Research Article

A public key cryptosystem and signature scheme based on numerical series

Khaled A. Nagaty[1]

Received: 20 March 2019 / Accepted: 20 December 2020 / Published online: 1 January 2020

Abstract

A public key cryptosystem and signature scheme based on a proposed key exchange algorithm are introduced. The key exchange algorithm is based on the difficulty of calculating the *n*th partial sum of infinite numerical series where no exponentiation computation is required to share a secret key between two parties as in the Diffie–Hellman algorithm. With the proposed public key cryptosystem ciphering and deciphering of messages, online signing documents and verifying signatures do not require exponentiation computation while providing higher security level than the state-of-the-art cryptosystems that depend on the difficulty of discrete logarithmic problem or factorizing large prime numbers. The proposed cryptosystem and signature scheme do not depend on elliptic curve cryptography or RSA cryptography that are computationally too slow, this makes the proposed cryptosystem and signature scheme computationally faster, easier to implement and more practical to be used in online transactions. It also provides a higher level of security as it provides forward secrecy and using large size symmetric keys to encrypt and decrypt messages in a significantly short time. Moreover, the proposed signature scheme can be used as a cryptographic hash function as the hash value significantly changes when a single letter in the document is changed. In comparison with the state-of-the art the experimental results show the superiority of the proposed key exchange algorithm and public key cryptosystem.

Keywords Cryptography · Key exchange · Public key · Partial sum · Subset sum · Diffie–Hellman algorithm · Numerical series · Security · Signature scheme

1 Introduction

In 1975 Diffie–Hellman proposed the public key cryptosystem [1]. It is a key exchange algorithm that is used to share public and private keys for ciphering and deciphering of messages in a secure way. To our best knowledge all cryptosystems are based on the difficulty of three problems namely the problem of factoring large integers, problem of computing *n*th residue classes and problem of computing discrete logarithms. Cryptosystems that are based on factoring large integers are the RSA cryptosystem that was described by Rivest–Shamir–Adleman [2] and probabilistic ciphering system that was described by Goldwasser and Micali [3]. Cryptosystems that are based on the difficulty of computing the *n*th residue classes are Paillier ciphering scheme described in [4], BGN described in [5] by Boneh–Goh–Nissim and Okamoto–Uchiyama in [6] which resembled the Paillier ciphering scheme. Taher ElGamal described a public cryptosystem and signature scheme based on the difficulty of computing discrete logarithms in [7]. Athena et al. introduced elliptic curve cryptography (ECC) uses the properties of elliptic curves to generate keys in [8]. In [9] Freeman used prime-order elliptic curve groups to construct secure pairing-based cryptosystem. In [10], Bahadori et al. proposed a method to speed up a secure generation of RSA public and private key values that are equipped on smart cards. In [11], Blackburn

✉ Khaled A. Nagaty, khaled.nagaty@bue.edu.eg | [1]Faculty of Informatics and Computer Science, The British University in Egypt, El-Sherouk City, Egypt.

SN Applied Sciences (2020) 2:142 | https://doi.org/10.1007/s42452-019-1928-8

et al. involved the certification authority to generate RSA-keys. The certification authority (CA) does not now about the user's private key. In [12], Sharma et al. presented a modified version of subset-sum problem over RSA algorithm called RSA algorithm using modified subset sum cryptosystem that relies on the fact that given a set of integers, does the sum of some non-empty subset equal exactly zero. However, this method still relies on RSA cryptography which makes it computationally too slow. Moreover, it is useless for authentication by cryptographic signing of documents. In [13], Ge et al. presented an efficient authentication method for secure key exchange among set of devices that have a single trusted administrator. In [14], Nagar et al. proposed a new method to exchange the values of the RSA-key offline generations between gateways. All generated key values are saved in tables within a database. Each key value includes the public and private key values. In [15], Patidar et al. proposed a modified form of RSA algorithm to speed up the implementation of RSA algorithm during data exchange across the network. The authors introduced a third prime number to produce modulus n that is not easily decomposed by intruders. In [16], Ashutosh Kumar Dubey et al. proposed a novel cloud-user security method which is based on RSA cryptography and MD5 for resource attestation and sharing in java environment. In [17], Wuling Ren et al. proposed a hybrid-ciphering algorithm based on the integration between DES and RSA to enhance the security of data transmission in Bluetooth communication. In this approach, DES algorithm is used for data transmission because of its higher efficiency in block ciphering, and RSA algorithm is used for the ciphering of the key of the DES because of its management advantages in key cipher. In [18], Arazi integrated the Diffie–Hellman key exchange into digital signature algorithm (DSA) by replacing a message in digital signature algorithm (DSA) with exchange key in the Diffie–Hellman to achieve mutual authentication of exchanged keys. In [19], Harn et al. proposed an enhanced integration of the Diffie–Helman and DSA for key exchange by providing three round protocols. In [20], Bernstein, released a key exchange protocol referred to as, Curve25519 that is an elliptic curve Diffie–Hellman key exchange protocol to establish a shared secret. More details on Curve25519 found in [21]. In [22, 23] a high-performance elliptic curve referred to as FourQ was introduced by Costello, C. and Longa, P. FourQ is 128-bit security level. In [24], Alvarez et al. showed that standard Diffie–Hellman is little slower than the elliptic curve variants in key pair generation and much slower than elliptic curve invariants in secure key exchange. In [25], Kumar et al. proposed an enhanced Diffie–Hellman algorithm for reliable key exchange by employing the concept of primitive roots. In this algorithm, the sender and receiver to obtain a shared-secret key use conventional Diffie–Hellman algorithm. Then the sender and receiver find the primitive root of their shared-secret key. The Diffie–Hellman algorithm is used again to get a second secret key. Finally, the sender and receiver multiply their second shared-secret key with their own random number and exchange it for getting a new-shared secret key for every message and even for the same message. This enhanced Diffie–Hellman algorithm still relies on the conventional Diffie–Hellman algorithm for two rounds and use exponentiation computation in employing the primitive roots which make this algorithm computationally too slow. All the previous public key cryptosystems based on the standard Diffie–Hellman key exchange algorithm that based on the difficulty of discrete logarithmic problem. All ciphering schemes based on discrete logarithms require exponentiation operations for ciphering and deciphering which is computationally time consuming. The modified subset sum cryptosystem relies on RSA cryptography that requires the generation of two large prime numbers p and q which is the most difficult process in the RSA algorithm. All methods rely on RSA cryptography are computationally slow. Moreover, the modified subset sum cryptosystem does not allow for secure key exchange between two ends of a conversation. The proposed key exchange algorithm allows for secure exchange of shared secret key that does not require exponential time computation because no exponentiation operations are used. RSA cryptosystems based on elliptic curves are difficult to implement and computationally too slow. The elliptic curve Diffie–Hellman (ECDH) key exchange algorithm allow for two parties to exchange a shared secret key but it is also based on Diffie–Hellman algorithm in integration with elliptic curve cryptography which makes this cryptosystem computationally too slow. Compared to all above cryptosystems, the proposed public key cryptosystem and signature scheme do not depend on exponentiation computation that makes them simpler, computationally faster and provides a higher security level.

This paper introduces a new public key cryptosystem that based on a proposed secure key exchange algorithm and signature scheme that both rely on the difficulty of computing the nth partial sum of an infinite numerical series over a finite field. In case the infinite numerical series, the first term in this series a and large number of terms k are all kept secret i.e. unknown, it believes that for an attacker to compute the nth partial sum of an infinite numerical series is computationally intractable. The real contribution of this paper is a new algorithm for a secure key exchange other than the standard Diffie–Hellman algorithm or the enhanced Diffie–Hellman algorithm which both rely on exponentiation computation which makes them computationally too slow. The proposed key exchange algorithm depends on the difficulty of calculating the nth partial sum of infinite numerical series without using exponentiation computation. This makes the proposed algorithm simple, computationally faster, easy to implement and can be used for online ciphering and deciphering of messages and online signing documents. The proposed signature verification scheme can be used as a cryptographic hash function as the hash value significantly changes if there is any change in the document.

This paper organized as follows: Sect. 2 explains infinite numerical series over a finite field and a way to find the nth partial sum. Section 3 shows how to implement the proposed key exchange algorithm based on the difficulty of computing the nth partial sum of an infinite numerical series then shows how the proposed key exchange algorithm used in ciphering and deciphering of messages. Section 4 introduces a proof of the secure key exchange algorithm. Section 5 introduces a new digital signature scheme based on infinite numerical series over a finite field. Section 6 introduces a security analysis of the new digital signature scheme. Section 7 gives some properties of the proposed public key cryptosystem and the signature scheme. Section 8 dedicated for the implementation and experimental results. Section 9 contains conclusion.

2 Sum of series

Definition 1 Suppose we have an infinite sequence of numbers:

$$u_1, u_2, u_3, \dots, u_n, \dots$$

The expression

$$u_1 + u_2 + u_3 + \cdots + u_n + \cdots \tag{1}$$

is called a numerical series, the numbers $u_1, u_2, u_3, \dots, u_n, \dots$ are called the terms of the series [26].

Definition 2 The sum of a finite number of terms (the first n terms) of a series called the nth partial sum of the series such that [26]:

$$s_n = u_1 + u_2 + u_3 + \cdots + u_n \tag{2}$$

If there exists a finite limit:

$$s = \lim_{n\to\infty} s_n \tag{3}$$

Then s called the sum the series in Eq. (1) and we say that the series converges.

If the $\lim_{n\to\infty} s_n$ does not exist i.e. $s_n \to \infty$ as $n \to \infty$ then the series has no sum and we say this series diverges.

3 Proposed public key system

Suppose that sender A and receiver B wants to share a secret key using two different infinite numerical series t_1 and t_2. Let A use series t_1, B use series t_2 and p is a large prime number that chosen by A. The two infinite numerical series t_1 and t_2 can randomly be chosen from a pool of infinite numerical series, the first term a and large number of terms k for each series are randomly chosen using true random number generator so that the last term k in each series is much greater than the first term a i.e. $k \gg a$. Assume that user A randomly chose the first term a and large number of terms k in numerical series t_1. In addition, user B randomly chose the first term b and large number of terms r in numerical series t_2. Assume S_A is the kth partial sum of k terms in t_1 and S_B is the rth partial sum of r terms in t_2. Both users A and B can securely share a secret key as follows:

3.1 Key exchange algorithm

- Select a very large prime number p.

Private keys

- S_A: computed by user A.
- S_B: computed by user B.

Public keys

- Let A computes y_A such that:

$$y_A = S_A \, mod \, p \tag{4}$$

- A sends (y_A, p) to B.
- Let B computes y_B as follows:

$$y_B = S_B \, mod \, p \tag{5}$$

- B sends y_B to A.
- A and B compute y_{AB} as follows:

$$y_{AB} = y_A \cdot y_B \tag{6}$$

Shared key

- A computes S_{AB} as follows:

$$S_{AB} = ((y_{AB} \cdot y_{\boldsymbol{B}}) \cdot S_A) \, mod \, p \tag{7}$$

- B computes S_{AB} as follows:

$$S_{AB} = ((y_{AB} \cdot y_{\boldsymbol{A}}) \cdot S_B) \, mod \, p \tag{8}$$

Note that:

$$((y_{AB} \cdot y_{\boldsymbol{B}}) \cdot S_A) \, mod \, p = ((y_{AB} \cdot y_{\boldsymbol{A}}) \cdot S_B) \, mod \, p \tag{9}$$

For an attacker it is difficult to compute the shared key S_{AB} as it is equivalent to computing the *k*th partial sum and *r*th partial sum of two different infinite numerical series t_1 and t_2 over two different finite fields without knowing the first term and the number of terms in each numerical series.

3.2 Ciphering

Suppose user *B* wants to encipher message *m* and sends it to user *A*. The ciphering algorithm works as follows:

- *B* computes the ciphered message m' using the shared secret key S_{AB}:

$$m' = m \cdot S_{AB} \tag{10}$$

- *B* sends m' to *A*.

Note that new S_A, S_B and S_{AB} are generated for every message to improve security. So S_A, S_B and S_{AB} are called ephemeral keys.

3.3 Deciphering

The deciphering algorithm works as follows:

- *A* receives m'.
- *A* uses the shared secret key S_{AB}.
- *A* recovers *m* by dividing m' by S_{AB}:

$$m = \frac{m'}{S_{AB}} = \frac{m \cdot S_{AB}}{S_{AB}} \tag{11}$$

4 Proof of secure key exchange algorithm

The proposed key exchange algorithm is based on the difficulty of calculating the *n*th partial sum of an infinite numerical series problem. Calculating the *n*th partial sum is based on the subset sum problem that is NP-complete [27]. The subset sum problem is the following: given a set of *n* positive integers $\{a_1, a_2, \ldots, a_n\}$ and a positive integer *S*, determine whether there is a subset of a_i that sum to *S*. The subset sum problem is a decision problem in the complexity theory that states that NP-complete problems are the hardest problems in NP in the sense that they are at least as difficult as every other problem in NP [27]. The problem of calculating the subset sum problem can be reduced to the *n*th partial sum of an infinite numerical series problem in polynomial time i.e. the *subset sum problem* $\leq_P$ *n*th *partial sum problem*. The following algorithm reduces the subset sum problem to *n*th partial sum problem in polynomial time as follows:

- *Step 1* Let *m* be the number of terms in numerical series *t* such that $m < n$, where *n* is the number of positive integers in the subset sum problem.
- *Step 2* Select *z* terms from the set of positive integers $\{a_1, a_2, \ldots, a_n\}$ of the subset sum problem such that $z \gg m$.
- *Step 3* Sort all the *z* terms in an ascending order.
- *Step 4* Repeat for each (z_i, z_{i+1}) in the sorted list of z terms:

  If $(z_i = z_{i+1})$ remove z_{i+1} from the list.

- *Step 5* Choose the first *m* terms form the list obtained from step 4 as the terms of the numerical sequence *t*.

The time complexity of step 1 is O(1), the time complexity of step 2 is O(z), the time complexity of sorting in step 3 is $O(z^2)$, the time complexity of step 4 is O(z) while the time complexity of step 5 is O(m). Therefore, the time complexity of the reduction algorithm is:

$$\begin{aligned} &O(1) + O(z) + O(z^2) + O(z) + O(m) \\ &\quad = O(z^2) + 2O(z) + O(1) + O(m) \end{aligned} \tag{12}$$

Since m ≪ z therefore O(m) ≪ O(z) and since $O(z^2)$ is the highest order in Eq. (12), therefore $O(z^2)$ is dominating the time complexity of the reduction algorithm. This means that the reduction algorithm required to reduce the subset sum problem to the *n*th partial sum problem is a polynomial time algorithm. Since the subset sum problem is NP-complete therefore calculating the *n*th partial sum of an infinite numerical series *t* is also NP-complete problem.

For an attacker to compute the private key S_A for user *A* or the private key S_B for user *B* who are involved in a conversation is equivalent to the problem of calculating the *n*th partial sum of an infinite numerical series without knowing what numerical series is used by each user, the first term and number of terms in each numerical series which is proved to be NP-complete. This proves the security of the proposed key exchange algorithm, proves the security of the proposed cryptosystem and proves the security of the proposed signature scheme.

Lemma *Let S be the nth partial sum of an infinite numerical series t. Since the problem of computing S is NP-complete then multiplying S by a constant C is also NP-complete.*

Proof Let $t = a_1 + a_2 + \cdots + a_n + \cdots$ is an infinite numerical series over a finite field. Let S be the nth partial sum of t as follows:

$$S = \sum_{i=a_1}^{a_n} a_i \tag{13}$$

Multiply S by a constant C to get S_C:

$$S_C = C \cdot S \tag{14}$$

Therefore:

$$\begin{aligned} S_C =& C \cdot \sum_{i=a_1}^{a_n} a_i \\ =& \sum_{i=a_1}^{a_n} Ca_i \end{aligned} \tag{15}$$

Substitute $d_i = Ca_i, \forall i = 1, \ldots, n$

Therefore:

$$S_C = \sum_{i=a_1}^{a_n} d_i \tag{16}$$

which still nth partial sum of the infinite numerical series t that is NP-complete. Therefore S_C is also NP-complete problem.

5 Proposed digital signature scheme

A new signature scheme is described where the private key y_r depends on the sequence (t_1, a, r) such that t_1 is an infinite numerical series over a finite field, a is the first term in t_1 and r is a large number of terms. The public key y_k depends on a different sequence (t_2, b, k) such that t_2 is a different infinite numerical series over a different finite field, b is the first term in t_2 and k is a large number of terms. Assume m is a document to be signed by user A such that:

$0 \leq m \leq p - 1$ where p is a very large prime number.

- A generates the private key y_r as follows:

$$y_r = S_r \ mod\ p \tag{17}$$

  where S_r is the rth partial sum of numerical series t_1.
- A generates the public key y_k as follows:

$$y_k = S_k \ mod\ p \tag{18}$$

  where S_k is the kth partial sum of numerical series t_2.

5.1 The signing procedure

- A signs document m with the private key y_r:

$$m_r = m \cdot y_r \tag{19}$$

- A signs document m with the public key y_k:

$$m_k = m \cdot y_k \tag{20}$$

- A calculates m_{rk} as follows:

$$m_{rk} = m_r \cdot m_k \tag{21}$$

- The signature for document m is (m_{rk}, m_r).
- A publishes the signature of document m and his public key y_k for any user to verify the signature.

5.2 The verification procedure

Assume user B wants to verify the signature of user A on document m by using the published sequence (m_{rk}, m_r, y_k) as follows:

- B computes $m_k^{'}$ as follows:

$$m_k' = m \cdot y_k \tag{22}$$

- B computes $m_{rk}^{'}$ as follows:

$$m_{rk}' = m_k' \cdot m_r \tag{23}$$

- If $m_{rk} = m_{rk}^{'}$ then the signature of user A is verified otherwise the signature is unverified or the original document m may be changed.

6 Security analysis

The security of the proposed digital signature scheme based on the assumption that given two infinite numerical series t_r and t_k it is computationally intractable to compute m_r from its partial sums S_r and compute m_k from its partial sum S_k given the following unknowns:

- The numerical series used.
- The first term a of series t_r.
- Large number of terms r of series t_r which are randomly chosen from $[1, n_r]$ where n_r is the total number of terms in the infinite numerical series t_r.
- The first term b of series t_k.
- Large number of terms k of series t_k which are randomly chosen from $[1, n_k]$ where n_k is the total number of terms in the infinite numerical series t_k.
- A very large prime number p.

In this section, we analyse the possible attacks on the signature scheme which are equivalent to computing the nth partial sum of an infinite numerical series over a finite field. Some of these attacks are focused on recovering the secret key y_r and other attacks focus on forging the signature (m_{rk}, m_r). The attacks to recover private key y_r is difficult because it requires recovering S_r from its y_r. Since y_r depends on S_r which is not used twice to sign a new document, and by using very large prime number p it is difficult for an intruder to recover S_r from y_r.

For attacks focus on forging the signature (m_{rk}, m_r) of user A a forger may try to find the sequences (t_r, a, r) and (t_k, b, k). The forger tries different infinite numerical series with different random values for a, r to compute S_r and different random values for b, k to compute S_k which are equivalent to compute the rth partial sum of infinite numerical series t_r and the kth partial sum of infinite numerical series t_k with unknown parameters a, r and b, k respectively. Suppose there is a pool of l different infinite numerical series with different finite fields to choose from, therefore the probability for choosing the correct infinite numerical series t_r is $\frac{1}{l}$. The probability to recover m_r is equivalent to the joint probability $p\left(t_r, n_r\right)$ of choosing the correct numerical series t_r and the correct number of terms n_r. As choosing t_r is independent of choosing n_r therefore the joint probability $p\left(t_r, n_r\right)$ is the product of the probabilities $p\left(t_r\right)$ and $p\left(n_r\right)$ where $p\left(n_r\right)$ is the probability of choosing the correct infinite numerical series t_r and $p\left(n_r\right)$ is the probability of choosing the correct n_r terms to calculate the partial sum S_r, therefore:

$$\begin{aligned} p(t_r, n_r) &= p(t_r) \times p(n_r) \\ &= \frac{1}{l} \times \frac{n_r}{N} = \frac{n_r}{l \times N} \end{aligned} \tag{24}$$

where N is the size of the numerical set an attacker can choose the n_r terms from it.

In order for the attacker to increase the possibilities to pick up the correct n_r terms, the size of the set of the numerical set N must increase.

Therefore:

$$\lim_{N \to \infty} \frac{n_r}{l \times N} = 0 \tag{25}$$

Also, the probability to recover m_k is equivalent to the joint probability $p\left(t_k, n_k\right)$ of choosing the correct numerical series t_k and the correct number of terms n_k. Since choosing t_k is independent of choosing n_k therefore the joint probability $p\left(t_k, n_k\right)$ is the product of the probabilities $p\left(t_k\right)$ and $p\left(n_k\right)$ where $p\left(n_k\right)$ is the probability of choosing the correct infinite numerical series t_k and $p\left(n_k\right)$ is the probability of choosing the correct n_k terms to calculate the partial sum S_k, therefore:

$$\begin{aligned} p(t_k, n_k) &= p(t_k) \times p(n_k) \\ &= \frac{1}{l} \times \frac{n_k}{N} = \frac{n_k}{l \times N} \end{aligned} \tag{26}$$

where N is the size of the numerical set an attacker can choose n_k terms from it.

For the attacker to increase the possibilities to pick up the correct n_k terms, the size of the set of elements N must increase.

Therefore:

$$\lim_{N \to \infty} \frac{n_k}{l \times N} = 0 \tag{27}$$

Since choosing the infinite numerical series t_r, t_k is independent of each other therefore the probability to recover m_{rk} from its components m_r and m_k is the product of the probabilities to recover m_r and m_k such that:

$$p(t_r, t_k, n_r, n_k) = p(t_r, n_r) \times p(t_k, n_k) \tag{28}$$

$$p(t_r, t_k, n_r, n_k) = \frac{n_r}{l \times N} \times \frac{n_k}{l \times N} = \frac{n_r \times n_k}{i^2 \times N^2} \tag{29}$$

Therefore:

$$\lim_{N \to \infty} \frac{n_r \times n_k}{i^2 \times N^2} = 0 \tag{30}$$

where N is the size of the numerical set an attacker can choose n_r, n_k terms from it.

We believe it is not feasible to calculate y_r by solving the following equation using S_r:

$$y_r = S_r \, mod \, p \tag{31}$$

Also, we believe it is not feasible to calculate y_k by solving the following equation using S_k:

$$y_k = S_k \, mod \, p \tag{32}$$

The proposed cryptosystem provides forward secrecy. Forward secrecy protects past encrypted messages from future compromises of shared secret keys. By generating a unique shared secret key for each message to be ciphered then the compromise of a single shared secret key will not affect any message other than that ciphered using that particular shared secret key. Since the proposed cryptosystem relies on the difficulty of computing the nth partial sum of an infinite numerical series this allows for an infinite numerical space to choose different values for the first term and the number of terms in the numerical series each time a new message is encrypted.

A semantic security ciphering scheme must fulfill the following requirement which is: "It is infeasible to learn anything about the plaintext from the cipher text". In other words: "Whatever an eavesdropper can compute about the plain text given the cipher text, he can also compute without the cipher text" [28]. Semantic security is commonly defined by the following game:

Let M be the set of all possible messages and T be the set of all possible numerical series.

- *Initialize* The challenger A gives the public key key_A to adversary B and keeps the shared key S_{AB} with himself.
- *Phase 1* The adversary asks different ciphering queries to encrypt message $m_i \in M, i = 1 \ldots N$. Each query uses different infinite numerical series $t_i \in T, i = 1 \ldots N$ with different first term a_i and different number of terms k_i such that:

  $m_i' = E(m_i, y)$ where E is the ciphering algorithm.

- *Challenge* When the adversary decides that phase 1 is over he chooses two equal length plaintext messages (m_i, m_j) such that $i \neq j$ on which he wishes to be challenged. The challenger picks message m_i and sends the adversary $m_i' = E\left(m_i, S_{AB}\right)$ as a challenge to the adversary.
- *Phase 2* The adversary issues more ciphering queries as in Phase 1.
- *Guess* The adversary outputs a guess m_i'' and wins the game if $m_i'' = m_i'$.

7 Proposed scheme properties

A. The proposed ciphering scheme has an additive homomorphic property: by the distribution property, it is clear that ciphering the sum of m_1 and m_2 produces a ciphertext equivalent to the sum of $E\left(m_1\right)$ and $E\left(m_2\right)$ so that:

$$E(m_1 + m_2) = E(m_1) + E(m_2) \tag{33}$$

Proof Given two different messages m_1 and m_2:

$$\begin{aligned} E(m_1 + m_2) =& (m_1 + m_2) \cdot S_{AB} \\ =& m_1 \cdot S_{AB} + m_2 \cdot S_{AB} \\ =& E(m_1) + E(m_2) \end{aligned} \tag{34}$$

Unfortunately, the proposed ciphering scheme has no multiplicative homomorphic property.

B. For the secure key exchange, ciphering and deciphering of messages no exponentiation computations are required which make the proposed scheme significantly faster and more practical than the state-of-the-art key exchange algorithms and cryptosystems that are based on the difficulty of discrete logarithmic problem or factoring large prime numbers while producing a higher security level.

C. For signing the documents no exponentiation computations are required and the same for signatures verification.

D. The signature scheme can be used as cryptographic hash function to verify that a document is not tampered with as the hash value of a document is changed significantly if there is any change in the document. For example, if an upper case letter is changed to lower case letter or vice versa the hash value of the document is changed significantly.

See "Appendix 2".

8 Implementation

The proposed key exchange cryptosystem is implemented on Intel Core i3 using MatLab R2017a. In this implementation a random number generator is used to choose the first term and a large number of terms in two different numerical series. The last term generated by the random number generator must be much greater than the first term in each numerical series.

User A enters a very large prime number p to be used in the ciphering and deciphering processes. Assume A chose an infinite numerical series n^2 as follows:

$1^2, 2^2, \ldots, i^2, \ldots$

The partial sum of its k terms starting from the ath term to the kth is:

$$\sum_{i=a}^{k} i^2 \tag{35}$$

where a the first term and k is the last term in the numerical series.

Assume B chose another infinite numerical series $2n + 1$ which is:

$3, 5, 7, 9, \ldots, 2i + 1, \ldots$

The partial sum of its r terms starting from the bth term is:

$$\sum_{i=b}^{r} (2i + 1) \tag{36}$$

where b is the first term and r is the last term in this numerical series.

Note that a, b, r and k are all chosen using a true random number generator such that $k \gg a$ and $r \gg b$.

User A calculates the kth partial sum S_A using his numerical series and calculates y_A using the chosen large prime number p. In this implementation, the length of p is 78 digits for a key of size 128 bits, 92 digits for keys of sizes 256 bits and 512 bits, 184 digits for keys of sizes 1024 bits and 693 digits for 2048 bits.

User A sends (y_A, p) to user B who calculates the rth partial sum S_B using his numerical series then calculates y_B using the large prime number p sent by user A. The function *share_key_generation()* implements the secure key exchange algorithm between the two parties in conversation so that each party will have a shared secret key. The function has no input but its outputs are the public key for user A which is y_A, the private key for user A which is S_A, the public key for user B which is y_B and the private key for user B which is S_B and the shared secret key S_{AB} between users A and B. The function *cipher*() implements the ciphering of message m by user B using the shared secret key with user A. This function has no inputs but the output is the ciphered message *mciph*. The function decipher **(***mciph***)** is implemented by the receiver user A, its input the ciphered message *mciph* and its output the deciphered message m. See "Appendix 1" for a practical example.

To significantly minimize the generation time for the kth partial sum S_A and rth partial sum S_B for these keys are evaluated using the number of terms used to generate keys of sizes < 512-bits. Finally, the final values for S_A and S_B are obtained by multiplying S_A with a large constant C_A and multiplying S_B with a large constant C_B to obtain the required key size. These large constants are selected empirically and must be changed for every new message to improve the security level.

Table 1 shows a comparison of key generation time, ciphering time, deciphering time and total execution time in milliseconds between the proposed cryptosystem, modified subset sum cryptosystem and standard RSA. Figure 1 shows a comparison in seconds between key generation time of 128-bit single key pair RSA, DH, P256, Curve25519, FourQ and 128-bits key of the proposed key exchange algorithm.

The proposed signature scheme is implemented using the functions: keys_generation() to generate the public key y_k and private key y_r, sign_document() and verify_signature().

Table 1 Comparison between the proposed cryptosystem, RSA algorithm using modified subset sum (MSSRSA) cryptosystem and the RSA cryptosystem

Ciphering/deciphering scheme	Key size	Number of elements	Key generation time (ms)	Ciphering time (ms)	Deciphering time (ms)	Total execution time (ms)
MSSRSA	128	32	16	235	78	329
RSA	128	32	16	94	62	172
Proposed scheme	128	32	0.216	4	0.245	5
MSSRSA	128	64	15	94	47	156
RSA	128	64	16	16	47	63
Proposed scheme	128	64	0.228	7	0.839	8
MSSRSA	256	32	–	–	–	–
RSA	256	32	–	–	–	–
Proposed scheme	256	32	0.267	3	0.241	4
MSSRSA	256	64	–	–	–	–
RSA	256	64	–	–	–	–
Proposed scheme	256	64	0.225	7	0.524	8
MSSRSA	512	32	125	1203	1766	3049
RSA	512	32	109	563	1719	2391
Proposed scheme	512	32	2.645	4	0.622	7
MSSRSA	512	64	63	344	875	1282
RSA	512	64	63	141	859	1063
Proposed scheme	512	64	1.724	8	1	11
MSSRSA	512	128	78	172	453	703
RSA	512	128	47	78	422	547
Proposed scheme	512	128	1.907	16	3	2
MSSRSA	1024	32	688	5407	11,328	17,423
RSA	1024	32	688	1719	12,172	14,579
Proposed scheme	1024	32	16	4	0.247	20
MSSRSA	1024	64	453	6593	5735	12,781
RSA	1024	64	453	2968	5688	9109
Proposed scheme	1024	64	14	8	0.803	23
MSSRSA	1024	128	562	516	2859	3937
RSA	1024	128	515	219	3344	4078
Proposed scheme	1024	128	16	14	2	32
MSSRSA	1024	512	12,812	187	781	13,780
RSA	1024	512	281	47	735	1063
Proposed scheme	1024	512	8	76	3	87
MSSRSA	2048	32	3735	9563	85,140	98,438
RSA	2048	32	3719	3688	85,672	93,079
Proposed scheme	2048	32	5	8	0.408	13
MSSRSA	2048	64	1563	3625	42,437	47,625
RSA	2048	64	1563	1688	43,234	46,485
Proposed scheme	2048	64	5	10	1	16
MSSRSA	2048	128	7125	6266	20,734	34,125
RSA	2048	128	7078	3829	21,406	32,313
Proposed scheme	2048	128	6	13	1	20
MSSRSA	2048	512	17,797	797	5281	23,875
RSA	2048	512	7703	375	6172	14,250
Proposed scheme	2048	512	7	53	8	68
MSSRSA	2048	1024	29,704	422	2797	32,923
RSA	2048	1024	2891	203	3406	6500
Proposed scheme	2048	1024	12	107	13	132

See "Appendix 2" for a practical example.

```
function share_key_generation()
/*User A calculates y_A
key_size=input('Enter key size:');
p = input('Enter the value of a very
large prime number p:');
a = rand();
k = rand();
if (key_size<512) C_A = 1;
else
  C_A=input('Enter value of C_A:');
end
S_A = 0;
for i = a : k
   S_A = S_A + i * i;
end
S_A = C_A * S_A;
y_A = mod(S_A ,p);
/* User A sends (key_size, y_A, p) to user B
/* User B calculates y_B
b = rand();
r = rand();
if (key_size < 512)  C_B = 1;
else
 C_B = input('Enter value of C_B:');
end
S_B = 0;
for i = b : r
   S_B = S_B + 2 * i + 1;
end
S_B = C_B * S_B;
y_B = mod(S_B, p);

/*Users A & B calculate y_AB
y_AB = y_A * y_B;

/*User A calculates shared secret
key S_AB */
S_AB = mod((y_AB * y_B) * S_A, p);

/*User B calculates shared secret
key S_AB */
S_AB = mod((y_AB * y_A) * S_B, p);
```

```
function cipher()
/*User B (sender) uses this function
to cipher message m */
m = input('Enter message:','s');
x = length(m);  c = 0;
for j= 1 : x
 for i=0:122
    if strcmp(m(j), char(i))
    c(j) = i;
 end
end
/*User B (sender) cipher the message
m using the shared secret key S_AB */
for j = 1 : x
 mciph (j) = c(j) * S_AB  /*ciphered
message*/
end
```

```
function decipher(mciph)
/*User A (receiver) uses this
function to decipher message mciph */
x = length(mciph);
/*User A (receiver) decipher mciph
using the shared secret key S_AB */
for j = 1 : x
 m(j) = mciph(j)/ S_AB; /*deciphered
message*/
end
```

```
function keys_generation( )
/*User A calculates private y_r */
p = input('Enter the value of p:');
a = rand();
k = rand();
S_r = 0;
for i = a : r
   S_r = S_r + i * i;
end
y_r = mod(S_r ,p);
/* User B calculates public key y_k
b = rand(); r = rand();
S_k = 0;
for i = b : k
   S_k = S_k + 2 * i + 1;
end
y_k = mod(S_k, p);
```

```
function sign_document(m,y_r,y_k)
/*User A signs document m*/
m = input ('Enter document to be
signed:', 's');
x=length(m);
c=0;
for j= 1:x
  for i=0:122
   if strcmp(m(j),char(i))
   c(j)=i;
  end
end
m_r=0;
m_k = 0;
m_rk= 0;
for j=1:x
 m_r=m_r + c(j) *  y_r(j);
 m_k=m_k + c(j) *  y_k(j);
end
m_rk=m_r *  m_k;
Publish (m_rk, m_r, y_k);
```

```
function verify_sign(m,m_rk,m_r, y_k)
/*User B vrifies signature of A on
document m */
c = 0;
for j= 1:x
 for i=0:122
    if strcmp(m(j),char(i))
    c(j) = i;
   end
end
m_k_1 = 0;
m_rk_1 =  0;
for j= 1:x
    m_k_1 = m_k_1 + c(j) * y_k;
end
m_rk_1 = m_r * m_k_1;
if (m_rk == m_rk_1)
 print('Verified');
else
 print ('Not a verified signature or
document may be changed')
```

From Table 1, it is clear that the average time for generating 128-bits key in RSA and MSSRSA is 16 ms while the average time for generating the same key size using the proposed key exchange algorithm is 0.222 ms which means that it is 98.61% faster than RSA and MSSRSA. The average time to cipher messages of sizes 32 and 64 elements using MSSRSA is 165 ms and using RSA is 55 ms while it takes 6 ms using the proposed cryptosystem. This means that the proposed cryptosystem is 96.36% faster than MSSRSA to cipher a message of 32 elements and 89.09% faster than RSA to cipher the same message. The average time to decipher messages of size 32 elements and 64 elements using the MSSRSA takes 63 ms and using the RSA it takes 55 ms while using the proposed cryptosystem it takes 0.542 ms. This means that the proposed cryptosystem is 99.14% faster than MSSRSA and 99.01% faster than RSA. The total average execution time for MSSRSA is 243 ms, 118 ms for RSA and 7 ms for the proposed cryptosystem. This means that the proposed cryptosystem is faster 97.12% than MSSRSA and 94.07% faster than RSA.

The average time for generating a key of size 512-bits using MSSRSA is 89 ms and by using RSA is 73 ms while the average key generation time for the same key size using the proposed key exchange algorithm is 2 ms which is 97.75% faster than MSSRSA and 97.26% faster than RSA. The average time for ciphering messages of size 32, 64 and 128 elements with 512-bits key size using MSSRSA is 573 ms, 261 ms for RSA and 9 ms for the proposed cryptosystem. This means that the proposed cryptosystem is 98.42% faster than MSSRSA and 96.55% faster than RSA. The average time for deciphering messages of size 32, 64 and 128 elements for MSSRSA is 1031 ms, for RSA is 1000 ms while for the proposed cryptosystem is 2 ms. This means that the proposed cryptosystem is faster than the MSSRSA by 99.81% and faster than RSA by 99.80%. The average total execution time for MSSRSA is 1678 ms, 1334 ms for RSA and 65 ms for the proposed cryptosystem. This means that the proposed cryptosystem is 96.12%, faster than MSSRSA and 95.13% faster than RSA.

The average time for generating a key of size 1024-bits using MSSRSA is 3629 ms and using RSA is 484 ms however by using the proposed key exchange algorithm is 14 ms ,which means that the proposed key exchange algorithm is 99.61% faster than MSSRSA and 97.11% faster than RSA. The average ciphering time for messages of size 32, 64, 128 and 512 elements using 1024-bits key size and using MSSRSA is 3176 ms, using RSA is 1238 ms while using the proposed cryptosystem it is 26 ms. This means that the proposed cryptosystem is 99.18% faster than MSSRSA and 97.89% faster than RSA. The average deciphering time

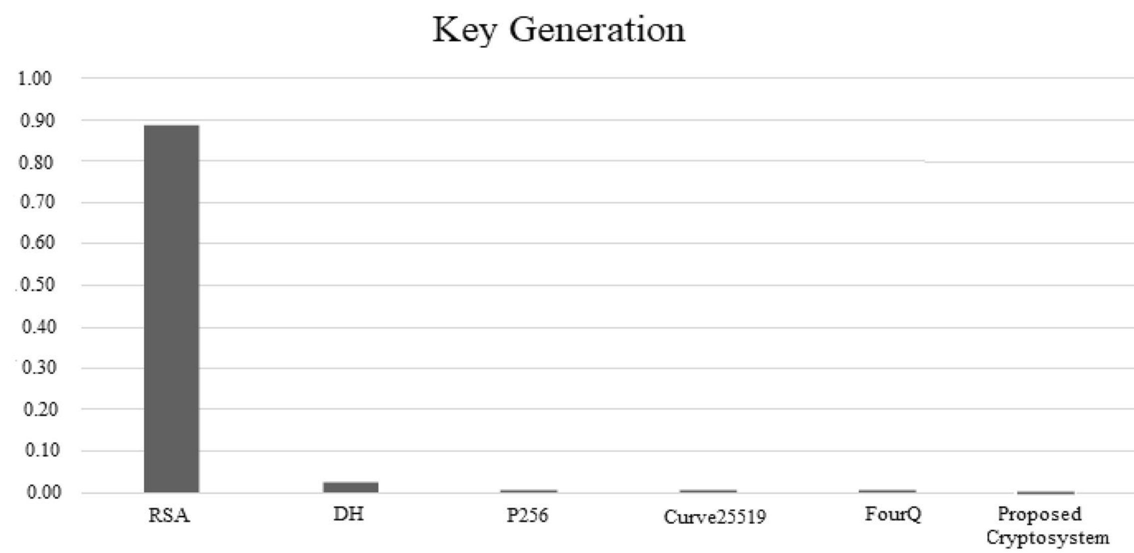

Fig. 1 Key generation time of 128-bit keys between the proposed cryptosystem, DH, P256, Curve25529 and Four Q. As adapted from [24]

for messages of size 32, 64, 128 and 512 elements using MSSRSA is 5176 ms, RSA is 5485 ms while using the proposed cryptosystem is 2 ms. It is clear that the proposed cryptosystem is 99.96% faster than MSSRSA and 99.96% faster than RSA for deciphering messages. The average total execution time for MSSRSA is 11,980 ms, 7207 ms for RSA and finally the average execution time for the proposed cryptosystem is 41 ms. The proposed scheme is faster 96.66% than MSSRSA and faster than RSA by 99.43%.

The key generation time for the key of size 2048-bits using MSSRSA is 11,985 ms and using RSA is 4591 ms while using the proposed key exchange algorithm is 7 ms which means that the proposed key exchange algorithm 99.94% faster than MSSRSA and 99.85% faster than RSA. The ciphering time for messages of size 32, 64, 128, 512 and 1024 elements for the MSSRSA is 4135 ms, 1957 ms for RSA and 38 ms for the proposed cryptosystem that means that the proposed cryptosystem is 99.08% faster than MSSRSA and 98.05% faster than RSA. The average deciphering time for messages of size 32, 64, 128, 512 and 1024 elements for MSSRSA is 31,278 ms, 31,978 ms for RSA and 5 ms for the proposed cryptosystem. This means that the proposed cryptosystem is 99.98% faster than MSSRSA and 99.98% faster than RSA. The average total execution time for MSSRSA is 47,397 ms, 38,525 ms for RSA while it is 50 ms for the proposed cryptosystem. This means that the proposed cryptosystem is 99.89% faster than MSSRSA and 99.87% faster than RSA.

Therefore, the general average time of key generation for MSSRSA is 3930 ms, and for RSA is 1291 ms while for the proposed key exchange algorithm is 6 ms. The general average time of message ciphering for MSSRSA is 3975 ms and for RSA is 878 ms while for the proposed cryptosystem is 4 ms. The general average time of message deciphering for MSSRSA is 9387 ms, and for RSA is 9630 ms while for the proposed cryptosystem is 2 ms.

Figure 1 shows a comparison of the key generation time between the proposed key exchange algorithm and state-of-the-art Curve25519, DH, P256 and FourQ cryptosystems. The average key generation time using the proposed key exchange algorithm is 0.00012 s which is less than the state-of-the-art cryptosystems. Table 1 shows the superiority of the proposed cryptosystem over the state-of-the art cryptosystems.

It is clear from the above analysis that the proposed cryptosystem and key exchange algorithm are significantly faster than the state-of-the-art key exchange algorithms and cryptosystems; this makes the proposed cryptosystem and signature scheme more practical to be used in practice and especially in online transactions.

9 Conclusion

This paper described a public key cryptosystem and a signature scheme which employ a proposed secure key exchange algorithm that is based on the difficulty of computing the *n*th partial sum of infinite numerical series over finite fields. Without using exponentiation for ciphering and deciphering of messages or signing documents and signature verifications the experimental results show that the proposed public key cryptosystem is easier to implement, computationally faster and more practical than state-of-the-art cryptosystems which are based on discrete logarithms, elliptic curves or factoring large prime numbers. Also, the proposed cryptosystem provides a higher level of security as the size of the shared secret key can be enlarged significantly with a very short time for key generation, ciphering and deciphering of messages. It supports forward secrecy because the first term and total number of terms in each numerical series involved in the key exchange algorithm are changed each time a message is ciphered. By generating a unique shared secret key each time a message is ciphered, the compromise of single shared secret key will not affect any ciphered message other than that specific message that is ciphered by that key. Finally, the signature scheme can be also used as a cryptographic hash function as the hash value changes significantly with any change in the document.

Compliance with ethical standards

Conflict of interest The author declares that there is no conflict of interest.

Appendix 1

For simplicity assume that user *A* chose a small prime number $p = 179$, which is not used in practice and was not in the experiments, the first term *a* of the numerical series n^2 is 1 and the number of terms is 7. User *A* calculates the partial sum $S_A = 20784$ and generates the key $y_A = 20$. User *A* sends his public key (y_A, *p*) to user B. User *B* uses the numerical series $2n + 1$, the first term *b* is 1 and the number of terms is 10. User *B* calculates the partial sum $S_B = 9960$ and generates the key $y_B = 115$. User *A* calculates $y_{AB} = 2300$ and user *B* calculates $y_{AB} = 2300$.

User *A* calculates his key as follows:

$$(y_{AB} \cdot y_B \cdot S_A) \bmod p = (2300 * 115 * 20784) \bmod 179 = 13.$$

User *B* calculates his key as follows:

$$(y_{AB} \cdot y_A \cdot S_B) \bmod p = (2300 * 20 * 9960) \bmod 179 = 13.$$

Assume user *B* wants to send an encrypted message to user A. Assume the message is "*Hello*".

User *B* finds the ASCII code of message *m*:

72 101 108 108 111 119

User *A* encodes this message by multiplying each ASCII code with the shared secret key $S_{AB} = 13$.

The ciphered message is:

936 1313 1404 1404 1443 1547

To decode the ciphered message user *A* divides each ciphered ASCI code by the shared secret key $S_{AB} = 13$

The ASCII Code of the deciphered message is:

72 101 108 108 111 119

Appendix 2

Assume that user *A* will sign document $m =$ "*Hello*" using the sign document scheme based on infinite numerical series. For simplicity assume that user *A* chose prime number $p = 179$ and the numerical series n^2 to generate his private key y_r in order to sign document *m*, the first term *a* and the number of terms *r* of the numerical series are randomly chosen assume the first term $a = 17$ and last term $r = 40$. Assume that user *A* chose a different numerical series $2n + 1$ to generate his public key y_k to be used by any other user to verify his signature, the first term *b* of the series and the number of terms *k* are also randomly selected assume $b = 19$, $k = 100$.

User *A* calculates the partial sum $S_r = 20644$ and generates his private key $y_r = 59$. Also, user *A* calculates the partial sum $S_k = 9840$ and generates his public key $y_k = 174$.

User *A* calculates $m_r = 36521$, $m_k = 107706$ and $m_{rk} = 3933530826$.

User *A* publishes the signature (3933530826, 36521) and publishes his public key $y_k = 174$.

Now assume user *B* wants to verify that user *A* signed document *m* suppose document *m* is not changed i.e. $m =$ "*Hello*".

User *B* calculates: $m_{k_1} = 107706$.

$$m_{rk_1} = m_{k_1} * m_r = 3933530826$$

Since $m_{rk_1} = m_{rk}$ then the signature of user *A* is verified and the document is not changed.

Now, assume the signature of user *A* is changed i.e. either m_{rk} or m_r is changed. Assume m_r is changed to be $m_{r_1} = 13666$, so user *B* calculates:

$$m_{k_1} = 107706, \; m_{rk_1} = m_{k_1} * m_{r_1} = 1471910196$$

since $m_{rk} = 3933530826$, $m_{rk_1} = 1471910196$ then $\left| m_{rk} - m_{rk_1} \right| = 2461620630$ which means that: $m_{rk} \neq m_{rk_1}$. Therefore, the signature is not verified.

Now, assume that a hacker changed document *m* to $m_1 =$ "*Hell*" with letter "*o*" removed. User *B* calculates $m_{k_1} = 87000$ and $m_{rk_1} = 3177327000$, since $m_{rk} = 3933530826$ then $\left| m_{rk} - m_{rk_1} \right| = 756203826$ which means the signature is not verified or the document changed. Assume that the hacker changed document *m* to $m_2 =$ "*hello*" with capital letter "H" is replaced by small letter "h" then user B calculates:$m_{k_1} = 108924$ and $m_{rk_1} = 3978013404$ so $\left|m_{rk} - m_{rk_1}\right| = 44482578$ which means that the signature is not verified or the document may be changed. It is clear from the previous two cases that the proposed signature scheme is too sensitive to any change in the document even if a letter is changed from uppercase to lowercase and vice versa. Hence, the proposed signature scheme can be used as a cryptographic hash function.

References

1. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inf Theory 22(6):644–654
2. Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public key cryptosystems. Commun ACM 21(2):120–126
3. Goldwasser S, Micali S (1982) Probabilistic ciphering and how play mental poker keeping secret all partial information. In: Proceedings of 14th symposium on theory of computing, pp 365–377
4. Paillier P (1999) Public key cryptosystem based on composite residue classes. In: Proceedings of advances in cryptology, EUROCRYPT99, pp 223–238
5. Boneh D, Goh E, Nissim K (2005) Evaluating 2-DNF formulas on ciphertexts. In: Proceedings of theory of cryptography, TCC'05, pp 325–341
6. Okamoto T, Uchiyama S (1998) A new public key cryptosystem as secure as factoring. In: Proceedings of advances in cryptology, EUROCRYPT'98, pp 308–318
7. ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31(4):469–472
8. Athena J, Sumathy V (2017) Survey on public key cryptography scheme for securing data in cloud computing. Circuits Syst 8:77–92
9. Freeman DM (2010) Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Proceedings of advances in cryptology, EUROCRYPT'10, pp 44–61
10. Bahadori M, Mali MR, Sarbishei O, Atarodi M, Sharifkhani M (2010) A novel approach for secure and fast generation of RSA public and private keys on smart card. In: 8th IEEE international NEWCAS conference (NEWCAS), pp 265–268
11. Blackburn SR, Galbraith SD (2000) Certification of secure RSA keys. Electron Lett 36:29–30
12. Sharma S, Sharma P, Dhakar RS (2011) RSA algorithm using modified subset sum cryptosystem. In: 2nd International conference on computer and communication technology (ICCCT-2011), Allahabad, pp 457–461. https://doi.org/10.1109/iccct.2011.6075138
13. Ge H, Tate SR (2006) Efficient authenticated key-exchange for devices with a trusted manager. In: Third international conference on information technology: new generations (ITNG'06), pp 198–203
14. Nagar SA, Alshamma S (2012) High speed implementation of RSA algorithm with modified keys exchange. In: 6th International conference on sciences of electronics, technologies of information and telecommunications (SETIT), Sousse, pp 639–6422012. https://doi.org/10.1109/setit.2012.6481987
15. Patidar R, Bhartiya R (2013) Modified RSA cryptosystem based on offline storage and prime number. In: IEEE international conference on computational intelligence and computing research, Enathi, pp 1–6. https://doi.org/10.1109/iccic.2013.6724176
16. Dubey AK, Dubey AK, Namdev M, Shrivastava SS (2011) Cloud-user security based on RSA and MD5 algorithm for resource attestation and sharing in java environment. Int J Adv Comput Res 1:2
17. Ren W, Miao Z (2010) A hybrid ciphering algorithm based on DES and RSA in bluetooth communication. In: Second international conference on modeling, simulation and visualization methods, pp 221–225
18. Arazi A (1993) Integrating a key cryptosystem into the digital signature standard. Electron Lett 29:966–967
19. Harn L, Mehta M, Hsin W-J (2004) Integrating Diffie–Hellman key exchange into the digital signature algorithm (DSA). IEEE Commun Lett 8:198–200
20. Bernstein DJ, Birkner P, Joye M, Lange T, Peters C (2008) Twisted Edwards curves. In: Proceedings of the international conference on cryptology in Africa, Casablanca, Morocco. Springer, Berlin, pp 389–405
21. Chou T (2015) Fastest Curve25519 implementation ever. In: Proceedings of the workshop on elliptic curve cryptography standards, Gaithersburg, MD, USA
22. Longa P (2016) FourQNEON: faster elliptic curve scalar multiplications on ARM processors. In Proceedings of the selected areas in cryptography-SAC, St. John's, NL, Canada
23. Costello C, Longa P (2015) FourQ: four-dimensional decompositions on a Q-curve over the Mersenne prime. In: Proceedings of the international conference on the theory and application of cryptology and information security, Kaoshiung, Taiwan. Springer, Berlin, pp 214–235
24. Alvarez R, Caballero-Gil C, Santonja J, Zamora A (2017) Algorithms for lightweight key exchange. Sensors 17:1517. https://doi.org/10.3390/s17071517
25. Kumar C, Vincent PM (2017) Enhanced Diffie–Hellman algorithm for reliable key exchange. In: IOP conference series: materials science and engineering, vol 263, p 042015. https://doi.org/10.1088/1757-899x/263/4/042015
26. Piskunov N (1974) Differential and integral calculus, vol II. MIR Publishers, Moscow, pp 253–295
27. Menezes AJ, van Oorschot PC, Vanstone SA (1997) Handbook of applied cryptography. CRC Press, Boca Raton
28. Goldwasser S, Micali S (1984) Probabilistic ciphering. J Comput Syst Sci 28:270–299