Research Article

White-Box Implementation of ECDSA Based on the Cloud Plus Side Mode

Jie Zhou ¹, Jian Bai ², and Meng Shan Jiang ²

¹School of Computer and Software Engineering, Xihua University, Chengdu 610039, China

²China Electronic Technology Cyber Security Co., Ltd., Chengdu 610041, China

Correspondence should be addressed to Jie Zhou; zibing357@163.com

Received 7 March 2020; Revised 20 October 2020; Accepted 29 October 2020; Published 19 November 2020

Academic Editor: Fulvio Valenza

Copyright © 2020 Jie Zhou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

White-box attack context assumes that the running environments of algorithms are visible and modifiable. Algorithms that can resist the white-box attack context are called white-box cryptography. The elliptic curve digital signature algorithm (ECDSA) is one of the most widely used digital signature algorithms which can provide integrity, authenticity, and nonrepudiation. Since the private key in the classical ECDSA is plaintext, it is easy for attackers to obtain the private key. To increase the security of the private key under the white-box attack context, this article presents an algorithm for the white-box implementation of ECDSA. It uses the lookup table technology and the “cloud plus side” mode to protect the private key. The residue number system (RNS) theory is used to reduce the size of storage. Moreover, the article analyzes the security of the proposed algorithm against an exhaustive search attack, a random number attack, a code lifting attack, and so on. The efficiency of the proposed scheme is compared with that of the classical ECDSA through experiments.

1. Introduction

1.1. Background. With the rapid development of computer and Internet technology, sensitive content containing military, political, economic, and other information is widely transmitted over networks. Because of the openness of the Internet, attackers usually use the Internet to obtain information illegally, which greatly threatens the lives and work of people. Network security has become an important part of network design, construction, and maintenance, and cryptography is the core barrier technology for protecting network information security. Traditional cryptography is based on the black-box model, which assumes that the operating environment of the cryptographic algorithm is safe. That is, the execution of the cryptographic algorithm cannot be observed nor tampered with, and an attacker can only observe and modify the information transmitted in a channel. However, as cryptography is widely used in e-mail, web access, digital rights management, e-government, and so forth, cryptographic algorithms often run in untrusted environments such as mobile phones, flat computers, and wearable electronic devices [1]. Attackers can easily use static

analysis or dynamic debugging to obtain the secret key in a cryptographic algorithm, leading to a disclosure of this information. The traditional cryptography model cannot meet the increasingly high-security requirements.

In 2002, Chow et al. [2] proposed the white-box attack context, which assumes the following:

- (1) A fully privileged attack software shares a host with cryptographic software, having complete access to the implementation of the algorithm.
- (2) Dynamic execution can be observed.
- (3) Internal algorithm details are completely visible and alterable at will.

People always use hardware or software to implement cryptographic algorithms. Regarding hardware, it is costly and has poor universality. Regarding software, the secret key will appear in the memory of the computing platform. It is easy for attackers to obtain this key by injecting malware. At present, there are two ways to prevent secret key leakage during the execution of a cryptographic algorithm. One is cloud collaboration [3], and the other is key decentralized

storage [4]. However, cloud collaboration cannot resist local key disclosure, and it also requires solving the problem of identification between the cloud and the terminal. Although key decentralized storage can mitigate the leak risk when the key is in static storage, the key must be composed when the cryptographic algorithm is running. Key decentralized storage will also make the whole key appear in the memory.

When the traditional cryptographic algorithms are implemented in software, the keys are easily leaked under the white-box attack context. Thus, it is urgent to design cryptographic algorithms that can resist white-box attacks [5]. The field of cryptography that can resist white-box attacks is called white-box cryptography. At present, the research on white-box cryptography has focused on white-box implementations and constructions. In the white-box implementations, the existing cryptographic algorithms are redesigned by using confusion, perturbation, and other methods to ensure the safeness of algorithms under the white-box attack context, with the original algorithm function kept. White-box construction is carried out to design a new cryptographic algorithm that can resist adversaries' attacks under the white-box attack context.

1.2. Previous Work. An important line of research focused on the white-box implementations of AES and DES. In 2002, Chow et al. proposed the first white-box cryptography algorithm—the white-box AES implementation [2]. However, Billet et al. [6] put forward the BGE (BGE stands for the first letter of each of the authors, Billet, Gilbert, and Ech-Chatbi) attack to extract the AES key from the white-box AES implementation of Chow et al., with a computational complexity of 2^{30} . The BGE attack motivated the design of a white-box AES implementation that can provide more resistance against key extraction. In 2006, Bringer et al. [7] proposed a new white-box AES implementation that can resist the BGE attack. Mulder et al. [8] attacked the Bringer's implementation with a computational complexity of 2^{17} . After that, Xiao and Lai [9] proposed another white-box AES implementation that was cryptanalyzed by Mulder et al. [10] with a computational complexity of 2^{32} . Recently, Xu et al. [11] proposed an AES-like cipher by replacing the AES S-boxes and MixColumn matrices with key-dependent components while keeping their good cryptographic properties. They claimed that the white-box implementation of their AES-like cipher can resist current known attacks. In 2014, Luo et al. proposed a new WBAC-oriented AES implementation [12], but Bai et al. [13] showed that the secret key of Luo's implementation can be recovered with time complexity of approximately 2^{44} . The earliest white-box DES implementation was proposed by Chow et al. [14] in 2003. A fault injection attack was used to extract the key from Chow's implementation by Jacob et al. [15]. In 2005, Link and Neumann [16] put forward an improved white-box DES implementation to resist the fault injection attack. However, Goubin et al. [17] and Wyseur et al. [18] both used differential cryptanalysis to extract the key from the implementation in [16]. In 2019, Amadori et al. [19] present a new DFA attack on a class of white-box implementations

presented a new DFA attack on a class of white-box implementations of AES. Lin et al. [20] present an overview of the classic methods for constructing white-box solutions.

In terms of new white-box cryptographic algorithms, Biryukov et al. [21] proposed a general method for designing white-box cryptography based on the ASASA (Affine-S-box) structure. They put forward two strong white-box variants of the ASASA symmetric scheme: one is based on Daemen's quadratic S-boxes and the other is based on random expanding S-boxes. Unfortunately, the ASASA construction was broken in [22, 23]. In 2020, Kwon et al. [24] use parallel table lookups to construct a secure white-box block cipher. Alpirez Bock et al. [25] present the security goals of white-box cryptography. Shi et al. [26] use state-dependent selectable random substitutions (SDSRS) to defeat various related white-box cryptanalytic approaches.

Currently, there are few studies on public-key white-box cryptography in the public literature. In 2018, Zhang et al. [27] proposed the first white-box implementation of the identity-based signature scheme. In 2019, Feng et al. [28] proposed a white-box implementation for the classical Shamir's IBS scheme.

1.3. Our Contributions. In this paper, we focus on a white-box implementation of the elliptic curve digital signature algorithm (ECDSA) [29]. ECDSA is the elliptic curve analogue of the digital signature algorithm and can provide integrity, authenticity, and nonrepudiation. The details of ECDSA can be found in several approved standards: ISO 14888-3 [30], ANSI X9.62 [31], IEEE Std 1363-2000 [32], and FIPS 186-2 [33]. Since ECDSA has a small key volume, small storage space, and high-security performance, it has become one of the most important digital signature algorithms.

The main contributions of this paper can be summarized as follows: (1) a white-box implementation of ECDSA based on the "cloud plus side" mode is proposed. To protect the security of the private key in ECDSA under the white-box attack context, we use lookup tables and permutations to protect the private key. The permutations are used in both the cloud side and the client side. The private key is confused in a series of lookup tables and is thus able to resist the white-box attack. (2) The storage size of the proposed schemes is reduced by the residue number system. The private key is divided into several small parts by the residue number system. The small part of the private key can be hidden by a smaller size of lookup tables. The storage needed by the proposed scheme is feasible. (3) The security of the proposed scheme is analyzed from aspects of an exhaustive search attack, a random number attack, and a code lifting attack. (4) The efficiency of the proposed scheme is compared with that of the classical ECDSA through experiments. Because of the improved security, the proposed scheme runs slower than the classical ECDSA. The operation speed of the proposed scheme is also acceptable.

The paper is organized as follows: in Section 2, we briefly review the signature and verification of the classical ECDSA and introduce the concept of the residue number system

(RNS). In Section 3, we propose a white-box implementation of ECDSA based on the RNS. The security analysis of our scheme is discussed in Section 4. In Section 5, the efficiency of the proposed scheme is compared with that of the classical ECDSA. Finally, we conclude in Section 6.

2. Preliminaries

2.1. Classical ECDSA. ECDSA is a digital signature algorithm over an elliptic curve with the domain parameters $D = (q, \text{FR}, a, b, G, n, h)$, where q is the order of field F_q , FR is the field representation of elements in F_q , a and b are the two coefficients used in an elliptic curve E over F_q (i.e., $y^2 = x^3 + ax + b$ in the case of a prime field, and $y^2 + xy = x^3 + ax^2 + b$ in the case of a binary field), $G = (x_G, y_G) \in E(F_q)$ has prime order n , and $h = \#E(F_q)/n$ is the cofactor. The user randomly selects an integer $d \in [1, n-1]$ as the private key, and the corresponding public key is $Q = dG$. The signature generation and verification of the classical ECDSA are as follows.

(1) ECDSA signature generation

Input: Domain parameters $D = (q, \text{FR}, a, b, G, n, h)$, private key d , and message m .

Output: Signature (r, s) .

- (1) Randomly select an integer $k \in [1, n-1]$.
- (2) Compute $kG = (x_1, y_1)$ and convert x_1 to an integer $\bar{x}_1$.
- (3) Compute $r = \bar{x}_1 \pmod{n}$. If $r = 0$ then go to (1).
- (4) Compute $k^{-1} \pmod{n}$, where $k^{-1}k = 1 \pmod{n}$.
- (5) Compute $e = H(m)$, where $H(m)$ is the hash value of the message m .
- (6) Compute $s = k^{-1}(e + dr) \pmod{n}$. If $s = 0$ then go to (1).
- (7) Return the signature (r, s) .

(2) ECDSA signature verification

Input: Domain parameters $D = (q, \text{FR}, a, b, G, n, h)$, public key Q , message m , and the signature (r, s) .

Output: “Reject the signature” or “Accept the signature.”

- (1) Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails, then return “Reject the signature.”
- (2) Compute $e = H(m)$, where $H(m)$ is the hash value of the message m .
- (3) Compute $w = s^{-1}$, where $s^{-1}s = 1 \pmod{n}$.
- (4) Compute $u_1 = ew \pmod{n}$ and $u_2 = rw \pmod{n}$.
- (5) Compute $X = (x_1, y_1) = u_1G + u_2Q$.
- (6) If $X = \infty$, then return “Reject the signature.” Otherwise, convert x_1 to an integer $\bar{x}_1$ and compute $v = \bar{x}_1 \pmod{n}$.
- (7) If $v = r$ then return “Accept the signature”; Else return “Reject the signature.”

2.2. Residue Number System. The residue number system (RNS) [34] has been widely studied and used in many applications, such as digital signal processing, multiple

precision arithmetic, and parallel computing [35, 36]. An RNS is defined by a set of pairwise coprime integers $\beta = \{p_1, p_2, \dots, p_t\}$. The set β is a basis of the RNS, and the elements in β are called the RNS moduli. Any integer x , $0 \leq x \leq P = \prod_{i=1}^t p_i$, can be uniquely represented as $\mathbf{x} = (x_1, x_2, \dots, x_t)$, where x_i is x modulo p_i , denoted as $x_i = |x|_{p_i}$. Note that only when x is not more than P and not less than zero can x be uniquely represented by the RNS with the basis β . The original value of x can be restored from $(x_1, x_2, \dots, x_t)$ by using the Chinese Remainder Theorem (CRT):

$$x = \sum_{i=1}^t P_i |x_i P_i^{-1}|_{p_i} \pmod{P}, \quad (1)$$

where $P_i = (P/p_i)$ and $P_i \cdot P_i^{-1} = 1 \pmod{p_i}$.

Assume that integers x and y can be uniquely represented as $\mathbf{x} = (x_1, x_2, \dots, x_t)$ and $\mathbf{y} = (y_1, y_2, \dots, y_t)$ under the basis β ; then

$$\mathbf{x} \odot \mathbf{y} = (|x_1 \odot y_1|_{p_1}, |x_2 \odot y_2|_{p_2}, \dots, |x_t \odot y_t|_{p_t}), \quad (2)$$

where $\odot \in \{+, -, \times\}$.

All the operations of ECDSA are performed on the finite field F_q . In the following, we transform the formulas on the finite field F_q into the residue number system.

- (1) Assume the formulas are computed over the integers, that is, without the mod q reduction. Let $|\omega|$ be the absolute maximum value generated by these operations.
- (2) Choose a basis $\beta = \{p_1, p_2, \dots, p_t\}$ of the RNS, where $P = p_1 \cdot p_2 \cdot \dots \cdot p_t \geq |\omega|$.
- (3) Represent every integer in RNS format based on the basis β .
- (4) Compute the formula in RNS format, and let $\mathbf{y} = (y_1, \dots, y_t)$ be the result.
- (5) Restore y by the Chinese Remainder Theorem: $y = \sum_{i=1}^t P_i |y_i P_i^{-1}|_{p_i} \pmod{P}$.
- (6) Compute $y' = y \pmod{q}$, and return y' .

Example. Compute $y' = ab \pmod{(2^{14} - 3)}$, where $a = 15242$ and $b = 11213$.

- (1) Since $a \leq 2^{14} - 3$ and $b \leq 2^{14} - 3$, the absolute maximum value of $|\omega| = |ab| \leq (2^{14} - 3)^2 = 268337161$.
- (2) Choose a basis $\beta = \{p_1, p_2, \dots, p_8\} = (3, 5, 7, 13, 17, 19, 23, 29)$ of the RNS, where $P = 3 \cdot 5 \cdot 7 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 = 294076965 > 268337161$.
- (3) Represent a as $\mathbf{a} = (2, 2, 3, 6, 10, 4, 16, 17)$, and represent b as $\mathbf{b} = (2, 3, 6, 7, 10, 3, 12, 19)$.
- (4) Compute $\mathbf{y} = \mathbf{ab} = (1, 1, 4, 3, 15, 12, 8, 4) = (y_1, y_2, \dots, y_8)$ by equation (2).
- (5) Compute $y = \sum_{i=1}^8 P_i |y_i P_i^{-1}|_{p_i} \pmod{P} = 170908546 \pmod{294076965}$ by equation (1).

- (6) Compute $y = 170908546 \pmod{(2^{14} - 3)} = 5573 \pmod{(2^{14} - 3)}$.

3. White-Box Implementation of ECDSA Based on the Residue Number System

3.1. Hide the Private Key. In the white-box attack context, attackers can completely access the implementation of ECDSA and know all the intermediate results of the algorithm. The attacker's objective is to extract the private key or forge a legitimate signature. From the details of ECDSA above, we know that the user's private key d is used only in step (6) of the process of ECDSA signature generation. Hence, to protect the security of ECDSA under the white-box attack context, we need to protect only the security of private key d during the procedure of computing:

$$s = k^{-1}(e + dr) \pmod{n}. \quad (3)$$

Similar to most white-box cryptographic schemes, the private key d in our scheme is hidden in lookup tables $\{T_1, T_2, \dots, T_t\}$ (see Table 1). Let $\beta = \{p_1, p_2, \dots, p_t\}$ be a basis of the RNS, let $\mathbb{Z}_{p_i}$ be the prime field with order p_i , and the private key d can be represented as $\mathbf{d} = (d_1, \dots, d_t)$, where $d_i = d \pmod{p_i}$. Let $f_i: \mathbb{Z}_{p_i} \rightarrow \mathbb{Z}_{p_i}$ and $g_i: \mathbb{Z}_{p_i} \rightarrow \mathbb{Z}_{p_i}$ be two permutations on $\mathbb{Z}_{p_i}$; f_i^{-1} and g_i^{-1} are the inverse mappings of f_i and g_i , respectively. An example of the table T_i generation pseudocode is as follows:

```

for  $j = 1$  to  $p_i$  do
  for  $k = 1$  to  $p_i$  do
     $T_i(j, k) = f_i^{-1}(j) + g_i^{-1}(k)d_i \pmod{p_i}$ ,
  end for;
end for.
```

Note that the table T_i hides not only part of the private key d_i but also the permutations f_i^{-1} and g_i^{-1} . If the permutations f_i^{-1} and g_i^{-1} are unknown, the attackers cannot derive the private key d_i by the input and output of table T_i . Since $1 \leq j, k, T_i(j, k) \leq p_i$, the storage space of table T_i is less than $p_i^2 \log_2 p_i$ bits. Moreover, the whole storage space of $\{T_1, \dots, T_t\}$ is less than $\sum_{i=1}^t p_i^2 \log_2 p_i$ bits.

3.2. Protect the Random Number k . Attackers can obtain the final signature (r, s) in the white-box attack context. To protect the private key d , we also need to protect the random number k used in the process of ECDSA signature generation. Otherwise, according to equation (3), the private key $d = r^{-1}(ks - e) \pmod{n}$ is easy to calculate by attackers. We use the "cloud plus side" mode to protect it. k is decomposed into two parts, namely, k_1 and k_2 , where k_1 is generated by the client and k_2 is generated by the cloud server. In the step of signature generation, the inverse of k_2 needs to be sent to the client. In order to prevent attackers in the client recovering $k = k_1 \cdot k_2$, we use a series of permutations to protect k_2 .

TABLE 1: Lookup table T_i , $i = 1, \dots, t$.

Index	Value
(1, 1)	$T_i(1, 1) = f_i^{-1}(1) + g_i^{-1}(1)d_i \pmod{p_i}$
(1, 2)	$T_i(1, 2) = f_i^{-1}(1) + g_i^{-1}(2)d_i \pmod{p_i}$
(1, 3)	$T_i(1, 3) = f_i^{-1}(1) + g_i^{-1}(3)d_i \pmod{p_i}$
...	...
(1, p_i)	$T_i(1, p_i) = f_i^{-1}(1) + g_i^{-1}(p_i)d_i \pmod{p_i}$
(2, 1)	$T_i(2, 1) = f_i^{-1}(2) + g_i^{-1}(1)d_i \pmod{p_i}$
(2, 2)	$T_i(2, 2) = f_i^{-1}(2) + g_i^{-1}(2)d_i \pmod{p_i}$
(2, 3)	$T_i(2, 3) = f_i^{-1}(2) + g_i^{-1}(3)d_i \pmod{p_i}$
...	...
(2, p_i)	$T_i(2, p_i) = f_i^{-1}(2) + g_i^{-1}(p_i)d_i \pmod{p_i}$
...	...
(p_i , 1)	$T_i(p_i, 1) = f_i^{-1}(p_i) + g_i^{-1}(1)d_i \pmod{p_i}$
(p_i , 2)	$T_i(p_i, 2) = f_i^{-1}(p_i) + g_i^{-1}(2)d_i \pmod{p_i}$
(p_i , 3)	$T_i(p_i, 3) = f_i^{-1}(p_i) + g_i^{-1}(3)d_i \pmod{p_i}$
...	...
(p_i , p_i)	$T_i(p_i, p_i) = f_i^{-1}(p_i) + g_i^{-1}(p_i)d_i \pmod{p_i}$

3.3. Our Scheme. In the following, we always assume that the cloud server is semihonest [37]. That is, (1) the cloud server stores the data without tampering with it; (2) the cloud server honestly executes every operation; and (3) the cloud server tries to learn the underlying information of the user data. In addition, the client is in a white-box attack context.

Now, we propose a white-box implementation of ECDSA. Before signing a message, we need an initialization phase to convert the private key d into a series of lookup tables $\{T_1, T_2, \dots, T_t\}$. A trusted third party (TTP) is needed in the initialization phase (see Figure 1). The initialization phase is performed in a secure environment.

Initialization:

Input: Private key d .

Output: A basis of the RNS $\{p_1, p_2, \dots, p_t\}$, the permutations $\{f_1, f_2, \dots, f_t\}$ and $\{g_1, g_2, \dots, g_t\}$, and the lookup tables $\{T_1, T_2, \dots, T_t\}$.

- (1) Client sends the private key d to TTP through a secure channel;
- (2) TTP randomly selects co-prime integers $p_1, p_2, \dots, p_t$ as a basis of the RNS, where $P = p_1 p_2 \dots p_t \geq n^2 + n$;
- (3) TTP randomly selects permutations f_i and g_i on $\mathbb{Z}_{p_i}$, $i = 1, \dots, t$;
- (4) TTP represents the private key as $\mathbf{d} = (d_1, d_2, \dots, d_t)$ according to the basis $\beta = \{p_1, p_2, \dots, p_t\}$, where $d_i = d \pmod{p_i}$;
- (5) TTP constructs table T_i as follows: $T_i(j, k) = f_i^{-1}(j) + g_i^{-1}(k)d_i \pmod{p_i}$, where f_i^{-1} and g_i^{-1} are the inverse mappings of f_i and g_i , respectively, with $j = 1, \dots, p_i$, $k = 1, \dots, p_i$;
- (6) TTP sends tables $\{T_1, \dots, T_t\}$ to client and sends permutations $\{f_1, \dots, f_t\}$ and $\{g_1, \dots, g_t\}$ to the cloud server through a secure channel. TTP then publishes the basis $\beta = \{p_1, p_2, \dots, p_t\}$.

FIGURE 1: Initialization.

The white-box implementation of ECDSA based on the RNS is as follows (see Figure 2).

Signature generation:

Input: Domain parameters $D = (q, FR, a, b, G, n, h)$, basis $\beta = \{p_1, p_2, \dots, p_t\}$, and message m .

Output: Signature $(\hat{r}, \hat{s})$.

- (1) Client computes $e = H(m)$, where $H(m)$ is the hash value of the message m .
- (2) Client randomly selects an integer $k_1 \in [1, n-1]$.
- (3) Client computes $P = k_1 G$.
- (4) Client sends e and P to the cloud server.
- (5) Cloud server randomly selects an integer $k_2 \in [1, n-1]$.
- (6) Cloud server computes $R = (x_1, y_1) = k_2 P$ and converts x_1 to an integer $\bar{x}_1$.
- (7) Cloud server computes the first part of the signature $\hat{r} = \bar{x}_1 \pmod{n}$. If $\hat{r} = 0$, then go to (2).
- (8) Cloud server computes k_2^{-1} , where $k_2^{-1} k_2 = 1 \pmod{n}$.
- (9) Cloud server computes $u = k_2^{-1} e \pmod{n}$, $v = k_2^{-1} \hat{r} \pmod{n}$.
- (10) Cloud server represents u under the basis β as $\mathbf{u} = (u_1, \dots, u_t)$ and confuses u_i by permutation f_i , i.e., $f_i(u_i)$. Similarly, cloud server represents v under the basis β as $\mathbf{v} = (v_1, \dots, v_t)$ and confuses v_i by permutation g_i , i.e., $g_i(v_i)$.
- (11) Cloud server sends $\hat{r}, f_1(u_1), \dots, f_t(u_t), g_1(v_1), \dots, g_t(v_t)$ to client.
- (12) Client computes $w_i = f_i^{-1}(f_i(u_i)) + g_i^{-1}(g_i(v_i))d_i \pmod{p_i}$ by looking up table T_i , where the input of table T_i is $(f_i(u_i), g_i(v_i))$ and the output of table T_i is w_i , where $d_i = d \pmod{p_i}$, $i = 1, \dots, t$.
- (13) Client restores w by Chinese Remainder Theorem.
- (14) Client computes k_1^{-1} , where $k_1^{-1} k_1 = 1 \pmod{n}$.
- (15) Client computes the second part of the signature $\hat{s} = k_1^{-1} w \pmod{n}$. If $\hat{s} = 0$, then go to (2).

(16) Client returns the signature $(\hat{r}, \hat{s})$.

From the above steps, the private key d does not appear in plaintext. If the basis β and permutations $\{f_1, \dots, f_t\}$ and $\{g_1, \dots, g_t\}$ are fixed, then tables $\{T_1, \dots, T_t\}$ correspond to the private key d are fixed. Tables $\{T_1, \dots, T_t\}$ are completely presented by the client, so attackers on the cloud server cannot directly obtain any information about the private key d .

Let $k = k_1 k_2$; then, $R = k_2 P = k_2 (k_1 G) = kG$. It is obvious that the first part of the signature is the same in the classical ECDSA and in our scheme.

In the classical ECDSA signature generation algorithm, the second part of the signature is

$$\begin{aligned}
 s &= k^{-1} (e + dr) \pmod{n} \\
 &= k_1^{-1} k_2^{-1} (e + dr) \pmod{n} \\
 &= k_1^{-1} (k_2^{-1} e + k_2^{-1} r d) \pmod{n} \\
 &= k_1^{-1} (u + v d) \pmod{n}.
 \end{aligned} \tag{4}$$

Let $\bar{w} = u + v d \pmod{n}$. If $\bar{w} = w$, then $s = \hat{s}$. We claim that $\bar{w} = w$. Since $u, v, d \leq n$, $|u + v d| \leq n^2 + n$. Note that the basis $\beta = \{p_1, \dots, p_t\}$ satisfies $P = p_1 \dots p_t \geq n^2 + n$. Using RNS theory, we can restore $\bar{w}$ by $(\bar{w}_1, \dots, \bar{w}_t)$, where $\bar{w}_i = u_i + v_i d_i \pmod{p_i}$, $u_i = u \pmod{p_i}$, $v_i = v \pmod{p_i}$, and $d_i = d \pmod{p_i}$, with $i = 1, \dots, t$. Since

$$\begin{aligned}
 \bar{w}_i &= u_i + v_i d_i \pmod{p_i} \\
 &= f_i^{-1}(f_i(u_i)) + g_i^{-1}(g_i(v_i))d_i \pmod{p_i} \\
 &= w_i,
 \end{aligned} \tag{5}$$

then $\bar{w} = w$.

Thus, the result $(\hat{r}, \hat{s})$ of the white-box implementation of ECDSA signature generation is consistent with the result (r, s) of the classical ECDSA signature generation. One advantage of our scheme is that we can directly use the classical ECDSA signature verification algorithm to verify the signature $(\hat{r}, \hat{s})$.

4. Security Analysis

4.1. The Security in the Cloud Server and the Client. Firstly, we analyze the security of our scheme from the aspects of the cloud server and the client. We assume that the client is in a white-box attack context and that the cloud server is semihonest. The goal of attackers is to extract the private key d . The security of our scheme is based on the difficulty of the discrete logarithm problem for elliptic curves (ECDLP) and the diversity of permutations.

4.1.1. The Security in the Cloud Server. From the details of our scheme, attackers in the cloud server can obtain the information of $e, P, k_2, \hat{r}$, and $\hat{s}$. Moreover, the following equation holds:

$$\hat{s} = k_1^{-1} k_2^{-1} (e + d\hat{r}) \pmod{n}. \tag{6}$$

FIGURE 2: The white-box implementation of ECDSA based on RNS.

If attackers can obtain the value of k_1 , they can extract the private key $d = \hat{r}^{-1}(k_1 k_2 \hat{s} - e) \pmod{n}$ from equation (6). Fortunately, because of the difficulty of the elliptic curve discrete logarithm, attackers cannot calculate k_1 by $P = k_1 G$.

4.1.2. The Security in the Client. From the details of our scheme, attackers in the client can obtain the information of $\{T_1, \dots, T_t\}$, e , k_1 , $f_1(u_1), \dots, f_t(u_t)$, $g_1(v_1), \dots, g_t(v_t)$, $\hat{r}$, and $\hat{s}$. There seem to be two ways for attackers to compute the private key: (i) from tables $\{T_1, \dots, T_t\}$ and (ii) from equation (6).

In case (i), table T_i is constructed by

$$T_i(j, k) = f_i^{-1}(j) + g_i^{-1}(k)d_i \pmod{p_i}, \quad (7)$$

where $i = 1, \dots, t, j = 1, \dots, p_i, k = 1, \dots, p_i$. After the initialization phase, the key d_i and permutations f_i and g_i are hidden in T_i . Since the permutations f_i and g_i are unknown

to the client, attackers in the client cannot derive the key d_i from table T_i directly. The diversity of permutation f_i (or g_i) is $p_i!$. Hence, the complexity of deriving d_i is $(p_i!)^2$ through an exhaustive search attack. The computational complexity of extracting the private key d from case (i) is $\prod_{i=1}^t (p_i!)^2$.

In case (ii), if attackers can obtain the value of k_2 , they can extract the private key d from equation (6). For the client, the information of k_2 is hidden in $f_1(u_1), \dots, f_t(u_t)$, $g_1(v_1), \dots, g_t(v_t)$, where $\mathbf{u} = k_2^{-1}e = (u_1, \dots, u_t)$ and $\mathbf{v} = k_2^{-1}\hat{r} = (v_1, \dots, v_t)$. Since the permutations are unknown to the client, the complexity of deriving k_2 is $\prod_{i=1}^t p_i!$ through an exhaustive search attack. The computational complexity of extracting the private key d from case (ii) is $\prod_{i=1}^t p_i!$.

Remark. The proposed scheme is a kind of cloud collaboration. The secret key is stored completely in the client in a form of lookup table. The attackers in the client cannot get the plaintext information of the key from the lookup table.

At the same time, even if the cloud is untrusted, the attacker in the cloud cannot obtain any information about the key. So, the proposed scheme alleviates the problems of local key disclosure and identity authentication in general cloud collaboration.

4.2. Random Number Attack. In the classical ECDSA, the random integer k must be different for different messages. Assume that, for different messages m' and m'' , we select the same random integer k to sign. Then, we have

$$\begin{cases} s' = k^{-1}(e' + r d) \pmod{n}. \\ s'' = k^{-1}(e'' + r d) \pmod{n}. \end{cases} \quad (8)$$

Thus, $d = (s' - s'')^{-1} r^{-1} (s'' e' - s' e'') \pmod{n}$. Usually, the information of e' , e'' , (r, s') , and (r, s'') is public; thus, attackers can easily obtain the information of the private key d .

In the white-box implementation of ECDSA, the random integers k_1 and k_2 are generated by the client and cloud server, respectively. For two messages m' and m'' , assume that k_1' and k_2' are used to compute the signature (r', s') of m' and that k_1'' and k_2'' are used to compute the signature (r'', s'') of m'' . Then, we have

$$\begin{cases} \hat{s}' = k_1'^{-1} k_2'^{-1} (e' + d r') \pmod{n}, \\ \hat{s}'' = k_1''^{-1} k_2''^{-1} (e'' + d r'') \pmod{n}. \end{cases} \quad (9)$$

Under the white-box attack context, the attackers in the client can obtain the information of k_1' , k_2' , e' , e'' , (r', s') , and (r'', s'') , and the attackers in the cloud server can obtain the information of k_1'' , k_2'' , e' , e'' , (r', s') , and (r'', s'') . We analyze the security of private key d in the following four cases:

(i) $k_1' = k_1''$, $k_2' = k_2''$:

From equation (9), we have $d = (\hat{s}' r'' - \hat{s}'' r')^{-1} (s'' e' - s' e'') \pmod{n}$. Thus, attackers in the client or the cloud server can derive the private key d under the white-box attack context.

(ii) $k_1' = k_1''$, $k_2' \neq k_2''$:

From equation (9), we have $d = (\hat{s}' k_2' r'' - \hat{s}'' k_2'' r')^{-1} (s'' k_2' e' - s' k_2'' e'') \pmod{n}$. Thus, attackers in the cloud server can derive the private key d under the white-box attack context.

(iii) $k_1' \neq k_1''$, $k_2' = k_2''$:

From equation (9), we have $d = (\hat{s}' k_1' r'' - \hat{s}'' k_1'' r')^{-1} (s'' k_1' e' - s' k_1'' e'') \pmod{n}$. Thus, attackers in the client can derive the private key d under the white-box attack context.

(iv) $k_1' \neq k_1''$, $k_2' \neq k_2''$:

Regardless of whether the attackers are in the client or the cloud server, they cannot simultaneously obtain the information of k_1' , k_1'' , k_2' , and k_2'' . Thus, attackers cannot obtain the information of private key d from equation (9) under the white-box attack context.

TABLE 2: Domain parameters $D = (q, FR, a, b, G, n, h)$ and basis β of RNS.

Parameters	Value
q	0x 80000000 00000000 00000000 00000000 00000000 00000000 00000000 00001103
FR	Integer representation
a	0
b	3
x_G	0x 2BB687F3 4C71B55A 06E63561 EBE3144E E1AA9C00 3C165107 2EE3631A A2729B88
y_G	0x 13FC9FB0 9308BC19 59B42204 8D81BDA5 B6A2183F 1DBCADD8 2C245EC3 3CEB8E60
n	0x 80000000 00000000 00000000 00000001 6288AEDF 18093517 F7A3E3CF F023CBFF
h	1
β	{97, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479}

In summary, the random integers k_1 and k_2 used in our scheme must be different for different messages.

4.3. Code Lifting Attack. One of the main purposes of the white-box cryptography is to protect the security of the algorithm implemented in the software used. It is well known that the algorithm implemented in the software is at risk of a code lifting attack (i.e., the entire code will be extracted and used as an equivalent secret key). To resist the code lifting attack, one common method is to bind the hardware information to the code. The method can also be used in the white-box implementation of ECDSA. Only two steps of our scheme proposed in Section 3 need to be modified: (i) the construction of table $\{T_1, \dots, T_t\}$ in the initialization phase and (ii) step (13) in the signature generation phase.

(i) The new lookup tables $\{T'_1, \dots, T'_t\}$ contain not only the private key $\mathbf{d} = (d_1, \dots, d_t)$ but also the unique identification information of the device $HI = (HI_1, \dots, HI_t)$, where $HI_i = HI \pmod{p_i}$. The new lookup table T'_i is constructed as follows:

$$T'_i(j, k) = f_i^{-1}(j) + g_i^{-1}(k) d_i + HI_i \pmod{p_i}, \quad (10)$$

where $i = 1, \dots, t$, $j = 1, \dots, p_i$, and $k = 1, \dots, p_i$.

(ii) The new step (13) is as follows: client restores w by Chinese Remainder Theorem and computes $w' = w - HI \pmod{n}$.

It is easy to check the correctness of the new scheme. The HI is hidden in lookup tables $\{T'_1, \dots, T'_t\}$. For each signature, the HI must be correctly extracted from the device. Through the hardware binding method, the code lifting

TABLE 3: Comparison of computation cost.

Operation	Scheme	
	Classical ECDSA (in millisecond)	Our scheme (in millisecond)
Initialization	0	4881.18
Signature generation	5.11	1749.61
Signature verification	5.98	6.07

TABLE 4: Comparison of signature generation cost.

The size of message (KB)	Classical ECDSA (in millisecond)	Our scheme (in millisecond)
1	2.99	1742.20
10	3.19	1758.44
100	4.79	1766.13
1	20.90	1797.68
10	195.27	1980.05
100	1857.29	3641.08
1	18547.54	20430.23

attack can be mitigated in the white-box implementation of ECDSA.

5. Performance Evaluation

In this section, we analyze the performance of our proposed scheme and compare it with the classical ECDSA [31]. The simulation is programmed by C language. Visual Studio 2015 is administered on a PC with an Intel Core i7-6700 3.40 GHz CPU and Windows 10 operating system using the MIRACL library. SM3 algorithm is used to calculate the hash values of the messages.

The domain parameters $D = (q, FR, a, b, G, n, h)$ on the elliptic curve $y^2 = x^3 + ax + b$, and the basis β of RNS are shown in Table 2. The size of the lookup table is about 35.4 MB. We run each experiment 100 times and calculate the average. All the times reported in Tables 3 and 4 are averages.

In Table 3, we compare the computation cost of our scheme with the classical ECDSA, where the size of the message is 100 KB. The classic ECDSA includes two stages: signature generation and signature verification. Compared with the classical ECDSA, our scheme proposed in Section 3 has one more stage: initialization. As shown in Table 3, the time of initialization of our scheme is 4881.18 milliseconds, which is the main cost of our scheme. Fortunately, this stage is usually executed by a TTP who possesses strong processing capability, so the time will be shorten. What is more, the initialization stage needs to be executed again only when the user needs to change the key. The user's private key is not often replaced in the actual application scenario, so the time of initialization is acceptable. The times of signature generation in classical ECDSA and our scheme are 5.11 milliseconds and 1749.61 milliseconds, respectively. Although the calculation time of our scheme is more, the security of

TABLE 5: Notations.

Symbol	Meaning
q	Prime number
F_q	The finite field with order q
$G = (x_G, y_G)$	Point in elliptic curve $E(F_q)$
d	Private key
$Q = dG$	The public key corresponding to d
(r, s)	A signature generated by classical ECDSA
$(\hat{r}, \hat{s})$	A signature generated by our scheme
k, k_1, k_2	Random number
$k^{-1}, k_1^{-1}, k_2^{-1}$	The inverses of k, k_1 , and k_2 modulo n , respectively
m	Message
e	The hash value of a message
$H(\cdot)$	Hash function
RNS	Residue number system
p_i	Prime number, $1 \leq i \leq t$
$\beta = \{p_1, \dots, p_t\}$	A basis of the RNS
P	$P = \prod_{i=1}^t p_i$, the product of $p_1, p_2, \dots$, and p_t
d, u, v	d modulo p_i , u modulo p_i , and v modulo p_i , respectively
$\mathbf{d}, \mathbf{u}, \mathbf{v}$	The RNS transformation of d, u , and v based on the basis β , respectively
$\mathbb{Z}_{p_i}$	The prime field with order p_i
f_i, g_i	Permutations on $\mathbb{Z}_{p_i}$, $1 \leq i \leq t$
f_i^{-1}, g_i^{-1}	The inverse mappings of f_i and g_i , respectively
$T_i(\cdot, \cdot)$	Lookup tables constructed as in Table 1, $1 \leq i \leq t$

the proposed scheme is higher. Note that the signature verification is the same in two schemes, so the times of signature verification are essentially equal.

In Table 4, we analyze the signature generation time of the message with different sizes, that is, 1 kB, 10 kB, 100 kB, 1 M, 10 M, 100 M, and 1 GB. The longer the size of the message is, the longer it takes to compute the hash value of the message. So, the time of signature generation is increasing when the size of the message is increasing in two schemes. Generally, our scheme takes an extra 1.7 to 1.9 seconds in the phase of signature generation.

6. Conclusions

A white-box implementation of ECDSA based on the RNS in the “cloud plus side” mode is proposed. Not only it can protect the security of the private key under the white-box attack context, but also it has good compatibility, has a low communication cost, and requires a small storage space. The proposed techniques can be extended to Schnorr signatures [38] or other ElGamal-based schemes [39]. The study of the white-box implementation of ECDSA can provide powerful theoretical support for the software implementation of the digital signature algorithm and expand the applied scenario of ECDSA. However, the white-box implementation of public-key cryptography is still in the preliminary stage, and its security analysis method is not complete. Our future work will focus on the definition and the security analysis of white-box public-key cryptography.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Appendix

The following Table 5 lists the symbols used in this article.

Acknowledgments

This work was supported in part by the National Key Research and Development Program under China Grant 2017YFC0801004.

References

- [1] Y. Shi, X. Wang, and H. Fan, "Light-weight white-box encryption scheme with random padding for wearable consumer electronic devices," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 1, pp. 44–52, 2017.
- [2] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot, "White-box cryptography and an AES implementation," in *SAC 2002, Lecture Notes in Computer Science*, vol. 2595, pp. 250–270, Springer, Berlin, Germany, 2002.
- [3] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.
- [4] Y. Law, R. Corin, S. Etalle, and P. Hartel, "A formally verified decentralized key management architecture for wireless sensor networks," in *PWC 2003, Lecture Notes in Computer Science*, vol. 2775, pp. 27–39, Springer, Berlin, Germany, 2003.
- [5] M. Beunardeau, A. Connolly, R. Géraud, and D. Naccache, "White-box cryptography: security in an insecure environment," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 88–92, 2016.
- [6] O. Billet, H. Gilbert, and C. Ech-Chatbi, "Cryptanalysis of a white box AES implementation," in *SAC 2004, Lecture Notes in Computer Science*, vol. 3357, pp. 227–240, Springer, Berlin, Germany, 2004.
- [7] J. Bringer, H. Chabanne, and E. Dottax, "White box cryptography: another attempt," *Cryptology ePrint Archive*, Report 2006/468, 2006, <http://eprint.iacr.org/2006/468>.
- [8] Y. D. Mulder, B. Wyseur, and B. Preneel, "Cryptanalysis of a perturbed white-box AES implementation," in *INDOCRYPT 2010, Lecture Notes in Computer Science*, vol. 6498, pp. 292–310, Springer, Berlin, Germany, 2010.
- [9] Y. Xiao and X. Lai, "A secure implementation of white-box AES," in *Proceedings of the 2nd International Conference on Computer Science and its Applications*, pp. 1–6, Jeju, South Korea, December 2009.
- [10] Y. D. Mulder, P. Roelse, and B. Preneel, "Cryptanalysis of the Xiao-Lai white-box AES implementation," in *SAC 2012, Lecture Notes in Computer Science*, vol. 7707, pp. 34–49, Springer, Berlin, Germany, 2012.
- [11] T. Xu, F. Liu, and C. Wu, "A white-box AES-like implementation based on key-dependent substitution-linear transformations," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 18117–18137, 2017.
- [12] R. Luo, X. Lai, and R. You, "A new attempt of white-box AES implementation," in *Proceedings of the International Conference on Security, Pattern Analysis, and Cybernetics*, pp. 423–429, Wuhan, China, October 2014.
- [13] K. Bai, C. K. Wu, and Z. Zhang, "Protect white-box AES to resist table composition attacks," *IET Information Security*, vol. 12, no. 4, pp. 305–313, 2018.
- [14] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot, "A white-box DES implementation for DRM applications," in *DRM 2002, Lecture Notes in Computer Science*, vol. 2696, pp. 1–15, Springer, Berlin, Germany, 2002.
- [15] M. Jacob, D. Boneh, and E. W. Felten, "Attacking an obfuscated cipher by injecting faults," in *DRM 2002, Lecture Notes in Computer Science*, vol. 2696, pp. 16–31, Springer, Berlin, Germany, 2002.
- [16] H. Link and W. Neumann, "Clarifying obfuscation: improving the Security of White-Box DES," in *Proceedings of the ITCC'05*, pp. 679–684, Las Vegas, NV, USA, April 2005.
- [17] L. Goubin, J.-M. Masereel, and M. Quisquater, "Cryptanalysis of white box DES implementations," in *SAC 2007, Lecture Notes in Computer Science*, vol. 4876, pp. 278–295, Springer, Berlin, Germany, 2007.
- [18] B. Wyseur, W. Michiels, P. Gorissen, and B. Preneel, "Cryptanalysis of white-box DES implementations with arbitrary external encodings," in *SAC 2007, Lecture Notes in Computer Science*, vol. 4876, pp. 264–277, Springer, Berlin, Germany, 2007.
- [19] A. Amadori, W. Michiels, and P. Roelse, "A DFA attack on white-box implementations of AES with external encodings," in *SAC 2019, Lecture Notes in Computer Science*, vol. 11959, pp. 591–617, Springer, Berlin, Germany, 2020.
- [20] T. Lin, H. Yan, X. Lai, and W. Qiu, "Several methods for constructing white-box solutions," *Journal of Physics: Conference Series*, vol. 1288, Article ID 012005, 2019.
- [21] A. Biryukov, C. Bouillaguet, and D. Khovratovich, "Cryptographic schemes based on the ASASA structure: block-box, white-box, and public-key (extended abstract)," in *ASIA-CRYPT 2014, Lecture Notes in Computer Science*, vol. 8873, pp. 63–84, Springer, Berlin, Germany, 2014.
- [22] H. Gilbert, J. Plüt, and J. Treger, "Key-Recovery attack on the ASASA cryptosystem with expanding S-boxes," in *CRYPTO 2015, Lecture Notes in Computer Science*, vol. 9215, pp. 475–490, Springer, Berlin, Germany, 2015.
- [23] B. Minaud, P. Derbez, P.-A. Fouque, and P. Karpman, "Key-recovery attacks on ASASA," *Journal of Cryptology*, vol. 31, no. 3, pp. 845–884, 2018.
- [24] J. Kwon, B. Lee, J. Lee, and D. Moon, "FPL: White-box secure block cipher using parallel table look-ups," in *CT-RSA 2020, Lecture Notes in Computer Science*, vol. 12006, pp. 106–128, Springer, Berlin, Germany, 2020.
- [25] E. Alpirez Bock, A. Amadori, C. Brzuska, and W. Michiels, "On the security goals of white-box cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 2, pp. 327–357, 2020.
- [26] Y. Shi, W. Wei, F. Zhang et al., "SDSRS: a novel white-box cryptography scheme for securing embedded devices in IIoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1602–1616, 2019.
- [27] Y. Zhang, D. He, X. Huang, D. Wang, and K. R. Choo, "White-box implementation of the identity-based signature scheme in the IEEE P1363 standard for public key cryptography," *Cryptology ePrint Archive: Report 2018/814*, 2018.
- [28] Q. Feng, D. He, H. Wang, N. Kumar, and K.-K. R. Choo, "White-box implementation of Shamir's identity-based

- signature scheme,” *IEEE Systems Journal*, vol. 14, no. 2, p. 1820, 2020.
- [29] D. Johnson, A. Menezes, and S. Vanstone, “The elliptic curve digital signature algorithm (ECDSA),” *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.
 - [30] ISO, *Information Technology-Security Techniques-Digital Signatures with Appendix-Part 3: Certificate Based Mechanisms*, ISO/IEC 14888-3, 1998, ISO–International Organization of Standards, Geneva, Switzerland.
 - [31] American National Standards Institute, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, ANSI X9.62, American National Standards Institute, New York, NY, USA, 1999.
 - [32] Institute of Electrical and Electronics Engineers, *Standard Specifications for Public Key Cryptography, IEEE Std 1363-2000*, Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 2000.
 - [33] National Institute of Standards and Technology, *Digital Signature Standard, FIPS 186-2*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2000.
 - [34] H. L. Garner, “The residue number system,” in *Proceedings of the IRE-AIEE-ACM’59*, pp. 146–153, San Francisco, CA, USA, March 1959.
 - [35] W. Jenkins and B. Leon, “The use of residue number systems in the design of finite impulse response digital filters,” *IEEE Transactions on Circuits and Systems*, vol. 24, no. 4, pp. 191–201, 1977.
 - [36] K. B. Sudeepa and G. Aithal, “Generation of maximum length non-binary key sequence and its application for stream cipher based on residue number system,” *Journal of Computational Science*, vol. 21, pp. 379–386, 2017.
 - [37] Q. Chai and G. Gong, “Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers,” in *Proceedings of the ICC 2012*, vol. 922, p. 917–, Ottawa, Canada, June 2012.
 - [38] C. P. Schnorr, “Efficient signature generation by smart cards,” *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
 - [39] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.