

# *Kriptografi Atasi Zarah* Key Encapsulation Mechanism (KAZ-KEM)

## **Algorithm Specifications and Supporting Documentation**

(KAZ-KEM 1.0)

Muhammad Rezal Kamel Ariffin<sup>1</sup> Abderrahmane Nitaj<sup>2</sup> Nor Azman Abu<sup>3</sup> Zahari Mahad<sup>1</sup>

<sup>1</sup>Universiti Putra Malaysia

<sup>2</sup>Université de Caen Normandie, France

<sup>3</sup>Universiti Teknikal Malaysia Melaka

# Table of Contents

<b>1 INTRODUCTION</b>	<b>1</b>
<b>2 THE DESIGN IDEALISME</b>	<b>1</b>
<b>3 THE HIDDEN NUMBER PROBLEM (HNP) (Boneh and Venkatesan, 2001)</b>	<b>2</b>
<b>4 COMPLEXITY OF SOLVING THE HNP</b>	<b>2</b>
<b>5 THE KAZ-KEM KEY ENCAPSULATION MECHANISM ALGORITHM</b>	<b>2</b>
5.1 Background	2
5.2 Utilized Functions	2
5.3 System Parameters	2
5.4 KAZ-KEM Algorithms	2
<b>6 PROOF OF CORRECTNESS</b>	<b>3</b>
<b>7 ALTERNATIVE ALGEBRA</b>	<b>4</b>
<b>8 IMPLEMENTATION OF THE HIDDEN NUMBER PROBLEM (HNP)</b>	<b>4</b>
<b>9 DISCRETE LOGARITHM PROBLEM ANALYSIS</b>	<b>4</b>
<b>10 DERIVING THE SECURITY LEVEL OF KAZ-KEM</b>	<b>5</b>
<b>11 IMPLEMENTATION AND PERFORMANCE</b>	<b>5</b>
11.1 Key Generation, Encapsulation and Decapsulation Time Complexity	5
11.2 Parameter sizes	5
11.3 Key Generation, Encapsulation and Decapsulation Ease of Implementation	5
11.4 Key Generation, Encapsulation and Decapsulation Empirical Performance Data	6
<b>12 ADVANTAGES AND LIMITATIONS</b>	<b>6</b>
12.1 Key Length	6
12.2 Speed	6
12.3 No Decapsulation Failure	6
12.4 Limitation	7
12.4.1 Based on not widely used problem, Hidden Number Problem (HNP)	7
<b>13 CLOSING REMARKS</b>	<b>8</b>
<b>14 ILLUSTRATIVE FULL SIZE TEST VECTORS</b>	<b>9</b>

**Name of the proposed cryptosystem:** KAZ-KEM 1.0

**Principal submitter:** Muhammad Rezal Kamel Ariffin  
Faculty of Science  
Universiti Putra Malaysia  
43400 UPM Serdang, Selangor  
Malaysia  
Email: rezal@upm.edu.my  
Phone: +60123766494

**Auxilliary submitters:** Abderrahmane Nitaj  
Nor Azman Abu  
Zahari Mahad

**Inventor of the cryptosystem:** Muhammad Rezal Kamel Ariffin

**Owner of the cryptosystem:** Muhammad Rezal Kamel Ariffin

**Alternative point of contact:** Zahari Mahad  
Institute for Mathematical Research  
Universiti Putra Malaysia  
43400 UPM Serdang, Selangor  
Malaysia  
Email: zaharimahad@upm.edu.my  
Phone: +60122437663

## 1. INTRODUCTION

The proposed KAZ Key Encapsulation Mechanism scheme, KAZ-KEM (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 Hidden Number Problem (HNP). The idea revolves around the difficulty of reconstructing an unknown product from a given public parameter. The target of the KAZ-KEM design is to be a quantum resistant key encapsulation mechanism candidate with short encryption, decryption keys and ciphertexts, decrypting correctly 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 plaintext-ciphertext space;
- (viii) The computation overhead for both encapsulation and decapsulation 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 ciphertext expansion ratio is kept to a minimum.

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

### 3. 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.

### 4. COMPLEXITY OF SOLVING THE HNP

The complexity to obtain  $\alpha \pmod{p}$  is  $O(p)$ . When deploying Grover's algorithm on a quantum computer, the complexity to obtain  $\alpha \pmod{p}$  is  $O(p^{\frac{1}{2}})$ .

### 5. THE KAZ-KEM KEY ENCAPSULATION MECHANISM ALGORITHM

#### 5.1 Background

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

#### 5.2 Utilized Functions

Let  $\ell(\cdot)$  be the function that outputs the bit length of a given input. Let  $\text{DLog}_u(v)$  modulo  $p$  be the function that outputs  $w$  such that  $v \equiv u^w \pmod{p}$ .

#### 5.3 System Parameters

From the given security parameter  $k$  (either 128, 192 or 256; depending on the security level needed), prepare a list of the first  $j$ -primes larger than 2,  $P = \{p_i\}_{i=1}^j$  (as of printing it is suggested  $j = 65, 96, 122$ ). Let  $N = \prod_{i=1}^j p_i$  be a public modulus. Choose random primes  $(g_1, g_2)$  where  $\text{DLog}_{g_1}(g_2)$  modulo  $N$  does not exist. Let  $O_{g_1N}$  be the order of  $g_1$  in  $\mathbb{Z}_N$ . Let  $O_{g_2N}$  be the order of  $g_2$  in  $\mathbb{Z}_N$ . The system parameters are  $(g_1, g_2, O_{g_1N}, O_{g_2N}, N)$ .

#### 5.4 KAZ-KEM Algorithms

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

---

**Algorithm 1** KAZ-KEM Key Generation Algorithm

---

**Input:** System parameters  $(g_1, g_2, O_{g_1N}, O_{g_2N}, N)$

**Output:** Public encapsulation key,  $(e_1, e_2)$  and private decapsulation key pair,  $(a_1, a_2)$ .

- 1: Choose random  $a_1 \in \mathbb{Z}_{O_{g_1N}}$  and  $a_2 \in \mathbb{Z}_{O_{g_2N}}$
  - 2: Compute  $e_1 \equiv g_1^{a_1} g_2^{2a_2} \pmod{N}$ .
  - 3: Compute  $e_2 \equiv g_1^{a_2} g_2^{a_1} \pmod{N}$ .
  - 4: Output public encapsulation key  $(e_1, e_2)$  and keep decapsulation key pair  $(a_1, a_2)$ .
- 

---

**Algorithm 2** KAZ-KEM Key Encapsulation Mechanism Algorithm

---

**Input:** System parameters  $(g_1, g_2, O_{g_1N}, O_{g_2N}, N)$ , public encapsulation key  $(e_1, e_2)$ , and plaintext to be encapsulated,  $m$ .

**Output:** Ciphertext,  $C$

- 1: Let  $m \in \mathbb{Z}_N$  be the plaintext to be encapsulated.
  - 2: Choose random ephemeral  $b_1 \in \mathbb{Z}_{O_{g_1N}}$  and  $b_2 \in \mathbb{Z}_{O_{g_2N}}$ .
  - 3: Compute  $B_1 \equiv g_1^{b_1} g_2^{b_2} \pmod{N}$ .
  - 4: Compute  $B_2 \equiv g_1^{b_2} g_2^{2b_1} \pmod{N}$ .
  - 5: Compute  $c \equiv m + e_1^{b_1} e_2^{b_2} \pmod{N}$ .
  - 6: Output ciphertext,  $C = (B_1, B_2, c)$ , and destroy  $(b_1, b_2)$ .
- 

---

**Algorithm 3** KAZ-KEM Decapsulation Algorithm

---

**Input:** Private decapsulation key pair,  $(a_1, a_2)$  and ciphertext,  $C$ .

**Output:** Message,  $m$

- 1: Compute  $y \equiv c - B_1^{a_1} B_2^{a_2} \pmod{N}$ .
  - 2: Output message  $m = y$ .
- 

## 6. PROOF OF CORRECTNESS

Observe the following,

$$\begin{aligned} c - B_1^{a_1} B_2^{a_2} &\equiv m + e_1^{b_1} e_2^{b_2} - B_1^{a_1} B_2^{a_2} \\ &\equiv m + g_1^{a_1 b_1} g_2^{2a_2 b_1} g_1^{a_2 b_2} g_2^{a_1 b_2} - g_1^{b_1 a_1} g_2^{b_2 a_1} g_1^{b_2 a_2} g_2^{2b_1 a_2} \\ &\equiv m + g_1^{a_1 b_1 + a_2 b_2} g_2^{2a_2 b_1 + a_1 b_2} - g_1^{b_1 a_1 + b_2 a_2} g_2^{b_2 a_1 + 2b_1 a_2} \\ &\equiv m \pmod{N} \end{aligned}$$

## 7. ALTERNATIVE ALGEBRA

An alternative algebra during encapsulation would be

$$c \equiv me_1^{b_1} e_2^{b_2} \pmod{N}$$

While the decapsulation would be

$$m \equiv cB_1^{-a_1} B_2^{-a_2} \pmod{N}$$

## 8. IMPLEMENTATION OF THE HIDDEN NUMBER PROBLEM (HNP)

It is clear that the following parameters are implementing the HNP:

1.  $e_1 \equiv g_1^{a_1} g_2^{2a_2} \pmod{N}$
2.  $e_2 \equiv g_1^{a_2} g_2^{a_1} \pmod{N}$
3.  $B_1 \equiv g_1^{b_1} g_2^{b_2} \pmod{N}$
4.  $B_2 \equiv g_1^{b_2} g_2^{2b_1} \pmod{N}$

Furthermore,  $c$  can be re-written as

$$c \equiv (x)(e_1^{b_1} e_2^{b_2}) \pmod{N} \quad (1)$$

for unknown pair  $x$ . It is obvious that (1) is the HNP.

## 9. DISCRETE LOGARITHM PROBLEM ANALYSIS

Since  $\text{DLog}_{g_1}(g_2)$  modulo  $N$  does not exist, an immediate DLP scenario does not occur with the equations  $(e_1, e_2, B_1, B_2)$ . However, assume that  $h$  is a primitive root in  $\mathbb{Z}_N$  such that we have  $h^{z_1} \equiv g_1 \pmod{N}$  and  $h^{z_2} \equiv g_2 \pmod{N}$ . One would now have the following equations:

$$e_1 \equiv h^{z_1 a_1 + 2z_2 a_2} \pmod{N} \quad (2)$$

$$e_2 \equiv h^{z_1 a_2 + z_2 a_1} \pmod{N} \quad (3)$$

Upon multiplying  $z_2$  on the DLP solution for (2) and  $z_1$  on the DLP solution for (3), one could proceed to obtain

$$2z_2^2 a_2 - z_1^2 a_2 \equiv (2z_2^2 - z_1^2) a_2 \pmod{\phi(N)} \quad (4)$$

Assuming  $\gcd(2z_2^2 - z_1^2, \phi(N)) = 1$  one would obtain  $a_2$ . Thus, the existence of such primitive root  $h \in \mathbb{Z}_N$  is the key towards the above cryptanalysis direction.

Recall that  $N$  is a product of small primes. With high probability  $\mathbb{Z}_N$  does not have a primitive root. As such, an adversary would now search for an element  $h_1 \in \mathbb{Z}_N$  such that the random walk of  $h_1$  generates both  $g_1$  and  $g_2$  in  $\mathbb{Z}_N$ . The complexity is  $O(N)$ .

## 10. DERIVING THE SECURITY LEVEL OF KAZ-KEM

The challenge faced by the adversary is to retrieve  $(a_1, a_2)$  from  $(e_1, e_2)$ . It is protected by the HNP.

Due to the strategies during key generation, we have the complexity  $O(a_1) > O(2^k)$  where  $k$  is the chosen security level, either 128, 192 or 256. When deploying Grover's algorithm on a quantum computer, the complexity to obtain  $N_1$  is greater than  $O(2^{\frac{k}{2}})$ .

## 11. IMPLEMENTATION AND PERFORMANCE

### 11.1 Key Generation, Encapsulation and Decapsulation Time Complexity

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

### 11.2 Parameter sizes

We will utilize  $g_1 = 65563$  and  $g_2 = 65617$ .

NIST Security Level	Security level, $k$	Number of primes in list $P$ , $j$	Order $(O_{g_1N}, O_{g_2N})$ size (bits)	Public key size $(e_1, e_2)$ (bits)	Private Key Size, $(a_1, a_2)$ (bits)	Ciphertext Size, $C$ (bits)
1	128	65	$\approx (128, 128)$	$2 \times 432 = 864$	256	$3 \times 432 = 1,296$
3	192	96	$\approx (192, 192)$	$2 \times 702 = 1,404$	384	$3 \times 702 = 2,106$
5	256	122	$\approx (256, 256)$	$2 \times 942 = 1,884$	512	$3 \times 942 = 2,826$

Table 1

### 11.3 Key Generation, Encapsulation and Decapsulation Ease of Implementation

The algebraic structure of KAZ-KEM 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, Encapsulation and Decapsulation 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 encapsulations and 100 decapsulations:

Security level	Time (ms)		
	Key generation	Encapsulation	Decapsulation
128 KAZKEM01	85	87	2
192 KAZKEM03	95	102	7
256 KAZKEM05	115	128	16

Table 2

## 12. ADVANTAGES AND LIMITATIONS

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

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

### 12.1 Key Length

KAZ-KEM key length is comparable to non-post quantum algorithms such as ECC and RSA. For 256-bit security, the KAZ-KEM key size is 1,884-bits. ECC would use 521-bit keys and RSA would use 15,360-bit keys.

### 12.2 Speed

KAZ-KEM'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 encapsulation and decapsulation is  $O(n^3)$  where parameter  $n$  here is the input length.

### 12.3 No Decapsulation Failure

From the proof of correctness, the probability of decapsulation failure is 0.

## **12.4 Limitation**

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

1. Based on not widely used problem, the Hidden Number Problem (HNP).

### **12.4.1 Based on not widely used problem, Hidden Number Problem (HNP)**

The HNP 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-KEM key encapsulation mechanism exhibits properties that might result in it being a desirable post quantum key encapsulation mechanism scheme.

To this end, the security is based on the HNP, which is not a widely used problem. We opine that, the acceptance of HNP 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-KEM key encapsulation mechanism, either findings that nullify its suitability as a post quantum key encapsulation mechanism scheme or findings that could enhance its deployment and use case in the future.

## 14. ILLUSTRATIVE FULL SIZE TEST VECTORS

The following are parameters that illustrate KAZ-KEM for 256-bit security.

### System Parameters

$N$  :

294218183941473459350361361353913759940241264053255766722273980374935594520081162  
835947090690978803191179463432813576314475560419038845862081616787105974697279997  
461798630453885591474074570682758159149149838963927578786839191890758982695509398  
68181179868469970964809582599153788719655

$g_1$  :

65563

$g_2$  :

65617

$O_{g_1N}$  :

99154693887499828557116081873795155652147461554242228686027806044656980768000  $\approx 2^{256}$

$O_{g_2N}$  :

297464081662499485671348245621385466956442384662726686058083418133970942304000  $\approx 2^{258}$

### Key generation

$a_1$  :

106732665057690615308701680462846682779480968671143352109289849544853387479432  $\approx 2^{256}$

$a_2$  :

323977999869557469144432644564867070808648082655844398809350209560252032063067  $\approx 2^{258}$

$A_1$  :

2233544582721045676693011394549873787904012851535551527301434103341770697560585094  
5603279213861038061473547836662152156897582176161121337100328029338821477826864792  
3029668381669914707000451938708095286393035451418857974283477712622569010106771729  
83145211500943055751567609604862790584

$A_2$  :

4135616808611205881148255230294436189247413503530220847281193647638518401538759923  
1425122373020822819983892047637816038594393882508743390176001914833826162164971819  
4469580842084651667854070821842231206950576819567963832750950246366015677565947284  
7883203846070704631177532370571404972

## Encapsulation

$b_1$  :

104066810900827461749919062303612358728884553709626539777574910471434356483636  $\approx 2^{256}$

$b_2$  :

248850397338699573933677633053195073562613307128507861912091178164029894782579  $\approx 2^{258}$

$B_1$  :

223825736993560944132345979730565432058984068600004548050567783316925477566596  
1587539967630463605399357436041575972435686611108191252847835915953890505769428  
1809185509866293571698026166515243079364633439848615216399768397265290249941656  
065475013787232567503576826032663722628368318488

$B_2$  :

2013353913179063411985909378891799706527108509807100232786670501999754915172004  
4448645543616500739725578538358097804071979410094932239288165083346508333084956  
4731760823860952586714707128231556281484155300119911208568579376312883639210770  
7243323256594800509549400217551725685727971102

$m$  :

2812698883649576134746893449974635753433639354353163897518348994748367336777549  
1172084040619673294316171931336792845269354935973394099124895338365061261246926  
9743533538879207868233984288204362277215794067931887899322666286078672284275791  
20516558372194772279939789361858815143186326574

$A_1^{b_1} A_2^{b_2} \pmod{N}$  :

2839049513089992979523652326478572339862829163948821263717471002635385021662072  
3204951988974462856799751398539433923678244887018338734726974637949968033361690  
0059817411671705327547919119560538371544241635372051438389954727571197832204568  
53471592961960916269366312661171237372284085293

$c \equiv m + A_1^{b_1} A_2^{b_2} \pmod{N}$  :

2597694192195575777024342229463530838657684292288310832900318092535657170964546  
5128429033935220842609827143986654989595899082730500097142736542548779807700080  
9668992194566578384009194979690505707268576547820141553519936091827431976555180  
76205069573828331826502742876313114027591193642

## Decapsulation

$B_1^{a_1} B_2^{a_2} \pmod{N}$  :

2839049513089992979523652326478572339862829163948821263717471002635385021662072  
3204951988974462856799751398539433923678244887018338734726974637949968033361690  
0059817411671705327547919119560538371544241635372051438389954727571197832204568  
53471592961960916269366312661171237372284085293

$c - B_1^{a_1} B_2^{a_2} \pmod{N}$  :

2812698883649576134746893449974635753433639354353163897518348994748367336777549  
1172084040619673294316171931336792845269354935973394099124895338365061261246926  
9743533538879207868233984288204362277215794067931887899322666286078672284275791  
20516558372194772279939789361858815143186326574

## References

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.