. From Conjecture 1, if one has the situation where $\hat{X}_1\hat{X}_2 \gg G_{Rg}qQ$, then there is no efficient algorithm to output all the roots of (1). That is, (1) usually has $G_{Rg}qQ$ many solutions, which is exponential in the bit-size of $G_{Rg}qQ$.

To this end, since both $\alpha^{\phi(Q)b}$ and $h^{\phi(qQ)\beta}$ are exponentially large, it is clear to conclude that $\hat{X}_1\hat{X}_2 \gg G_{Rg}qQ$. This implies, there is no efficient algorithm to output all the roots of (1).

8.12 Implementation of the Hidden Number Problem (HNP)

From S, let us denote as follows:

1. $x_1 \equiv \alpha^{\phi(Q)b} \pmod{G_{Rg}qQ}$
2. $x_2 \equiv \phi(qQ)\beta$

Thus, S can be re-written as

$$S \equiv (x_1)(h^{x_2}) \pmod{G_{Rg}qQ} \tag{2}$$

for unknown pair (x_1, x_2). It is obvious that (2) is the HNP.

9. SPECIFICATION OF KAZ-SIGN

The challenge faced by the adversary is to retrieve α from $V_1 \equiv \alpha \pmod{G_{Rg}q}$. It is protected by the MRP. The MRP representation is given as follows:

$$t = \frac{\alpha - V_1}{G_{Rg}q}$$

Due to the strategies during key generation, we have the complexity $O(t) = O(q)$.

As such, the complexity of solving the MRP via $V_1 \equiv \alpha \pmod{G_{Rg}q}$ will be the determining factor in identifying the suitable key length for each security level.

The following is the security specification for $\delta = 0.23$.

Number of primes in P	$\ell(q)$	$n = \ell(N)$	Total security level, k
180	134	1509	128
258	198	2321	192
342	264	3241	256

Table 1

10. IMPLEMENTATION AND PERFORMANCE

10.1 Key Generation, Signing and Verification Time Complexity

It is obvious that the time complexity for all three procedures is in polynomial time.

10.2 Parameter sizes

We provide here information on size of the key and signature based on security level information from Table 1 (for $\delta = 0.23$).

NIST Security Level	Number of primes in P	Security level, k	Length of parameter N (bits)	Public key size, (V_1, V_2) (bits)	Private key size, α (bits)	Signature Size (S) (bits)	ECC key size (bits)
1	180	128	1509	≈ 440	≈ 352	≈ 350	256
3	258	192	2321	≈ 605	≈ 530	≈ 465	384
5	342	256	3241	≈ 750	≈ 700	≈ 570	521

Table 2

In the direction of the research, we also make comparison to ECC key length for the three NIST security levels. KAZ-SIGN key length did not achieve its immediate target of having approximately the same key length as ECC, but further research might find means and ways.

10.3 Key Generation, Signing and Verification Ease of Implementation

The algebraic structure of KAZ-SIGN has an abundance of programming libraries available to be utilized. Among them are:

1. GNU Multiple Precision Arithmetic Library (GMP); and
2. Standard C libraries.

10.4 Key Generation, Signing and Verification Empirical Performance Data

In order to obtain benchmarks, we evaluate our reference implementation on a machine using GCC Compiler Version 6.3.0 (MinGW.org GCC-6.3.0-1) on Windows 10 Pro, Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz and 8.00 GB RAM (64-bit operating system, x64-based processor).

We have the following empirical results when conducting 100 key generations, 100 signings and 100 verifications:

Security level	Time (ms)		
	Key generation	Signing	Verification
128 - KAZ1509	438	281	328
192 - KAZ2321	797	719	703
256 - KAZ3241	1938	1031	1391

Table 3

11. ADVANTAGES AND LIMITATIONS

As we have seen, KAZ-SIGN can be evaluated through:

1. Key length
2. Speed
3. No verification failure

11.1 Key Length

KAZ-SIGN key length is comparable to non-post quantum algorithms such as ECC and RSA. For 256-bit security, the KAZ-SIGN key size is approximate 750-bits. ECC would use 521-bit keys and RSA would use 15360-bit keys.

11.2 Speed

KAZ-SIGN's speed analysis results stem from the fact that it has short key length to achieve 256-bit security plus its textbook complexity running time for both signing and verifying is $O(n^3)$ where parameter n here is the input length.

11.3 No verification failure

It is apparent that the execution of **KAZ-SIGN parameter suitability detection procedure** together with **KAZ-SIGN digital signature forgery detection procedure type – 1, type – 2, type – 3, and type – 4** within the verification procedure will enable the verification computational process by the recipient to verify or reject a digital signature that was received by the recipient with probability equal to 1. That is, the probability of verification failure is 0.

11.4 Limitation

As we have seen, limitation of KAZ-SIGN can be evaluated through:

1. Based on unknown problem, the Modular Reduction Problem (MRP)

11.4.1 Based on unknown problem, the Modular Reduction Problem (MRP)

The MRP is not a known hard mathematical problem which is quantum resistant and is subject to future cryptanalysis success in solving the defined challenge either with a classical or quantum computer.

12. CLOSING REMARKS

The KAZ-SIGN digital signature exhibits properties that might result in it being a desirable post quantum signature scheme. In the event that new forgery methodologies are found, as long as the procedure can also be done by the verifier, then one can add the new forgery methodology into the verification procedure. At the same time, the same forgery methodology can be inserted into the signing procedure in order to eliminate any chances the signer will produce a signature that will be rejected.

To this end, the security of the MRP is an unknown fact. We opine that, the acceptance of MRP as a potential quantum resistant hard mathematical problem will come hand in hand with a secure cryptosystem designed upon it. We welcome all comments on the KAZ-SIGN digital signature, either findings that nullify its suitability as a post quantum digital signature scheme or findings that could enhance its deployment and use case in the future.

Finally, we would like to put forward our heartfelt thanks to Prof. Dr. Abderrahmane Nitaj from Laboratoire de Mathématiques Nicolas Oresme, Université de Caen Basse Normandie, France for insights, comments, and friendship throughout the process. Next, special thanks to Prof. Dr. Daniel J. Bernstein from University of Illinois at Chicago, United States of America who has given his thoughts and efforts throughout versions 1.0 until 1.5β.2 of KAZ-SIGN which lead towards version 1.5 being announced. Today, our participation in this NIST exercise has lead us towards new collaborations. We would like to thank discussion opportunities with Kai Chieh Chang (Jay) and the team at Phison Architecture Design Department which triggered discussions that lead towards version 1.6, which resulted in reduced number of steps for KAZ-SIGN key gen, sign and verify algorithms.

13. ILLUSTRATIVE FULL SIZE TEST VECTORS – 1

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a valid KAZ-SIGN signature S. The valid KAZ-SIGN signature will pass all 4 KAZ-SIGN digital signature forgery detection procedure types.

N :
1665409992402569056088099162882616633362634244067356501888501184798944673390411604901732676624210376510769252181354174828223286340057028944019913396694146511184563726950707696198631319714142415860488628031406604720665322220735346993365959753415679244320546140681916938894958694783504509315984550444746877596669802184487731229941008215513808488975493742420953323598722589641742694189807070615662303109862713346329626534198736305288472594133321899608520755 $\approx 2^{1509}$

g :
6007

G_g :
66425249147392035103359575563682919206231140688573787652572381678879876350990985890249087277450456295776000 $\approx 2^{355} \approx 2^{0.235(1509)} \approx N^{0.235}$

R :
6151

G_{Rg} :
964284630129748924872876000 $\approx 2^{90} \approx N^{0.059}$

q :
18206603144869985452951603889167263698321

Q :
116431182179248680450031658440253681535

Key generation

α :
313886091767574972903718880094474567029275754185116197555190628845429405962348501671219343303278467038948 9

V_1 :
67168802080980751204385976782636900452151669616894408269695246494 89

a :
31124469535369889998760669402905941965381575875722313979988618293305109648860101
7795034148856729047612231 47

b :
38008181542021124523527095378283058542111770662857075921113373906226298378711388
10434856176185874841600001

V_2 :
412146251595839294135076300117086199254327735635171214834836626851178755173950

MRP complexity upon t

$t = \dfrac{\alpha - V_1}{G_{Rg} q}$:
178787808994295008768054827105930283065 $\approx 2^{128}$

Signing

h :
115760211758497538494681238275199133241749118140161026323854746247503823966177

r :
20076204133860574052715906836441534409715361631868200238417964000871425655152540
8142126629776279869405271 83

β :
23773746421608517633363078233668147526240430764602693720937944239610258967130543
7326719016839979991040000 1

S :
50269022193093585855417540036563114031198834186914551759148989779047194598444970
1989407269738968160560001

Verification

KAZ-SIGN digital signature forgery detection procedure type – 1

w_0 :
0

KAZ-SIGN digital signature forgery detection procedure type – 2

S_{F1} :
38682952988576219150640551409543845283304109692563701095106941328333662408761955
7433837457808533303140001

w_1 :
11586069204517366704776988627019268747894724494350850664042048450713532189683014
4555569811930434857420000 $\neq 0$

KAZ-SIGN digital signature forgery detection procedure type – 3

S_{F2} :
4641445340113508875934918466907165790901113131988036046872796 0001

w_2 :
1220405228505443354304138116763711770168 6 $\neq 0$

KAZ-SIGN digital signature forgery detection procedure type – 4

w_3 :
41214625159583929413507630011708619925432773563517121483483662685117875517 3950

w_4 :
0

Final verification

y_1 and y_2 :
46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
12933612525179831181176823388284533269026658118 3249793

14. ILLUSTRATIVE FULL SIZE TEST VECTORS – 2

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and $S \equiv S_V + G_{Rg}qQ$ where S_V is a valid signature as in illustrative full size test vectors – 1. This signature will fail the <u>**KAZ-SIGN digital signature forgery detection procedure type – 1**</u>.

S_V :
50269022193093585855417540036563114031198834186914551759148989779047194598444970
19894072697389681605600001

S :
25467965253584280531464303313011846046433665548629220158074724672883911005477848
91287109935825918676420001

<u>KAZ-SIGN digital signature forgery detection procedure type – 1</u>

w_0 :
$-$20441063034274921945922549309355534643313782129937764982159825694979191545633351
89297702666086950515860000 $\neq 0$

<u>Final verification</u>

y_1 and y_2 :
46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
1293361252517983118117682338828453326902665811832497 93

15. ILLUSTRATIVE FULL SIZE TEST VECTORS – 3

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and $S \equiv (S_V \pmod{\frac{G_{Rg}qQ}{e}}) + G_{Rg}qQ$ where S_V is a valid signature as in illustrative full size test vectors –1 and e is an n integer consisting some or all common primes between G_{Rg} and Q. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 1**.

e :

1963788631084825545

$S_V \pmod{\dfrac{G_{Rg}qQ}{e}}$:

10960411747382341419721925468692979387817611690725134829420225094195128650707331
7872001

S :

20441063034274921947018590484093768785285974676807062920941586864051705028575374
40239653952594023833732001

KAZ-SIGN digital signature forgery detection procedure type – 1

w_0 :

$-$20441063034274921945922549309355534643313782129937764982159825694979191545633351
89297702666086950515860000 $\neq 0$

Final verification

y_1 and y_2 :

46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
129336125251798311811768233882845332690266581183249793

16. ILLUSTRATIVE FULL SIZE TEST VECTORS – 4

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and conduct the CRT upon the equation pair $Y_1 = V_2 Q^{-1}$ and $Y_2 \equiv (V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}Q}$ to obtain a forge signature. That is, $S = CRT([Y_1, Y_2], [q, G_{Rg}Q])$. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 2**.

Y_1 :
3539827079667798623596677967908505931970

Y_2 :
85433692190473199370893534166960445882929079736216378764625940001

S :
38682952988576219150640551409543845283304109692563701095106941328333662408761955
743383745780853330314 0001

KAZ-SIGN digital signature forgery detection procedure type – 2

S_{F1} :
38682952988576219150640551409543845283304109692563701095106941328333662408761955
7433837457808533303140001

w_1 :
0

Final verification

y_1 and y_2 :
46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
1293361252517983118117682338828453326902665811 83249793

17. ILLUSTRATIVE FULL SIZE TEST VECTORS – 5

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and the forge signature is constructed as per steps 3 – 14 during verification. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 3**.

VQ :
5117711724152298763291800001

S :
46414453401135088759349184669071657909011131319880360468727960001

KAZ-SIGN digital signature forgery detection procedure type – 3

S_{F2} :
46414453401135088759349184669071657909011131319880360468727960001

w_2 :
0

Final verification

y_1 and y_2 :
4647119433975657005962404465491628708750162181016725532757289803044235132694189037504401385310131407970914858764681617610162668339230621373312650620303273534119848400601495764003669377999263643224175420550204068805801303070648527095877218776911636611584698044439533590459161995783196475481067279677875618250798049274374715631805710062494033838968858401210740778754967862825972302913827862713538395964129336125251798311811768233882845332690266581183249793

18. ILLUSTRATIVE FULL SIZE TEST VECTORS – 6

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and $S \equiv (V_1^{(\phi(Q))})(h^{(\phi(qQ))}) \pmod{G_{Rg}qQ}$. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 4**.

S :
71887167484691852861114131036532368075204095381612126814996063428153080445754585
704125522851985721072001

KAZ-SIGN digital signature forgery detection procedure type – 4

w_3 :
1497986136490727392643324334923533440849773369679031554383066641907531209319 25

w_4 :
$-26234763794676655487074386662473285516935039866726805939652996266042563424202 5$
$\neq 0$

Final verification

y_1 and y_2 :
46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
129336125251798311811768233882845332690266581183249793

19. ILLUSTRATIVE FULL SIZE TEST VECTORS – 7

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and conduct the CRT upon the equation pair $Y_1 \equiv 1 \pmod{\frac{qQ}{w}}$ where $w = \gcd(Q, G_{Rg})$ and $Y_2 \equiv (V_1^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}}$ to obtain a forge signature. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 4**.

Y_1 :
1

Y_2 :
511771172415229876329180001

S :
53498737379472412574332403469027722090916364071715199263321192825012228060225632
3984001

KAZ-SIGN digital signature forgery detection procedure type – 4

w_3 :
116431182179248680450031658440253681535

w_4 :
$-412146251595839294135076300117086199254211304452991966154386595192738501492415$
$\neq 0$

Final verification

y_1 and y_2 :
46471194339756570059624044654916287087501621810167255327572898030442351326941890
37504401385310131407970914858764681617610162668339230621373312650620303273534119
84840060149576400366937799926364322417542055020406880580130307064852709587721877
69116366115846980444395335904591619957831964754810672796778756182507980492743747
15631805710062494033838968858401210740778754967862825972302913827862713538395964
129336125251798311811768233882845332690266581183249793

20. ILLUSTRATIVE FULL SIZE TEST VECTORS – 8

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 180$. That is, $P = \{3, 5, 7, \ldots, 1087\}$. In this illustration, we provide a forged KAZ-SIGN signature S where the system parameters, $(N, g, q, Q, G_g, R, G_{Rg}, \alpha, V_1, V_2, h)$ are the same as in **ILLUSTRATIVE FULL SIZE TEST VECTORS – 1** and construct a forged signature with a forged α of the form $A = V_1 + G_{Rg}qT$ for some $T \in \mathbb{Z}$ and forged α is a prime. The constructed forged signature is of the form $S \equiv (A^{\phi(Q)})(h^{\phi(qQ)}) \pmod{G_{Rg}qQ}$. This signature will fail the **KAZ-SIGN digital signature forgery detection procedure type – 4**.

T :
213352556447705824662451401171525808097

A :
3745691637964433232174737397640681502576758773237370062062335477612628515179529368862426178981301088661489

S :
1132620244051064588046059774948475803791016202583942063347441396426498105782549901009794143756006850580001

KAZ-SIGN digital signature forgery detection procedure type – 4

w_3 :
149798613649072739264332433492353344084977336967903155438306664190753120931925

w_4 :
$-149798613649072739264332433492353344084977336967903155438306664190753120931925 \neq 0$

Final verification

y_1 and y_2 :
4647119433975657005962404465491628708750162181016725532757289803044235132694189037504401385310131407970914858764681617610162668339230621373312650620303273534119848400601495764003669377999263643224175420550204068805801303070648527095877218776911636611584698044439533590459161995783196475481067279677875618250798049274374715631805710062494033838968858401210740778754967862825972302913827862713538395964129336125251798311811768233882845332690266581183249793

References

Ajtai, M. (1998). The shortest vector problem in L2 is NP-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19.

Bleichenbacher, D. and May, A. (2006). New attacks on RSA with small secret CRT-exponents. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*, pages 1–13. Springer.

Boneh, D. and Venkatesan, R. (2001). Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Advances in Cryptology-CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, pages 129–142. Springer.

Girault, M., Toffin, P., and Vallée, B. (1990). Computation of approximate L-th roots modulo n and application to cryptography. In *Advances in Cryptology—CRYPTO'88: Proceedings 8*, pages 100–117. Springer.

Herrmann, M. and May, A. (2008). Solving linear equations modulo divisors: On factoring given any bits. In *Advances in Cryptology-ASIACRYPT 2008: 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings 14*, pages 406–424. Springer.

Hoffstein, J., Pipher, J., Silverman, J. H., and Silverman, J. H. (2008). *An introduction to mathematical cryptography*, volume 1. Springer.

Nguyen, P. Q. (2004). Can we trust cryptographic software? Cryptographic flaws in GNU Privacy Guard v1. 2.3. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 555–570. Springer.