information

Article

Profiling Attack against RSA Key Generation Based on a Euclidean Algorithm

Sadiel de la Fe [1,*], Han-Byeol Park [2], Bo-Yeon Sim [3], Dong-Guk Han [4] and Carles Ferrer [1]

[1] Department of Microelectronic and Systems, Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain; carles.ferrer@uab.cat
[2] Department of Financial Information Security, Kookmin University, Seoul 04718, Korea; hanbyeol@kookmin.ac.kr
[3] Department of Mathematics, Kookmin University, Seoul 04718, Korea; qjdusls@kookmin.ac.kr
[4] Department of Mathematics and Financial Information Security, Kookmin University, Seoul 04718, Korea; christa@kookmin.ac.kr
* Correspondence: sadiel.delafe@e-campus.uab.cat

Abstract: A profiling attack is a powerful variant among the noninvasive side channel attacks. In this work, we target RSA key generation relying on the binary version of the extended Euclidean algorithm for modular inverse and GCD computations. To date, this algorithm has only been exploited by simple power analysis; therefore, the countermeasures described in the literature are focused on mitigating only this kind of attack. We demonstrate that one of those countermeasures is not effective in preventing profiling attacks. The feasibility of our approach relies on the extraction of several leakage vectors from a single power trace. Moreover, because there are known relationships between the secrets and the public modulo in RSA, the uncertainty in some of the guessed secrets can be reduced by simple tests. This increases the effectiveness of the proposed attack.

Keywords: Euclidean algorithm; GCD; RSA key generation; side channel attack; profiling attack; machine learning-based attack

Citation: de la Fe, S.; Park, H.-B.; Sim, B.-Y.; Han, D.-G.; Ferrer, C. Profiling Attack against RSA Key Generation Based on a Euclidean Algorithm. *Information* **2021**, *12*, 462. https://doi.org/10.3390/info12110462

Academic Editors: Berk Gulmezoglu and Koksal Mus

Received: 23 September 2021
Accepted: 4 November 2021
Published: 9 November 2021

1. Introduction

Modular inversions are widely used in cryptography. For example, modular inversions are used in signature schemes such as the ECDSA (elliptic curve digital signature algorithm) and the key generation stage of the RSA (Rivest, Shamir and Adleman cryptosystem). Some multiplicative masking techniques also employ modular inversions. Montgomery multiplication, which is extensively used for multiprecision operands, requires inversion as well.

A well-known method to solve $a^{-1} mod\ p$ is the algorithm based on Fermat's Little Theorem (FLT), even though it is valid only for the prime modulus [1]. The Montgomery inversion is another method that has a variant (almost Montgomery Inversion) that produces the modular inverse multiplied by 2^k [2]. The latter can be conveniently used to accelerate the computation. Other methods to produce $a^{-1} mod\ p^k$ were introduced in [3–6]. The special case of these methods for modulo 2^k is useful to compute the Montgomery constant. Some of these methods are based on the Euclidean algorithm.

A binary variant of the Euclidean algorithm, introduced in [7], computes the greatest common divisor (GCD) of two integers. The extended variant (BEEA, binary extended Euclidean algorithm) also gives the inverse of one of the inputs modulo the other [8]. This is a very helpful algorithm, because it can work modulo a prime and modulo a composite as well. In fact, it is currently used in one of the most widely used cryptographic schemes to date, namely, the above-mentioned RSA. The key generation stage of the RSA performs (after the primes generation) coprimality tests and, at least, an inversion modulo a composite. These two procedures can be performed using the same BEEA algorithm.

Such implementation can be found in many products, for example, banking smartcards. Unfortunately, some successful side channel attacks (SCAs) have been reported against BEEAs.

The execution flow of the Euclidean algorithm is directly related to its inputs. This allowed Acıiçmez et al. in [9] to deduce the inputs by first identifying the specific branches executed in a classical BEEA implementation [10]. Similar vulnerabilities related to BEEA power consumption have also been exploited in [11], where the authors retrieve by simple power analysis (SPA) all of the secret bits of an ECDSA nonce during the modular inverse operation using a low number of power traces. The authors claim their attack is still effective in the presence of the countermeasures introduced in [12]. Moreover, the authors discuss some countermeasures to overcome the SPA risks. In [13], an SPA is conducted against an RSA key generation implementation based on a binary Euclidean algorithm. In this work, the authors retrieve the secret by deducing the whole execution flow of the BEEA (implemented as in [10]). The power traces of hardware and software implementations given in the work show that the right-shifts and the subtraction operations can be easily distinguished, thus acting as markers of the conditional branches. This attack exploits the difference between the bit length of the inputs, and then makes it easy to infer the input being manipulated at each branch. A similar SPA is described in [14], except that the target is the specific case of the GCD.

With one exception, all of the aforementioned attacks were performed over nonprotected implementations of the binary Euclidean algorithm. In all cases, they exclusively focused on the SPA. However, as we will discuss in the next section, countermeasures can be used to prevent this kind of attack.

On the other hand, the authors of [15] conduct an attack against an ECDSA method based on an NIST curve. The first stage of their attack targets a variant of the Euclidean algorithm (which they do not know, but they infer from an SPA) that apparently is used to invert the nonce involved in the signature generation. A template attack (TA) is carried out after the initial stage, but the authors give new details about its implementation.

Due to the COVID-19 pandemic, non-face-to-face services have gained more importance to reduce human contact. In many cases, user certification is essential to provide this kind of service. Secure authentication through a public-key infrastructure (PKI) is used for this purpose. This work addresses a relevant security issue that may affect the key generation mechanism that is a critical part of a PKI.

Profiling attacks (PAs) are a powerful form of SCA that have not been sufficiently considered to exploit the power-based leakage that may exist in a BEEA implementation. The key contribution in this work is the introduction of a PA against an RSA key generation relying on the BEEA. The provided theoretical demonstration shows that the attack succeeds even in the presence of a certain masking technique, while it is possible to extract several leakage vectors from a single power trace. In addition, a machine learning-based profiling attack (ML-based PA) is performed to validate our approach. Moreover, based on existing relationships between secret and public values, a simple test is suggested to strengthen the reliability of the obtained profiles. Consequently, the use of more effective protection in BEEA implementations is recommended.

This paper is organized as follows: In Section 2, an introduction to RSA key generation is given, and some of the BEEA details are emphasized. Section 3 addresses the PAs fundamentals. An ML-based PA on BEEA-based RSA key generation is described in Section 4. The experimental results are shown in Section 5. Finally, conclusions are given in Section 6.

2. RSA Key Generation

The RSA cryptosystem security relies on the intractability of factorizing a large integer, namely, $N = p \cdot q$, where p and q are randomly chosen large prime numbers [16]. This

method is widely used for encryption and authentication purposes. The main operations of the RSA are given by:

$$s = m^d mod\ N, \tag{1}$$

and

$$m = s^e mod\ N, \tag{2}$$

where m is the input message and s is its corresponding signature.

Additionally, the private key is obtained following:

$$d = e^{-1} mod\ \phi(N), \tag{3}$$

where e is the public key and $\phi(N)$ is the Euler totient function that is computed as

$$\phi(N) = (p-1) \cdot (q-1). \tag{4}$$

There are two characteristics of $\phi(N)$ that are quite important for the objective of this work. $\phi(N)$ is divisible by four because it results from the multiplication of two even numbers. Besides, it is known that $\phi(N)$ shares approximately half of its most significant bytes (MSB) with N.

The key pair generation in RSA starts looking for two large prime numbers. This may be a long process due to the primality tests that must be carried out for the randomly chosen candidates. The algorithm based on FLT is often used to perform these tests.

Once the prime factors are generated, the public exponent e is chosen. Then, to guarantee that e is coprime with $p-1$ and $q-1$, coprimality tests are performed. The Euclidean algorithm is widely employed to carry out these tests. Finally, d is computed as above, and the Euclidean algorithm is usually used in this process as well.

In resource-constrained devices, for efficiency, the public key may be pre-established with a convenient value (e = 65,537 is commonly used). The coprimality tests involving e, $p-1$ and $q-1$ are computed separately to update p or q if necessary.

The following steps resume a key generation stage on the RSA, where e is pre-established:

1. Generate two random primes p and q;
2. Verify that e is coprime with $p-1$ and $q-1$;
3. Compute the modulus $N = p \cdot q$;
4. Compute $\phi(N) = (p-1) \cdot (q-1)$;
5. Compute the private key $d = e^{-1} mod\ \phi(N)$.

2.1. The Binary Extended Euclidean Algorithm

The BEEA computes the inverse of a modulo m by solving Bézout's identity [17]. The fact that this algorithm works for both prime and composite moduli makes it quite versatile. In addition to being used in the RSA key generation, BEEA can also be employed to invert the nonce k in ECDSA. These can be considered the two currently most commonly used asymmetric cryptosystems.

The Euclidean algorithm is a suitable choice for hardware devices. It substitutes costly divisions by shifts, allowing more efficient circuits. Recently, it was demonstrated that a BEEA variant implementation performs better than FLT in the processors based on the residue number system (RNS) [18,19]. RNS arithmetic allows for the construction of efficient parallel circuits for large operands.

In the remainder of this paper, we will refer to a generic BEEA coded as Algorithm 1, which is its classical definition [8]. Note that we avoid including the oddness verification of the inputs for simplicity. We write the algorithm following a notation corresponding to the RSA private key computation: BEEA(e, $\phi(N)$), although it will be used later to analyze the coprimality test that has different inputs namely, GCD(e, $p-1$) and GCD(e, $q-1$). BEEA and GCD are conveniently used to emphasize the different operations required at certain key generation steps. However, particularly since Section 3, BEEA and GCD always refer

to a unique implementation of the binary extended Euclidean algorithm that gives both the modular inverse of one input and the greatest common divisor of both inputs.

It is observed from Algorithm 1 that the *v-loop* (step 12) will be executed first because $v = \phi(N)$ is even, but e is not. In addition, the *v-loop* will run twice because $\phi(N)$ is divisible by four. Upon entering this loop for the first time, D is odd; thus, the ELSE branch (step 17) will be executed. In the second iteration, and C and D are both even; thus, the flow will go through the IF branch (step 14). Once the shifted v becomes odd, the *v-loop* ends, and the algorithm flow continues with the verification at step 18. In the case of the coprimality tests of RSA, only one iteration of the *v-loop* can be predicted, because it can only be ensured that both secrets $p-1$ and $q-1$ are even.

Algorithm 1 Binary Extended Euclidean Algorithm.

Inputs: e and $\phi(N)$; where e is odd
Outputs: GCD $(e, \phi(N))$ and $d = e^{-1} mod\ \phi(N)$

1. $u = e$
2. $v = \phi(N)$
3. $A = D = 1$
4. $B = C = 0$
5. **while** $u \neq 0$ do
6. **while** (u is even) do ---------- u-loop
7. $u = u/2$
8. **if** (A is even) **and** (B is even) **then**
9. $A = A/2;\ \ B = B/2$
10. **else**
11. $A = (A + \phi(N))/2;\ \ B = (B - e)/2$
12. **while** (v is even) do ---------- v-loop
13. $v = v/2$
14. **if** (C is even) **and** (D is even) **then**
15. $C = C/2;\ \ D = D/2$
16. **else**
17. $C = (C + \phi(N))/2;\ \ D = (D - e)/2$
18. **if** $u \geq v$ **then**
19. $u = u - v$
20. $A = A - C$
21. $B = B - D$
22. **else**
23. $v = v - u$
24. $C = C - A$
25. $D = D - B$
26. **return** $(v = GCD(e, \phi(N)), d = C\ mod\ \phi(N))$

2.2. Countermeasures against SCA on BEEA

A large bit length difference in BEEA inputs is exploited to predict the execution of the algorithm and extract sensitive information, as mentioned above. Therefore, the countermeasures that have been proposed to date focus on making both operands the same size or directly mask the sensitive input.

In [14], two methods were patented to protect the GCD algorithm from an SPA. The first one follows the property:

$$GCD(X - 1; e) = GCD(X - 1 + r \cdot e; e). \tag{5}$$

It consists of applying additive masking to $X - 1$, where X is a secret prime (p or q) in an RSA scheme, e is the public key, and r is a random. If r is a sufficiently large nonce (e.g.,

$sizeof(X) - sizeof(e)$), then all of the bits of the secret are masked. In such a case, even if an adversary is able to follow the complete algorithm flow, no relevant information will be disclosed.

Note that, even assuming the case in which an adversary obtains the masked $X' = X - 1 + r \cdot e$ through an SPA and X' is reduced modulo e, the brute force complexity to obtain X will make the attack unfeasible.

This is because the length of X has been properly chosen (e.g., $\geq$512 bits), and $sizeof(X)$ is much larger than $sizeof(e)$, which is the common scenario.

The second method introduced by Chartier in [14] relies on the property:

$$GCD(X - 1; e) = GCD(GCD(X - 1; r \cdot e); e). \quad (6)$$

In [11,20], the proposed countermeasure to protect the BEEA from SPA is based on Equation (6). The same technique is also used to protect the modular inversion in the well-known open source library Mbed TLS [21]. This countermeasure ensures that both operands have the same bit length to avoid the algorithm's flow prediction. In this case, if the random number is a sufficiently large nonce, the specific conditional branch that the algorithm takes at each iteration cannot be guessed by an SPA on the power trace. This removes the vulnerability exploited in [11]. The steps to blind the secret k are as follows:

1. Select r at random such that $0 < r < p$;
2. Compute $a = k \cdot r \ mod \ p$;
3. Compute $b = a^{-1} \ mod \ p$;
4. Compute $k^{-1} = r \cdot b \ mod \ p$.

These two methods, although they were originally intended to protect the coprimality tests involving e and the candidates for p and q, are equally useful to protect the private key computation in the case it is performed by means of the Euclidean algorithm (BEEA(e, $\phi(N)$)).

In Equation (5), the secret is blinded before the GCD execution. On the other hand, notice how the same secret is manipulated in plaintext in Equation (6). This is not relevant when SPA is the only threat being considered, and we emphasize that these countermeasures aim to mitigate such attacks. However, another kind of SCA may take advantage of this plaintext secret.

3. Profiling Attacks

In electronic devices, under certain conditions, a power consumption profile of some operations can be constructed. Using this power profile, an adversary may be able to extract sensitive data from the target device by matching its power consumption with the computed profile. When this PA is realized through the use of templates, the procedure is known as TA. This kind of attack was introduced by Chari et al. in [22], and is considered one of the strongest types of SCA reported to date. A necessary condition is that the device has side channel leakage related to the manipulated data.

An ideal scenario for performing a PA is to acquire both profiling and attack power traces from the same device in the same acquisition campaign. For this purpose, it is advantageous to have a profiling operation similar to the target operation in which inputs and outputs can be controlled. Under such conditions, the power consumption characterization will be close to optimal.

An attack method using deep learning technology in the process of constructing a profile is called ML-based PA. Deep learning is a machine learning technique that learns features from the input data through a neural network. The multilayer perceptron (MLP) is a deep-learning technology that has been proposed to overcome the limitations of a single-layer perceptron. The MLP consists of an input layer, hidden layers, and an output layer. The hidden layers learn the features from the data and finally compute the output value through an activation function.

Machine learning techniques, including MLP, have been evaluated for SCA improvements, particularly as a TA alternative, for the last decade [23–26]. Martinasek et al. demonstrated the feasibility of using MLP that, in some aspects, may overcome traditional profiling techniques, pointing out the difficult problem of parametrization. As an aside, the reader is recommended the work of Benadjila et al. [27], where the authors discuss several options for the parametrization problem in MLP when applied to SCA.

In this paper, the process of building a power consumption profile is performed based on the MLP model. The specific method of ML-based PA consists of the following two steps.

[Step1 : Learning Phase]

- First, the attacker collects several power consumption traces from a device running a BEEA and calculates an intermediate value for each of them;
- Then, the intermediate value corresponding to the power consumption is used as a label, and the profile is constructed through a training process by the MLP model.

[Step2 : Attack Phase]

- The attacker calculates the matching probability between the power consumption measured from the attack device and the profile constructed in Step 1;
- The value with the highest probability is adopted as the correct intermediate value; then, the secret value is recovered.

Single Trace Profiling Attacks

In a convenient scenario for an adversary, a large number n_a of power traces can be acquired in the attack phase, where the same target value (k') is manipulated. In such a case, a ranking among the n_a probabilities $P_i(k')$ is built to improve the prediction accuracy. A single trace profiling attack (STPA) is performed when only one power trace is available in the attack phase. Because of its probabilistic basis, the STPA results will be less accurate.

In some methods, like RSA key generation and ECDSA, the secret is a nonce; thus, they impose the limitation of only one power trace available in the attack phase. However, there are successful cases of STPA in the recent literature, such as [28,29].

4. Profiling Attack on the Euclidean Algorithm

In this section, a PA on a masked BEEA implementation is proposed. We consider the RSA key generation process as the scope of the attack and, more exactly, the coprimality tests and the private key computation, assuming that they are implemented using the Euclidean algorithm.

For the feasibility of this attack, the following assumptions are made:

- The hardware device(s) employed for the PA have a source of leakage;
- The target RSA key generation uses a classical Euclidean algorithm to compute d and/or $GCD(e, x-1)$;
- The BEEA implementation uses the masking method defined in Equation (6);
- An adversary can acquire sufficient power (or electromagnetic) signals from the target device (preferred) or similar signals running the BEEA or a similar function;
- An adversary can acquire the power (or electromagnetic) signals of the whole target RSA key generation phase.

In this work, we consider as the power trace (or indistinctly, a leakage vector), the vector of the power consumption measurements where a target operation manipulates a secret value. The points of interest (PoIs) are those within the resultant leakage vector after applying a compression method, as suggested in [30]. In addition, every execution of a target operation will be herein referred to as an attack point.

Usually, once an RSA key pair is generated, it is stored to be used several times. The coprimality tests are performed only once involving p and q (the two finally successful prime candidates). A similar procedure is used in ECDSA with the nonce k. These facts impose a restriction, as the target secrets are used in only one algorithm execution.

Therefore, our proposed attack is specifically an STPA. Nevertheless, due to the nature of the Euclidean algorithm and $\phi(N)$, we can target at least eight attack points in a single trace to obtain eight leakage vectors (see Section 4.1 for further details). This approach is still formally an STPA but is equivalent to an 8-trace PA where the success probabilities are higher.

Let us denote by P_{ap} the probability of a successful attack on a single attack point; then,

$$P_{STPA} = 1 - (1 - P_{ap})^8, \tag{7}$$

will be the probability of a successful attack on the key generation process of the RSA. Furthermore, if an adversary can acquire n traces of the key renewal procedure, the probability of a successful attack increases and can be defined by:

$$P_{n-STPA} = 1 - (1 - P_{ap})^{8n}. \tag{8}$$

The proposed attack can also be conducted on the ECDSA when the nonce k is inverted. However, in this case, the conditions for performing the attack are less convenient than those of the RSA key generation.

4.1. Attack Points on Private Key Generation

In the case of private key generation, the *v-loop* of the Algorithm 1 is executed first and at least twice because $\phi(N)$ is divisible by four. The operation sequence for the first and second iterations can be easily predicted. Table 1 shows the ordered execution of the relevant operations involved.

Table 1. Target operations on BEEA (e, $\phi(N)$).

	Operation 1	**Operation 2**
1st iteration	$v = v/2$	$C = (C + \phi(N))/2$
2nd iteration	$v = v/2$	$C = C/2$
	Result 1	**Result 2**
1st iteration	$v = \phi(N)/2$	$C = \phi(N)/2$
2nd iteration	$v = \phi(N)/4$	$C = \phi(N)/4$

All of these operations that we define as attack points perform a right shift followed by a copy-to-register. Furthermore, the manipulated data are only the secret $\phi(N)$; thus, the leakage of both the shift and the copy-to-register operations is uniquely related to that value. At the low level, every operation in Table 1 manipulates the secret (at least) twice, shifting it and copying it to a register. This should generate two sets of POIs per attack point. In fact, having as many POIs as possible is an advantage because TA is probabilistic; thus, its success is strongly linked to the number of available samples.

The manipulation of $\phi(N)$ and $\phi(N)/2$ does not prevent the consideration of these operations as equivalent attack points operating with the same value. The guessed bytes in the second iteration that shifted one bit to the left corresponded to those guessed in the first iteration; alternatively, the guessed bytes in the first iteration that shifted one bit to the right corresponded to those guessed in the second iteration. Therefore, (from Table 1) operations 1 and 2 for the first and second iterations of BEEA (e, $\phi(N)$) can be considered four equivalent attack points from which four leakage vectors are obtained.

If $\phi(N)$ has more than two trailing zero bits, it will be easily detected by a visual inspection of the signal trace. This is because the power profile of the following iterations (into the *v-loop*) will be quite similar to the previous profile. This can be observed for hardware and software implementations in the power signal figures given by Cartaya et al. in [13], where the *v-loop* profile looks very different from the subtraction profile. Therefore, the presence of more than two trailing zero bits in $\phi(N)$ yields additional attack points. Thus, it can be said that the more trailing zero bits there are in $\phi(N)$, the higher the

probability of a successful attack. Table 2 shows how the guesses should be performed for four trailing zero bits.

Table 2. Target operations on BEEA $(e, \phi(N))$ after 2nd iteration.

	Operation 1	**Operation 2**
3rd iteration	$v = v/2$	$C = C/2$ or $C = (C + \phi(N))/2$
4th iteration	$v = v/2$	$C = C/2$ or $C = (C + \phi(N))/2$
	Result 1	**Result 2**
3rd iteration	$v = \phi(N)/8$	$C = \phi(N)/8$ or $C = 5\phi(N)/8$
4th iteration	$v = \phi(N)/16$	$C = \phi(N)/16$ or $C = 13\phi(N)/16$

Note that for our attack, it is assumed that the masking method in Equation (6) is used, and then the public key e is randomized. In this case, the evenness of D cannot be predicted. Therefore, the conditional branch that will execute into the *v-loop* cannot be predicted either. For this reason, in Table 2, the two possible results from the execution of the second operation are given.

4.2. Attack Points on the Coprimality Tests

The analysis to define the attack points in the coprimality tests is rather similar to that in the previous section. Let e and $x - 1$ be the GCD inputs, where x is either the secret p or q that are indeed the targets. Then, since $x - 1$ is even, the *v-loop* is executed first. It cannot be ensured that $x - 1$ is divisible by four as $\phi(N)$; thus, only the first iteration can be predicted. This allows us to define the attack points as the first and second operations in Table 1 but only within the first iteration. These operation results are $v = x/2$ and $C = x/2$ for the computation of GCD $(e, x - 1)$.

Because the coprimality tests are performed for $p - 1$ and $q - 1$, we have another four attack points. Moreover, the latter analysis in Section 4.1 regarding the probable further trailing zeroes can be applied herein. Therefore, with some probability, more than four leakage vectors can be obtained from the whole coprimality tests of p and q.

4.3. Profiles and Guesses Verification

The reliability of the obtained profile (templates in the case of TA) and the accuracy of the guesses can be verified by exploiting some known bytes. It is known that $\phi(N)$ shares roughly half of its most significant bytes with N, which is public. Then, the leakage vectors on the private key generation can be exploited starting by the known bytes of $\phi(N)$. A significant correlation should appear among the MSB of $\phi(N)$ for each of the known bytes if the profile is good enough. In a convenient (although casuistic) scenario for an adversary, one or more correlations corresponding to known bytes can also appear among the lower half of $\phi(N)$, which is indeed the secret part.

On the other hand, matching the guessed p and q with N is a straightforward approach for carrying out a partial verification of the attack phase on the coprimality test. It can be demonstrated that, for a certain byte length, $N_H = p_H \cdot q_H$ and $N_L = p_L \cdot q_L$, where X_H and X_L are the most and least significant bytes of X, respectively.

Let N be l-bytes long; then, p_H, q_H, p_L and q_L have approximately $l/2$ bytes (assuming p and q have the same byte length, as commonly occurs). Then, N can be written as a function of the primes as given by:

$$N = 2^{l/2}(p_H \cdot q_H) + 2^{l/4}(p_H \cdot q_L + q_H \cdot p_L) + p_L \cdot q_L. \tag{9}$$

Equation (9) shows that approximately $l/4$ of the MSB of N depends on half of the MSB of p and q. On the other hand, one quarter of the LSB of N strictly depends on half of the LSB of p and q. These relationships allow us to verify the accuracy of some of the guessed values of p and q and to tune them to achieve the match with N.

5. Experimental Results

In this section, we describe the experimental results of the attack on the protected BEEA algorithm using ML-based PA. In the first subsection, we describe how to measure the power consumption trace of the protected BEEA algorithm while it is running. In the second subsection, we describe the multilayer perceptron (MLP) model used to perform the attack and describe the method of cross-validation. The success rate of the attack is described in the last subsection.

5.1. Experiment Environment

The attack is applied against the countermeasure in Equation (6) that protects a BEEA implementation. More specifically, we target the inner operation where the secret $(X-1)$ is manipulated in plaintext. We coded and implemented an 8-bit-based BEEA following Algorithm 1, which is a classical definition [8].

We measure the power consumption that includes the operations of Table 1. The traces are collected while the algorithm runs 50,000 times on the ChipWhisperer-Lite power collection board, at a sampling rate of 29.54 ms/s. To obtain a profile for the secret value $\phi(N) = (p-1)\cdot(q-1)$, we select $\phi(N)$ as a random 128-bit value that is divisible by four. The options for compiling the 8-bit-based BEEA algorithm code for uploading on the ChipWhisperer-Lite board were configured as shown in Table 3.

Table 3. Optimization level for each option.

Optimization Option	Description of Optimization Level
-O0	None
-O1	Optimization of speed (Low)
-O2	Optimization of speed (Medium)
-O3	Optimization of speed (High)
-Os	Optimization of size

5.2. Attack Model

To perform the ML-based PA, we must first construct a profile for the value $\phi(N)$ that we seek to recover. In other words, we train the MLP model by using the value $\phi(N)$ as a label. We concatenate the power consumption traces of the four attack points, namely, points of interest (PoIs), shown in Figure 1 . The concatenated trace is an input to the neural network. Of the 50,000 power consumption traces collected, 40,000 were used for training, and 5000 were used for verification. The remaining 5000 traces are used for the attack, and only one trace is used for each attack, that is, we select one trace from 5000 traces and perform the attack. The hyperparameters of the MLP model used in this paper are shown in Table 4. The $(x\cdot 4)$ in the input layer indicates the input data size when a range of each PoI is set to x. This is done to generalize the power consumption trace collected by each optimization option since four PoI ranges exist and are different in size.

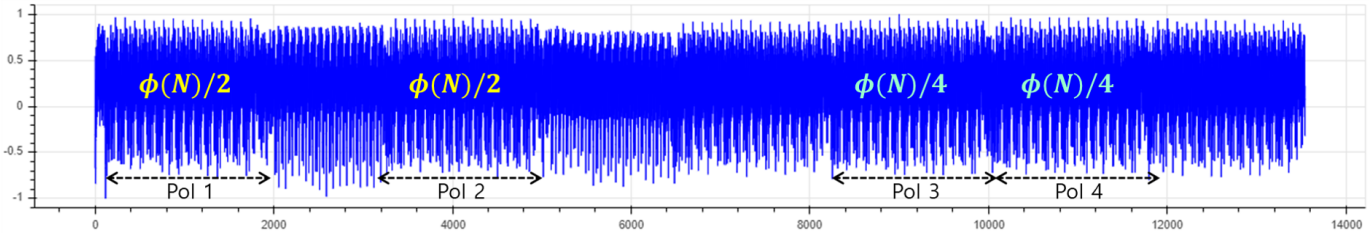

Figure 1. Power trace of protected BEEA with optimization option -Os.

For example, the power consumption trace collected with optimization option -Os is shown in Figure 1. We applied an SPA to identify PoIs. The four PoIs presented in Section 4.1 are the same parts of PoI 1, PoI 2, PoI 3, and PoI 4, respectively, shown in the power consumption trace of Figure 1. The time positions of each section are 150–1850, 3286–4986, 8274–9974 and 9849–11549, namely, the x of Table 4 is 1700. These four locations are concatenated and used as the input to the MLP model. Similarly, for power consumption traces collected at -O0, -O1, -O2, and -O3, four PoIs found in the trace were used as the input to the neural network.

Table 4. Hyperparameters used in MLP model.

Optimization Option	Description of Optimization Level
Input Layer	$(x \cdot 4)$
Hidden Layer	Dense(200) with activation function ELU BatchNormalization Dropout(0.3) Dense(200) with activation function ELU BatchNormalization Dropout(0.3)
Output Layer	256
Optimizer	NAdam(lr = 0.002, beta1 = 0.9, beta2 = 0.999, decay = 0.004)
Loss Function	categorical cross entropy
Epoch	500
Batch Size	32

5.3. Experiment Results

In this experiment, of the 50,000 power consumption traces collected, 40,000 were used for training, 5000 were used for the validation of training, and 5000 were used for attack processes. For attack, only one trace selected from 5000 traces is used, that is, the attack is a single-trace attack. Then, all of the data were divided into a total of ten cases, as shown in Figure 2 and K-Fold cross-validation [31] was performed.

The attack was performed using one power trace, and the average success rate was calculated by repeating it 1000 times for each case. In other words, we repeated the task of choosing one out of 5000 traces and performing the attack 1000 times. The success rates derived when these attacks are carried out are shown in Table 5 for each optimization option. An examination of the results in Table 5 shows that each byte 8-bit value of the secret value $\phi(N)$ is recovered with a probability of more than 98.90% in a single-trace attack, regardless of the optimization option used to collect the power consumption trace.

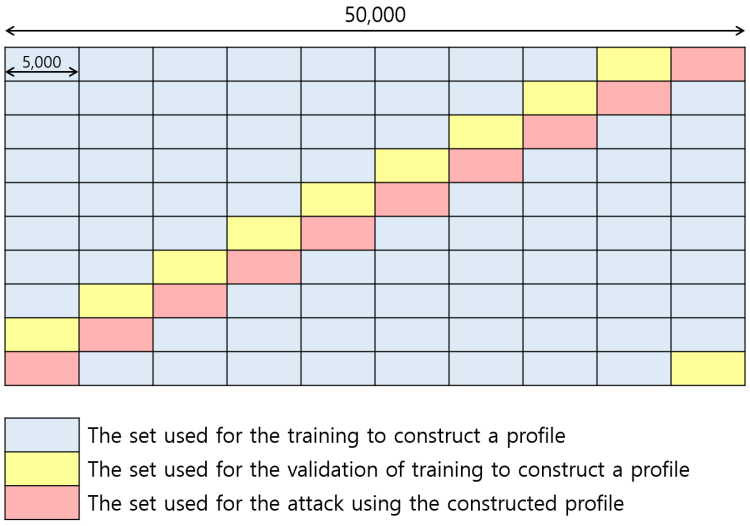

Figure 2. Selection of data for K-fold cross-validation.

Table 5. Success rate of the single-trace attack for each optimization option.

	-O0	-O1	-O2	-O3	-Os
Case 1	100.00%	100.00%	98.90%	99.20%	99.30%
Case 2	100.00%	100.00%	98.90%	99.20%	99.30%
Case 3	100.00%	100.00%	98.80%	99.30%	99.20%
Case 4	100.00%	100.00%	99.20%	99.50%	98.90%
Case 5	100.00%	100.00%	99.30%	99.40%	99.70%
Case 6	100.00%	100.00%	98.10%	99.10%	99.40%
Case 7	100.00%	100.00%	98.80%	99.40%	99.40%
Case 8	100.00%	100.00%	98.70%	99.00%	96.70%
Case 9	100.00%	99.80%	98.70%	88.00%	99.40%
Case 10	100.00%	99.98%	98.82%	97.46%	99.03%
Average	100.00%	100.00%	98.90%	99.20%	99.30%

6. Conclusions

In this work, an ML-based PA on an RSA key generation is proposed that targets a protected BEEA that is used to obtain the prime secrets, p, q, and the private key.

It was demonstrated that at least eight leakage vectors can be extracted from a single power trace. This is not common in single-trace attacks, and such a finding denotes the high risk of a naive implementation of the BEEA. According to recent works, we believe this attack is feasible in 8-bit devices and can also be extended to 16-bit devices.

Furthermore, we have pointed out that one of the most well-known countermeasures (described by Equation (6) and actually intended to prevent SPA) is not effective against our attack. On the other hand, the countermeasure described by Equation (5) prevents both the SPA and the PA presented herein. Therefore, we suggest the use of the countermeasure described by Equation (5), or even the combination of both, as proposed in [14] for ensuring safe BEEA implementation.

Author Contributions: Conceptualization, S.d.l.F.; methodology, S.d.l.F., H.-B.P. and B.-Y.S.; software, S.d.l.F., H.-B.P. and B.-Y.S.; validation, H.-B.P. and B.-Y.S.; formal analysis, S.d.l.F., H.-B.P. and B.-Y.S.; investigation, S.d.l.F.; writing—original draft preparation, S.d.l.F., H.-B.P. and B.-Y.S.; writing—review and editing, D.-G.H. and C.F.; supervision, D.-G.H. and C.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00903, Development of Physical Channel Vulnerability-based Attacks and its Countermeasures for Reliable On-Device Deep Learning Accelerator Design).

Data Availability Statement: Not applicable.

Acknowledgments: The authors wish to thank A. Cabrera Aldaya and D. Hernández for their comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liskov, M. Fermat's Little Theorem. In *Encyclopedia of Cryptography and Security*; van Tilborg, H.C.A., Ed.; Springer: Boston, MA, USA, 2005.
2. Kaliski, B.S. The Montgomery inverse and its applications. *IEEE Trans. Comput.* **1995**, *44*, 1064–1065. [CrossRef]
3. Dussé, S.R.; Kaliski, B.S., Jr. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology—EUROCRYPT 90*; Springer: Berlin/Heidelberg, Germany, 1990; pp. 230–244.
4. Arazi, O.; Qi, H. On calculating multiplicative inverses modulo 2^m. *IEEE Trans. Comput.* **2008**, *57*, 1435–1438. [CrossRef]
5. Koç, C.K. A New Algorithm for Inversion mod p^k. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 411.
6. De la Fé, S.; Ramis, C. A Secure algorithm for inversion modulo 2^k. *Cryptography* **2018**, *2*, 23. [CrossRef]
7. Stein, J. Computational problems associated with Racah algebra. *J. Comput. Phys.* **1967**, *1*, 397–405. [CrossRef]
8. Knuth, D.E. The Art of Computer Programming. In *Seminumerical Algorithms*; Addison-Wesley Longman Publishing Co.: Boston, MA, USA, 1997.

9. Acıiçmez, O.; Gueron, S.; Seifert, J.-P. New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. *Cryptogr. Coding* **2007**, *4887*, 185–203.
10. Menezes, A.J.; van Oorschot, P.; Vanstone, S. *Handbook of Applied Cryptography*; CRC Press: New York, NY, USA, 1997.
11. Cabrera Aldaya, A.; Cabrera Sarmiento, A.J.; Sánchez-Solano, S. SPA vulnerabilities of the binary extended Euclidean algorithm. *J. Cryptogr. Eng.* **2016**, *7*, 273–285. [CrossRef]
12. Aravamuthan, S.; Thumparthy, V. R. A parallelization of ECDSA resistant to simple power analysis attacks. In Proceedings of the 2nd International Conference on Communication Systems Software and Middleware, Bangalore, India, 7–12 January 2007; pp. 1–7.
13. Cabrera Aldaya A.; Cuiman Márquez R.; Cabrera Sarmiento A.J.; Sánchez-Solano S. Side-channel analysis of the modular inversion step in the RSA key generation algorithm. *Int. J. Circuit Theory Appl.* **2017**, *45*, 199–213. [CrossRef]
14. Chartier, M. Method to Protect a Binary GCD Computation against SPA Attacks. Patent WO/2013/092265, 27 June 2013.
15. De Mulder, E.; Hutter, M.; Marson, M.E.; Pearson, P. Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: Extended version. *J. Cryptogr. Eng.* **2014**, *4*, 33–45. [CrossRef]
16. Rivest, R.L; Shamir, A.; Adleman, L.M. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
17. Hardy, G.H.; Wright, E.M.; Wiles, A.J. *An Introduction to the Theory of Numbers*, 6th ed.; Oxford University Press: Oxford, UK, 2008; pp. 3–4.
18. Deschamps, J.-P; Sutter, G. Hardware implementation of finite-field division. *Acta Appl. Math.* **2006**, *93*, 119–147. [CrossRef]
19. Bigou, K.; Tisserand, A. Improving modular inversion in RNS using the plus-minus method. In *Cryptographic Hardware and Embedded Systems—CHES 2013, LNCS 8086*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 233–249.
20. Galindo, D.; Großschädl, J.; Liu, Z.; Vadnala, P.K.; Vivek, S. Implementation of a leakage-resilient ElGamal key encapsulation mechanism. *J. Cryptogr. Eng.* **2016**, *6*, 229–238. [CrossRef]
21. ARM Limited. Mbed TLS: Open Source Embbeded TLS Library. Available online: https://tls.mbed.org/ (accessed on 30 June 2021).
22. Chari, S.; Rao, J.; Rohatgi, P. Template Attacks. In *Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 51–62.
23. Yang, S.; Zhou, Y.; Liu, J.; Chen, D. Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. *Proc. Int. Conf. Inf. Secur. Cryptol. (ICISC)* **2011**, *7259*, 169–185.
24. Martinasek, Z.; Zeman, V. Innovative method of the power analysis. *Radioengineering* **2013**, *22*, 586–594.
25. Martinasek, Z.; Malina, L.; Trasy, K. Profiling Power Analysis Attack Based on Multi-layer Perceptron Network. In *Computational Problems in Science and Engineering: Lecture Notes in Electrical Engineering*; Springer: Berlin/Heidelberg, Germany, 2008; p. 343.
26. Martinasek, Z.; Dzurenda, P.; Malina, L. Profiling power analysis attack based on MLP, DPA contest V4.2. In Proceedings of the 39th International Conference on Telecommunications and Signal Processing, Vienna, Austria, 27–29 June 2016; pp. 223–226.
27. Benadjila, R.; Prouff, E.; Strullu, R.; Cagli, E.; Dumas, C. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.* **2020**, *10*, 163–188. [CrossRef]
28. Wagner, M.; Heyse, S. Single-Trace Template Attack on the DES Round Keys of a Recent Smart Card. Available online: https://eprint.iacr.org/2017/057.pdf (accessed on 7 November 2021).
29. Järvinen, K.; Balasch, J. Single-trace side-channel attacks on scalar multiplications with precomputations. *CARDIS* **2016**, *10146*, 137–155.
30. Choudary, O.; Kuhn, M.G. Efficient Template Attacks. *CARDIS 2013* **2013**, *8419*, 253–270.
31. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006.