





Article

# Fast Constant-Time Modular Inversion over $\mathbb{F}_p$ Resistant to Simple Power Analysis Attacks for IoT Applications

Anissa Sghaier <sup>1</sup>, Medien Zeghid <sup>1,2</sup>, Chiraz Massoud <sup>1</sup>, Hassan Youusif Ahmed <sup>2</sup>, Abdellah Chehri <sup>3,\*</sup>  
and Mohsen Machhout <sup>1</sup>

- <sup>1</sup> Electronics and Micro-Electronics Laboratory, Faculty of Sciences, University of Monastir, Monastir 5000, Tunisia; sghaier.anissa@gmail.com (A.S.); medien.zeghid@fsm.rnu.tn (M.Z.); massoud.chiraz@hotmail.fr (C.M.); machhout@yahoo.fr (M.M.)
- <sup>2</sup> Electrical Engineering Department, College of Engineering at Wadi Aldawaser, Prince Sattam Bin Abdulaziz University, Wadi Aldawaser 11991, Saudi Arabia; hassanuofg@gmail.com
- <sup>3</sup> Department of Applied Sciences, University of Quebec in Chicoutimi (UQAC), Chicoutimi, QC G7H 2B1, Canada
- \* Correspondence: akehri@uqac.ca

**Abstract:** The advent of the Internet of Things (IoT) has enabled millions of potential new uses for consumers and businesses. However, with these new uses emerge some of the more pronounced risks in the connected object domain. Finite fields play a crucial role in many public-key cryptographic algorithms (PKCs), which are used extensively for the security and privacy of IoT devices, consumer electronic equipment, and software systems. Given that inversion is the most sensitive and costly finite field arithmetic operation in PKCs, this paper proposes a new, fast, constant-time inverter over prime fields  $\mathbb{F}_p$  based on the traditional Binary Extended Euclidean (BEE) algorithm. A modified BEE algorithm (MBEEA) resistant to simple power analysis attacks (SPA) is presented, and the design performance area-delay over  $\mathbb{F}_p$  is explored. Furthermore, the BEE algorithm, modular addition, and subtraction are revisited to optimize and balance the MBEEA signal flow and resource utilization efficiency. The proposed MBEEA architecture was implemented and tested on Xilinx FPGA Virtex #5, #6, and #7 devices. Our implementation over  $\mathbb{F}_p$  (length of  $p = 256$  bits) with 2035 slices achieved one modular inversion in only  $1.12 \mu\text{s}$  on Virtex-7. Finally, we conducted a thorough comparison and performance analysis to demonstrate that the proposed design outperforms the competing designs, i.e., has a lower area-delay product (ADP) than the reported inverters.

**Keywords:** IoT; PKCs; prime field; modular inversion; BEEA; modular addition and subtraction; SPA; ADP; FPGA



**Citation:** Sghaier, A.; Zeghid, M.; Massoud, C.; Ahmed, H.Y.; Chehri, A.; Machhout, M. Fast Constant-Time Modular Inversion over  $\mathbb{F}_p$  Resistant to Simple Power Analysis Attacks for IoT Applications. *Sensors* **2022**, *22*, 2535. <https://doi.org/10.3390/s22072535>

Academic Editors: Alexandru Lavric, Liliana Anchin and Adrian I. Petrariu

Received: 26 January 2022

Accepted: 22 March 2022

Published: 25 March 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The IoT encompasses the idea that everyday objects can be connected to the Internet and interfere with each other. Thus, these objects are capable of exchanging and storing data. The concept of IoT is to create a link between the real world and the digital world. However, several high-profile incidents have highlighted the vulnerability of IoT security because a common device was used to penetrate a larger network.

Connected objects are being used more and more, especially with the new connected cities projects. This increases the security risks of the IoT, especially when there is no monitoring or management of the devices. For example, any security breach in medical devices in healthcare applications (wearable or implantable ones) needs increasing attention to protect patient privacy.

In an IoT environment, there are several challenges to securing devices and ensuring end-to-end security. Additionally, since IoT is still an emerging market, many manufacturers and designers are more concerned about launching their products to market than about designing and building security at the beginning.

One of the most frequently cited security concerns with IoT is the use of hard-coded or default passwords.

Furthermore, since a large number of IoT devices are designed to be “set and forget”—placed on a machine or the field and left at the end of their lifespans—they rarely receive security or patch updates. Adding security upfront can be costly, slow development, and prevent the device from working properly.

Another security challenge is connecting legacy assets that are not designed for IoT connectivity. It would be prohibitively expensive to replace legacy infrastructure with connected technologies. In spite of this, there are still some objects that probably have never been updated or protected against modern threats. As a result, the potential attack surface has grown.

There are also a limited number of industry-recognized standards for IoT security. Despite the existence of several IoT security frameworks, there is no single framework that has been agreed upon.

Security and privacy must be prioritized by product manufacturers and service providers. For example, default encryption and authorization should be included.

Smart homes, connected cars, and manufacturing plants are all examples of environments that may experience IoT security breaches. For example, an attack that disables the brakes on a connected car or on a connected healthcare device can have life-threatening effects. Likewise, an attack on critical infrastructures—such as an oil well, energy grid, or water supply—could have dire consequences. Therefore, relevant industries are adopting procedures and implementing measures to ensure their safety. An analysis method that processes the data generated from all security equipment, and a measure based on previous attacks against control systems, have been developed [1].

The rise of IoT has sparked worries about the security of data transmitted between IoT devices and the edge. Indeed, Kim et al. in 2019 [2] conducted a study to address the security flaws of existing IoT devices such as sensor multi-platforms, and they proposed a model that addressed their security vulnerability.

The Elliptic/Hyperelliptic Curve (ECC/HECC) has gained increased attention in recent years because it provides shorter private key lengths with the same level of security as other PKCs such as RSA [3]. At present, cryptosystems based on asymmetric algorithms such as ECC/ECDSA are employed by many IoT devices to safeguard their data and connections [4–6].

In the ECC/ECDSA algorithms, protecting the private and ephemeral keys ( $d$ ,  $k$ ) is essential because, if an adversary obtains these keys, they could modify messages and signatures, thus making the algorithms useless. Several physical attacks aim to retrieve private and ephemeral keys ( $d$ ,  $k$ ) [7,8]. A side-channel attack (SCA) in cryptography is used to extract cryptographic keys and other secret information from a device such as IoT sensors, a smart card, and an integrated circuit.

The power consumption of an electronic device while processing secret data can reveal some of those details. SPA, which uses the existence of visually recognizable power consumption patterns that may expose the sequence of operations conducted by an algorithm, is one of the approaches that can be used to recover hidden information [9,10]. Hence, leakage power consumption is determined by tracking voltages of the device and then dividing by the transition count leakage and Hamming weight, with the first reflecting the number of 1-bit bits treated in time and the second reflecting the number of state variables loaded at a time [11]. Thus, if a secret value is required for this sequence of operations, the implementation is SPA-vulnerable.

Many state-of-the-art studies have examined the protection of the IoT environment against power analysis attacks [12]. Through a bit-checking mechanism, Moon et al. [13] proposed a side-channel attacks countermeasure for IoT systems. In order to eliminate branching in modulus operations, bit checking was introduced. In 2021, a statistical experimental design was proposed to optimize the power attack parameters of an IoT transducer with a minimum cost [14].

In the same year, against differential power analysis (DPA) attacks, two novel countermeasures were proposed. The two methods are based on the back-gate bias technique of fully depleted silicon on insulator (FD-SOI) technology [15]. Using the proposed countermeasures, the required number of traces to recover the secret was increased, as demonstrated by the experimental results.

Finally, SPA has been successfully implemented against a variety of cryptographic algorithms. The ECC/ECDSA key generation technique in the IoT environment is one of the algorithms that has been targeted by this type of attack.

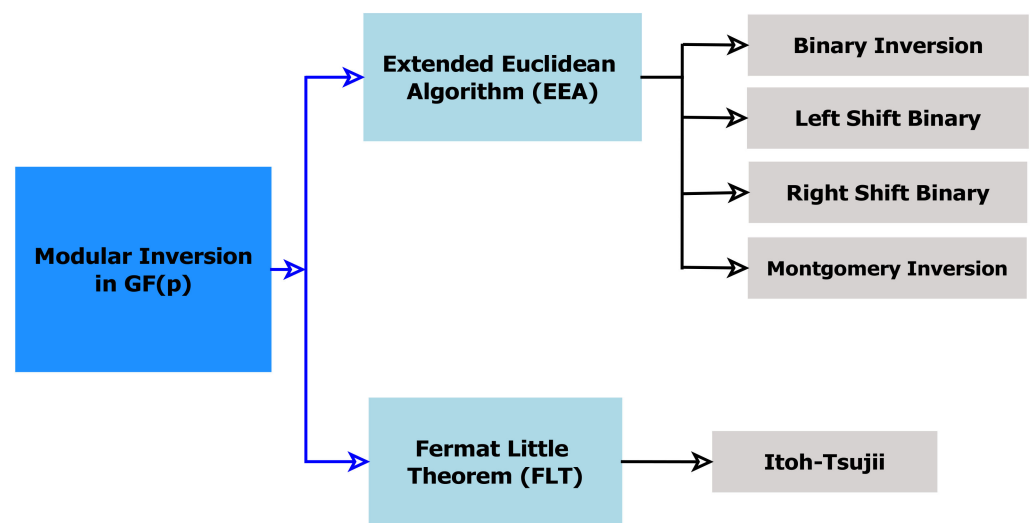
Jérémy et al. [16] published a review on passive attacks on ECC scalar multiplication algorithms in 2016, including leakage sources and frequent errors exploited to attack the ECDSA system. This work described the link between lattice attacks and partial leakage to show how tiny leaks affect ECDSA security. In the same year, Genkin et al. [17] investigated the susceptibility of mobile devices' ECC implementations to side-channel key extraction, finding that these implementations are vulnerable to electromagnetic and power side-channel assaults.

The authors demonstrated partial key leakage from OpenSSL running on Android and from iOS' CommonCrypto, and complete key extraction from OpenSSL running on iOS devices. In 2018, to recover the ECDSA secret key to SM2 Digital Signature Algorithm (SM2-DSA), Zhang et al. [18] extended the new lattice-based attack introduced by Later Nguyen and Shparlinski. SM2-DSA is a Chinese version of ECDSA. They tested the security of the SM2-DSA on the Atmega128 microcontroller using a lattice attack.

In 2019, Wunan et al. [19] published a threat analysis on the broken digital signature of data transactions and an improved SPA against ECDSA. ECDSA's private key can be obtained by using the described attack method combined with a power trace. Moreover, Wunan et al. proposed a side-channel attacks countermeasure for blockchain devices by inserting empty operations into ECC-(point doubling/point addition) operations. In 2021, Thiebault et al. [20] presented a high-quality hardware ECDSA core and provided a complete open-source ECDSA attack artifact. They demonstrated an effective PAA against its FPGA implementation.

In ECC/ECDSA cryptosystems, the SPA leakage-based side-channel research concentrates on the modular-inversion process necessary to generate an ECC/ECDSA private key. The conversion of coordinates in an ECC implementation from projective to affine based on conventional modular inversion, for example, can lead to the disclosure of information, and an adversary could obtain the secret key. Additionally, ECDSA inverts the per-message random secret after scalar multiplication to generate a digital signature [21].

A challenge for cryptographic implementations is a modular inversion because it is one of the most time-consuming field operations in ECC computations. Computing modular inverses can be performed in a variety of ways. Using Fermat's little (FLT) theorem, the Extended Euclidean algorithm (EEA), or any binary variant based on Montgomery's Modular Inverse (MMI) algorithm, it is possible to compute the inverse function. Figure 1 shows the two popular methods, which are the FLT (its variation is the Itoh-Tsujii technique) and the EEA (its variations are the Binary Inversion Algorithm, Left Shift Binary Algorithm, Right Shift Binary Algorithm, and the Montgomery Inversion Algorithm) [22].



**Figure 1.** Modular inversion methods in  $\mathbb{F}_p$ .

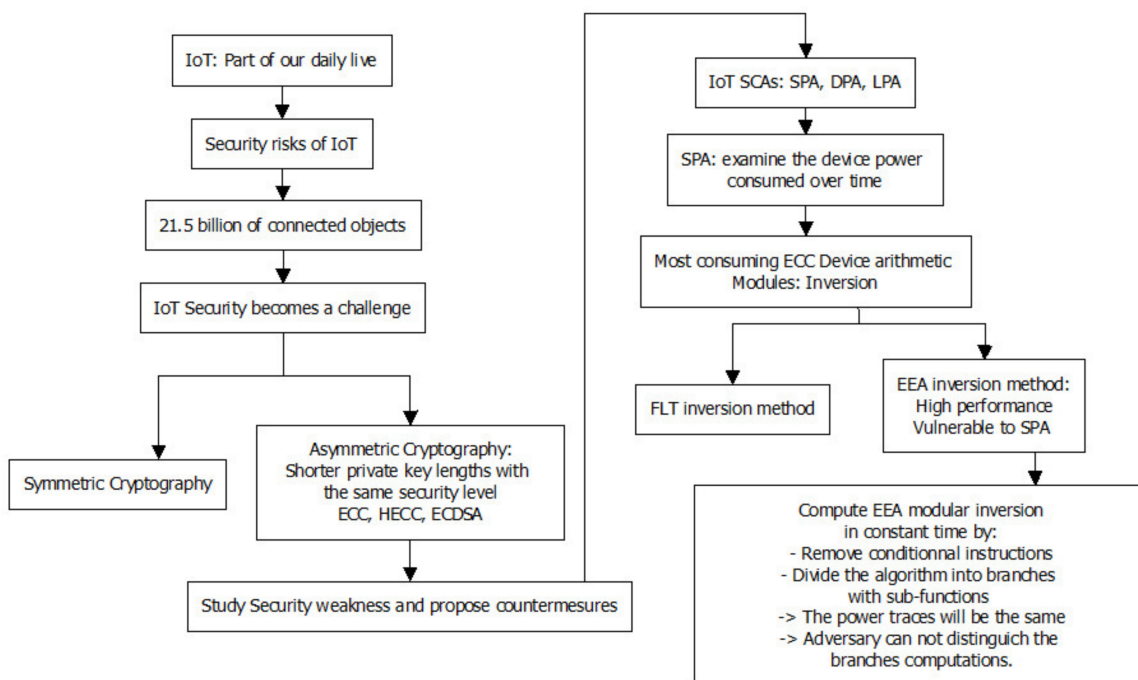
A comparison between the FLT, EEA, and BEEA methods is presented in Table 1. It shows that the BEEA binary version has high performance and efficiency, which meets the requirements of electronic devices (smart cards, RFID tags, mobile phones, IoT devices, etc.).

**Table 1.** Comparison between FLT, EEA, and BEEA methods.

FLT	EEA	BEEA
<ul style="list-style-type: none"> <li>• More complicated than EEA</li> <li>• Higher complexity <math>O(\log^3 n)</math></li> <li>• Slower (consumes a lot of time compared to EEA)</li> <li>• Uses a large number of repetitive multiplications</li> <li>• Secure against SPA and timing attack</li> </ul>	<ul style="list-style-type: none"> <li>• More efficient than FLT</li> <li>• Less complexity <math>O(\log^2 n)</math></li> <li>• Very fast and commonly used for large operands</li> </ul>	<ul style="list-style-type: none"> <li>• Suitable for hardware implementation because it replaced expensive divisions with shift-right operations</li> <li>• Faster than EEA</li> <li>• Less complicated (uses only ordinary additions and subtractions)</li> <li>• No need for multiplications or divisions</li> </ul>

The BEEA method is commonly favored to eliminate the divisions needed in the EEA algorithm since it uses right-shift operations to replace multi-precision divisions. This improvement results in high-performance software-hardware implementations. The BEEA's execution flow is heavily reliant on its inputs. This attribute was investigated in an attempt to find side-channel vulnerabilities that could seriously affect the privacy of a secret variable processed by BEEA.

It is possible to leak information as a result of a typical BEEA implementation. This is because BEEA's execution flow is heavily reliant on its inputs. Hence, the time required to calculate the result is dependent on the inputs. As a countermeasure, one of the most important elements is to ensure that the implementation has constant run-time (worst-case), meaning that the implementation's execution time is unaffected by the input. This is usually accomplished by removing any data-dependent BEEA code branches. Hence, the main idea of this work is to compute an inverse using the same number of constant-time iterations each time. Figure 2 shows the class diagram of this work.



**Figure 2.** UML class diagram of IoT/modular inversion/SPA.

In addition, a high-performance modular inverter is required to speed up the calculation of a PKC system. However, key sizes grow as security levels increase, and this becomes a limiting factor in inverter design over prime field  $\mathbb{F}_p$  due to the carry propagation problem. As a result, adding two prime numbers (A and B) in  $\mathbb{F}_p$ , where  $p$  is a prime of length  $> 128$  bits has a direct impact on the efficiency of calculations in hardware/software (HW/SW) inverter implementations.

Over  $\mathbb{F}_p$ , carry skip (CSK), carry select (CSL), carry-save (CSA), carry-lookahead (CLA), and parallel prefix (PPF) adders are some of the fast binary adders proposed in the literature [23]. The Kogge–Stone adder (KSA) is the fastest adder in the literature because it has a lower fan-out at each stage, which then increases its performance, and is thus widely considered as a standard adder in the industry for high-performance arithmetic circuits [24]. However, it takes more area to be implemented because it computes the carries in parallel. Therefore, KSA performs parallel additions in microprocessors, DSPs, mobile devices, and other high-speed applications.

Based on the above discussions, this paper proposes an alternative method for computing modular inversion in constant time in order to prevent SPA attacks. We modify the (non-constant-time) BEEA approach by removing the conditional instructions and dividing the algorithm into four branches with sub-functions. Since the power traces will be the same regardless of input changes, the adversary cannot distinguish the branch's computation. Therefore, a novel BEEA-based inverter over prime fields is proposed and implemented (MBEEA).

The BEE algorithm, modular addition, and subtraction are revisited to perform MBEEA concurrently, resulting in competitive time and area complexities. Since addition is an essential operation in MBEEA, this paper introduces the generic G-KSA/S adder/subtractor to be reused for any prime number length.

The remainder of the paper is structured as follows: The (modular inversion-SPA) related works are introduced in Section 2. A brief description of BEEA is given in Section 3. Section 4 develops the constant-time BEEA-based inverter. The MBEEA architecture design is presented in Section 5. Section 6 contains the results and analysis of performance. The conclusion is given in Section 7.

## 2. Related Works

In the literature, several algorithms have been proposed for computing modular inversion. Some works were interested in analyzing SPA leakages of modular inversion implementation and their countermeasures. In 2014, to prevent combinational attacks (SPA and lattice techniques), Joppe W. Bos [25] modified the non-constant-time approach form of the Montgomery Inversion based on the EE greatest common divisor algorithm to compute both the classical and the MMI in constant time. He demonstrated that when the modulus has a special form, such as the Curve25519 (Elliptic curve with 256-bit key size offering 128-bit security), the FLT performance is comparable to the BEEA. However, the software implementation is done on ARM 32-bit, and the results showed that the constant-time almost inversion is much slower than the classic one.

The famous and easiest method to resist the leakage of information is the blinding technique using simple multiplicative masking [26]. When  $Z$  is the  $z$ -coordinate in projective coordinates of a given point and  $u$  is a random element in  $\mathbb{F}_p$  (unknown to the attacker), the inversion will be performed for the product  $Zu$  instead of directly inverting  $Z$  using the BEEA; finally, the result  $(Zu)^{-1}$  is multiplied by  $u$  to obtain  $Z^{-1}$ . However, this method requires two additional modular multiplication operations. Since the modular inversion based on FLT is based on modular exponentiation, Xu et al. constructed, in 2017, a secure and efficient modular exponentiation using Montgomery friendly primes described by NIST to obtain an efficient and constant-time modular inversion over  $\mathbb{F}_p$  [27]. Their improvement allows reducing the number of modular multiplications (a saving of 90%), which are computed using only subtractions and shifts.

In [28], the power consumption traces of two different BEEA implementations are investigated in detail in order to extract the SPA leakages that the binary EEA implementation may exhibit during the computation of the RSA key generation algorithm.

In 2018, Savaş et al. proposed two efficient constant-time Montgomery Inversion Algorithms that can be used to counter SCA [29].

The first algorithm is based on the BEEA method, and the second uses Kaliski's method. The two algorithms have comparable performance. The number of iterations for the Montgomery Inversion Algorithm based on BEEA was  $2n$  (where  $n$  is the bit length of modulo  $p$ ) which is less than for the Montgomery Inversion Algorithm using Kaliski's method ( $2n + 1$ ). Furthermore, to speed up (upper-bounded by 2) computation in the software implementation of a multi-core processor, the author proposed a simple parallel algorithm of Montgomery Inversion.

In 2019, Bernstein et al. introduced streamlined constant-time variants of Euclid's algorithm for polynomial and integer inputs [30]. They presented simple, fast, constant-time "division steps" that work the same for integer and polynomial inputs. After applying the main algorithm to GCD, they studied the software speed of modular inversion of polynomials as part of a key generation in the NTRU cryptosystem.

In 2021, Awaludin et al. presented a high-speed ECC processor for arbitrary Weierstrass curves over GF ( $p$ ) [31]. The proposed processor works in constant time and is suitable for applications that require high speed and SCA resistance. To preserve the SCA-resistance, they used constant-time Fermat's little theorem to perform field inversion operations.

Similarly, Sarna et al. presented a constant-time modular inversion algorithm development process capable of achieving a high level of security against timing and SPA Attacks [32].

In their work, Aldaya et al. present a novel SPA of the BEEA algorithm that reveals some exploitable power consumption-related leakages [33]. Using traces, they used the ECDSA protocol to apply SPA to reveal standardized private key sizes. An investigation was conducted into three countermeasures for eliminating SPA leakages from BEEA implementations.

### 3. BEE Algorithm

Several cryptographic algorithms use modular inverses. The BEEA is one of the most common ways to carry out the inversion operation. Given two integer numbers 'a' and 'p', BEEA computes  $x$  and  $y$  such that  $ax + py = \gcd(a, p)$ . When  $\gcd(a, p) = 1$ , the calculated value for  $x$  refers to the multiplicative inverse of  $(a \bmod p)$ .

The BEEA has been reported in a few different forms in the literature. The original BEEA can be shortened when it is known in advance that the modulus  $p$  is a prime number, as in ECDSA. Algorithm 1 illustrates the updated version of the BEEA.

According to Algorithm 1, 'v-loop' refers to the loop that divides 'v' by 2, while 'u-loop' refers to the loop that divides 'u' by 2.

Additionally, the term sub-step is associated with subtraction operations executed in step 5.5 to update  $u$  and  $x$  and  $v$  and  $y$ , respectively.

According to Algorithm 1, in accordance with the least significant bit (LSB) of 'a', only the u-loop during the first iteration can be executed since 'v' is equal to 'p' and 'p' is a prime number. Additionally, for the rest of the iterations, due to the subtraction at step 5.5, only one loop is executed per iteration. Hence, 'u' and 'v' are odd integers just before the execution of the sub-step stage.

A BEEA side-channel analysis was presented in [33]. It concluded that a full recovery of the algorithm's inputs can be achieved when the adversary can collect  $Z_i$  and SUBS[i] for all iterations, where  $Z_i$  is the number of times that an  $x$ -loop ( $x = u$  or  $v$ ) is executed at iteration 'i' and SUBS[i] is the output of the sub-step function of Algorithm 1 at iteration. When  $u \geq v$ , SUBS[i] =  $u$ , else SUBS[i] =  $v$ .

**Algorithm 1:** Pseudo-code of the BEE algorithm.

```

1. Input: Integers  $a$  and  $p$  such that  $\gcd(a, p) = 1$ 
2. Output:  $b = a^{-1} \bmod p$ 
3. Define;  $u = a; v = p$ 
4. Define;  $x1 = 1; x2 = 0$ 
5. While ( $u \neq 1$  and  $v \neq 1$ ) {
5.1. While  $u$  is even {
5.1.1.  $u \leftarrow u/2;$ 
5.1.2. If  $x1$  is even {
5.1.2.1.  $x1 \leftarrow x1/2;$ 
5.1.2.2. Else
5.1.2.3.  $x1 \leftarrow (x1 + p)/2;$ 
5.1.3. }
5.2. }
5.3. While  $v$  is even {
5.3.1.  $v \leftarrow v/2;$ 
5.3.2. If  $x2$  is even {
5.3.2.1.  $x2 \leftarrow x2/2;$ 
5.3.2.2. Else
5.3.2.3.  $x2 \leftarrow (x2 + p)/2;$ 
5.3.3. }
5.4. }
5.5. If  $u \geq v$  {
5.5.1.  $u \leftarrow u - v;$ 
5.5.2.  $x1 \leftarrow x1 - x2;$ 
5.5.3. Else
5.5.4.  $v \leftarrow v - u;$ 
5.5.5.  $x2 \leftarrow x2 - x1;$ 
5.6. }
5.7. }
6. If ( $u = 1$ ) {
6.1. Return  $x1 \bmod p;$ 
6.2. Else
6.3. Return  $x2 \bmod p;$ 
6.4. }

```

The diagram illustrates the flow of the algorithm with labels for 'u-loop', 'v-loop', and 'sub-step'. The 'u-loop' label is positioned to the right of the code block, with a dashed line indicating the scope of the loop from line 5.1 to 5.2. The 'v-loop' label is positioned to the right of the code block, with a dashed line indicating the scope of the loop from line 5.3 to 5.4. The 'sub-step' label is positioned to the right of the code block, with a dashed line indicating the scope of the loop from line 5.5 to 5.6.

By knowing only  $Z_i$ , the authors were able to recover some SUBS[i] and to express a certain number of bits from one of the BEEA inputs as a function of the other.

It is very interesting to consider this approach in the context of power-based side-channel analysis since it appears very difficult to extract the SUBS[i] directly from power traces, while it is easier to extract the  $Z_i$  when the sub-step can be differentiated.

We can segment BEEA iterations into x-loops followed by sub-steps. Consequently, the  $Z_i$  is based on the duration between two consecutive sub-steps. Therefore, if an adversary can distinguish between the sub-steps of the BEEA in the power trace, then he can exploit the  $Z_i$ .

Remarkably, Algorithm 1 is based on conditional instructions (loops while and if statements). Therefore, the execution time of Algorithm 1 depends on 'a' and 'p' values. Furthermore, the input 'a' presents the scalar multiplication algorithm results in affine coordinates, and a modular inversion will convert it to projective coordinates. Hence, the inversion should be secure against SPA attacks to prevent information leakage. For this reason, our contribution is to ensure a countermeasure to prevent Algorithm 1 from a specific SPA attack.

#### 4. Constant-Time Modular Inversion

Input-dependent execution flows of the BEEA have prompted the development of anti-side-channel measures. A modified version of BEEA that operates in constant time and is resistant to SPAs is proposed in this section. As a countermeasure, we propose eliminating the conditional branches in Algorithm 1 in order to reduce the data dependency.

Algorithm 1's execution time depends on both 'a' and 'p'. In such an algorithm, when different inputs are used, the number of iterations in the while loop that have to be removed at the end of the algorithm may vary significantly.

Algorithm 1 may be converted into a constant-time algorithm by meeting the following conditions:

- The same amount of time is always taken to compute an iteration. In constant time, this requires computing all four branches of Algorithm 1 and selecting the appropriate values. As a result, the computed time of each iteration is independent of the branch taken; however, it may be increased to (at most) that of the computation of the sum of all the branches.
- The algorithm always has the same number of iterations. As a result, the worst-case number of 'K' iterations should always be calculated. It can be achieved by determining when Algorithm 1 terminates (when we reach  $v = 1$  or  $u = 1$ ).

The constant-time version of Algorithm 1 is described in Algorithm 2. The first two branches are computed in every cycle (constant run-time) regardless of whether the values of  $u$  and  $v$  are even or odd. Thus, the sequence of the parity computation deduced from the power trace does not reveal any information about the inputs (the bits of a).

In Algorithm 2, the comments showing which branches from Algorithm 1 are being computed are displayed after an '#'. As shown in Algorithm 2, the "while loops" of Algorithm 1 are replaced by "for loops" whose higher bound is 'K'. 'K' presents the cycle number of the biggest number inverse in the finite field  $\mathbb{F}_p$ , such that  $(a < p)$ . Therefore, K presents the worst case of iterations.

It is not difficult to determine the number of iterations ('K') in the worst-case scenario of Algorithm 1. In every iteration, either 'u' or 'v' are reduced by at least a factor of two, so the maximum number of iterations is  $2\log_2(p)$ , where  $p$  is a prime number of n-bit length. It follows that the minimum number of iterations is  $\log_2(p)$ . This reveals the bounds on the exponent 'k' when the algorithm terminates. The functions presented in Algorithm 2 are:

- The shift-by-one function denoted by  $lshift_1(z, x)$ : this function shifts  $x$  by one position to the left and stores the result in  $z$ .



- The subtraction and addition functions are denoted respectively by  $\text{sub}(z, x, y)$  and  $\text{add}(z, x, y)$  computing  $z \leftarrow x - y$  and  $z \leftarrow x + y$ . Those two functions are computed by the adder/subtractor G-KSA/S which is discussed in the next section.

For the function  $\text{add}(x_1, x_1, (p' \wedge x_1(0)))$ , we create a bitmask  $x_1(0)$  since the LSB of  $x_1$  determines the parity of  $x_1$  (1 indicates odd, 0 indicates even) in order to calculate the iteration 5.1.2 in Algorithm 1.

$$\text{add}(x_1, x_1, (p' \wedge x_1(0))) = \begin{cases} \text{if } x_1 \text{ is even then } x_1(0) = 0 \rightarrow x_1 = \frac{x_1}{2} \\ \text{if } x_1 \text{ is odd then } x_1(0) = 1 \rightarrow x_1 = \frac{x_1}{2} + p' \end{cases} \quad (1)$$

- The function  $\text{mem}_u$  and  $\text{mem}_v$  are used respectively to save the values of  $u$  and  $x_1$  if  $u$  is even ( $\text{mem}_u = 0$ , RAM active 0), and the values of  $v$  and  $x_2$  if  $v$  is even ( $\text{mem}_v = 0$ , RAM active 0). As the LUT area is  $2n$ , we need four look-up tables to memorize  $u$ ,  $x_1$ ,  $v$ , and  $x_2$ .
- The comparison function is denoted by  $\text{comp}(u, v)$ . This function is used to compare 'u' and 'v' values.

$$\begin{cases} \text{if } u \geq v \text{ and } u \neq 1 \text{ then } \text{sel}_1 = 1, \text{ else } \text{sel}_1 = 0 \\ \text{if } v \geq u \text{ and } u \neq 1 \text{ then } \text{sel}_2 = 1, \text{ else } \text{sel}_2 = 0 \end{cases} \quad (2)$$

- The function denoted  $\text{mem}_u(\text{sel}_1)$

$$\begin{cases} \text{if } \text{sel}_1 = 1 \text{ then } u \text{ and } x_1 \text{ alookuped in the look up tables} \\ \text{else no values are stored} \end{cases} \quad (3)$$

- The occurrence function is denoted by  $\text{occur}(\text{RAM}, u, 1, i)$ , which means the first occurrence of 1 in RAM of  $u$  indicates the corresponding  $i$ , then:

$$\begin{cases} \text{If } i < j \text{ then } b = x_1 \\ \text{If } j > i \text{ then } b = x_2 \end{cases} \quad (4)$$

**Algorithm 2:** Pseudo-code of the proposed modified BEE algorithm

---

```

1. Input:  $p$  and  $a \in \mathbb{F}_p$ 
2. Output:  $b = a^{-1} \bmod p$ 
3. Define;  $u = a; v = p$ 
4. Define;  $x1 = 1; x2 = 0$ 
5. Define;  $p' = \text{Rshift}(p,1)$ 
Inversion steps
6. for ( $i = 1; i \leq k; i++$ ) {
6.1.  $\text{lshift}_1(u, u)$ 
6.2.  $\text{lshift}_1(x_1, x_1)$ 
6.3.  $\text{add}(x_1, x_1, (p' \wedge x_1(0)))$ 
6.4.  $\text{mem}_u(u(0))$ 
6.5.  $\text{lshift}_1(v, v)$ 
6.6.  $\text{lshift}_1(x_2, x_2)$ 
6.7.  $\text{add}(x_2, x_2, (p' \wedge x_2(0)))$ 
6.8.  $\text{mem}_v(v(0))$ 
6.9.  $\text{comp}(u, v)$ 
6.10.  $\text{sub}(u, u, v) \wedge \text{sel}_1$ 
6.11.  $\text{sub}(x_1, x_1, x_2) \wedge \text{sel}_1$ 
6.12.  $\text{mem}_u(\text{sel}_1)$ 
6.13.  $\text{sub}(v, v, u) \wedge \text{sel}_2$ 
6.14.  $\text{sub}(x_2, x_2, x_1) \wedge \text{sel}_2$ 
6.15.  $\text{mem}_v(\text{sel}_2)$ 
6.16.  $\text{occur}(\text{RAM}, u, 1, i)$ 
6.17.  $\text{occur}(\text{RAM}, v, 1, j)$ 
6.18.  $\text{return}(i, j)$ 
7. }

```

```

# u ← SHIFT(u,1)
# x1 ← SHIFT(x1,1)
# x1 ← SHIFT(ADD(x1,p),1)

# v ← SHIFT(v,1)
# x2 ← SHIFT(x2,1)
# x2 ← SHIFT(ADD(x2,p),1)

# If u ≥ v
# u ← SUB(u,v)
# x1 ← SUB(x1, x2)
# v ← SUB(v,u)
# x2 ← SUB(x2, x1)

# Return x1 mod p if u equal 1
# Return x2 mod p if v equal 1

```

---

Overall, Algorithm 2 in  $\mathbb{F}_p$  has two specified inputs: an element ' $a$ ' and a prime number ' $p$ '. The output of the algorithm is the multiplicative inverse of ' $a$ ', i.e.,  $ba = 1 \pmod p$ . Algorithm 2 employs three steps in total to produce the multiplicative inverse result  $p$ :

- The initial step: ' $a$ ' and ' $p$ ' are assigned to ' $u$ ' and ' $v$ ', respectively. Moreover, in this step, ' $1$ ' and ' $0$ ' are assigned to  $x1$  and  $x2$ , respectively.
- Updating step:  $u$ ,  $v$ ,  $x1$ , and  $x2$  values are updated by three different operations: add, sub, left shift (*lshift*),  $mem_u$ , and  $mem_v$ . The updating process is executed throughout the entire loop ( $(1 < i < k)$  execution time) and managed by two signals: sel1 and sel2. The new ( $u$ ,  $v$ ,  $x1$ , and  $x2$ ) values are assigned according to sel1 and sel2, and these control signals are updated in step 12.9 during each iteration of the "for loop".
- Return ' $b$ ' after  $k$  iterations.

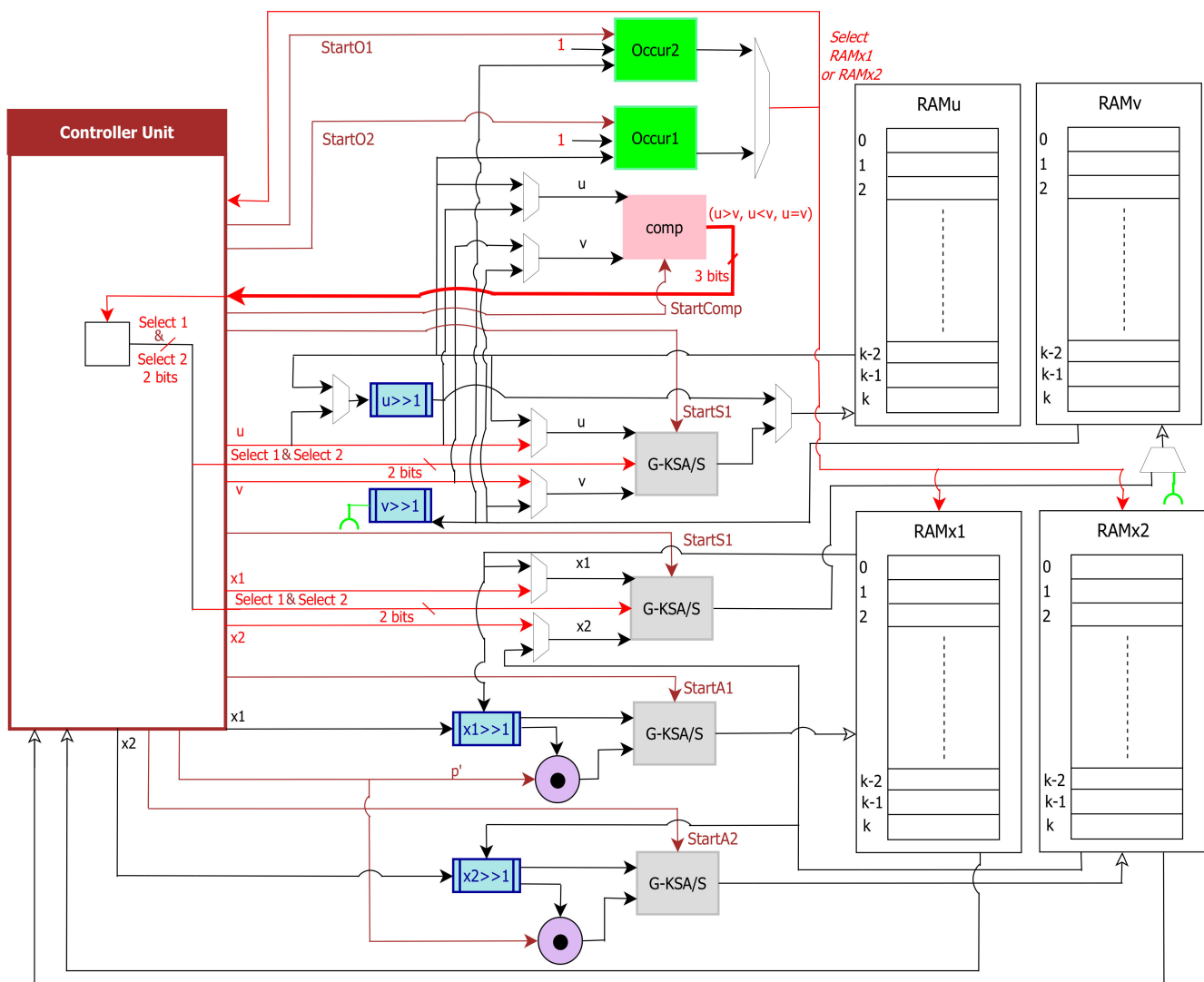
## 5. MBEEA Implementation

As shown in Algorithm 2, inversion is realized mainly by subtracting and shifting operations. Additionally, shift operations are easily implemented in hardware at no cost. We can therefore consider in our design a case that allows performing as many shifts as possible in one clock cycle to reduce the number of clock cycles.

As seen in Algorithm 2, 'Reg\_u', 'Reg\_v', 'Reg\_x1', and 'Reg\_x2'  $n$ -bit registers are essential for implementing the hardware architecture of MBEEA over a prime field. By applying this algorithm, the calculation of division, such as ' $u/2$ ', ' $v/2$ ', ' $x1/2$ ', and ' $x2/2$ ', is based on comparisons of parity and magnitude. There are two multiplexers for selecting ' $u$ ', ' $v$ ', ' $x1$ ', and ' $x2$ '. However, exact comparisons can only be made through full  $n$ -bit subtractions, and this has the effect of delaying decisions about the next calculation. To perform the additions or subtractions, we used  $n$ -bit KSA. In the implemented design, all possible ' $u$ ', ' $v$ ', ' $x1$ ', and ' $x2$ ' updated values are computed at once with multiplexers, and the new values for ' $u$ ' and ' $v$ ' are selected.

Figure 3 depicts the proposed MBEEA inverter design, which is based on Algorithm 2 and includes the following units:

- *Controller unit*: This generates control signals for all the MBEEA architecture units and the data flow in the inverter design, and the movement of data between the adder-subtractor units, the memory units, the comparator unit, and the occurrence units.
- *Adder/Subtractor (G-KSA/S)*: This performs the addition or the subtraction of two values according to the controller decision.
- *Memory unit (MU)*: The main purpose of this unit is to store different parameters such as  $u$ ,  $v$ ,  $x1$ , and  $x2$ , and their intermediate results. It constitutes of four blocks of RAM (RAM<sub>u</sub>, RAM<sub>v</sub>, RAM<sub>x1</sub>, and RAM<sub>x2</sub>) and multiplexers that are used to read operands ( $u$ ,  $v$ ,  $x1$ , and  $x2$ ) from the MU using the corresponding control signals.
- *Comp unit*: This is a comparator between  $u$  and  $v$ . It has a 3-bit output to indicate whether  $u > v$  or  $u < v$  or  $u = v$ .
- *Occur unit*: This searches for  $u = 1$  occurrence. Return  $x1 \pmod p$  if  $u$  equal 1. Otherwise, if  $v$  is 1, return  $x2 \pmod p$ .



**Figure 3.** Proposed MBEEA architecture.

### 5.1. Kogge–Stone Adder/Subtractor Unit: Design and Implementation

Adders are widely recognized as the fundamental building blocks for more complex arithmetic operators such as multipliers and inverters. Usually, subtraction and addition are implemented using a single circuit. The addition of  $x$ ,  $\bar{y}$ , and 1 can be used to compute the subtraction of  $x - y$ , where  $\bar{y}$  is the bitwise complement of  $y$ . Therefore, hardware addition can be used to support the subtraction function. To perform the addition and subtraction in Algorithm 2, four adders/subtractors are needed. The data path of the adder directly affects the delay of the arithmetic inversion data path. KSA is the fastest adder and shows good hardware performance. Thus, KSA was selected for use in the MBEEA architecture for performing addition and subtraction. A parallel prefix adder such as KSA is suitable for additions with longer word lengths. The tree network of KSA allows reducing the latency to  $O(\log_2 n)$  where ‘ $n$ ’ represents the number of bits.

The KSA algorithm is composed of three stages: preprocessing, parallel prefix network, and post-processing. In the preprocessing stage, two signals are calculated simultaneously for each input: Propagate ( $P$ ) and Generate ( $G$ ). The intermediate carries are then generated using these signals in the parallel prefix network. Finally, in the post-processing, we calculate the sum by XORing the intermediate carries and the propagate signals.

In the MBEEA design, a generic adder/subtractor based on the G-KSA/S algorithm was developed. The bit-level version of the G-KSA/S algorithm is shown in Algorithm 3,

which employs n-bit registers to compute three intermediate results  $P, G,$  and  $C$ . Overall, Algorithm 3 has two specified inputs:  $A$  and  $B$  in  $\mathbb{F}_p$  and a pre-computed vector  $S$ . The pre-computed vector  $S$  is represented by a set of elements ' $s_i = 2^i$ ' where  $i \in [0, m - 1]$ , and  $m$  represents the stage number which depends on the length of the prime number  $p (n)$ , such that  $m = \log_2(n)$ . Table 2 presents the ' $m$ ' and ' $S$ ' values for a prime number of lengths  $n = 4, 8, 16, 32, 64, 128,$  and  $256$ . Hence for  $m = 3, S = \{s_0, s_1, s_2\} = \{2^0, 2^1, 2^2\}$ .

**Table 2.** Values of  $m$  and  $S$  for  $n = 4, 8, 16, 32, 64, 128,$  and  $256$  bits.

$n$	$m$	$S$
4	2	1,2
8	3	1,2,4
16	4	1,2,4,8
32	5	1,2,4,8,16
64	6	1,2,4,3,16,32
128	7	1,2,4,3,16,32,64
256	8	1,2,4,3,16,32,64,128

$A$  and  $B$  in  $\mathbb{F}_p$ , two  $n$ -bit inputs, are processed in the first stage bit-by-bit, to compute the generation signals  $G_0$  and propagation signals  $P_0$  which are presented in steps 2.1 and 2.2 of Algorithm 3. To perform the subtraction, the *carry\_in* and the input  $B$  are just XORed, such as presented in step 1 in Algorithm 3.

**Algorithm 3:** G-KSA/S algorithm

```

Input:  $(A,B) \in \mathbb{F}_p$ 
Define:  $n; n = \text{bit length of } p$ 
Pre-computed:  $S = [1,2,4,8,16,32,64,128], j \leftarrow 1, k \leftarrow 1$ 
Output:  $(Sum = (A + B) \bmod p; c_{out})$ 
Preprocessing
Step 1: For  $(i = 0; i \leq n - 1; i++) \{$ 
 $C(i) := c_{in} \oplus B(i);$ 
 $\};$ 
Step 2: For  $(i = 0; i \leq n - 1; i++) \{$ 
2.1:  $P_0(i) := A(i) \oplus C(i);$ 
2.2:  $G_0(i) := A(i) \otimes C(i);$ 
 $\};$ 
Parallel Prefix Network
Step 3: For  $(i = 0; i \leq j-1; i++) \{$ 
3.1:  $G_k(i) := G_{k-1}(i),$ 
3.2:  $P_k(i) := P_{k-1}(i)$ 
 $\};$ 
Step 4: For  $(i = 0; i \leq n-j-1) \{$ 
4.1:  $G_k(i+j) := (P_{k-1}(i+j) \otimes G_{k-1}(i)) \vee G_{k-1}(i+j);$ 
4.2:  $P_k(i+j) := (P_{k-1}(i+j) \otimes P_{k-1}(i));$ 
 $\};$ 
 $k++; j := S[k-1], T := m - 1;$ 
IF  $(T > 0)$  go to step 3 ELSE go to step 5
Step 5: For  $(i = 0; i \leq n-1; i++) \{$ 
5.1:  $C(i) := G_{NP-1}(i) \vee (c_{in} \otimes P_{NP-1}(i));$ 
 $\};$ 
Post-processing
Step 6: 6.1:  $sum(0) = P_0(0) \oplus c_{in};$ 
6.2: For  $(i = 1; i \leq n-1; i++) \{$ 
6.2.1:  $sum(i) = P_0(i) \oplus C(i - 1);$ 
 $\};$ 
Step 7:  $c_{out} = C(N - 1);$ 
Return Sum,  $c_{out}$ 

```

Figure 4 presents the KSA process for 8 bits. As shown in Figure 3, the parallel prefix network for G-KSA/S is composed of three levels. The carry propagation network is in charge of transmitting the carry signal from the preceding bit lines.  $P_k$  and  $G_k$  are generated via ‘carry propagation’ steps, steps 4.1.1 and 4.1.2, as shown in Algorithm 3. Finally, the carry signal obtained in step 5 is used to calculate the sum in the last stage (step 6). The output step performs the XOR operation between the previous bits of the ‘carry’ signal ( $c_{i-1}$ ) and the current bits of the propagation signal ( $P_i$ ), as shown in steps 6 and 7.

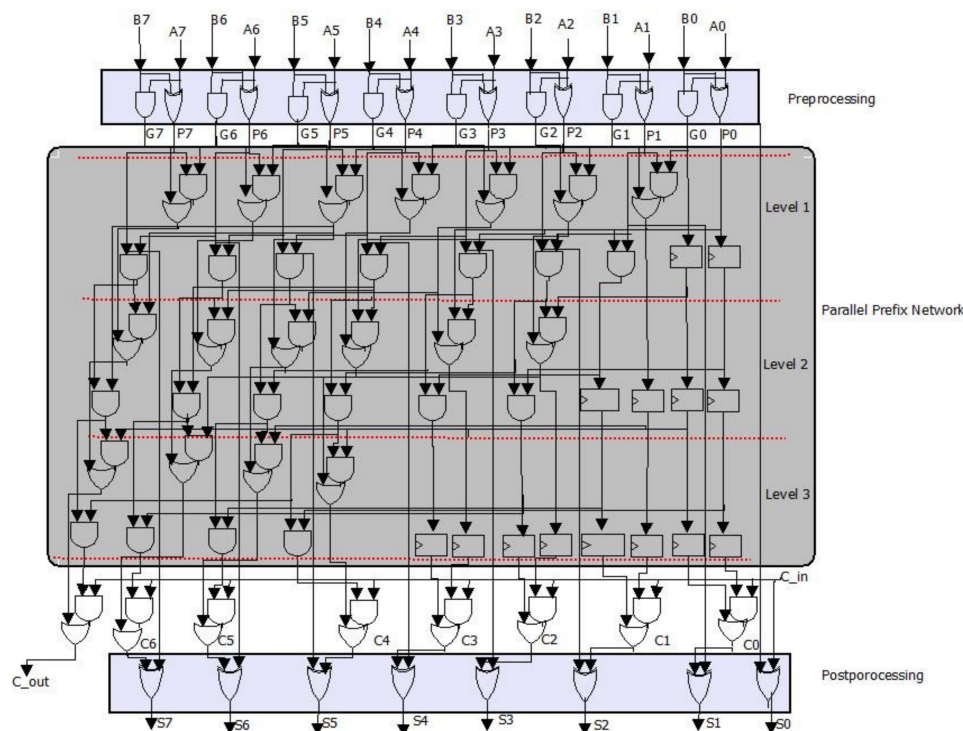


Figure 4. Eight-bit G-KSA/S data-path.

### 5.2. The Modular Reduction

The modular reduction in large values is a fundamental operation in the majority of common PKs that involve intensive computations in prime fields. At the end of Algorithm 2, the result will be reduced modulo the prime number ‘ $p$ ’. The prime number should be chosen carefully to ensure efficient reduction and high performance. The modular reduction can be calculated in three ways:

- General module forms (i.e., Barrett and Montgomery algorithms), which are slower and present an expansive part of the arithmetic operation.
- Based on the LUT method (which is based on pre-computed values); however, this requires a large amount of memory.
- Modulo form of prime numbers, such as pseudo-Mersenne numbers. Their special form makes them appropriate for modular reduction. The pseudo-Mersenne prime number  $p$  is presented with the special form:  $p = 2^\alpha - c$ , where  $\alpha = n$  ( $n$  represents the security level) and ‘ $c$ ’ is a positive integer that is relatively small compared to the modulus. An integer  $0 \leq z < (2^\alpha - c)^2$  can be represented in radix- $2^\alpha$  by spilling  $z$  up into a lower part  $Z_L$  and a higher part  $Z_H$ :  $Z_H 2^\alpha + Z_L$ . Then, using the fact that  $2^\alpha \equiv c \pmod{p}$ , we have  $Z_H 2^\alpha + Z_L \equiv Z_H c + Z_L \pmod{p}$  where  $0 \leq Z_H c + Z_L < (c + 1)2^\alpha$ . Hence, a multiplication of  $Z_H$  by the constant  $c$  is needed and the final result is obtained after the addition of  $Z_L$ . For the three values of  $n$  (128, 192, and 256), the resulting primes satisfy  $p \equiv 3 \pmod{4}$ .

### 5.3. Controller Unit

The main component of the proposed inverter design is the control unit (CU), which is in charge of all communications between all MBEEA units.

At the beginning of the algorithm, the start signal is equal to '1', and initial values 'a', 'p', '1', '0', and 'p/2' are assigned to 'u', 'v', 'x1', 'x2', and 'p' registers, respectively, by input multiplexers. The computed intermediate results are stored in the memory units. The control unit activates the addition/subtraction chains after receiving the input data (p and a) and the signals "CLK", "Reset" = '1', and "Start" = '1'. Hence KSA-start takes '1' and Cf becomes '0'. If KSA completes the addition function ("KSA-done" = '1'), the control unit sends signals of writing to the memory unit to update the 'u', 'v', 'x1', and 'x2' values. The update process is controlled by five functions belonging to three blocks: addition and subtraction functions (KSA block), left shifting function (shift registers), writing, and reading functions (block memory). These new values are determined by two distinct control signals: sel1 and sel2. These control signals are updated after the KSA block performs the 'u-v' function as shown in step 12.9 in Algorithm 2. Hence, the CU generates the signals for the G-KSA/S components, and the read and write addresses for the memory units. The MBEEA state machine describing the design data flow is presented in Figure 5.

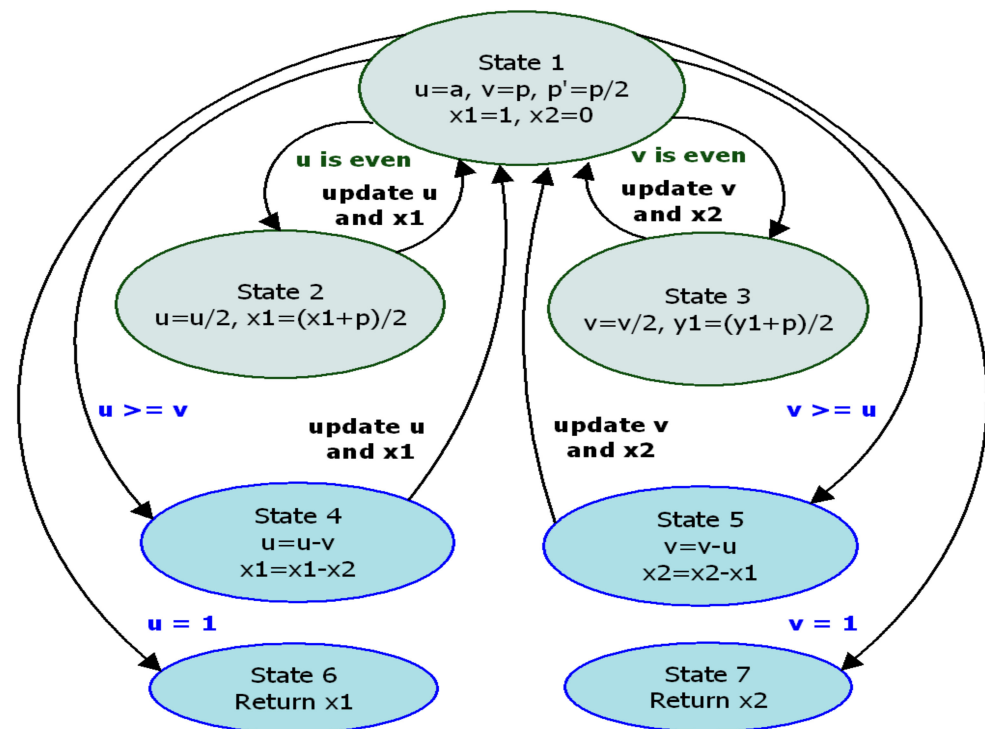


Figure 5. MBEEA state machine.

To implement the MBEEA, FSM is made up of seven states: St#1 is a state of inactivity, whereas during St#2 to St#7, necessary signals for updating 'u', 'v', 'x1', and 'x2' are generated. Of these seven states, two states (S#2, S#3) are executed in parallel. Hence two additions, four shifting operations, and four reading/writing operations are required. Similarly, in states 4 and 5, two subtractions and two reading/writing operations are needed. Finally, based on the output of the comparator unit, 'u' and 'x1' or 'v' and 'x2' are updated.

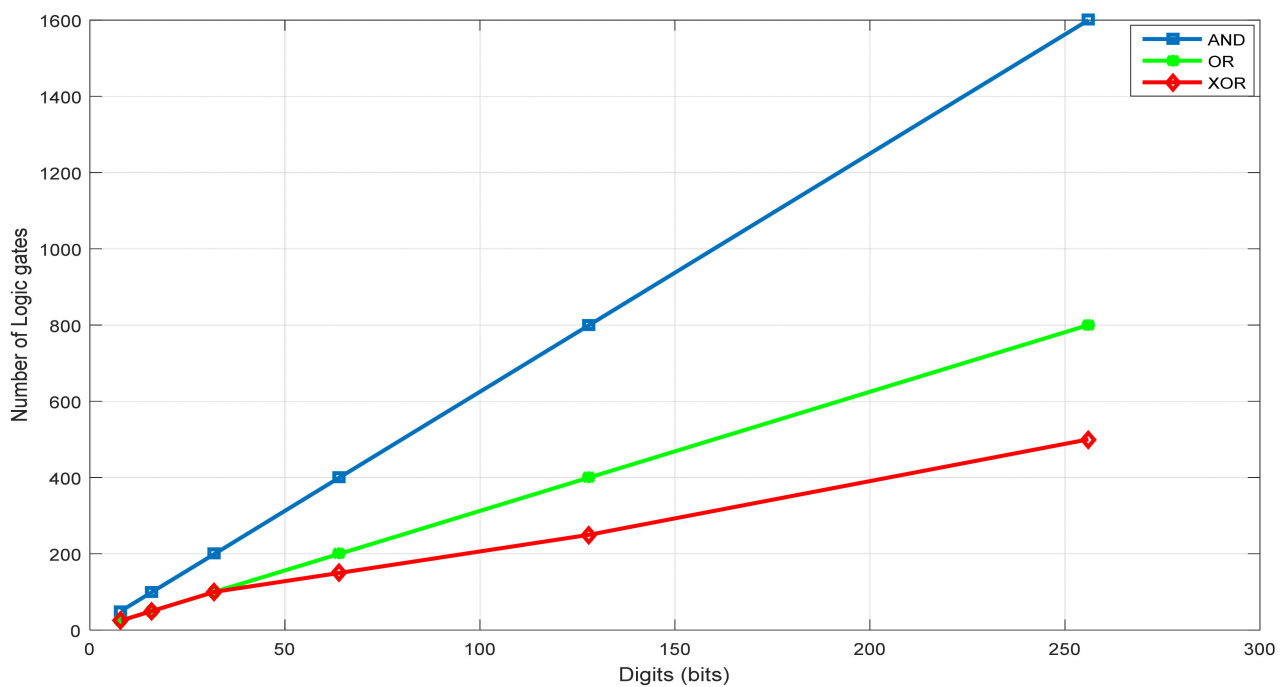
## 6. Results and Performance Analysis

The proposed MBEEA inverter and G-KSA/S designs were then developed for a different prime field of lengths 8, 16, 32, 64, 128, and 256 bits. Modelsim was used to validate the proposed designs, which were coded in VHDL. The G-KSA/S and MBEEA

designs were then implemented on various devices (Spartan 3E, Virtex-E, Virtex-II, Virtex 5, Virtex 7) using Xilinx 14.7. Tables 3 and 4 show the obtained results after place and route, which include the maximum frequency ( $F_{max}$ , MHz), area usage (slices/CLB), latency ( $\mu s$ ), and ADP ( $ADP = \#slices \times Latency$ ).

### 6.1. G-KSA/S Implementation Results

The G-KSA/S algorithm was successfully implemented for  $\mathbb{F}_p$  ( $p$  is a prime number of length 8, 16, 32, 64, 128, 256 bits). Figure 6 presents the required logic gates (AND, OR, and XOR) for the GKSA design (for G-KSA/S, we have to add one XOR gate for subtraction). We can see that the number of required logic gates grows linearly.



**Figure 6.** Logic gates versus  $\mathbb{F}_p$ .

For a fair comparison with related works, the G-KSA/S algorithm was implemented on Spartan3E and Virtex-5, as shown in Table 3. As shown in Table 3, the proposed design clearly outperforms the existing different prime field-length KSA implementations. We can see from Table 3 that our proposed design is much smaller (almost 66% for  $\mathbb{F}_p$  ( $p$  is a prime of length 8, 16, and 32), and almost 41% for  $\mathbb{F}_p$  ( $p$  is a prime of length 64, 128, and 256)) and considerably faster than those of [34,35].

Furthermore, the G-KSA/S proposed design always achieves the best ADP value in all the prime fields compared to the existing designs. For example, over  $\mathbb{F}_p$  with a length of  $p = 256$  bits, compared with [35] on the Virtex-5 device, the ADP of the proposed design has a 28.62% smaller ADP.

**Table 3.** Comparison of the proposed and existing G-KSA/S designs for different prime field lengths.

Designs	Platform	$n = \text{Bit Length of } p$	Area (Slices)	Delay (ns)	ADP ( $10^{-9}$ )	Gain %
[34]	Spartan-3E	8	83	5.776	479.408	74.71%
G-KSA/S	Spartan-3E		47	3.6	169.2	
[34]	Spartan-3E	16	166	10.85	1801.1	61.28%
G-KSA/S	Spartan-3E		98	7.3	715.4	
[34]	Spartan-3E	32	332	20.56	6825.92	68.65%
G-KSA/S	Spartan-3E		174	12.3	2140.2	
[35]	Virtex-5	64	449	30.5	13,694.5	41.13%
G-KSA/S	Virtex-5		289	27.9	8063.1	
[35]	Virtex-5	128	1111	57.3	63,660.3	35.16%
G-KSA/S	Virtex-5		641	64.4	41,280.4	
[35]	Virtex-5	256	1345	106.7	143,511.5	28.62%
G-KSA/S	Virtex-5		737	139	102,443	

### 6.2. MBEEA Implementation Results

Table 4 summarizes the MBEEA hardware implementation results on a contemporary Xilinx Virtex-7 (x7vx330t-2.g1157) FPGA. The proposed MBEEA generic architecture designs were implemented over different prime field lengths ( $n$  (bit length of  $p$ ) = 8, 16, 32, 64, 128, and 256 bits). In practice, 128- and 256-bit lengths for ECC/HECC and pairings systems, over prime fields, are very useful for modern security applications. The post place and route-static timing report was used to calculate the minimum clock period.

**Table 4.** FPGA implementation performance for the proposed MBEEA design in Virtex-7.

Design	$n = \text{Bit Length of } p$	Freq. (MHz)	Area (Slices)	Latency ( $\mu\text{s}$ )
MBEEA	8	530	545	0.179
	16	480	770	0.27
	32	420	1060	0.346
	64	380	1237	0.428
	128	310	1532	0.851
	256	250	2035	1.24

In Table 5, we compare the corresponding FPGA implementation results with those of reports available in the literature to additionally assess the actual performance of the proposed inverter design over  $\mathbb{F}_{256}$ .  $\mathbb{F}_p$  for a prime of length 256 bits is considered to be the large prime field size. Table 5 lists all related performance metrics: area, frequency (MHz), latency ( $\mu\text{s}$ ), and ADP. As an overall performance metric, we used ADP for all related designs to ensure a fair comparison.

**Table 5.** Performance analysis of the proposed and the existing modular inversion designs over  $\mathbb{F}_{256}$ .

Ref.	FPGA Device	Freq (MHz)	Time ( $\mu\text{s}$ )	Area (Slices)	ADP ( $\times 10^{-9}$ )
[36]	Virtex-7	146.23	2.329	1480	3.44
[37]	Kintex 7	142.38	2.33	1480	3.45
[38]	Virtex-6	151	3.39	1190	4.04
[39]	Virtex-6	146	3.52	1340	4.72
[40]	Virtex-5	129	7.937	592	4.7
[41]	Virtex-7	138.3	2.45	1577	3.87
[42]	Virtex-II	55.70	6.2	5863	36.35
[43]	Virtex-II	37	4.98	9213	45.88
[44]	Virtex-II	68.17	11.60	2085	24.19
[10]	Virtex-II	34	14.6	9146	133.53
[45]	Virtex-II	40.68	15.22	14,844	225.26
[46]	Virtex-II	50	6.4	5477	35
MBEEA	Virtex-E	106	2.92	2830	8.26
MBEEA	Virtex-II	175	1.77	2530	4.47
MBEEA	Virtex-5	208	1.49	2318	3.45
MBEEA	Virtex-6	240	1.29	2140	2.76
MBEEA	Virtex-7	276	1.12	2035	2.28

Table 5 clearly shows that the proposed design outperforms the existing large prime field-size modular inversion implementations.



In terms of latency, on the Virtex-7 device, the proposed design is 51.2% and 54.29% faster than those of [36,41], respectively. At the same time, on the Virtex-6 device, the proposed design is 61% and 63.35% faster than the competing designs of [38,39], respectively. On the Virtex-II, the proposed design remarkably outperforms all the existing designs of [10,42–46] in terms of latency and area. For instance, the works presented in [43,46] utilize 72.54% and 53.81%, respectively, more FPGA slices than this work.

At the same time, Table 5 shows that the proposed design ensures significantly lower ADP than the existing designs. Compared with [36,37,41] on the Virtex-7 device, the ADP of the proposed design has 23.81%, 23%, and 41.18% smaller ADP, respectively.

## 7. Conclusions

Securing the IoT is one of the major, if not the major, challenges for IT systems today. If the IoT is not sufficiently protected, critical events can occur, such as water or power outages, or worse, manipulated processes resulting in bacteria in water and faulty products such as cars, etc., that pose security risks. This urgent need for security, combined with the lack of security of IoT devices that are simultaneously connected to the Internet, which has a high risk of threat, illustrates the importance of the topic. The data and connections of many IoT devices are secured by cryptographic algorithms such as ECC and ECDSA. ECC/ECDSA is one of the algorithms targeted by the SPA attack. The modular BEEA inversion process used for generating an ECC/ECDSA private key is the focus of the leakage-based side-channel study. During the execution of the cryptographic algorithm, SPA can easily distinguish the conditional branches outcomes since a device consumes power differently and the execution time is not constant.

A new version of the BEE inversion algorithm was proposed that works in constant time by avoiding the data dependency of the classical BEEA. To ensure the traces are the same, thus preventing an attacker from distinguishing the computation across branches, the proposed method removes conditional instructions and divides the algorithm into separate branches with sub-functions. The classical BEEA is secured against SPA attacks due to the new countermeasure.

The new algorithm was designed so that the calculation of the modular inversion is easy and efficient for hardware implementation. The algorithm complexity is suitable even for a device having limited resources such as an IoT device. To achieve high performance on an FPGA, a variety of optimization techniques, including algorithmic reformulation and architectural optimization, are used. On a Xilinx Virtex-7 FPGA, our design can achieve a maximum clock rate of 276 MHz and it takes only 1.12  $\mu$ s to perform the 256-bit prime modular inversion.

The proposed design for  $\mathbb{F}_{256}$  provides an ADP figure of 2.28 on a Virtex-7, which is less than the relevant state-of-the-art solutions. Furthermore, our architecture outperforms others in terms of FPGA area (slices) and delay. Therefore, the proposed design is suitable for constrained implementations of cryptographic primitives, such as IoT, wireless sensor nodes, and RFID devices.

**Author Contributions:** A.S., M.Z., C.M. and H.Y.A. designed the research. A.S., M.Z., H.Y.A. and A.C. performed the code programming. C.M. carried out simulation analysis. A.S., M.Z., C.M., H.Y.A. and M.M. performed the analysis of the results and the interpretation of the results. A.S., M.Z. and A.C. wrote the first draft together, supported by H.Y.A. and M.M. commented on the draft and all authors finalized it. A.C. is the corresponding author. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

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

## References

1. Lee, S.; Huh, J.-H. An effective security measures for nuclear power plant using big data analysis approach. *J. Supercomput.* **2019**, *75*, 4267–4294. [CrossRef]
2. Kim, S.-K.; Kim, U.-M.; Huh, J.-H. A Study on Improvement of Blockchain Application to Overcome Vulnerability of IoT Multiplatform Security. *Energies* **2019**, *12*, 402. [CrossRef]
3. Miller, V.-S. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO '85 Proceedings*. CRYPTO 1985; Lecture Notes in Computer Science; Springer, Inc.: New York, NY, USA, 1986; pp. 417–426.
4. Islam, T.; Youki, R.-A.; Chowdhury, B.-R.; Hasan, A.-S.M.T. An ECC Based Secure Communication Protocol for Resource Constraints IoT Devices in Smart Home. In *Proceedings of the International Conference on Big Data, IoT, and Machine Learning*; Lecture Notes on Data Engineering and Communications Technologies; Arefin, M.S., Kaiser, M.S., Bandyopadhyay, A., Ahad, M.A.R., Ray, K., Eds.; Springer: Singapore, 2022; Volume 95.
5. Lohachab, K.A. ECC based inter-device authentication and authorization scheme using MQTT for IoT networks. *J. Inf. Secur. Appl.* **2019**, *46*, 1–12. [CrossRef]
6. Marin, L.; Pawlowski, M.-P.; Jara, A. Optimized ECC Implementation for Secure Communication between Heterogeneous IoT Devices. *Sensors* **2015**, *15*, 21478–21499. [CrossRef]
7. Varchola, M.; Drutarovsky, M.; Repka, M.; Zajac, P. Side-channel attack on multi-precision multiplier used in protected ecDSA implementation. In *Proceedings of the 2015 International Conference on ReConfigurable Computing and FPGAs (ReConFig)*, Riviera Maya, Mexico, 7–9 December 2015; pp. 1–6.
8. Fan, J.; Verbauwhede, I. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 265–282.
9. Rafik, M.-B.-O.; Mohammed, F. The impact of ecc's scalar multiplication on wireless sensor networks. In *Proceedings of the 2013 11th International Symposium on Programming and Systems (ISPS)*, Algiers, Algeria, 22–24 April 2013; pp. 17–23.
10. Ghosh, S.; Mukhopadhyay, D.; Roychowdhury, D. Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable GF(p) arithmetic unit. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2011**, *58*, 1798–1812. [CrossRef]
11. Kocher, P.C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology—CRYPTO'96*; Lecture Notes in Computer Science; Koblitz, N., Ed.; Springer: Berlin/Heidelberg, Germany, 1996; Volume 1109.
12. Shanmugham, S.-R.; Paramasivam, S. Survey on power analysis attacks and its impact on intelligent sensor networks. *IET Wirel. Sens. Syst.* **2018**, *8*, 295–304. [CrossRef]
13. Moon, J.; Junga, I.-Y.; Park, J.-H. IoT application protection against power analysis attack. *Comput. Electr. Eng.* **2018**, *67*, 566–578. [CrossRef]
14. Arpaia, P.; Bonavolontà, F.; Cioffi, A.; Moccaldi, N. Reproducibility Enhancement by Optimized Power Analysis Attacks in Vulnerability Assessment of IoT Transducers. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 1–8. [CrossRef]
15. Dao, B.A.; Hoang, T.T.; Le, A.T.; Tsukamoto, A.; Suzuki, K.; Pham, C.K. Exploiting the Back-Gate Biasing Technique as a Countermeasure Against Power Analysis Attacks. *IEEE Access* **2021**, *9*, 24768–24786. [CrossRef]
16. Dubeuf, J.; Hely, D.; Beroulle, V. ECDSA Passive Attacks, Leakage Sources, and Common Design Mistakes. *ACM Trans. Des. Autom. Electron. Syst.* **2016**, *21*, 1–24. [CrossRef]
17. Genkin, D.; Pachmanov, L.; Pipman, I.; Tromer, E.; Yarom, Y. ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 24–28 October 2016; pp. 1626–1638.
18. Zhang, K.; Xu, S.; Gu, D.; Gu, H.; Liu, J.; Guo, Z.; Liu, R.; Liu, L.; Hu, X. Practical Partial-Nonce-Exposure Attack on ECC Algorithm. In *Proceedings of the 13th International Conference on Computational Intelligence and Security (CIS)*, Hong Kong, China, 15–18 December 2017.
19. Wunan, W.; Hao, C.; Jun, C. The Attack Case of ECDSA on Blockchain Based on Improved Simple Power Analysis. In *Artificial Intelligence and Security*. ICAIS 2019; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; Volume 11635.
20. Thibault, J.-P.; O'Flynn, C.; Dewar, A. Ark of the ECC: An Open-Source ECDSA Power Analysis Attack on an FPGA Based Curve P-256 Implementation. *Cryptology ePrint Archive: Report 2021/1520*. 2021. Available online: <https://eprint.iacr.org/2021/1520> (accessed on 25 January 2022).
21. Sghaier, A.; Zeghid, M.; Massoud, C.; Mahchout, M. Design and Implementation of Low Area/Power Elliptic Curve Digital Signature Hardware Core. *Electronics* **2017**, *6*, 46. [CrossRef]
22. Choi, P.; Lee, M.K.; Kong, J.T.; Kim, D.K. Efficient Design and Performance Analysis of a Hardware Right-shift Binary Modular Inversion Algorithm in GF(p). *J. Semicond. Technol. Sci.* **2017**, *17*, 425–437.
23. Alhazmi, B.; Gebali, F. Fast Large Integer Modular Addition in GF(p) Using Novel Attribute-Based Representation. *IEEE Access* **2019**, *7*, 58704–58719. [CrossRef]
24. Janwadkar, S.; Dhavse, R. Qualitative and Quantitative Analysis of Parallel-Prefix Adders. In *Advances in VLSI and Embedded Systems*; Lecture Notes in Electrical Engineering; Patel, Z., Gupta, S., Kumar, Y.B.N., Eds.; Springer: Singapore, 2020; Volume 676.
25. Bos, J.-W. Constant time modular inversion. *J. Cryptogr. Eng.* **2014**, *4*, 275–281. [CrossRef]
26. Liu, Z.; Großschädl, J.; Li, L.; Xu, Q. Energy-efficient elliptic curve cryptography for MSP430-based wireless sensor nodes. In *Proceedings of the 21st Australasian Conference on Information Security and Privacy*, Melbourne, Australia, 4–6 July 2016; Springer: Cham, Switzerland, 2016; Volume 9722, pp. 94–112.

27. Xu, S.; Gu, H.; Wang, L.; Guo, Z.; Liu, J.; Lu, X.; Gu, D. Efficient and Constant Time Modular Inversions Over Prime Fields. In Proceedings of the 13th International Conference on Computational Intelligence and Security (CIS), Hong Kong, China, 15–18 December 2017; pp. 524–528.
28. Aldaya, A.C.; Márquez, R.C.; Sarmiento, A.-J.C.; 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. [\[CrossRef\]](#)
29. Savaş, E.; Koç, Ç.K. Montgomery inversion. *J. Cryptogr. Eng.* **2018**, *8*, 201–210. [\[CrossRef\]](#)
30. Bernstein, D.-J.; Yang, B.-Y. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**, 340–398. [\[CrossRef\]](#)
31. Awaludin, A.-M.; Larasati, H.-T.; Kim, H. High-Speed and Unified ECC Processor for Generic Weierstrass Curves over GF(p) on FPGA. *Sensors* **2021**, *21*, 1451. [\[CrossRef\]](#)
32. Sarna, S.; Czerwinski, R. RSA and ECC universal, constant time modular inversion. In Proceedings of the AIP Conference Proceedings, Crete, Greece, 29 April–3 May 2020; Volume 2343. [\[CrossRef\]](#)
33. Aldaya, A.-C.; Sarmiento, A.-J.C.; Sánchez-Solano, S. Spa vulnerabilities of the binary extended Euclidean algorithm. *J. Cryptogr. Eng.* **2017**, *7*, 273–285. [\[CrossRef\]](#)
34. Rajnish, D.; Jitendra, J. An Efficient Processing by using Kogge-Stone High-Speed Addition Technique. *Int. J. Comput. Appl.* **2015**, *131*, 21–23.
35. Vitoroulis, K.; Al-Khalili, A.-J. Performance of Parallel Prefix Adders implemented with FPGA technology. In Proceedings of the 2007 IEEE Northeast Workshop on Circuits and Systems, Montreal, QC, Canada, 5–8 August 2007. [\[CrossRef\]](#)
36. Hossain, M.-S.; Kong, Y. High-Performance FPGA Implementation of Modular Inversion over F256 for Elliptic Curve Cryptography. In Proceedings of the 2015 IEEE international conference on Data Science and Data Intensive Systems, Sydney, NSW, Australia, 11–13 December 2015; pp. 169–174. [\[CrossRef\]](#)
37. Hossain, M.-S.; Kong, Y.; Saeedi, E.; Vayalil, N.C. High-performance elliptic curve cryptography processor over NIST prime fields. *IET Comput. Digit. Tech.* **2016**, *11*, 33–42. [\[CrossRef\]](#)
38. Javeed, K.; Wang, X. Low latency flexible FPGA implementation of point multiplication on elliptic curves over GF (p). *Int. J. Circuit Theory Appl.* **2017**, *45*, 214–228. [\[CrossRef\]](#)
39. Javeed, K.; Wang, X.; Scott, M. High-performance hardware support for elliptic curve cryptography over general prime field. *Microprocess. Microsyst.* **2017**, *51*, 331–342. [\[CrossRef\]](#)
40. Mrabet, A.; El-Mrabet, N.; Bouallegue, B.; Mesnager, S.; Machhout, M. An efficient and scalable modular inversion/division for public-key cryptosystems. In Proceedings of the 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, Tunisia, 8–10 May 2017; pp. 1–6. [\[CrossRef\]](#)
41. Kudithi, T.; Sakthivel, R. High-performance ECC processor architecture design for IoT security applications. *J. Supercomput.* **2019**, *75*, 447–474. [\[CrossRef\]](#)
42. Liu, Z.; Liu, D.; Zou, X. An efficient and flexible hardware implementation of the dual field elliptic curve cryptographic processor. *IEEE Trans. Ind. Electron.* **2017**, *64*, 2353–2362. [\[CrossRef\]](#)
43. Lee, J.W.; Chung, S.C.; Chang, H.-C.; Lee, C.Y. Efficient power analysis resistant dual field elliptic curve cryptographic processor using heterogeneous dual processing element architecture. *IEEE Trans. Very Large Scale Integr. (VLSI)* **2014**, *22*, 49–61. [\[CrossRef\]](#)
44. Vliegen, J.; Mentens, N.; Genoe, J.; Braeken, A.; Kubera, S.; Touha, A.; Verbauwhede, I. A compact FPGA based architecture for elliptic curve cryptography over prime fields. In Proceedings of the ASAP 2010—21st IEEE International Conference on Application-Specific Systems, Architectures and Processors, Rennes, France, 7–9 July 2010; pp. 313–316. [\[CrossRef\]](#)
45. McIvor, C.J.; McLoone, M.; McCanny, J.V. Hardware elliptic curve cryptographic processor over GF(p). *IEEE Trans. Circuits Syst. I Regul. Pap.* **2006**, *53*, 1946–1957. [\[CrossRef\]](#)
46. Daly, A.; Marnane, W.; Kerins, T.; Popovici, E. An FPGA Implementation of a GF(p) ALU for encryption processors. *Microprocess. Microsyst.* **2004**, *28*, 253–260. [\[CrossRef\]](#)