

# High-Speed Secure Vanity Address Generator for User Convenience in Blockchain Networks

Md. Mainul Islam

*Dept. of Computer Science and Engineering*  
Texas A&M University  
College Station, USA  
Email: mainul.islam@ieee.org

Md. Kamrul Islam

*Dept. of Electrical and Electronic Engineering*  
Shahjalal University of Science and Technology  
Sylhet, Bangladesh  
Email: kamrulsust333@gmail.com

Mohammad Hammoudeh

*Dept. of Information and Computer Science*  
King Fahd University of Petroleum and Minerals  
Dahran, Saudi Arabia  
Email: mohammad.hammoudeh@kfupm.edu.sa

**Abstract**—Generating a vanity address (VA) for user convenience in blockchain networks is a memoryless brute-force exercise that involves testing millions of probable private keys. It requires a high-speed processor to compute millions of public keys and corresponding addresses in a short time. In this paper, we propose a novel split-key-multiplicative online VA generator for crypto wallet that provides faster VA generation compared with the existing vanity pools. The proposed approach is capable of improving the speed (in keys per second) of conventional VA generation by 2–7 times without reconfiguring the processor. We also propose a high-speed offline VA generator that offers blockchain users the ability to create a VA with a random vanity word using a low-performance processor without reliance on a vanity pool. To the best of our knowledge, this is the first research paper on VA generation in blockchain networks.

**Index Terms**—Vanity address, crypto address, crypto wallet, blockchain, elliptic curve cryptography.

## I. INTRODUCTION

A vanity address (VA) is a crypto address that starts with a specific set of characters to distinguish among multiple crypto wallets of the same or different holders. Creating a VA is analogous to solving the mathematical puzzle of a consensus algorithm, such as proof-of-work (PoW), in which a hash value is desired to start with a certain number of zeros that makes the hash value smaller than a predefined hash value [1]. The process of generating a VA is the same as generating a general crypto address, but it is repeated until the desired combination of characters is found. An indefinite number of public keys on an elliptic curve (i.e., secp256k1) [2] and corresponding wallet addresses are generated by randomly varying candidate private keys through an indefinite loop. The loop is stopped when the leading letters of an address match a predefined word or name. The process is memoryless, which means that the generated addresses have no correlation; instead, they are computed randomly using some collision-resistant, one-way hash functions (e.g., SHA256, RIPEMD160, and Keccak256) [3]. The difficulty of finding a VA increases exponentially with

the length of expected vanity word with which the address starts. The probability of finding a VA within a certain amount of time depends on the speed (in keys per second) of address generation. The speed can be boosted by employing a powerful graphics processing unit (GPU); however, it can take months or years to find an address that starts with a vanity word whose length is greater than or equal to 8 letters. Therefore, it is unwise to look for a vanity word more than 7 letters long unless the searcher is prepared for a long wait. Table I presents the difficulties of finding VAs for different lengths of vanity words and the corresponding average time required by a machine capable of searching  $10^6$  keys/s [4].

A blockchain user can hold numerous addresses at the same time for security and privacy. These addresses are long strings of random alphanumeric characters that are not easy to recognize and remember. Moreover, it is difficult for users to distinguish their addresses from one another. These issues are solved by creating vanity addresses (VAs) that these users can easily recognize and differentiate.

Fig. 1a and Fig. 1b demonstrate cryptographic paper wallets with a general address and a VA, respectively. Placing the vanity word “CLOUDS” rather than “Kh88Xa” at the beginning of the wallet address makes it easy to recognize. This type of addresses increases the security of a payment, because a distinctive address prevents an attacker from substituting in another address and fooling the payer into paying the attacker rather than the real payee. VAs are also helpful for a blockchain network that adopts decentralized identifiers (DIDs) for serverless user authentication [5]. Users can personalize their DIDs by creating VAs. However, VAs can raise security vulnerabilities, because an attacker can create a VA that is analogous to someone’s VA and fool that person during a transaction if the person is unaware of the ending letters of the recipient’s address. Therefore, payers are always advised to check the ending letters of a receiving address. VAs can be created by one of the following methods. First, they can be

TABLE I  
VARIATIONS IN DIFFICULTY AND AVERAGE TIME OF VA GENERATION  
WITH VANITY WORD LENGTH

Vanity word	Difficulty	Average time
A	22	<1 s
Be	1330	<1 s
Cat	77,178	<1 s
Deer	4,476,342	<10 s
Eagle	259,627,881	3 min
Falcon	1.506E+10	3 h
Gorilla	8.734E+11	1 week
Hedgehog	5.066E+13	1 year

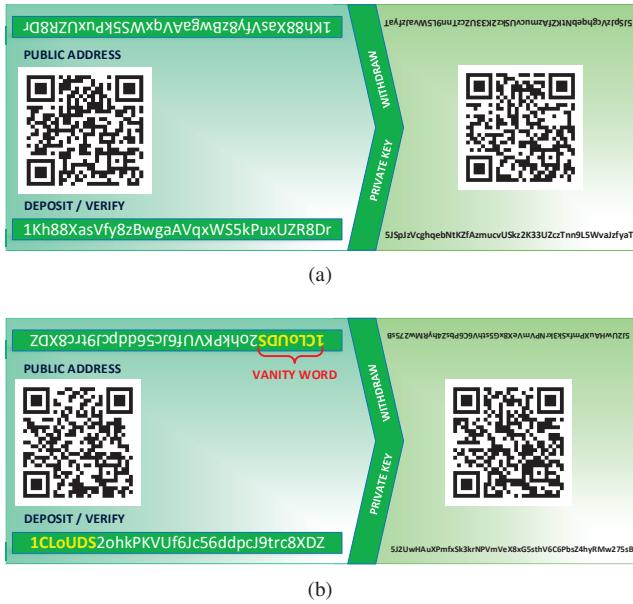


Fig. 1. Cryptographic paper wallets. (a) General address. (b) VA.

created by users using a software. This is the safest method, because the private and public keys of a user are not revealed to a third party; however, there is a risk of leaking the private key to the software provider. Second, they can be created by miners through a vanity pool. Miners invest their computing in finding the desired addresses of users and deliver to them via email after payment. Although the process can be quick, it might not be as secure as the first method, because the miners could hold a user's private key corresponding to the delivered VA and steal the assets that belong to the address.

Because most users do not have a high-performance processor that can create a VA within a reasonable amount of time, they have to depend on a vanity pool and pay for its service. However, these users may not be satisfied with a vanity pool that only provides non-split-key VA delivery. In a non-split-key system, the same private and public keys are shared by a miner and a user, which does not ensure a safe system. To meet the security demands of users, some vanity pools offer split-key [6] option, in which users do not share their private keys with miners and thus, their assets remain safe.

Existing vanity pools that provide split-key VA generation

service compute a complimentary private key of 256 bits in each iteration while performing the brute force exercise of VA generation [7]–[9]. However, it is not necessary to fix the complimentary private key as 256-bit, because the private key that corresponds a VA is not the final private key of a user in a split-key system. After receiving a VA and the corresponding complimentary private key from a vanity pool, a user creates the final private key that is different from the complimentary private key. Hence, the size of the candidate private keys in public keys computation can be reduced to obtain a higher key rate and find a VA quickly. This paper shows that the speed of VA generation in existing non-split and split-key systems could be boosted by 2–7 times without reconfiguring the processor. This paper also presents a high-speed offline VA generator, with which users can create a VA with a vanity word of 2 to 6 letters in a short time using a low-performance processor.

The remainder of this paper is organized as follows: The mathematical background of this research study is explained in Section II. Section III presents the conventional and proposed designs for VA generation. The performance of the proposed online and offline VA generators is analyzed in Section IV. Finally, this paper concludes in Section V.

## II. BACKGROUND

This section describes the mathematical background behind crypto address generation and the importance of private key security.

An elliptic curve over a prime field  $\mathbb{F}_p$  is given by the equation [10]:

$$E_{a,b} : y^2 = x^3 + ax + b \quad (1)$$

where  $p$  is a prime number,  $a$  and  $b$  are arbitrary constants, and  $x, y \in [1, p - 1]$  with

$$4a^3 + 27b^2 \neq 0. \quad (2)$$

Secp256k1 is a version of the elliptic curve, which is used in most blockchain networks for digital signature generation and verification. For this elliptic curve,  $a = 0$ ,  $b = 7$ , and  $p = 2^{256} - 2^{32} - 977$ . Therefore, the secp256k1 curve can be expressed as follows [2], [11]:

$$E : y^2 = x^3 + 7. \quad (3)$$

Generating a crypto address requires to create a public key that is a point on an elliptic curve and obtained by the elliptic curve scalar multiplication (ECSM) operation. ECSM includes several elliptic curve group operations, such as point addition and point doubling [12].

Let us consider two points on  $E$  in affine coordinates, such as  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$ , in the range of  $[1, p - 1]$ . Then the point addition  $P_1 + P_2$  makes another point  $P_3(x_3, y_3)$  on  $E$ , which is calculated as follows [13]:

$$\begin{aligned} P_3(x_3, y_3) &= P_1(x_1, y_1) + P_2(x_2, y_2) \in E \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \quad (4)$$

**Algorithm 1** Double-and-add algorithm-based ECPM [9]

---

**Input :**  $P(x, y), k = \sum_{i=0}^{l-1} k_i 2^i; k_i \in \{0, 1\}, k_{l-1} = 1$

**Output :**  $Q(x, y)$

- 1:  $P \leftarrow P(X = x, Y = y, Z = 1);$  // affine to projective
- 2:  $Q(X, Y, Z) \leftarrow P;$
- 3: **for**  $i$  from  $l - 2$  to 0 **do**
- 4:    $Q \leftarrow 2Q;$  // point doubling
- 5:   **if**  $k_i = 1$  **then**
- 6:      $Q \leftarrow Q + P;$  // point addition
- 7:   **end if;**
- 8: **end for;**
- 9:  $Q(x, y) \leftarrow Q(X/Z^2, Y/Z^3);$  // projective to affine
- 10: **return**  $Q;$

---

where  $\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$  and  $P_1 \neq P_2.$

Doubling of the point  $P_1(x_1, y_1)$  on  $E$  is performed as follows [13]:

$$\begin{aligned} P_3(x_3, y_3) &= 2P_1(x_1, y_1) \in E \\ x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \quad (5)$$

where  $\lambda = (3x_1^2)(2y_1)^{-1}$  and  $P_1 = P_2.$

Point addition and doubling in affine coordinates involve several division operations over  $\mathbb{F}_p$  [10] that are considerably time consuming, because a single modular division operation is equivalent to 80 modular multiplication operations in terms of latency. To reduce the time required by the modular division or inversion operations, point addition and doubling are performed in projective coordinates by representing the points  $P_1$  and  $P_2$  as  $(X_1 = x_1, Y_1 = y_1, Z_1 = 1)$  and  $(X_2 = x_2, Y_2 = y_2, Z_2 = 1)$ , respectively.

The formula for projective point addition is given by the equation [11]:

$$\begin{aligned} P_3(X_3, Y_3, Z_3) &= P_1(X_1, Y_1, Z_1) + P_2(X_2, Y_2, Z_2) \in E \\ X_3 &= \alpha^2 - \beta^3 - 2X_1Z_2^2\beta^2 \\ Y_3 &= \alpha(X_1Z_2^2\beta^2 - X_3) - Y_1Z_2^3\beta^3 \\ Z_3 &= Z_1Z_2\beta \end{aligned} \quad (6)$$

The formula for projective point doubling is given by the equation [11]:

$$\begin{aligned} P_3(X_3, Y_3, Z_3) &= 2P_1(X_1, Y_1, Z_1) \in E \\ X_3 &= 9X_1^4 - 8X_1Y_1^2 \\ Y_3 &= 3X_1^2(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned} \quad (7)$$

where  $\alpha = Y_2Z_1^3 - Y_1Z_2^3$  and  $\beta = X_2Z_1^2 - X_1Z_2^2.$

A public key  $Q$  is computed by performing ECSM in which a base point  $P$  on  $E$  is multiplied by a scalar or private key  $k$  such that  $Q = kP.$  The generation time of a crypto address mostly depends on this mathematical operation.

ECSM can be performed by adding  $P$  to itself  $k - 1$  times as follows [14]:

$$Q = P + \underbrace{P + \dots + P}_{k-1 \text{ times}} \quad (8)$$

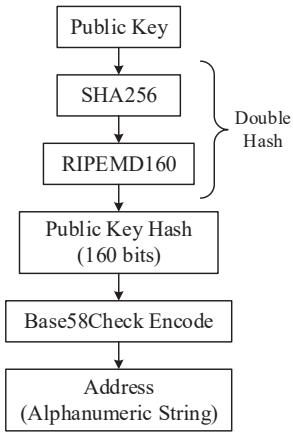


Fig. 2. Public key to crypto address generation.

If  $k$  is expressible as a power of 2,  $Q$  can be obtained by doubling  $P$  on itself  $\log_2 k$  times as follows [14]:

$$Q = \underbrace{\dots 2(2(P))}_{\log_2 k \text{ times}}. \quad (9)$$

A crypto address is a 34-character alphanumeric string that is generated by converting an uncompressed public key (i.e., "04"|| $Q_x$ || $Q_y$ ) into a hash value using the SHA256 and RIPEMD160 hash functions, consecutively, as shown in Fig. 2. The hash value is then encoded using the Base58Check encoding technique, which has an alphabet of 1–9, a–z, and A–Z excluding 0 (zero), O (capital o), I (capital i), and l (small L) [6]. The purpose of using a crypto address in peer-to-peer transaction rather than a public key is 2-fold; 1) to shorten and obfuscate a public key and 2) to provide more security through the hash functions used to generate an address with one-way and collision-resistant properties.

### III. PROBLEM DEFINITION AND SYSTEM MODEL

This section defines the problems exist in conventional vanity pools and presents the proposed system models for online and offline VA generation.

#### A. Problems Exist in Conventional Vanity Pools

Fig. 3a demonstrates the conventional non-split-key system for VA generation. Here, a user requests a miner to generate a VA according to a desired vanity word  $V_w.$  To access this service, the user have to pay money to the miner based on the length of  $V_w.$  After creating a VA  $V_a$  that starts with  $V_w,$  the miner delivers the VA along with the corresponding private and public keys  $\beta$  and  $Q,$  respectively, to the user. The drawback of this method is that the same private key is shared between the miner and the user, which carries a risk of money theft from the user wallet in the case of a dishonest miner.

Fig. 3b presents the conventional split-key-additive system for VA generation, in which the user first generates a complementary public key  $Q$  by performing ECSM with a complementary private key  $\alpha.$  Then, the user provides  $Q$  and  $V_w$  to the miner and requests for a VA and corresponding

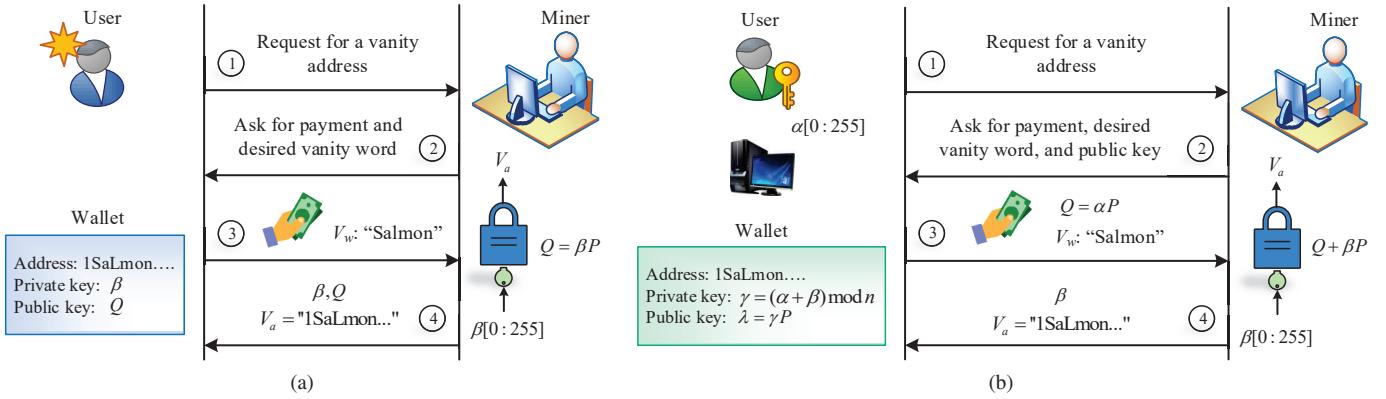


Fig. 3. Conventional vanity pools [7], [8]. (a) Non-split key (insecure). (b) Split-key-additive (secure).

private key. The miner selects a complementary private key  $\beta$ , computes the corresponding public key  $\beta P$ , adds  $Q$  to  $\beta P$ , and generates a VA  $V_a$  that starts with  $V_w$  using the combined public key. After receiving  $V_a$  and  $\beta$  from the miner, the user computes a new private key  $\gamma$  by combining the 2 complementary private keys, such that  $\gamma = (\alpha + \beta) \bmod n$ , where  $n$  is the order of  $E$ . The user also generates a new public key  $\lambda$  using  $\gamma$ , which corresponds to  $V_a$ . In this design, the miner cannot steal the money that are stored in the user's wallet, because the miner is unable to generate a valid signature with  $\beta$ . Thus, the wallet becomes more secure and the secrecy of the private key in it is ensured.

The conventional split-key-multiplicative system is analogous to the split-key-additive one, but the mechanism of key generation is slightly different. In a split-key-multiplicative system, the point addition  $Q + \beta P$  is replaced by a point multiplication operation  $\beta Q$ . Similarly, the final private key  $\gamma$  is generated by the modular multiplication of  $\alpha$  and  $\beta$  rather than modular addition.

The drawback of the conventional split-key methods is that the size of the complementary private keys is fixed to 256 bits (64 characters), which increases the time required to perform numerous ECSM operations. In the additive method, an extra point addition operation  $Q + \beta P$  is performed in every iteration while searching a VA, which increases the overall latency. In the multiplicative method, the multiplication of 256-bit  $\alpha$  and  $\beta$  results in approximately a 512-bit integer, which is again reduced to a 256-bit value by modular reduction at the time of  $\gamma$  computation. All these issues are solved in our proposed adjustable split-key-multiplicative system.

### B. Proposed VA Generators

Fig. 4 illustrates the proposed online VA generator, in which the conventional fixed length split-key approach is replaced by an adjustable one. Here, the complimentary private keys  $\beta$  and  $\alpha$  are taken as  $m$  and  $(256 - m)$  bits in size, respectively, where  $32 \leq m \leq 128$  and the speed of VA generation is inversely proportional to  $m$ . This type of split-key approach significantly reduces the latency of ECSM without decreasing the output size (256 bits) and does not require a modular reduction

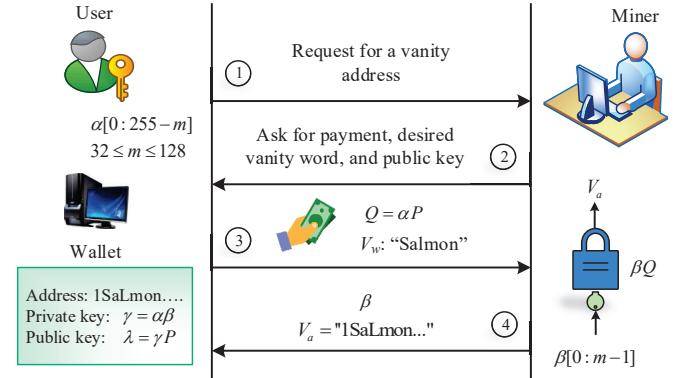


Fig. 4. Proposed online VA generator (adjustable split-key-multiplicative).

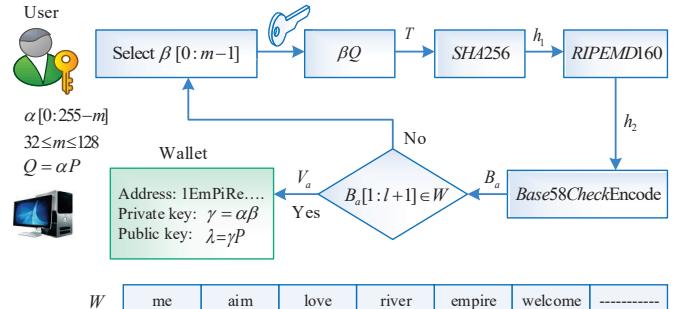


Fig. 5. Proposed offline VA generator.

operation at the time of  $\gamma$  computation. Consequently, VA generation in this approach becomes much faster compared with the conventional approaches. The lower limit of  $m$  is fixed to 32, because while choosing the candidate private keys randomly through a loop operation, the probability of repeating a private key increases as the value of  $m$  is reduced.

Fig. 5 illustrates the proposed high-speed offline VA generator, in which the user generates a VA by choosing a random vanity word from a dictionary  $W$  that contains a definite number of words ranging from 2 to 7 letters in length. Algorithm 2 demonstrates the proposed algorithm for offline

TABLE II  
TEST RESULTS OF VA GENERATION BY THE PROPOSED ONLINE VA GENERATOR

<i>m</i>	Test results
	$\alpha$ : f89dc6332307aa4e0432f67a926dcd5f $Q$ : 02cf2eef2ef7b83902c08c39d86289088214b2127904fe42a850102127a7531f5f $\beta$ : fb6ca95f30e3ad34737ecde3f2a8e7c8 $V_a$ : 1LoVEWusHQrpy5XKk2XkyrS3D8Y87kxHTS
128	Elapsed time to generate 194,215 addresses: 931.558 s Speed: 208 keys/s $\gamma$ (Hex) : f42c38605aca4b964b5dc75189b2f89784b2ebe791d466b969a93c530e712b38 $\gamma$ (WiFi) : 5KfpesTjYwuwk1AHKBZYaqdF4TpV4ugDfCVRgdTpu3wodL16whz $\lambda$ : 0335790d183691e30a95ad26d755f17e37fd879bcff9656dead4dc82e699ab4a3a
	$\alpha$ : eff8f18b189b8f85b68a54026a8a6aecfcdf291ea22752bd58b5307c $Q$ : 023dcfcacc2b333adbc6729cd22c2d6d1e2d699ead9ff765977ce2c1a446b949 $\beta$ : c3182f03 $V_a$ : 1LoVe6b5Y7rVEy4kWfGjiWcyYHXXaBcheU
32	Elapsed time to generate 90,983 addresses: 126.763 s Speed: 717 keys/s $\gamma$ (Hex) : b6e14b651d0c2b0e27c0b60a406cda03341674721f8d42d71af9106c4da65574 $\gamma$ (WiFi) : 5KCq1NABNUD9jm8m3Ze6pZeij1Pkprz9JLEM2GKLFFkEgWWsB2F $\lambda$ : 02b0d74d9808ad8b306ca2e1288ad1a537911a50ae79eda3650c9d7c3569e8cb4b

**Algorithm 2** Proposed offline VA generation

Curve parameter: generator point  $P(x, y)$ , curve order  $n$   
Input: adjustable length  $m$ , length of vanity word  $l$ , and wordlist  $W$   
Output: VA  $V_a$ , private key  $\gamma$ , and public key  $\lambda$

```

1: Classify  $W$  according to the word length (i.e., 2–7 letters).
2: Take desired vanity word length  $l$ .
3: Copy  $l$ -letter words from the classified  $W$  to an array (i.e.,  $D$ ).
4: Compute a complementary  $m$ -bit private key  $\beta$ .
5: Generate a complementary 256-bit public key by ECPM:  $Q = \beta P$ .
6: Compute the SHA256 digest of  $Q$ :  $h_1 = SHA256(Q)$ .
7: Compute the RIPEMD160 digest of  $h_1$ :  $h_2 = RIPEMD160(h_1)$ .
8: Generate the address by encoding  $h_2$ :  $V_a = Base58Encode(h_2)$ .
9: Extract the vanity string from  $V_a$ :  $V_w = V_a[1 : l + 1]$ .
10: Define a temporary variable  $T = 0$ .
11: for  $i$  in range( $l$ ) do
12:   if  $V_w[i]$  is digit then
13:     Go to step 4.
14:   end if
15: end for
16: if  $V_w.lower()$  exists in  $D$  then
17:   Save the VA  $V_a$ .
18:   Compute another complementary (256 –  $m$ )-bit private key  $\alpha$ .
19:   Compute the final private key  $\gamma$ :  $\gamma = \alpha\beta$ .
20:   Generate the final public key  $\lambda$ :  $\lambda = \gamma P$ .
21: else
22:   Go to step 4.
23: end if
```

VA generation. First, the words in  $W$  are classified according to their length. The user defines the preferred vanity word length  $l$ , rather than a vanity word. Then, the system starts to generate a VA in a manner similar to the proposed split-key multiplicative approach and stops when a generated address contains an  $l$ -letter vanity word that belongs to  $W$ . In this way, the user can create a VA with a low performance processor, which eliminates dependency on a third-party vanity pool and saves time and money.

**IV. SYSTEM IMPLEMENTATION AND PERFORMANCE ANALYSES**

The proposed online and offline VA generators are implemented in Python. The systems are tested generating VA with a workstation (processor: Intel(R) Xeon (R) Gold 5218 CPU @ 2.30 GHz; graphics card: NVIDIA Quadro RTX 5000 GPU; and memory: 64 GB).

The test results of finding a VA with the vanity word “love” using the proposed online VA generator are shown in Table II. Firstly, a 128-bit complementary private key  $\alpha$  is chosen and the corresponding 256-bit public key  $Q$  is computed such that  $Q = \alpha P$ . Then, the public key  $Q$  is multiplied by another 128-bit complementary private key  $\beta$  and a 34-character VA  $V_a$  is generated by hashing and encoding the resultant  $\beta Q$ . Finally, the final 256-bit private and public keys  $\gamma$  and  $\lambda$ , respectively, are calculated using  $\alpha$  and  $\beta$ . The whole process is repeated continuously and a VA that starts with the vanity word “Love” is found after 15.5 min. Note that the vanity word in a VA can be in any case (upper or lower), because matching the letter-case of desired vanity word increases the difficulty. To reduce the time, the same vanity word is sought by taking  $\alpha$  and  $\beta$  as 224 and 32-bit, respectively. A VA is found after 2.1 min with a key rate of 717 keys/s, which provides an approximation of the maximum possible speed of VA generation using this scheme.

Different sizes of private keys are compared in terms of the ECSM and address generation speeds in Fig. 6. As shown in the figure, decreasing the size of the private keys reduces the time required to perform ECSM and generate addresses. The sizes of the ECSM outputs and addresses are constant for different sizes of private keys.

The implementation results of the proposed offline VA generator using the workstation and a notebook (processor:

TABLE III  
IMPLEMENTATION RESULTS OF OFFLINE VA GENERATION WITH RANDOM VANITY WORD

Vanity word	Number of available words	Workstation		Notebook	
		Average time		Average time	
		$m = 32$ @ 667 keys/s	$m = 128$ @ 201 keys/s	$m = 32$ @ 335 keys/s	$m = 128$ @ 147 keys/s
2-letter	46	41 ms	158 ms	137 ms	221 ms
3-letter	587	98 ms	196 ms	226 ms	430 ms
4-letter	2293	901 ms	1.17 s	1.62 s	4.82 s
5-letter	4265	7.72 s	15.95 s	20.6 s	46.19 s
6-letter	6936	2.78 min	5.26 min	4.49 min	6.16 min
7-letter	1371	1 day (not found)	1 day (not found)	1 day (not found)	1 day (not found)

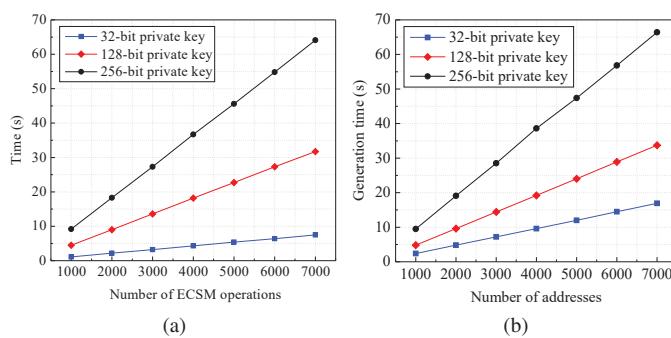


Fig. 6. Comparisons of 32-, 128-, and 256-bit private keys in terms of ECPM and address generation speeds. (a) ECPM time. (b) Address generation time.

TABLE IV  
SPEED COMPARISON OF CONVENTIONAL AND PROPOSED ONLINE VA GENERATORS

Vanity string	Conventional system		Proposed system	
	Average time		Average time	
Fixed size $\beta$ @ 103 keys/s	$m = 32$ @ 688 keys/s		$m = 128$ @ 203 keys/s	
A	353 ms		72 ms	153 ms
Be	3.14 s		778 ms	2.33 s
Cat	2.18 min		9.90 s	29.90 s
Deer	1.32 h		5.45 min	28.58 min
Eagle	1 day (not found)		5.28 h	12.86 h

Intel Core i3-7020U@2.30 GHz; memory: 4 GB) is presented in Table IV. Both the workstation and the notebook generate VAs within a short time duration for vanity words of 2 to 6 letters from a predefined dictionary, where the average key rates of the processors are 667 and 335 keys/s, respectively, for  $m = 32$ . They can not find a VA with a 7-letter vanity word within a day because of high difficulty of such address generation. However, the time required to obtain a VA in the proposed offline VA system is considerably less than that in the proposed online VA system for the same size of vanity word.

Table III shows a comparison of the conventional and proposed online VA generators in terms of speed, where the vanity words are user defined. The same workstation is used in

both systems, and the average key rate in the proposed system is 2–6.8 times higher than that in the conventional system. It is observed that as the length of the vanity word increases, the number of iterations and time required to find a VA rise exponentially. The average time in our system to find the same vanity word is less than that of the conventional system. For a 5-letter vanity word (i.e., eagle), it becomes almost impossible to create a VA in the conventional system, whereas our system finds a VA having the vanity word “eagle” after 5.28 and 12.86 hours for the cases of  $m = 32$  and  $m = 128$ , respectively.

The security of a crypto wallet depends on the size of the private key used to generate the wallet address. Although our proposed online and offline VA generators adopt a split-key technique in which the size of an auxiliary private key is less than the standard size (256 bits), the final private key produced through the method is 256-bit. Therefore, both the conventional and proposed VAs provide 128-bit security level.

## V. CONCLUSION

High-speed online and offline VA generating systems are proposed in this paper for crypto wallet by exploiting a new approach to reduce ECM time in VA generation. The performance of the proposed systems is evaluated by measuring the time required to generate VAs of different sizes. A comparison of the proposed approach and the traditional methods of creating a VA is provided. The proposed online VA generator achieves a higher key rate in VA generation compared with conventional vanity pools. The proposed offline VA generator is capable of generating a VA with a vanity word of fewer than 7 letters within 7 min. Based on the overall performance analyses, it can be concluded that the proposed VA generators could be good choices for the generation of user-friendly and secure wallet addresses.

## REFERENCES

- [1] M. C. K. Khalilov and A. Levi, “A survey on anonymity and privacy in bitcoin-like digital cash systems,” *IEEE Commun. Surv. Tutor.*, vol. 20, no. 3, pp. 2543–2585, Mar. 2018.
- [2] M. Qu, “SEC 2: Recommended elliptic curve domain parameters,” Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6, 1999.
- [3] L. Wang, X. Shen, J. Li, J. Shao, and Y. Yang, “Cryptographic primitives in blockchains,” *J. Netw. Comput. Appl.*, vol. 127, pp. 43–58, Feb. 2019.
- [4] Bitcoin Wiki. 2020. *Difficulty of finding a vanity address*. Accessed on: Oct. 25, 2020. [Online]. Available at: <https://en.bitcoin.it/wiki/Vanitygen>.

- [5] M. M. Islam, M. K. Islam, M. Shahjalal, M. Z. Chowdhury, and Y. M. Jang, "A low-cost cross-border payment system based on auditable cryptocurrency with consortium blockchain: Joint digital currency," *IEEE Trans. Services Comput.*, early access, Sep. 16, 2023, doi: 10.1109/TSC.2022.3207224.
- [6] W. J. Buchanan. 2020. *Vanity bitcoin address generation*, Asecuritysite. Accessed on: Oct. 25, 2020. [Online]. Available at: <https://asecuritysite.com/encryption/vanity/>.
- [7] BitAddress.org. 2013. *Open source javascript client-side bitcoin wallet generator*. Accessed on: Oct. 25, 2020. [Online]. Available at: <https://www.bitaddress.org>.
- [8] Vanity-eth.tk. *ETH vanity address generator*. Accessed on: Oct. 13, June 2023. [Online]. Available at: <https://vanity-eth.tk/>.
- [9] Vante.me. 2016. *Generate your vanity address*. Accessed on: Oct. 25, 2020. [Online]. Available at: <https://vante.me>.
- [10] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. New York, NY, USA: Springer-Verlag, 2004.
- [11] M. M. Islam and H. P. In, "A privacy-preserving transparent central bank digital currency system based on consortium blockchain and unspent transaction outputs," *IEEE Trans. Services Comput.*, early access, Dec. 1, 2022, doi: 10.1109/TSC.2022.3226120.
- [12] M. M. Islam, M. S. Hossain, M. K. Hasan, M. Shahjalal and Y. M. Jang, "FPGA implementation of high-speed area-efficient processor for elliptic curve point multiplication over prime field," *IEEE Access*, vol. 7, pp. 178811–178826, Dec. 2019.
- [13] M. S. Hossain and Y. Kong, "High-performance FPGA implementation of modular inversion over F\_256 for elliptic curve cryptography," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Sys.*, Sydney, Australia, Dec. 2015, pp. 169–174.
- [14] M. M. Islam, M. S. Hossain, M. K. Hasan, M. Shahjalal, and Y. M. Jang, "Design and implementation of high-performance ECC processor with unified point addition on twisted Edwards curve," *Sensors*, vol. 20, no. 18, p. 5148, 2020.