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

Algorithm Specifications and Supporting Documentation

(Version 1.2)

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 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 $f(x_1,x_2,\dots,x_k) = a_1x_1 + a_2x_2 + \dots + a_kx_k$ be a linear polynomial. One can hope to solve the modular linear equation $f(x_1,x_2,\dots,x_k) \equiv 0 \pmod{N}$, that is to be able to find the set of solutions $(y_1,y_2,\dots,y_k) \in \mathbb{Z}_N^k$, when the product of the unknowns are smaller than the modulus. More precisely, let X_i be upper bounds such that $|y_i| \le X_i$ for $1,\dots,k$. Then one can roughly expect a unique solution whenever the condition $\prod_i X_i \le N$ holds (see Herrmann and May (2008)). It is common knowledge that under the same condition $\prod_i X_i \le N$ the unique solution $(y_1,y_2,\dots,y_k)$ 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 $\prod_i X_i \ge N^{1+\varepsilon}$ then the linear equation $f(x_1,x_2,\dots,x_k) = \sum_{i=1}^{k} a_i x_i \equiv 0 \pmod{N}$ usually has N^{ε} many solutions, which is exponential in the bit-size of N.*

Remark 2. *From Conjecture 1, there is 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 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.

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 $\phi(G_g)$. Let β be the largest factor of G_{Rg}. The system parameters are $(g, n, n_{\phi(G_g)}, N, R, G_g, G_{Rg}, \beta)$.

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 $(g, n, n_{\phi(G_g)}, N, R, G_g, G_{Rg}, \beta)$, security parameter, k.
Output: Public verification key tuple, $V = (V_1, V_2, V_3)$, and private signing key, α

1: Choose random $\alpha \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
2: Compute public verification key – 1, $V_1 \equiv \alpha \pmod{G_{Rg}}$.
3: Choose a random k-bit prime ρ, where k is the security parameter. The public verification key – 2, is given by $V_2 = \beta\rho$.
4: Compute public verification key – 3, $V_3 \equiv \alpha \pmod{V_2}$.
5: Output public verification key tuple, $V = (V_1, V_2, V_3)$ and private signing key α.

Algorithm 2 KAZ-SIGN Signing Algorithm

Input: System parameters $(g, n, n_{\phi(G_g)}, N, R, G_g, G_{Rg}, \beta)$, private signing key, α, and message to be signed, $m \in \mathbb{Z}_N$
Output: Signature tuple, $S = (S_1, S_2, S_3)$.

1: Compute the hash value of the message, $h = H(m)$.
2: Choose random ephemeral prime $r \in (2^{n_{\phi(G_g)}-2}, 2^{n_{\phi(G_g)}-1})$.
3: Compute $S_1 \equiv R^{r \pmod{G_{Rg}}} \pmod{G_g}$.
4: Compute $S_2 \equiv (\alpha^{r \pmod{V_2}} + h)r^{-1} \pmod{G_{Rg}V_2}$.
5: Compute $S_3 \equiv r \pmod{V_2}$.
6: Compute $r_F \equiv r \pmod{G_{Rg}}$.
7: Compute Chinese Remainder Theorem upon $w_4 \equiv (V_3^{S_3} + h)S_3^{-1} \pmod{\rho}$ and $w_5 \equiv (V_1^{S_3} + h)r_F^{-1} \pmod{G_{Rg}}$ to obtain $w_6 \pmod{\rho G_{Rg}}$.
8: Compute $w_7 = w_6 - S_2$.
9: **if** $w_7 = 0$ **then**
10: Repeat from Step 2
11: **else** Continue step 13
12: **end if**
13: Output signature tuple, $S = (S_1, S_2, S_3)$, and destroy r.

Steps 6, 7, 8, 9, 10, 11, and 12 during signing are known as the **KAZ-SIGN parameter suitability detection procedure**.

Algorithm 3 KAZ-SIGN Verification Algorithm

Input: System parameters $(g, n, n_{\phi(G_g)}, N, R, G_g, G_{Rg}, \beta)$, public verification key tuple, $V = (V_1, V_2, V_3)$, message, m, and, signature tuple, $S = (S_1, S_2, S_3)$.

Output: Accept or reject

1: Compute the hash value of the message to be verified, $h = H(m)$.
2: Compute $w_0 \equiv (S_2 S_3) - h \pmod{V_2}$.
3: Compute $w_1 \equiv V_3^{S_3} \pmod{V_2}$.
4: **if** $w_0 \neq w_1$ **then**
5:     Reject signature $\perp$
6: **else** Continue step 8
7: **end if**
8: Compute $r_F = \mathrm{DLog}_R S_1 \pmod{G_g}$.
9: Compute $w_2 \equiv \left(\frac{G_{Rg}}{\beta}\right) S_3 \pmod{G_{Rg}}$.
10: Compute $w_3 \equiv \left(\frac{G_{Rg}}{\beta}\right) r_F \pmod{G_{Rg}}$.
11: **if** $w_2 \neq w_3$ **then**
12:     Reject signature $\perp$
13: **else** Continue step 15
14: **end if**
15: Compute Chinese Remainder Theorem upon $w_4 \equiv (V_3^{S_3} + h) S_3^{-1} \pmod{\rho}$ and $w_5 \equiv (V_1^{S_3} + h) r_F^{-1} \pmod{G_{Rg}}$ to obtain $w_6 \pmod{\rho G_{Rg}}$.
16: Compute $w_7 = w_6 - S_2$.
17: **if** $w_7 = 0$ **then**
18:     Reject signature $\perp$
19: **else** Continue step 21
20: **end if**
21: Compute $y_1 \equiv g^{S_1^{S_2} \pmod{G_g}} \pmod{N}$.
22: Compute $z_0 \equiv R^h \pmod{Gg}$.
23: Compute $z_1 \equiv R^{V_1^{S_3} \pmod{G_{Rg}}} \pmod{G_g}$.
24: Compute $y_2 \equiv g^{z_0 z_1} \pmod{N}$.
25: **if** $y_1 = y_2$ **then**
26:     accept signature
27: **else** reject signature $\perp$
28: **end if**

Steps 2, 3, 4, 5, 6, and 7 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 1**, steps 8, 9, 10, 11, 12, 13 and 14 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 2**, and steps 15, 16, 17, 18, 19 and 20 during verification are known as the **KAZ-SIGN digital signature forgery detection procedure type – 3**.

8. THE DESIGN RATIONALE

8.1 Proof of correctness (Verification steps 21, 22, 23, 24, 25, 26, 27 and 28)

We begin by discussing the rationale behind steps 21, 22, 23, 24, 25, 26, 27 and 28 with relation to the verification process. Observe the following,

$$g^{S_1^{S_2}} \equiv g^{R^{r(\alpha^{r \pmod{V_2}}+h)r^{-1}}} \equiv g^{R^{r(V_1^{S_3}+h)r^{-1}}} \equiv g^{R^{(V_1^{S_3}+h)}} \equiv g^{z_0 z_1} \pmod{N}.$$

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.

8.2 Proof of correctness (Verification steps 2, 3, 4, 5, 6, and 7: KAZ-SIGN digital signature forgery detection procedure type – 1)

In order to comprehend the rationale behind steps 2, 3, 4, 5, 6, and 7, one has to observe the following,

$$S_2 S_3 - h \equiv V_3^{S_3} \pmod{V_2}.$$

Hence, $w_0 = w_1$.

8.3 Proof of correctness (Verification steps 8, 9, 10, 11, 12, 13 and 14: KAZ-SIGN digital signature forgery detection procedure type – 2)

In order to comprehend the rationale behind steps 8, 9, 10, 11, 12, 13 and 14, one has to observe the following;

From, $\frac{G_{Rg}}{\beta} V_2 \equiv 0 \pmod{G_{Rg}}$, we have

$$\left(\frac{G_{Rg}}{\beta}\right) S_3 \equiv \left(\frac{G_{Rg}}{\beta}\right) r \equiv \left(\frac{G_{Rg}}{\beta}\right) r_F \pmod{G_{Rg}}.$$

Hence, $w_2 = w_3$.

8.4 Proof of correctness (Verification steps 15, 16, 17, 18, 19 and 20: KAZ-SIGN digital signature forgery detection procedure type – 3)

In order to comprehend the rationale behind steps 15, 16, 17, 18, 19 and 20, upon computing

$$w_6 \pmod{\rho G_{Rg}}$$

it is clear from $S_2 \equiv (\alpha^{r \pmod{V_2}} + h)r^{-1} \pmod{G_{Rg}V_2}$, one will obtain

$$w_7 = w_6 - S_2 \neq 0.$$

Moreover, **KAZ-SIGN parameter suitability detection procedure** has ensured a valid signature will not produce $w_7 = 0$.

8.5 Complexity of deriving forged signature tuple, $(S_{1f1}, S_{2f1}, S_{3f1})$ – For the case of random $S_{3f1} \in \mathbb{Z}_{V_2}$ and $\gcd(S_{3f1}, G_{Rg}) = 1$

We have $\rho = \frac{V_2}{\beta}$. An adversary utilizing a random r_0 computes the corresponding $S_{1f1} \equiv R^{r_0 \pmod{G_{Rg}}} \pmod{G_g}$, chooses a random k-bit $S_{3f1} \in \mathbb{Z}_{V_2}$ where $\gcd(S_{3f1}, G_{Rg}) = 1$ and then computes the following:

$$z_1 \equiv \left(V_3^{S_{3f1}} + h\right) S_{3f1}^{-1} \pmod{\rho}$$
$$z_2 \equiv \left(V_1^{S_{3f1}} + h\right) r_0^{-1} \pmod{G_{Rg}}$$

From the fact that $\gcd(\rho, G_{Rg}) = 1$, the adversary will solve z_1 and z_2 using the Chinese Remainder Theorem to obtain $S_{2f1} \pmod{\rho G_{Rg}}$.

Observe that

$$g^{S_{1f1}^{S_{2f1}}} \equiv g^{R^{r_0\left(V_1^{r_0 \pmod{V_2}} + h + G_{Rg}x\right)r_0^{-1}}} \equiv g^{R^{r_0\left(V_1^{S_{3f1}} + h\right)r_0^{-1}}} \equiv g^{R^{\left(V_1^{S_{3f1}} + h\right)}} \equiv g^{z_0 z_1} \pmod{N}.$$

where $z_0 \equiv R^h \pmod{G_g}$ and $z_1 \equiv R^{V_1^{S_{3f1}} \pmod{G_{Rg}}} \pmod{G_g}$.

But before verification steps 21, 22, 23, 24, 25, 26, 27 and 28 are conducted, the verifier needs to execute verification steps 2 - 20.

For steps 2, 3, 4, 5, 6, and 7 we have:

$$S_{2f1}S_{3f1} - h \not\equiv V_3^{S_{3f1}} \pmod{V_2}.$$

This is due to the following angebraic reasoning:

$$S_{2f1} \equiv (V_3^{S_{3f1}} + h)S_{3f1}^{-1} \pmod{\rho}$$
$$S_{2f1} \equiv (V_1^{S_{3f1}} + h)r_0^{-1} \pmod{G_{Rg}}$$

which means

$$S_{2f1} = (V_3^{S_{3f1}} + h)(S_{3f1}^{-1}) + \rho t$$

for some $t \in \mathbb{Z}$, which in turn implies

$$(V_3^{S_{3f1}} + h)(S_{3f1}^{-1}) + \rho t \equiv (V_1^{r_0} + h)r_0^{-1} \pmod{G_{Rg}}.$$

Solving for t, we obtain

$$t \equiv \left((V_1^{r_0} + h)r_0^{-1} - (V_3^{S_{3f1}} + h)(S_{3f1}^{-1})\right)\left(\rho^{-1}\right) \pmod{G_{Rg}}.$$

Note that,

$$(V_1^{r_0} + h)r_0^{-1} - (V_3^{S_{3f1}} + h)(S_{3f1}^{-1}) \not\equiv 0 \pmod{\beta}$$

due to the fact that

$$S_{3f1} \not\equiv r_0 \pmod{\beta}.$$

Upon substitution, we obtain

$$S_{2f1} = (V_3^{S_{3f1}} + h)(S_{3f1}^{-1}) + \rho t \pmod{\rho G_{Rg}}.$$

From the fact that

$$\rho G_{Rg} \equiv 0 \pmod{V_2},$$

we end up with the situation

$$S_{2f1} \not\equiv (V_3^{S_{3f1}} + h)(S_{3f1}^{-1}) \pmod{V_2}.$$

Thus, we will obtain $w_0 \neq w_1$ and hence satisfies the condition to reject the signature.The above also holds for

$$S_{2f1} := S_{2f1} + G_{Rg}x \pmod{V_2 G_{Rg}}$$

for random $x \in \mathbb{Z}_{\rho G_{Rg}}$ (including $x = 0$).

As a note we have the following relation,

$$S_{2f1}S_{3f1} - h \equiv V_3^{S_{3f1}} \pmod{\rho} \tag{1}$$

However (1) is not part of the verification procedure.

For steps 8, 9, 10, 11, 12, 13 and 14, we have:

$$\left(\frac{G_{Rg}}{\beta}\right) S_{3f1} \not\equiv \left(\frac{G_{Rg}}{\beta}\right) r_0 \not\equiv \left(\frac{G_{Rg}}{\beta}\right) r_{0F} \pmod{G_{Rg}}$$

where $r_{0F} = \mathrm{DLog}_R S_{1f1} \pmod{G_{Rg}}$. Thus, we will obtain $w_2 \neq w_3$ and hence satisfies the condition to reject the signature.

For steps 15, 16, 17, 18, 19 and 20, if $w_6 := S_{2f1}$ was the result of the Chinese Remainder Theorem upon z_1 and z_2, we will have

$$S_{2f1} \equiv w_4 \equiv (V_3^{S_3} + h)S_3^{-1} \pmod{\rho} \quad \text{and} \quad S_{2f1} \equiv w_5 \equiv (V_1^{S_3} + h)r_F^{-1} \pmod{\rho}.$$

Hence, steps 15, 16, 17, 18, 19 and 20 will output

$$w_7 = w_6 - S_2 = 0.$$

Thus, satisfies the condition to reject the signature.

8.6 Complexity of deriving forged signature tuple, $(S_{1f1}, S_{2f1}, S_{3f1})$ – For the case of random $S_{3f1} \equiv r \pmod{V_2}$ and $\gcd(S_{3f1}, G_{Rg}) = 1$

We have $\rho = \frac{V_2}{\beta}$. An adversary utilizing a random r_0 computes the corresponding $S_{1f1} \equiv R^{r_0 \pmod{G_{Rg}}} \pmod{G_g}$, sets $S_{3f1} \equiv r_0 \pmod{V_2}$ where $\gcd(S_{3f1}, G_{Rg}) = 1$ and then computes the following:

$$z_1 \equiv \left(V_3^{S_{3f1}} + h\right) S_{3f1}^{-1} \pmod{\rho}$$
$$z_2 \equiv \left(V_1^{S_{3f1}} + h\right) r_0^{-1} \pmod{G_{Rg}}.$$

That is,

$$z_1 \equiv \left(V_3^{r_0} + h\right) r_0^{-1} \pmod{\rho}$$
$$z_2 \equiv \left(V_1^{r_0} + h\right) r_0^{-1} \pmod{G_{Rg}}.$$

From the fact that $\gcd(\rho, G_{Rg}) = 1$, the adversary will solve z_1 and z_2 using the Chinese Remainder Theorem to obtain $S_{2f1} \pmod{\rho G_{Rg}}$.

Observe that

$$g^{S_{1f1}^{S_{2f1}}} \equiv g^{R^{r_0\left(V_1^{r_0 \pmod{V_2}} + h + G_{Rg}x\right)r_0^{-1}}} \equiv g^{R^{r_0\left(V_1^{S_{3f1}} + h\right)r_0^{-1}}} \equiv g^{R^{\left(V_1^{S_{3f1}} + h\right)}} \equiv g^{z_0 z_1} \pmod{N}.$$

where $z_0 \equiv R^h \pmod{G_g}$ and $z_1 \equiv R^{V_1^{S_{3f1}} \pmod{G_{Rg}}} \pmod{G_g}$.

But before verification steps 21, 22, 23, 24, 25, 26, 27 and 28 are conducted, the verifier needs to execute verification steps 2 - 20.

For steps 2, 3, 4, 5, 6, and 7 we have:

$$S_{2f1}S_{3f1} - h \equiv V_3^{S_{3f1}} \pmod{V_2}$$

This is due to the following algebraic reasoning:

Since $S_{3f1} \equiv r_0 \pmod{V_2}$, we proceed to solve the following via CRT,

$$
\begin{aligned}
S_{2f1} &\equiv (V_3^{r_0} + h) r_0^{-1} \pmod{\rho} \\
S_{2f1} &\equiv (V_1^{r_0} + h) r_0^{-1} \pmod{G_{Rg}}
\end{aligned}
$$

which means

$$S_{2f1} = (V_3^{r_0} + h)(r_0^{-1}) + \rho t$$

for some $t \in \mathbb{Z}$, which in turn implies

$$(V_3^{r_0} + h)(r_0^{-1}) + \rho t \equiv (V_1^{r_0} + h) r_0^{-1} \pmod{G_{Rg}}.$$

Solving for t and doing the necessary substitution, we obtain

$$S_{2f1} = (V_3^{r_0} + h) r_0^{-1} + \rho t \pmod{\rho G_{Rg}}$$

where

$$
\begin{aligned}
t &\equiv (V_1^{r_0} - V_3^{r_0}) r_0^{-1} \left(\frac{1}{\rho}\right) \pmod{G_{Rg}} \\
&\equiv \left((\alpha - G_{Rg} t_1)^{r_0} - (\alpha - \beta \rho t_2)^{r_0}\right) r_0^{-1} \left(\frac{1}{\rho}\right) \pmod{G_{Rg}} \\
&\equiv (\beta \Delta) r_0^{-1} \left(\frac{1}{\rho}\right) \pmod{G_{Rg}}
\end{aligned}
$$

for some $t_1, t_2, \Delta \in \mathbb{Z}_{G_{Rg}}$.

That is, $t \equiv 0 \pmod{\beta}$.

And from the fact that,

$$\frac{V_2}{\beta} G_{Rg} \equiv 0 \pmod{V_2},$$

we end up with the situation

$$S_{2f1} \equiv (V_3^{r_0} + h)(r_0^{-1}) \pmod{V_2}.$$

and

$$S_{2f1} S_{3f1} - h \equiv V_3^{S_{3f1}} \pmod{V_2}$$

Thus, $w_0 = w_1$ and hence does not satisfy the condition to reject the signature.

For steps 8, 9, 10, 11, 12, 13 and 14, we have:

$$\left(\frac{G_{Rg}}{\beta}\right) S_{3f1} \equiv \left(\frac{G_{Rg}}{\beta}\right) r_0 \equiv \left(\frac{G_{Rg}}{\beta}\right) r_{0F} \pmod{G_{Rg}}$$

where $r_{0F} = \mathrm{DLog}_R S_{1f1} \pmod{G_{Rg}}$. Thus, we will obtain $w_2 = w_3$ and hence does not satisfy the condition to reject the signature.

For steps 15, 16, 17, 18, 19 and 20, if $w_6 := S_{2f1}$ was the result of the Chinese Remainder Theorem upon z_1 and z_2, we will have

$$S_{2f1} \equiv w_4 \equiv (V_3^{S_3} + h)S_3^{-1} \pmod{\rho} \quad \text{and} \quad S_{2f1} \equiv w_5 \equiv (V_1^{S_3} + h)r_F^{-1} \pmod{\rho}.$$

Hence, steps 15, 16, 17, 18, 19 and 20 will output

$$w_7 = w_6 - S_2 = 0.$$

Thus, satisfies the condition to reject the signature.

8.7 Complexity of deriving forged signature tuple, $(S_{1f2}, S_{2f2}, S_{3f2})$

Continuing the discussion, an adversary utilizing a random r_0 computes the corresponding $S_{1f2} \equiv R^{r_0 \pmod{G_{Rg}}} \pmod{G_g}$, $S_{2f2} \equiv (V_1^{r_0 \pmod{V_2}} + h + G_{Rg}x)r^{-1} \pmod{G_{Rg}V_2}$ and $S_{3f2} \equiv r_0 \pmod{V_2}$ for the hash value of a message m that the adversary wishes to forge a signature upon it and a random $x \in \mathbb{Z}_{G_{Rg}}$, including $x = 0$. Observe that

$$g^{S_{1f2}^{S_{2f2}}} \equiv g^{R^{r_0\left(V_1^{r_0 \pmod{V_2}} + h + G_{Rg}x\right) r_0^{-1}}} \equiv g^{R^{r_0\left(V_1^{S_{3f2}} + h\right) r_0^{-1}}} \equiv g^{R^{\left(V_1^{S_{3f2}} + h\right)}} \equiv g^{z_0 z_1} \pmod{N}.$$

But before verification steps 21, 22, 23, 24, 25, 26, 27 and 28 are conducted, the verifier needs to execute verification steps 2, 3, 4, 5, 6, and 7.

Let $S_{3f2} = r_0 \pmod{V_2}$. The verifier will obtain

$$S_{2f2}S_{3f2} - h \equiv V_1^{r_0 \pmod{V_2}} + G_{Rg}x \not\equiv V_3^{S_{3f2}} \pmod{V_2}$$

For the above equation to hold, the adversary needs to identify S_{3f2} satisfying

$$V_3^{S_{3f2}} - S_{2f2}S_{3f2} + h \equiv 0 \pmod{V_2}$$

Since,

$$S_{2f2} \equiv \left(V_1^{S_{3f2}} + h + G_{Rg}x\right) S_{3f2}^{-1} \pmod{G_{Rg}V_2}$$

This will imply,

$$V_3^{S_{3f2}} - \left(V_1^{S_{3f2}} + h + G_{Rg}x\right) + h \equiv 0 \pmod{V_2}$$

$$V_3^{S_{3f2}} - V_1^{S_{3f2}} - G_{Rg}x \equiv 0 \pmod{V_2} \tag{2}$$

Then upon obtaining S_{3f2} we set

$$S_{1f2} \equiv R^{S_{3f2}} \pmod{G_g}$$

We can then have

$$g^{S_{1f2}^{S_{2f2}}} \equiv g^{R^{S_{3f2}\left(V_1^{S_{3f2}}+h+G_{Rg}x\right)S_{3f2}^{-1}}} \equiv g^{R^{\left(V_1^{S_{3f2}}+h\right)}} \equiv g^{z_0 z_1} \pmod{N}$$

However, to solve equation (2), the complexity is $O(V_2)$. When deploying Grover‘s algorithm on a quantum computer, the complexity will be $O\left(V_2^{\frac{1}{2}}\right)$. Furthermore, V_2 contains the factor ρ which is a k-bit prime number (where k is either 128 or 192 or 256 bits). The adversary will not be able to execute the Chinese Remainder Theorem to reduce this complexity.

8.8 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 continue this direction by obtaining $r_F \equiv (V_1^{r \pmod{V_2}} + h)S_2^{-1} \pmod{G_{Rg}}$.

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

$$S_2 \equiv (\alpha^{r \pmod{V_2}} + h)r^{-1} \equiv (V_1 + h)r_F^{-1} \pmod{G_{Rg}}$$

which implies

$$r_F \alpha^{r \pmod{V_2}} - (V_1 + h)r + hr_F \equiv 0 \pmod{G_{Rg}}. \tag{3}$$

Let $\hat{\alpha}$ be the upper bound for $\alpha^{r \pmod{V_2}}$ 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 equation (3). That is, equation (3) usually has G_{Rg} many solutions, which is exponential in the bit-size of G_{Rg}.

To this end, since $\alpha^{r \pmod{V_2}}$ is exponentially large, it is clear to conclude that $\hat{\alpha}\hat{r} \gg G_{Rg}$. This implies, there is no efficient algorithm to output all the roots of equation (3).

8.9 Implementation of the Hidden Number Problem

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

8.10 Another "Expensive" Problem Related To KAZ-SIGN: The 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 $\delta \to 0$. That is, $g^{Gg} \equiv 1 \pmod N$. 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 $Z_{\phi(N)}$. Such Q, has it own natural order in $Z_{\phi(G_g)}$. 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 relation given by

$$g^{Q^x \pmod{\phi(N)}} \equiv A \pmod N \tag{4}$$

one has solved the Discrete Logarithm Problem (DLP) upon equation (4) 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 $x \in \mathbb{Z}_{\phi(G_g)}$ where $x \approx \phi(G_g)$ such that equation (4) holds.

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 equation (4), a collision would occur prior to the full cycle of g. As such, the process of solving the DLP upon equation (4) to obtain $X \approx N^\delta$ would occur in polynomial time on a classical computer. And since $T_1 < \phi(N)$ and $T_1 \approx N_1$, the relation $Q^x \not\equiv X \pmod{\phi(N)}$ will hold.

Furthering on the discussion, one has the relation $g^{G_g} \equiv 1 \pmod N$. As such, from the value $X < G_g$ obtained from equation (4), 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 $O(N^{\frac{1-\delta}{2}})$.

To this end, note that if one proceeds to solve the DLP upon $Q^x \equiv X \pmod{G_g}$, one can obtain the value $x_0 \equiv x \pmod{G_{Qg}}$. From the preceding sections, this is in fact the MRP. It is easy to see that with correct choice of parameters (x, G_{Qg}), the complexity of 2-DLP and MRP can be made the same. Hence, a more "non-expensive" method in discussing the needs of the KAZ-SIGN is directly via the MRP.

9. KEY GENERATION, SIGNING AND VERIFICATION TIME COMPLEXITY

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

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
127	989	128
200	1713	192
257	2311	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 1 (for $\delta = 0.3$).

NIST Security Level	Number of primes in P, j	Security level, k	Length of parameter N (bits)	Public key size, (V_1, V_2, V_3, N) (bits)	Private key size, α (bits)	Signature Size (S_1, S_2, S_3) (bits)	ECC key size (bits)
1	127	128	989	≈ 1350	≈ 270	≈ 620	256
3	200	192	1713	≈ 2210	≈ 390	≈ 890	384
5	257	256	2311	≈ 2980	≈ 520	≈ 1190	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 - KAZ989	238	280	163
192 - KAZ1713	244	526	375
256 - KAZ2311	277	1063	910

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 2311-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 parameter suitability detection procedure** together with **KAZ-SIGN digital signature forgery detection procedure type – 1, type – 2, and type – 3** 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.

12.4 Limitation

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

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

12.4.1 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.

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 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.

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 $j = 127$. That is, $P = \{3, 5, 7, \ldots, 719\}$. We provide a valid KAZ-SIGN signature tuple $S = (S_1, S_2, S_3)$ ~~and two types of forged KAZ-SIGN signature tuple~~ $\cancel{S = (S_{1f1}, S_{2f1}, S_{3f1})}$ ~~and~~ $\cancel{S = (S_1, S_{2f2}, S_3)}$.

N :

4962053072862893011815686085054943995857661934414654393269559561100268468433879052996996579124346821800802464304723640429503137601278290422409955273127096762893555100070212926092147189104881374461018100187690751198809547084086962840136426056988521987231393663009223478164952125897464044412149392265 $\approx 2^{989}$

g :

3792095925705048180187726800782032155915363113946453085039449402737696831381307646578422599137337997987536804167062476101209355561530828643855480333744328890869141524787483553455706606949037853108642642195623853886679262820034445333309151932645086653983408254647919756524877986169280211550967441529

G_g :

6300586713693401606004827190506565575604406149541213937396264879495638528941024000 $\approx 2^{272} \approx 2^{0.275(989)} \approx N^{0.3}$

R :

713206001856918918690182577320611894139047803582796412383918512134658852406679143629516090411185110233302723295119632067831315317917844556902288772698398235864225536018956230377314019768259027323615151761787263863944858128592122228870249761848708840263638416125194479107341024351702968758449937061

G_{Rg} :

15185584726452738974376000 $\approx 2^{84} \approx N^{0.085}$

α :

4529042971682937974629650569493312922541909068092062210018400142139573 6
2551591767 $\approx 2^{268}$

V_1 :

187088687015794757845576 7

V_2 :

3505173080381366320384504685008474189772 9

$= (3938396719529625079083713129223004707 61)(89) \approx 2^{128+7} = 2^{135}$

V_3 :

2386247881679418449775730878590491789428 6

$t_{\alpha V_1} = \dfrac{\alpha - V_1}{G_{Rg}}$:

2982462021230903963657425139662778987933135816440461438 6 $\approx 2^{185}$

$t_{\alpha V_3} = \dfrac{\alpha - V_3}{V_2}$:

1292102520423948249327541084836404225248 9 $\approx 2^{134}$

h :

1021182001152369575276187180417632279038644129681046911202527623997453 1
2594006

r :

3028162554059738180396497724439423808765392275578399824613060999434364 6
2287626427

S_1 :

2031471405439651092286597557704680632868152787030250062758029003331133 7
06478644221

S_2 :
348200403871761959820595478816884359752686314408086214800793156007

S_3 :
899728926510320300276504075753721623351

w_0 for valid signature (S_1, S_2, S_3) :
1894385819300418162690239011192143163649 8

w_1 for valid signature (S_1, S_2, S_3) :
18943858193004181626902390111921431636498

r_F for valid signature (S_1, S_2, S_3) :
993786368636405135933842 7

w_2 for valid signature (S_1, S_2, S_3) :
1706245474882330221840000

w_3 for valid signature (S_1, S_2, S_3) :
1706245474882330221840000

w_6 for valid signature (S_1, S_2, S_3) :
1320632861118211878425713780793542048149618522269662060025268007

$w_7 = w_6 - S_2$:
$-$346879771010643747942169765036090817704536695885816552740767888000

y_1 and y_2 for (S_1, S_2, S_3) :
410587225894278505846942594397675330726893954932378830575415082907581 19
878323421620814728130302647688744602610775640087168493183121604619270 10
262619846399097407415666211971214551327233056374275465322910726894893 79
514293326715986093701665628143898199811099252057772871258277638686621 56
96928065903204

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 $j = 127$. That is, $P = \{3, 5, 7, \ldots, 719\}$. We provide here a forged KAZ-SIGN signature tuple $S = (S_{1f1}, S_{2f1}, S_{3f1})$ for the case where $S_{3f1} \equiv r_0 \pmod{V_2}$.

r_0 :
36848086515502374510356024641985617894101287869942957813611949337390505
5371550551

S_{1f1} :
29262830137860333122986254255432246368143368821285949894239321044350730
61372778061

S_{2f1} :
2633622113187822484456675070340549130480662557839585121561974859

S_{3f1} :
29589512199279852817850592903884428838279

w_0 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
202519866323929090846496573079057396022 73

w_1 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
20251986632392909084649657307905739602273

r_F for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
939778711161214804811055 1

w_2 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
29006173072999613771280 00

w_3 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
2900617307299961377128000

w_6 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
2633622113187822484456675070340549130480662557839585121561974859

$w_7 = w_6 - S_2 : 0$

y_1 and y_2 for $(S_{1f1}, S_{2f1}, S_{3f1})$:
48549625150966063887887625409276520145873166315545951303524420440236037
68527676157130338698974355770893047761261918454855379934435962382168546
28713105994348785696296363988073363030916431638638469202704615101865385
55118164449943896095542360245949634102602800042034026788666829022116185
44563930783034

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 $j = 127$. That is, $P = \{3, 5, 7, \ldots, 719\}$. We provide here a forged KAZ-SIGN signature tuple $S = (S_{1f1}, S_{2f1}, S_{3f1})$ for the case where $S_{3f1} \in \mathbb{Z}_{V_2}$ is chosen randomly.

r_0 :
36848086515502374510356024641985617894101287869942957813611949337390505
5371550551

S_{1f1} :
29262830137860333122986254255432246368143368821285949894239321044350730
61372778061

S_{2f1} :
4586383035861356540086585811090320752811133462616987391874676443

S_{3f1} :
484482419480347199371882349267932668353

w_0 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
32663622426448784578397119051119938565605

w_1 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
6276364405600296548536241085325807024618

r_F for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
9397787111612148048110551

w_2 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
11431844681711612486328000

w_3 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
2900617307299961377128000

w_6 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$ is not needed to be calculated.

$w_7 = w_6 - S_2$ is not needed to be calculated.

y_1 and y_2 for $(S_{1f1}, S_{2f1}, S_{3f1})$:
133607164280613543711717893317622855650404034981172935052266888787706395271251869510611270876206528173739768237745130621617980217475077160838149829870773740344780530557429955268022871642745290422578973438351563162550880560580499633568406565985955055410959997632349716119426578259172514774016544859

17. 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 $j = 127$. That is, $P = \{3, 5, 7, \ldots, 719\}$. We provide here a forged KAZ-SIGN signature tuple $S = (S_{1f1}, S_{2f1}, S_{3f1})$ for the case where $S_{3f1} \equiv r_0 \pmod{V_2}$ is chosen randomly and $S_{2f1} := S_{2f1} + G_{Rg}x \pmod{V_2 G_{Rg}}$.

r_0 :
36848086515502374510356024641985617894101287869942957813611949337390505
5371550551

S_{1f1} :
29262830137860333122986254255432246368143368821285949894239321044350730
61372778061

x :
691604549986101413229541331013615291236888067504485367306676281251204031
11047

S_{2f1} :
71419315504439264635380276012705112565924963819236972853192244443

S_{3f1} :
484482419480347199371882349267932668353

w_0 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
23000441233897682825011334691995890604567

w_1 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$:
6276364405600296548536241085325807024618

r_F for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$ is not needed to be calculated.

w_2 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$ is not needed to be calculated.

w_3 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$ is not needed to be calculated.

w_6 for forged signature $(S_{1f1}, S_{2f1}, S_{3f1})$ is not needed to be calculated.

$w_7 = w_6 - S_2$ is not needed to be calculated.

y_1 and y_2 for $(S_{1f1}, S_{2f1}, S_{3f1})$:
1336071642806135437117178933176228556504040349811729350522668887877063952712518695106112708762065281737397682377451306216179802174750771608381498298707737403447805305574299552680228716427452904225789734383515631625508805605804996335684065659859550554109599976323497161194265782591725147740165448594774016544859

18. 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 $j = 127$. That is, $P = \{3, 5, 7, \ldots, 719\}$. We provide here a forged KAZ-SIGN signature tuple $S = (S_1, S_{2f2}, S_3)$ which utilizes (S_1, S_3) from test vectors – 1 and a forged S_2 denoted as S_{2f2}.

S_1 :
2031471405439651092286597557704680632868152787030250062758029003331133706478644221

x :
71981845501063606179217140969068507332847841021747432232028236584640888049041135912329293820081715173743605265084161888219882804609251915033997167715191406773167

S_{2f2} :
394407130394973462889758868570984590683930905807407593727983436007

S_3 :
8997289265103203002765040757537216233 51

w_0 for forged signature (S_1, S_{2f2}, S_3) :
23955945495202555451299964948041941606729 5

w_1 for forged signature (S_1, S_{2f2}, S_3) :
18943858193004181626902390111921431636498

r_F for forged signature (S_1, S_{2f2}, S_3) is not needed to be calculated.

w_2 for forged signature (S_1, S_{2f2}, S_3) is not needed to be calculated.

w_3 for forged signature (S_1, S_{2f2}, S_3) is not needed to be calculated.

w_6 for forged signature (S_1, S_{2f2}, S_3) is not needed to be calculated.

$w_7 = w_6 - S_2$ is not needed to be calculated.

y_1 and y_2 for (S_1, S_{2f2}, S_3) :
410587225894278505846942594397675330726893954932378830575415082907581198783234216208147281303026476887446026107756400871684931831216046192701026261984639909740741566621197121455132723305637427546532291072689489379514293326715986093701665628143898199811099252057772871258277638686621569692806590320 4

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.