Kriptografi Atasi Zarah Digital Signature (KAZ-SIGN)

Algorithm Specifications and Supporting Documentation

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]

[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
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

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
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 Second Order Discrete Logarithm Problem (2-DLP). The idea revolves around the difficulty of reconstructing a Discrete Logarithm Problem (DLP) from a given parameter in order to proceed to identify the secret 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 Second Order Discrete Logarithm Problem (2-DLP). It is defined in the following section.

3. SECOND ORDER DISCRETE LOGARITHM PROBLEM (2-DLP)

Let N be a composite number, g a random prime in $\mathbb{Z}_N$ of order G_g where at most $G_g \approx N^\delta$ for $\delta \in (0,1)$ and $\delta \to 0$. Choose a random prime $Q \in \mathbb{Z}_{\phi(N)}$ of order G_Q, where $G_Q \approx \phi(N)^\varepsilon$ for $\varepsilon \to 1$. That is, choose Q with a large order in $\mathbb{Z}_{\phi(N)}$. Such Q, has its own natural order in $\mathbb{Z}_{\phi(G_g)}$. Let that order be denoted as G_{Qg}. We can observe the natural relation given by $Q^{G_{Qg}} \equiv 1 \pmod{G_g}$ and $\phi(N) \equiv 0 \pmod{G_g}$.

Then choose a random integer $x \in \mathbb{Z}_{\phi(G_g)}$ where $x \approx \phi(G_g)$. Suppose from the equation given by

$$g^{Q^x \pmod{\phi(N)}} \equiv A \pmod{N} \tag{1}$$

one has solved the Discrete Logarithm Problem (DLP) upon equation (1) in polynomial time on a classical computer and obtained the value X where $Q^x \not\equiv X \pmod{\phi(N)}$ and $g^X \equiv A \pmod{N}$. The relation $Q^x \not\equiv X \pmod{\phi(N)}$ would result in the non-existence of the discrete logarithm solution for $Q^x \equiv X \pmod{\phi(N)}$.

The 2-DLP is, upon given the values (A,g,N,Q), one is tasked to determine $x \in Z_{\phi(G_g)}$ where $x \approx \phi(G_g)$ such that the relation (1) holds.

4. COMPLEXITY OF SOLVING THE 2-DLP

Let $n_{\phi(G_g)} = \ell(\phi(G_g))$ be the bit length of $\phi(G_g)$. The complexity to obtain x is $O(2^{n_{\phi(G_g)}})$. When deploying Grover's algorithm on a quantum computer, the complexity to obtain x is $O(2^{\frac{n_{\phi(G_g)}}{2}})$. In other words, since $\phi(G_g) \approx G_g \approx N^\delta$, the complexity to obtain x is $O(N^\delta)$. When deploying Grover's algorithm on a quantum computer, the complexity to obtain x is $O(N^{\frac{\delta}{2}})$.

5. COMPLEXITY PRE-DETERMINING PARAMETERS TO SATISY 2-DLP

Obtaining the relation $Q^x \not\equiv X \pmod{\phi(N)}$

Let $Q^x \equiv T_1 \pmod{\phi(N)}$. From the predetermined order of $g \in \mathbb{Z}_N$, during the process of solving the DLP upon (1), a collision would occur prior to the full cycle of g. As such, the process of solving the DLP upon (1) to obtain $X \approx N^\delta$ would occur in polynomial time on a classical computer. And since $T_1 < \phi(N)$ and $T_1 \approx N$, the relation $Q^x \not\equiv X \pmod{\phi(N)}$ will hold.

6. 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 with computing the entire shared secret key itself.

7. 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* $f(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} a_i x_i \equiv 0 \pmod N$ *usually has* N^{ε} *many solutions, which is exponential in the bit-size of* N.

Remark 2. *From Conjecture 1, there is no hope to find efficient algorithms that in general improve on this bound, since one cannot even output all roots in polynomial time.*

8. THE KAZ-SIGN DIGITAL SIGNATURE ALGORITHM

8.1 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.

8.2 Utilized Functions

Let $H(\cdot)$ be a hash function. Let $\mathrm{DLog}(\cdot)$ be the discrete anti-logarithm function. That is, from $g^x \equiv \beta \pmod N$, upon given (β, g, N) one computes $x = \mathrm{DLog}_g(\beta \pmod N)$. Let $\phi(\cdot)$ be the usual Euler-totient function. Let $\ell(\cdot)$ be the function that outputs the bit length of a given input.

8.3 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 $\delta \to 0$. 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 $Q^{G_{Rg}} \equiv 1 \pmod{G_g}$ 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 G_{Rg}. The system parameters are $(g, n, n_{\phi(G_g)}, N, \phi(N), \phi(\phi(N)), R, G_g)$.

8.4 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 $(g, n, n_{\phi(G_g)}, N, \phi(N), \phi(\phi(N)), R, G_g)$
Output: Public verification key, V, and private signing key, α

1: Choose random $\alpha \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
2: Compute verification key, $V \equiv g^{R^{\alpha} \pmod{\phi(N)}} \pmod{N}$.
3: Compute the discrete logarithm $v = \mathrm{DLog}_g(V \pmod{N})$.
4: Compute $z_1 = v - R^{\alpha} \pmod{\phi(N)}$.
5: **if** $z_1 \equiv 0 \pmod{\phi(N)}$ **then**
6: repeat steps 1 till 4.
7: **else** continue step 9
8: **end if**
9: Compute the discrete logarithm $z_2 = \mathrm{DLog}_R(v \pmod{\phi(N)})$.
10: **if** z_2 has a solution **then**
11: repeat steps 1 till 9.
12: **else** continue step 14
13: **end if**
14: Output public verification key V and private signing key α.

Algorithm 2 KAZ-SIGN Signing Algorithm

Input: System parameters $(g, n, n_{\phi(G_g)}, N, \phi(N), \phi(\phi(N)), R, G_g)$, private signing key, α, and message to be signed, $m \in \mathbb{Z}_N$
Output: Signatures, (S_1, S_2), salt, σ.

1: Generate a random salt, $\sigma \in \{0,1\}^{32}$ corresponding to message, m.
2: Compute the hash value of the message, $h = H(m \| \sigma)$.
3: Choose random ephemeral prime $r \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
4: Compute $S_0 \equiv g^{R^{r} \pmod{\phi(N)}} \pmod{N}$.
5: Compute the discrete logarithm $S_1 = \mathrm{DLog}_g(S_0 \pmod{N})$.
6: Compute $z_3 = S_1 - R^r \equiv 0 \pmod{\phi(N)}$.
7: **if** $z_3 = S_1 - R^r \equiv 0 \pmod{\phi(N)}$ **then**
8: Repeat steps 3 till 6.
9: **else** Continue step 11
10: **end if**
11: Compute the discrete logarithm $z_4 = \mathrm{DLog}_R(S_1 \pmod{\phi(N)})$.
12: **if** z_4 has a solution **then**
13: Repeat steps 3 till 11.
14: **else** Continue step 16
15: **end if**
16: Compute $S_2 \equiv (\alpha + h) r^{-1} \pmod{\phi(\phi(N))}$.
17: Compute the discrete logarithm $v = \mathrm{DLog}_g(V \pmod{N})$.
18: Compute the discrete logarithm $S_{2f} = \mathrm{DLog}_{S_1}(vR^h \pmod{\phi(N)})$.

19: **if** $S_2 \equiv S_{2f} \pmod{\phi(\phi(N))}$ **then**
20: Repeat steps 3 till 18
21: **else** Continue step 23.
22: **end if**
23: Compute $\alpha_F = \text{DLog}_R(v \pmod{G_g})$.
24: Compute $W_0 \equiv (\alpha_F + h)S_2^{-1} \pmod{\phi(\phi(N))}$.
25: **if** W_0 does not exist **then**
26: Repeat steps 1 till 24.
27: **else** Continue 29.
28: **end if**
29: Compute $w_1 \equiv g^{S_1} \pmod{N}$.
30: Compute $w_2 \equiv g^{R^{W_0} \pmod{\phi(N)}} \pmod{N}$.
31: **if** $w_1 = w_2$ **then**
32: Repeat steps 1 till 30.
33: **else** Continue 35.
34: **end if**
35: Output signature (S_1, S_2), salt, σ and destroy r.

Steps 17, 18, 19 and 20 during signing are known as the **KAZ-SIGN digital signature forgery detection procedure type-1**. While steps 23, 24, 25, 26, 27, 28, 29, 30, 31 and 32 are known as the **KAZ-SIGN parameter suitability detection procedure**.

Algorithm 3 KAZ-SIGN Verification Algorithm

Input: System parameters $(g, n, n_{\phi(G_g)}, N, \phi(N), \phi(\phi(N)), R, G_g)$, public verification key, V, message, m, signatures, (S_1, S_2) and salt corresponding to M, σ.

Output: Accept or reject

1: Compute the hash value of the message and its corresponding salt, σ to be verified, $h = H(m\|\sigma)$.
2: Compute the discrete logarithm $v = \mathrm{DLog}_g(V \pmod{N})$.
3: Compute the discrete logarithm $S_{2f} = \mathrm{DLog}_{S_1}(vR^h \pmod{\phi(N)})$.
4: **if** $S_2 \equiv S_{2f} \pmod{\phi(\phi(N))}$ **then**
5:     reject signature $\perp$
6: **else** continue step 9
7: **end if**
8: Compute $\alpha_F = \mathrm{DLog}_R(v \pmod{G_g})$.
9: Compute $W_0 \equiv (\alpha_F + h)S_2^{-1} \pmod{\phi(\phi(N))}$.
10: Compute $w_1 \equiv g^{S_1} \pmod{N}$.
11: Compute $w_2 \equiv g^{R^{W_0} \pmod{\phi(N)}} \pmod{N}$.
12: **if** $w_1 = w_2$ **then**
13:     reject signature $\perp$
14: **else** continue step 16
15: **end if**
16: Compute $y_1 \equiv g^{S_1^{S_2} \pmod{\phi(N)}} \pmod{N}$.
17: Compute $y_2 \equiv v^{R^h \pmod{\phi(N)}} \pmod{N}$.
18: **if** $y_1 = y_2$ **then**
19:     accept signature
20: **else** reject signature $\perp$
21: **end if**

Steps 2, 3, 4 and 5 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type-1**. While steps 8, 9, 10, 11, 12 and 13 are known as the **KAZ-SIGN digital signature forgery detection procedure type-2**.

9. THE DESIGN RATIONALE

9.1 Proof of correctness (Verification steps 16, 17, 18 and 19)

$$g^{S_1^{S_2}} \equiv g^{R^{r(\alpha+h)r^{-1}}} \equiv g^{R^{\alpha}R^{h}} \equiv v^{R^h \pmod{\phi(N)}} \pmod{N}.$$

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.

9.2 Proof of correctness (Verification steps 2, 3, 4 and 5: KAZ-SIGN digital signature forgery detection procedure type-1)

In order to comprehend the rationale behind steps 2, 3, 4 and 5, one has to observe that due to small parameters, an adversary would be able to compute $v = \mathrm{DLog}_g(V \pmod{N})$ and $S_{2f} = \mathrm{DLog}_{S_1}(vR^h \pmod{\phi(N)})$ in polynomial time on a classical computer. Observe the following,

$$g^{S_1^{S_{2f}}} \equiv g^{vR^h} \equiv V^{R^h \pmod{\phi(N)}} \pmod{N}.$$

Hence, the verifier would have accepted the pair (S_1, S_{2f}) as a legitimate KAZ-SIGN signature pair. In retrospect, the verifier could also compute the values v and S_{2f} in polynomial time on a classical computer. As such, steps 2, 3, 4 and 5 during verification will identify an attempt to forge S_2, and upon identifying such situation, the verifier can reject the signature.

9.3 Proof of correctness (Verification steps 8, 9, 10, 11, 12 and 13: KAZ-SIGN digital signature forgery detection procedure type-2)

In order to comprehend the rationale behind steps 8, 9, 10, 11, 12 and 13, one has to observe that due to small parameters, an adversary would be able to compute $\alpha_F = \mathrm{DLog}_R(v \pmod{G_g})$ in polynomial time on a classical computer. If an adversary utilizing a random r constructs the corresponding S_1 and then computes $S_{2f_2} = (\alpha_F + h)r^{-1} \pmod{\phi(\phi(N))}$ for the hash value of a message m that the adversary wishes to forge a signature upon it, and then upon relaying the parameters (S_1, S_{2f_2}) to the verifier, we can observe the following during verification,

$$g^{S_1^{S_{2f_2}}} \equiv g^{R^{r(\alpha_F+h)r^{-1}}} \equiv g^{R^{\alpha_F}R^h} \equiv g^{vR^h} \equiv V^{R^h \pmod{\phi(N)}} \pmod{N}$$

Hence, the verifier would have accepted the pair (S_1, S_{2f_2}) as a legitimate KAZ-SIGN signature pair.

As such, from steps 8, 9, 10, 11, 12 and 13 during verification, the verifier will identify an attempt to forge S_2. From steps 8, 9, 10, 11, 12 and 13 during verification, the verifier would obtain the following

$$g^{R^{W_0} \pmod{\phi(N)}} \equiv g^{R^{\frac{(\alpha_F+h)r}{\alpha_F+h}} \pmod{\phi(N)}} \equiv g^{R^r} \equiv g^{S_1} \pmod{N}.$$

That is, $w_2 = w_1$. Hence, the verifier would reject the signature.

On the other hand, if the verifier obtains a valid signature pair, due from steps 23, 24, 25, 26, 27, 28, 29, 30, 31 and 32 from the signing procedure and from steps 8, 9, 10, 11, 12 and 13 during the verification procedure, he will obtain the following

$$g^{R^{W_0} \pmod{\phi(N)}} \equiv g^{R^{\frac{(\alpha_F+h)r}{(\alpha+h)}} \pmod{\phi(N)}} \not\equiv g^{R^r} \equiv g^{S_1} \pmod{N}.$$

That is, $w_2 \neq w_1$. Hence, the verifier would proceed to verify the signature.

9.4 Another complexity analysis to solve the 2-DLP

One has the relation $g^{G_g} \equiv 1 \pmod{N}$. As such, from the value $X < G_g$ obtained from equation (1), one can construct the set of solutions given by $T_0 = X + G_g t$ for $t = 0, 1, 2, 3, \ldots$. Now let $Q^x \equiv T_1 \pmod{\phi(N)}$. Following through, since T_1 is an element from the set of solutions, one can have the relation

$$t_{T_1} = \frac{T_1 - X}{G_g}.$$

Since $G_g, X \approx N^{\delta}$ and $\phi(N) \approx N$, the complexity to obtain t_{T_1} is $O(N^{1-\delta})$. When deploying Grover's algorithm on a quantum computer, the complexity to obtain t_{T_1} is $O(N^{\frac{1-\delta}{2}})$. This complexity is much higher than the complexity to guess x in equation (1), which is $O(N^{\delta})$ for $\delta \to 0$.

9.5 Modular linear equation of S_2.

Let G_{Rg} be the order of R in $\mathbb{Z}_{G_g}$ where $R^{G_{Rg}} \equiv 1 \pmod{G_g}$.

We begin by analyzing $\alpha_F = \mathrm{DLog}_R(v \pmod{G_g})$ which implies $R^{\alpha_F} \equiv v \pmod{G_g}$ and consequently that $\alpha_F \equiv \alpha_0 \pmod{G_{Rg}}$.

We continue this direction by analyzing $r_F = \mathrm{DLog}_R(S_1 \pmod{G_g})$ which implies $R^{r_F} \equiv S_1 \pmod{G_g}$ and consequently that $r_F \equiv r_0 \pmod{G_{Rg}}$.

From the above, observe that one can analyze S_2 as follows,

$$S_2 \equiv (\alpha + h)r^{-1} \equiv (\alpha_0 + h)r_0^{-1} \pmod{G_{Rg}}$$

which implies

$$r_0\alpha - (\alpha_0 + h)r + hr_0 \equiv 0 \pmod{G_{Rg}}. \tag{2}$$

Let $\hat{\alpha}$ be the upper bound for α and $\hat{r}$ be the upper bound for r. From Conjecture 1, if one has the situation where $\hat{\alpha}\hat{r} \gg G_{Rg}$, then there is no efficient algorithm to output all the roots of (2). That is, (2) usually has G_{Rg} many solutions, which is exponential in the bit-size of G_{Rg}.

To this end, we have both $\hat{\alpha}$ and $\hat{r} \approx 2^{n_{\phi(G_g)}}$. Thus $\hat{\alpha}\hat{r} \approx 2^{2n_{\phi(G_g)}}$. And since we have chosen the element $R \in \mathbb{Z}_{\phi(G_g)}$ with order G_{Rg}, where G_{Rg} is at most $2^{n_{\phi(G_g)}}$, we can conclude that $\hat{\alpha}\hat{r} \gg G_{Rg}$. This implies, there is no efficient algorithm to output all the roots of (2).

9.6 Implementation of the Hidden Number Problem

From S_2 to obtain α or r, is the hidden number problem.

10. SPECIFICATION OF KAZ-SIGN

The following is the security specification for $\delta = 0.3$.

Number of primes in P, j	$n = \ell(N)$	Total security level, k
68	458	128
100	738	192
125	970	256

Table 1

11. IMPLEMENTATION AND PERFORMANCE

11.1 Key Generation, Signing and Verification Time Complexity

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

11.2 Parameter sizes

We provide here information on size of the key and signature based on security level information from Table 2 (for $\delta = 0.3$).

NIST Security Level	Number of primes in P, j	Security level, k	Length of parameter N (bits)	Key size, (V,N) (bits)	Signature Size (S_1,S_2) (bits)	ECC key size
1	68	128	458	916	590	256
3	100	192	738	1476	930	384
5	125	256	970	1940	1220	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.

11.3 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.

11.4 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 - KAZ458	1406	9955	2696
192 - KAZ738	4280	20822	10306
256 - KAZ970	8276	43319	22650

Table 3

12. ADVANTAGES AND LIMITATIONS

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

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

12.1 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 970-bits. ECC would use 521-bit keys and RSA would use 15360-bit keys.

12.2 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.

12.3 No verification failure

It is apparent that the execution of KAZ-SIGN digital signature forgery detection procedure type-1 within steps 17, 18, 19 and 20 together with KAZ-SIGN parameter suitability detection procedure within steps 23, 24, 25, 26, 27, 28, 29, 30, 31 and 32 during signing 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. This is achievable by the recipient as per execution of KAZ-SIGN digital signature forgery detection procedure type-1 in steps 2, 3, 4 and 5 during verification and the KAZ-SIGN digital signature forgery detection procedure type-2 in steps 8, 9, 10, 11, 12 and 13 during verification.

12.4 Limitation

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

1. Based on unknown problem, the Second Order Discrete Logarithm Problem (2-DLP)

12.4.1 Based on unknown problem, the Second Order Discrete Logarithm Problem (2-DLP)

The 2-DLP 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.

13. 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 2-DLP is an unknown fact. We opine that, the acceptance of 2-DLP 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.

14. ILLUSTRATIVE FULL SIZE TEST VECTORS -1

The following are parameters that illustrate KAZ-SIGN for 128-bit security (refer to Table 2). This is an example for $j = 68$. That is, $P = \{3, 5, 7, \ldots, 347\}$. This is the case where adversary is not able to generate S_{2f} (i.e. there does not exist S_{2f}).

N :
37470874733837919416563211326754079989324849463818175868172713496859968
436633910633680216649416805806774541289433279788468718778634973256 5

$\approx 2^{458}$

$\phi(N)$:
71467427390759841729059757466289459181369050713019533645557376916391119
99763706870879597933663696434550639916639846400000000000000000000

$\approx 2^{455}$

g :
37337841543021527447924528338800360404907597638975774468833719703953986
640424245333940843404362488655570562547596485848440650654105417515 7

G_g :
4647420081498856225747178719543948128000

$\approx 2^{132} \approx 2^{0.29(458)} \approx N^{0.3}$

R :
56649467415797035426833950941618577643554746014304810200407453298702899
388975013642190017670254089031691771627671453016895621451465031029

G_R :
37780750794040061026159837604616367499689184180646736913638079474868225
742729956556800000000000000000000

$\approx 2^{345} \approx 2^{0.76(455)} \phi(N)^{0.76}$

α :
340220954190025314480923429400932119169

α_F :
728864143169

V :
25866442924975776240800138071249435949798661733028100730316281988378898
8455025085636730013602437867546222655844280985350098854390051978397

v :
780840645036182383233589252746450007669

h :
62472778471879427263483730054672741426741022889380468967830179446174955
835497

r :
207380663228983046860356253517041976201

S_0 :
92594056727630943065724157541020972855919594994390865675224423794137249
153868709944441146309582451052880451950091230577526489649311892292

S_1 :
1058126240615153382953495372797133459029

S_2 :
83442359262307195180000653887392692592583010904033713996707826885970980
73438574391231665570921221497761840498601574481688393337798045466

S_{2f} (Verification Step 2) :
FAIL – No solution exists

y_1 and y_2 :

1730643972786312919837377952592972441794203634605389631704395974968676336 6
5844431367400376683379429243159432265465219336003824498498807927

W_0 :

17286294947364018101572702012771789874609996309106504341604534630015632312
674617759407007219477942103008746068111236752225301744487 40201

w_1 :

925940567276309430657241575410209728559195949943908656752244237941 37249
153868709944441146309582451052880451950091230577526489649311892292

w_2 :

216524965380402271875369019905771821069639914232700007651671141585 30835
0987093706203694781142283948857387689677788848554952213236236901117

15. 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 $j = 68$. That is, $P = \{3, 5, 7, \ldots, 347\}$. This is the case where adversary is able to generate S_{2f} (i.e. there exist S_{2f}).

N :
3747087473383791941656321132675407998932484946381817586817271349685996 8
43663391063368021664941680580677454128943327978846871877863497325 65
$\approx 2^{458}$

$\phi(N)$:
7146742739075984172905975746628945918136905071301953364555737691639111 9
99763706870879597933663696434550639916639846400000000000000000000
$\approx 2^{455}$

g :
3730412091359611871584962630692178886079871953152362640713297785772976 9
41440959824124802546815664508455943438306920260777840881202097646 71

G_g :
1440700225264645429981625403058623919680 00
$\approx 2^{137} \approx 2^{0.28(458)} \approx N^{0.3}$

R :
4800960061283836270063303469344116903945420179025163315701288521689323 3
38471739366003908739377168749078543201569487988600863391907406575 7

G_R :
142954192193665095774658844990440309458283399602447112646198138553555448756275511296000000000000000000000
$\approx 2^{340} \approx 2^{0.75(455)}\phi(N)^{0.75}$

α :
80345722927735983873595641676804320878 99

α_f :
190961639579

V :
1082591771867822303548994528724001208973502160292176288529564498063229387110007162805914790404792842918282335030730890746947405352639828 46

v :
112315556753750599540886433443945935169893

h :
11203892680223557709740725439319621629913778086884094445517066373243819 8835556

r :
562095288101035727934131386339072197 4061

S_0 :
12533958362578133287158298848952793749023178033447093397720380949356865552573026080504390172438792767885475371937577663448484565892120634 1

S_1 :
724715659378818715083113865248354328157 57

S_{2f} (Verification Step 2) :
556088066917164171085127621657323749220776461773352361564632971059033653702569130817106874143671670799 5

y_1 and y_2 (to test verifying the pair (S_1, S_{2_f})) :
1089828122168885480060194588599124610341175782265998970143623592353803263669606171392158043246194996150166396433332825504597142745791326 11

W_0 (to test verifying the pair (S_1, S_{2_f})) :
176228591781623110556587863347424976422920983261708212087522877096884050628832521601928073853999804046915018499433458893032673800856177 3

w_1 (to test verifying the pair (S_1, S_{2_f})) :
125339583625781332871582988489527937490231780334470933977203809493568655525730260805043901724387927678854753719375776634484845658921206341

w_2 (to test verifying the pair (S_1, S_{2_f})) :
320622669038820744604402619313346586837436621208577922720694948515299391010199855633036570832144154181926281316391308765968591888217346 96

16. 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 $j = 68$. That is, $P = \{3, 5, 7, \ldots, 347\}$. This is the case where adversary is able to generate S_{2f} (i.e. utilizing α_F and $w_1 = w_2$).

N :
3747087473383791941656321132675407998932484946381817586817271349685996843663391063368021664941680580677454128943327978846871877863497325 65
$\approx 2^{458}$

$\phi(N)$:
71467427390759841729059757466289459181369050713019533645557376916391119997637068708795979336636964345506399166398464000000000000000000000
$\approx 2^{455}$

g :
37264577947418073527219311012684352466996219161009705980277080418489051184782940236591658620834680947693828596558921358623812001562746821 9

G_g :
480233408421548476660541801019541306560 00
$\approx 2^{136} \approx 2^{0.29(458)} \approx N^{0.3}$

R :
66664243434711203266642091153713838731668193722745467827816912424088057245654978843244510888654644295974331443262963345499491933368065623

G_R :
1322326277791402135915594316161572862489121446322635791977332781620387900995548479488000000000000000000000
$\approx 2^{346} \approx 2^{0.75(455)}\phi(N)^{0.75}$

α :
5396936944544249571843104215168603175869

α_f :
203216267869

V :
19762686571584032493214033226823173499894688211433473673804640625975773816609384480211987445931744211122051297973798119869334698695295058 9

v :
1014325162324478012135845576394984878066 3

h :
6492521605251393317830410091739490201681100074496078236006157704693049968035 4

r :
4829484744733867592063978505815779164799

S_0 :
2939502369814153480722764744575472561161416404642315455091255824401791397560306924310762185724132387880143084128067554013044681644363484 89

S_1 :
3331196655519889688159330563697340035568 7

S_{2f_2} (Forged S_2 using α_F) :
4939445551833085406626712385834528119595037784925212810573667946879425613203257021817780396746869541681870862579115899901797521297981377

y_1 and y_2 (to test verifying the pair $(S_1, S_{2_{f_2}})$) :
21926358402111017410539995740808256881444870205557190113824238748231410617131199368485454979882121265871450991914172899286811745559726379 4

W_0 (to test verifying the pair $(S_1, S_{2_{f_2}})$) :
48294847447338675920639785058157791647 99

w_1 (to test verifying the pair $(S_1, S_{2_{f_2}})$) :
2939502369814153480722764744575472561161416404642315455091255824401791397560306924310762185724132387880143084128067554013044681644363484 89

w_2 (to test verifying the pair $(S_1, S_{2_{f_2}})$) :
2939502369814153480722764744575472561161416404642315455091255824401791397560306924310762185724132387880143084128067554013044681644363484 89

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.