IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 6, 2018, accepted November 29, 2018, date of publication December 10, 2018,
date of current version January 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2885989

Efficient Algorithm for Multi-Bit Montgomery Inverse Using Refined Multiplicative Inverse Modular 2^K

YOUNG-SIK KIM, (Member, IEEE)
Department of Information and Communication Engineering, Chosun University, Gwangju 61452, South Korea

e-mail: iamyskim@chosun.ac.kr

This work was supported by the Institute for Information and Communications Technology Promotion Grant through the Korea Government (MSIT) (Research on Lightweight Post-Quantum Crypto-systems for IoT and Cloud Computing) under Grant R-20160229-002941.

ABSTRACT This paper makes two major contributions. First, we propose an algorithm to compute a multiplicative inverse modulo 2^k without using integer division by refining the Arazi–Qi algorithm. We then generalize this algorithm into a scalable algorithm to compute an inverse modulo 2^{kl} by iteratively exploiting a k-bit-based process when the inverse modulo 2^k is given. Because of its scalability, this generalized algorithm can be cost-effectively implemented on hardware and applied to larger modulus operations by iteratively applying a processing unit. Furthermore, we highlight the possibility of the derivation of a binary representation of an inverse P^{-1} in terms of a representation of P and demonstrate this operation for up to eight bits. Second, using this direct relationship, we propose a multi-bit Montgomery inverse algorithm that is at least three times faster than the original version. Finally, we derive various properties of this algorithm and compare it with previous algorithms. This fast calculation of a Montgomery inverse is important for public key cryptographic applications, such as elliptic curve cryptosystems, because the relative speed of calculating a modular inverse for modular multiplication is a critical parameter for deciding which coordinate systems to adopt.

INDEX TERMS Cryptography, high speed arithmetic-algorithms, modular arithmetic, modular inverse, Montgomery inverse, Montgomery multiplication, public key cryptosystem.

I. INTRODUCTION

In public key cryptographic applications, modular operations are the fundamental building blocks for encrypting and decrypting messages [1], [2]. For efficient hardware implementations, Montgomery modular multiplication is frequently used because it requires no trivial division and integer division is replaced by 2^k division, which can be implemented using a simple k-bit right shift in the hardware [3], [4]. A good introduction to this field is [4].

However, before applying Montgomery modular multiplication, integer inputs A and B must be transformed into $A2^r$ and $B2^r$. Here, r is a parameter related to the bit size of the modulus P such that $P < 2^n \leq 2^r$, where n is the number of bits in the binary representation of P. In this paper, we only consider the case of $n = r$ for simplified analysis. Furthermore, for distinction, we assume that $A2^n$ is in the *Montgomery domain* and A is in the integer domain, even if A and $A2^r$ are in the same integer ring Z_p, where P is an odd integer.

Montgomery modular multiplication takes three inputs $A2^r$, $B2^r$, and a modulus P, and produces an output in the form $(A2^n) \cdot (B2^n) \cdot 2^{-n} \equiv (AB)2^n \bmod P$, which is also an integer in the Montgomery domain. Because of the homomorphic property, modular additions and subtractions also produce outputs in the Montgomery domain. Therefore, the transformation from the integer domain to the Montgomery domain is applied before applying any operations and its inverse transformation is applied after all operations are finished.

For modular inversion, the extended Euclidean algorithm is widely used to compute modular inverses in many applications. However, this algorithm is not preferred in embedded systems with restricted environments. Many approaches without direct executions of the extended Euclidean algorithm have been proposed for such systems. If the modulus P is a prime, then the multiplicative inverse of an integer A can be derived using Fermat's little theorem such that $A^{P-2} \equiv A^{-1} \pmod P$, which can be calculated using modular

exponentiation algorithms. If P is a prime and A is a composite number that is relatively prime to P, then Arazi's inversion formula can be used to modify the problem from a derivation of $P^{-1} \pmod{A}$ into $A^{-1} \pmod{P}$ [6].

The Multiplicative inverse modulo 2^k is used in many computer arithmetic problems, such as the discrete logarithm and exponential residue problems. Additionally, for the Montgomery multiplication algorithm with a radix 2^k, we must derive the multiplicative inverse of P, $1/P \pmod{2^k}$, where P is a modulus such that $AB2^{-r} \pmod{P}$ [7].

Kaliski [8] defined a new type of modular inverse called the *Montgomery inverse*. A Montgomery inverse of an integer A is defined as $A^{-1}2^r \pmod{P}$. He also proposed a simple algorithm to compute a Montgomery inverse. Kaliski's algorithm generates an inverse in the Montgomery domain from an integer in the integer domain. However, because every intermediate value is computed in the Montgomery domain, there has been great interest in designing a Montgomery inverse algorithm that has both inputs and outputs in the Montgomery domain. Savaş and Koç proposed a modified Montgomery inverse algorithm that can compute a Montgomery inverse $A^{-1}2^r \pmod{P}$ from an integer $A^{-1}2^r$ in the Montgomery domain [9].

The speed of operations in elliptic curve arithmetic is dependent on which coordinate system is selected [10]. The performance of each coordinate system is determined by the speed of individual field operations, particularly modular multiplication/reduction and multiplicative inversion over a finite field. Because the speed of field multiplication is generally much faster than field inversion, the coordinate system that requires the least field inversion operations is typically preferred, even if it requires more field multiplications. However, if we develop a faster method to calculate field inversions, this situation may change. It has been shown that if the speed of a field inversion is faster than that of 11 field multiplications, the affine coordinate system is more advantageous than the projective coordinate system for $GF(2^n)$ with a Galois field containing 2^n elements [11]. Generally, if the speed of a field inversion is faster than that of six field multiplications, then there is a faster way to perform elliptic curve arithmetic [12]. In other words, the speed increases for field inversions have a strong relationship to the speed of elliptic curve cryptosystems.

We propose a novel multi-bit Montgomery inverse algorithm. In the proposed algorithm, intermediate values are reduced by multiple bits in a single cycle. In simulations, for a modulus with a 27-bit length, the proposed algorithm calculates the result at least three times faster than Kaliski's original algorithm on average. Additionally, we derive various properties of the proposed algorithm and compare its performance to previous algorithms. This remainder of this paper is organized as follows: In Section II, some notations and previous results are presented. In Section III, we propose our multi-bit Montgomery inverse algorithm. The performance of the proposed multi-bit Montgomery inverse algorithm is compared to that of two previous algorithms in Section IV. Finally, we summarize our conclusions in Section V.

II. PRELIMINARIES

In this section, we define related notations and useful concepts. For the sake of self-containedness and comparison, we also introduce previous inverse modulo 2^k algorithms.

A. NOTATIONS

We use the following notations in this paper:

- P: A modulus, an odd integer.
- $Z_P = \{0, 1, 2, \ldots, P-1\}$: An integer ring with P elements.
- A and B: Integers in Z_P.
- X: Inverse of P.

B. INHERITANCE PROPERTY

Matula *et al.* [13] first introduced the inheritance property. This property states that a function $f(a_{k-1}a_{k-2}\ldots a_1\ a_0) = b_{k-1}b_{k-2}\ldots b_1\ b_0$ with the inheritance property shares lower digits with the output of the function $f(a_{k-2}a_{k-3}\ldots a_1\ a_0)$, except for the most significant bit (MSB). In the binary representation of an integer X, $X \pmod{2^k}$ is equivalent to taking the k-bit least significant digit (LSD) from X. Because $X \pmod{2^r}$ with $r < k$ corresponds to taking the r-bit least significant bit (LSB) from a k-bit $X \pmod{2^k}$, the inheritance property holds trivially for the inverse modulo 2^k operation.

C. DUSSÉ-KALISKI ALGORITHM

In 1990, Dussé and Kaliski [14] introduced an efficient algorithm to derive the inverse modulo 2^k, which is shown in Algorithm 1.

Algorithm 1 Phase 1 (Dussé and Kaliski [14], Calculation of Modular Inverse 2^k)

input : $P = (p_{k-1}, p_{k-2}, \ldots, p_1, 1)$
output: $X = (x_{k-1}, x_{k-2}, \ldots, x_1, 1)$, where $X \equiv P^{-1} \bmod 2^k$

```
1 S = 1;
2 for i = 2 to k do
3 |   if 2^{i-1} < P × S mod 2^i then
4 |   |   S = S + 2^{i-1}
5 |   end
6 end
7 return X = S;
```

Algorithm 1 uses single precision addition and $k \times k$ multiplications to compute a modular inverse X. Note that the inheritance property is used in the above algorithm. Because the $(i-1)$th S is congruent to 1 $(\bmod 2^{i-1})$, $PS \pmod{2^i}$ takes one of two possible values, 1 $(\bmod 2^i)$ or $1 + 2^{i-1} \pmod{2^i}$, in the ith loop. Thus, by adding 2^{i-1} in Step 4, it is possible to maintain $PS \equiv 1 \pmod{2^i}$ at the

end of each loop. The addition in Step 4 substitutes 0 with 1 in the $(i-1)$th bit and propagates the carry, which is why the $(k-1)$-bit addition is needed in the addition of Step 4 in the worst case. Therefore, the above algorithm can determine one bit at a time of the inverse X, moving from the LSB to the MSB. Because of Step 3, the $k \times k$ modular multiplications between the current intermediate value S and an integer P must be calculated for the generation of the next bit.

D. ARAZI-QI ALGORITHM

In 2008, Arazi and Qi [15] pointed out that a straightforward method of calculating $X \equiv P^{-1} \bmod 2^k$ is to add the left shifts of P in a controlled manner. Based on this observation, they proposed a method to calculate $P^{-1} \pmod{2^k}$ by first computing $2^{-k} \pmod P$ and then recovering $P^{-1} \pmod{2^k}$, as shown in Algorithm 2. Let $X = (x_{k-1}, x_{k-2}, \ldots, x_1, x_0)$ be the modular inverse of P, that is, $X \equiv P^{-1} \pmod{2^k}$. In Stage 1 of Algorithm 2, the final value of S is $2^{-k} \pmod P$. Using Stage 2 of Algorithm 2, we can convert S into $X \equiv P^{-1} \pmod{2^k}$.

Algorithm 2 Arazi and Qi [15], Calculation of Modular Inverse 2^k

Stage 1: Calculating $2^{-k} \pmod P$ through k sequential divisions of $2 \pmod P$.
input : P and k
output: $S \equiv 2^{-k} \bmod P$

1 **begin**
2 $S = 1$;
3 **for** $i = 1$ **to** k **do**
4 **if** S *is odd* **then** $S = S + P$;
5 $S = S/2$;
6 **end**
7 **return** S *and* k
8 **end**

Stage 2: Recovering $X \equiv P^{-1} \pmod{2^k}$ from $S \equiv 2^{-k} \pmod P$
input : $S \equiv 2^{-k} \bmod P$
output: $X \equiv P^{-1} \bmod 2^k$

9 **begin**
10 $T = S \cdot 2^k$;
11 $U = (T-1)/P$;
12 **return** $X = (2^k - U)$;
13 **end**

Note that in Step 11 of Stage 2, we must divide an integer $T-1$ by P. Even if the remainder of this division is always zero, an integer division is still required to obtain the inverse. This makes Algorithm 2 less efficient. However, with a simple modification, we can remove the procedures in Stage 2 in Algorithm 2.

III. REFINEMENT OF ARAZI-QI ALGORITHM

We start with a straightforward method of calculationg $X \equiv P^{-1} \bmod 2^k$ by adding the left shifts of P in a controlled manner. The congruent $PX \equiv 1 \pmod{2^k}$ can be rewritten as:

$$P\sum_{i=0}^{k-1} x_i 2^i = \sum_{i=0}^{k-1} P x_i 2^i \equiv 1 \bmod 2^k. \tag{1}$$

From the inheritance property, the $(i-1)$th LSB of the inverse modulo 2^i is the same as the inverse modulo 2^{i-1}. Therefore, the lower bits of the modular inverse can be derived from the LSBs sequentially. First, we have $x_0 = 1$ because P is odd. The next LSB of the inverse should satisfy the relationship $x_1 P \times 2^1 + 1 \times P \times 2^0 \equiv 1 \bmod 2^2$. Therefore, using a 1-bit right shift, we obtain $x_1\ P + (P-1)/2 \equiv 0 \bmod 2$. Thus, the second LSB is given by $x_1 \equiv -(P-1)/2 \bmod 2$ because the inverse is given by $P^{-1} \equiv 1 \bmod 2$. Using a similar procedure, we can drive the next LSB and repeat the process sequentially.

Note that this approach is similar to the derivation of quotients in the high-radix Montgomery modular multiplication algorithm [7], [16]. In Montgomery modular multiplication, the quotient q must be determined to obtain all zeros for the k LSBs of the sum of the previous result S, partial product bA, and modular reduction qP. Therefore, q is given by:

$$q \equiv -\frac{1}{P}(S + bA) \bmod 2^k.$$

Note that this is an application of the inverse $P^{-1} \bmod 2^k$.

By using the above approach, we can refine the Arazi-Qi algorithm. In Stage 1 of Algorithm 2, let x_i be the LSB of S in the ith iteration, where $1 \le i \le k$. Since the initial value of S is equal to 1, x_0 can be considered to always be 1. After k iterations in Stage 1 of Algorithm 2, we can represent S as follows:

$$\begin{aligned} S &= \frac{1}{2^k} + \frac{P}{2^k}\left(1 + x_1 2 + x_2 2^2 + x_3 2^3 + \ldots + x_{k-1} 2^{k-1}\right) \\ &= \frac{1}{2^k} + \frac{PX}{2^k}. \end{aligned}$$

Therefore, $-X$ is the inverse of $P \pmod{2^k}$ because $2^k S = 1 + PX$. This allows the result to be obtained one bit at a time during the k iterations in Stage 1 of Algorithm 2 instead of obtaining the entire S at the end of Stage 1.

Furthermore, in order to remove the final subtraction in Stage 2 of Algorithm 2, we can set the initial value to $(P-1)/2$, which is equivalent to a one-bit right shift and discarding the LSB of P (i.e., $P \gg 1$). Even if the LSB of P is discarded, there is no loss of information because we already know that P is always odd.

Using these observations, we can refine Algorithm 2 as shown in Algorithm 3.

In Algorithm 3, we can directly obtain the multiplicative inverse of P without any integer division or a final subtraction. In (1), each $x_i = s_0$ can be sequentially determined by accumulating Ps_0 where s_0 is the LSB of the accumulated

Algorithm 3 Main Result 1 (Refined Arazi-Qi Algorithm)

Stage 1: Calculate $2^{-k} \pmod P$ through k sequential divisions of $2 \pmod P$.
input : $P = (p_{k-1}, p_{k-2}, \ldots, p_1, 1)$
output: $X = (x_{k-1}, x_{k-2}, \ldots, x_1, 1)$, where $X \equiv P^{-1} \bmod 2^k$

```
begin
    S = (P − 1)/2 // To remove final subtraction in the Stage 2 of Algorithm 2;
    for i = 1 to k − 1 do
        x_i = s_0 // The LSB of the intermediate value S;
        S = S + x_i P;
        S = S/2;
    end
    return X
end
```

TABLE 1. Comparison of numbers of operations in three algorithms: Dussè-Kaliski, Arazi-Qi, and refined algorithms.

	Dussé-Kaliski	Arazi-Qi	Refined
# of Loop	$k-1$	k	$k-1$
k-bit Additions (on average)	$k/2$	$k/2$	$k/2$
Comparison	$k-1$	None	None
$k \times k$ Multiplication	$k-1$	1	None
Division by 2	None	k	$k-1$
Division by M	None	1	None
Subtraction	None	1	None

value in the previous loop. By adding Ps_0 to the 1-bit right shifted accumulated value $S/2$, the LSB of the sum $Ps_0 + S/2$ becomes zero. In other words, if the LSB of $S/2$ is 1, the LSB of $Ps_0 + S/2$ is 0 because an odd integer P will be added. Otherwise, there is no addition of P to the right shifted accumulated value.

The numbers of operations in the Dussé-Kaliski, Arazi-Qi, and refined algorithms are listed in Table 1. Note that the proposed algorithm only requires simple operations, such as k-bit addition and right shifting, while the others use more complicated operations, such as $k \times k$ multiplication and integer division.

It is not difficult to implement Algorithm 3 in hardware. In the implementation of Algorithm 3, a single bit of $X \equiv P^{-1} \pmod 2^k$ will be derived in each cycle from the LSB to the MSB. However, by utilizing Algorithm 3, it is possible to represent all k bits of X in terms of each component of P, p_i $(0 \le i \le k-1)$, which is explained in the next subsection.

A. DIRECT CALCULATION OF 8-BIT MODULAR INVERSE

For the fast calculation of intermediate quotients, it is necessary to compute modular 2^k in a fast manner. In this subsection, we propose a direct 8-bit relationship between an odd number and its inverse modulo 2^k. This calculation is based on the component-wise relationship between an integer P and its inverse X. In other words, it is possible to derive the relationship between the input $P = (p_{k-1}, p_{k-2}, \ldots, 1)$ and output $P^{-1} = (x_{k-1}, x_{k-2}, \ldots, 1)$. For a concrete example, we will demonstrate the relationship between the 8-bit components of P and X.

Let $P = (p_7, p_6, \ldots, p_1, 1)$. For $k = 8$, the lower eight bits of the modular inverse X can be represented in terms of components of P as described in the following proposition.

Proposition 1: The lower eight bits of the modular inverse X of an integer P modular 2^8 is given by:

$$
\begin{aligned}
&x_0 = 1, \quad x_1 = p_1,\ x_2 = p_2,\\
&x_3 = s_3 \bmod 2,\ \text{where } s_3 = p_3 + p_2 + p_1,\\
&x_4 = s_4 \bmod 2,\ \text{where } s_4 = p_4 + p_3 + \lfloor s_3/2 \rfloor,\\
&x_5 = s_5 \bmod 2,\ \text{where } s_5 = p_5 + p_4 + p_3 + p_1 + \lfloor s_4/2 \rfloor,\\
&x_6 = s_6 \bmod 2,\ \text{where } s_6 = p_6 + p_3p_2 + p_3p_1 + p_2p_1\\
&\qquad + p_5 + p_4 + p_2 + \lfloor s_5/2 \rfloor\\
&x_7 = s_7 \bmod 2,\ \text{where } s_7 = p_7 + p_6 + p_5 + p_4 + p_3p_2p_1\\
&\qquad + (p_4 + p_3 + 1)(p_2 + p_1) + \lfloor s_6/2 \rfloor.
\end{aligned}
$$

It is necessary to consider that there are carry propagations from the lower bit to the next bit.

Proof: The derivation of Proposition 1 is provided in Appendix. □

It is also possible to derive the higher bits of the modular inverse X. However, there is a tradeoff between the size of the modular inverse and the carry propagation delay and logical complexity. As a rule of thumb, it seems that an adequate size is less than eight bits. Furthermore, by using this proposition, we construct a multi-bit Montgomery inverse algorithm in Section IV. Furthermore, we similarly derive the lower eight bits of the relationship of the modular inverse in the binary extension field $GF(2^m)$ in Appendix. Note that, recently, another relationship between P and $-P^{-1} \bmod 2^8$ was proposed in [18]. However, our proposed relationship is more simple than the relationship proposed in [18] at the cost of carry propagation.

B. EXTENSION FROM K-BIT TO 8K-BIT

In order to calculate $X \equiv P^{-1} \pmod{2^{2k}}$ given $P_L^{-1} \pmod{2^k}$, where P_L and P_H denote the upper and lower half of the binary representation of a $2k$-bit value P, respectively, Arazi and Qi proposed the following theorem.

Theorem 2 (Arazi and Qi [15]): Given B and R as k-bit values where $R \equiv B^{-1} \pmod{2^k}$, and given P as a $2k$-bit value where $P_L = B$, the value $X \equiv P^{-1} \pmod{2^{2k}}$ can be efficiently obtained by calculating its lower half X_L and upper half X_H (both are independent k-bit values). The lower half of X can be calculated as $X_L = R$ and the upper half of X can be calculated as:

$$X_H \equiv -[[(R \cdot B)_H + (R \cdot P_H)_L] \cdot R] \bmod 2^k$$

where X is formed by the concatenation of X_H and X_L such that $X = X_H || X_L$.

As explained in [15], Theorem 2 can be applied to more general cases. For example, suppose that we want to calculate the multiplicative inverse X of an $8k$-bit integer P given the k-bit X_0 of X. Here, X can be represented as a series of concatenations of k-bit blocks of X, $X = X_7||X_6||X_5||X_4||X_3||X_2||X_1||X_0$. Then, using Theorem 2, we can sequentially compute the following values:

1) $X_1||X_0 \equiv P^{-1} \bmod 2^{2k}$,
2) $X_3||X_2||X_1||X_0 \equiv P^{-1} \bmod 2^{4k}$,
3) and finally $X \equiv P^{-1} \bmod 2^{8k}$.

In order to compute $X \equiv P^{-1} \pmod{2^{mk}}$, $\log_2 m$ consecutive doublings of partial blocks of X, $X_i||\ldots||X_0$, yield X. However, for each doubling, the operands of the internal multiplications double in size compared to the previous multiplications. It is not easy to implement this operation in hardware unless the architecture of the hardware is based on the maximum size, which results in high implementation costs. In other words, this algorithm is not scalable in hardware.

However, using a simple observation, we can transform it into a scalable algorithm. As a starting example, we describe a calculation scheme for the inverse modulo 2^{2k}, which is similar to Theorem 2. This scheme can calculate a $2k$-bit modular inverse using a given k-bit lower modular inverse.

The inverse X of P modulo 2^{2k} can be divided into an upper k-bit X_H and lower k-bit X_L. Suppose that the lower k-bit X_L is determined by Proposition 1 in Subsection III-A. Then, PX_L can be calculated using a $2k \times k$ multiplication. Because we only need the lower $2k$-bit multiplication result, it is possible to simplify the $2k \times k$ multiplication by omitting the generation circuit for the most significant k bits in the result because of the following modular operation. Thus, PX_L can always be represented as:

$$PX_L = (\underbrace{\times, \times, \ldots, \times, \times}_{2k\text{-bit}}, \underbrace{0, 0, \ldots, 0, 1}_{k\text{-bit}})_2 \tag{2}$$

where the $\times$s denote arbitrary bits in the upper $2k$-bit result. Here, the upper k bits will be discarded because of the modulus 2^k in (3). In (2), the lower k bits are always given as $(0, 0, \ldots, 0, 1)_2$ by the definition of X_L. Therefore, even though the lower k bits of PX_L are rounded by k-bit right shift, there is no loss of information. Next, we must derive an X_H that satisfies the following congruence.

$$PX_H + (PX_L - 1)/2^k \equiv 0 \bmod 2^k. \tag{3}$$

Because $\gcd(P, 2^k) = 1$, it is possible to obtain X_H from the following congruence.

$$\begin{aligned} X_H &\equiv -\frac{1}{P}\left(\frac{PX_L - 1}{2^k}\right) \bmod 2^k \\ &\equiv -X_L\left(\frac{PX_L - 1}{2^k}\right) \bmod 2^k \end{aligned}$$

Then, $X_H||X_L$ is the $2k$-bit modular inverse of P. This procedure is depicted in Fig. 1.

Note that there are two types of multipliers: $2k \times k$ and $k \times k$ multipliers. Prior to $k \times k$ multiplication, X_L must be multiplied by -1, which is equivalent to replacing $X_L = (x_k, x_{k-1}, \ldots, x_1, 1)_2$ with $-X_L = (1 - X_k, 1 - X_{k-1}, \ldots, 1 - x_1, 1)_2$. Because $2k \times k$ and $k \times k$ multiplications can be parallelized, it is possible to obtain a $2k$-bit modular inverse X with little delay.

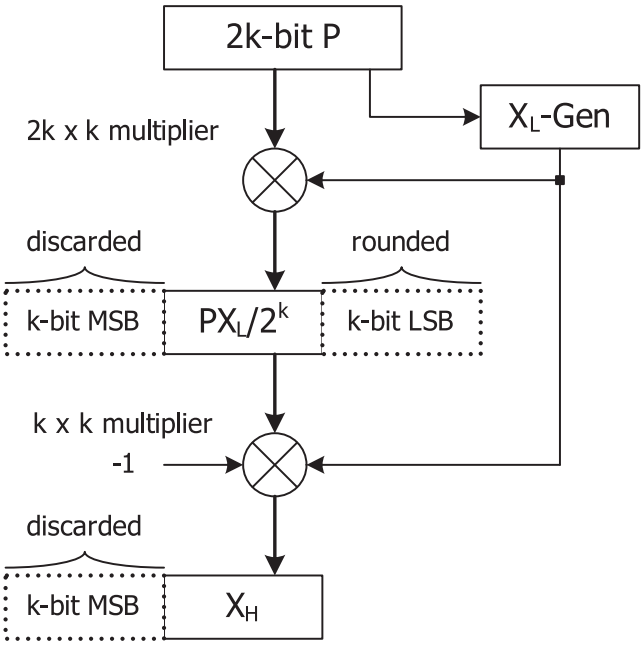

FIGURE 1. Calculation procedure for $2k$-bit inverse from k-bit inverse.

Algorithm 4 Derivation of kl-Bit Modular Inverse From Multiple k-Bit Modular Inverses

input : $P = (p_{kl-1}, p_{kl-2}, \ldots, p_1, 1)$
output: $X = (x_{kl-1}, x_{kl-2}, \ldots, x_1, 1)$, where $X \equiv P^{-1} \bmod 2^{kl}$

```
begin
    Generate the lowest k-bit modular inverse X_0 of X.;
    S = (P · X_0 − 1)/2^k // Scalable multiplication and addition;
    for i = 1 to l − 1 do
        X_i = −X_0 · S mod 2^k // k × k multiplication, fixed size;
        S = S + P · X_i // Scalable multiplication and addition;
        S = S/2^k;
    end
    return X = X_{l−1}|| · · · ||X_0
end
```

We will now generalize the scheme described above as in Algorithm 4. In Algorithm 4, it is possible to derive a kl-bit modular inverse by iteratively using a k-bit modular inverse $l - 1$ times.

Proof of Correctness of Algorithm 4: Suppose that a modular inverse X can be divided into l sub-blocks with k-bit lengths, $X = X_{l-1}||\ldots||X_1||X_0$. By the definition of the

modular inverse X_0, we have:

$$S_0 = P \cdot X_0 = (\underbrace{\times, \times, \ldots, \times, \times}_{(l-1)k\text{-bit}}, \underbrace{0, 0, \ldots, 0, 1}_{k\text{-bit}})_2.$$

After rounding the lower k bits, X_1 can be found using the following congruent relationship.

$$P \cdot X_1 + (S_0 - 1)/2^k \equiv 0 \bmod 2^k$$

which is equivalent to:

$$X_1 \equiv -\frac{1}{P} \cdot \frac{S_0 - 1}{2^k} \bmod 2^k.$$

We then have the next intermediate values as:

$$\begin{aligned} S_1 &= P \cdot X_1 + (S_0 - 1)/2^k \\ &= (\underbrace{\times, \times, \ldots, \times, \times}_{(l-2)k\text{-bit}}, \underbrace{0, 0, \ldots, 0, 0}_{k\text{-bit}})_2. \end{aligned}$$

Generally, for a given S_{i-1}, $(0 < i \leq l - 1)$, we can obtain X_i as:

$$X_i \equiv -\frac{1}{P}(S_{i-1}/2^k) \bmod 2^k.$$

Then, the i intermediate value S_i are given by:

$$\begin{aligned} S_i &= P \cdot X_i + S_{i-1}/2^k \\ &= (\underbrace{\times, \times, \ldots, \times, \times}_{(l-(i+1))k\text{-bit}}, \underbrace{0, 0, \ldots, 0, 0}_{k\text{-bit}})_2 \end{aligned}$$

□

It is easy to see that Algorithm 4 is a generalized version of Algorithm 3. What needs to be emphasized is that Algorithm 4 is scalable in hardware. In Step 6 (and Step 3) of Algorithm 4, $P \cdot X_i + S$ can be implemented in a scalable manner. In other words, it can be computed by using a given fixed-length multiplication-addition block. In Step 5 of Algorithm 4, we utilize $k \times k$ multiplication and the lower k-bit result without considering the total size of the operands.

Example 3: Let $P = 7919 = (1111011101111)_2$. Then $X_0 = 15$ from Proposition 1. $S_0 = (PX_0 - 1)/2^4 = 7424$. In order to distinguish the value S for each step, we add the subscript to S_i where i is the step number.

$$\begin{aligned} X_1 &= -X_0 \cdot S \bmod 2^4 = 0, S_1 = (S_0 + P \cdot X_0)/2^4 = 464 \\ X_2 &= -X_0 \cdot S \bmod 2^4 = 0, S_2 = (S_1 + P \cdot X_1)/2^4 = 29 \\ X_3 &= -X_0 \cdot S \bmod 2^4 = 13. \end{aligned}$$

Then $X = 13 \times 2^{4\times 3} + 0 \times 2^{4\times 2} + 0 \times 2^4 + 15 = 53263$. It is easy to check that $X \times P = 421,789,697 = 1 \bmod 2^{16}$.

IV. NEW MULTI-BIT MONTGOMERY INVERSE ALGORITHM

In this section, we propose a multi-bit Montgomery inverse algorithm by exploiting Proposition 1. First, we will briefly introduce previous Montgomery inverse algorithms.

A. KALISKI'S MONTGOMERY INVERSE ALGORITHM

The Montgomery inverse of an integer $A \in [1, P - 1]$ is defined as an integer $X \equiv A^{-1}2^n \bmod P$, where P is an odd prime and $n = \lceil \log_2 P \rceil$. Kaliski [8] proposed the concept of the Montgomery inverse of an integer. He proposed an algorithm to compute a Montgomery inverse from an integer in the integer domain. Kaliski's algorithm consists of two phases. The first phase computes the almost Montgomery inverse of A, defined in [9], which is given by:

$$\mathrm{MI}_1(A) = A^{-1}2^k \bmod P \tag{4}$$

where $n \leq k \leq 2n$. The second phase then reduces the exponent k of 2 in (4) to n as follows.

$$\mathrm{MI}_2(B) = B2^{n-k} \bmod P. \tag{5}$$

Therefore, the Montgomery inverse can be sequentially computed using (4) and (5)

$$\begin{aligned} \mathrm{MI}_1(A) &= \mathrm{MI}_2(\mathrm{MI}_1(A)) = (A^{-1}2^k)2^{n-k} && (6) \\ &\equiv A^{-1}2^n \bmod P. && (7) \end{aligned}$$

Savas and Koç [9] modified Phase 2 to obtain a final result in the Montgomery domain from an integer in the Montgomery domain as:

$$\mathrm{MI}'_2(\mathrm{MI}_1(A)) = A^{-1}2^{2n} \bmod P.$$

This procedure can be implemented using Algorithm 5. In [9], it was pointed out that the second phase can be implemented by using a Montgomery multiplication instead of successive addition and shift operations in Phase 2 of Algorithm 5.

As pointed out in [8], this algorithm satisfies the following invariants.

$$\begin{aligned} AR &\equiv -U2^k \bmod P && (8) \\ AS &\equiv V2^k \bmod P && (9) \end{aligned}$$

which can be proved by induction [8]. During iteration, U and V are monotonically decreasing and eventually reach one. Note that if $U = 1$ or $V = 1$, we have $R \equiv -A^{-1}2^k \bmod P$ or $S \equiv A^{-1}2^k \bmod P$, respectively. Therefore, there is no need to wait to reach $V = 0$ in Step 3 of Algorithm 5.

By multiplying V and U as (8) and (9), respectively, we can obtain:

$$A(US + VR) \equiv UV2^k - VU2^k = 0 \bmod P$$

Because $\gcd(A, P) = 1$, we have $US + VR \equiv 0 \pmod P$. From the initial conditions, we obtain:

$$US + VR = P \tag{10}$$

Invariants (8)–(10) are inherited in the following two algorithms in Subsections IV-B and IV-C.

Algorithm 5 Kaliski's Montgomery Inverse Algorithm

Phase 1 (Algomost Montgomery Inverse Phase)
input : P and $A \in [1, P-1]$)
output: $D \in [1, P-1]$ and k where $D = A^{-1}2^k \bmod P$ and $n \le k \le 2n$

begin
  $U = P$, $V = A$, $R = 0$, and $S = 1$. **while** $V > 0$ **do**
    if U *is even* **then** $U = U/2$, $S = 2S$;
    else if V *is even* **then** $V = V/2$, $R = 2R$;
    else if $U > V$ **then** $U = (U-V)/2$, $R = R+S$, $S = 2S$;
    else if $V \ge U$ **then** $V = (V-U)/2$, $S = R+S$, $R = 2R$;
    $k = k+1$
  end
  if $R \le P$ **then** $D = R - P$;
  return $D = P - R$ *and* k
end

Phase 2 (Correction)
input : $D \in [1, P-1]$, P, and k from Phase 1
output: $X \in [1, P-1]$, where $X \equiv A^{-1}2^n \bmod P$

begin
  for $i = 1$ **to** $k - n$ **do**
    if D *is even* **then** $U = U/2$;
    else $D = (D+P)/2$;
  end
  return $X = D$
end

B. MULTI-BIT SHIFTING MONTGOMERY INVERSE ALGORITHM

Gutub *et al.* [17] proposed a hardware algorithm for the multi-bit shifting Montgomery inverse algorithm, which is shown in Algorithm 6. In this algorithm, the intermediate values U and V are represented in the binary form:

$$U = (u_{n-1}, u_{n-2}, \ldots, u_1, u_0)_2$$
$$V = (v_{n-1}, v_{n-2}, \ldots, v_1, v_0)_2.$$

Note that Algorithm 6 is a slightly modified version of the Gutub-Tenca-Savaş-Koç algorithm presented in [17] because the algorithm in [17] is intended for implementation in hardware. Here, we omit certain parameters that were adopted for hardware specific reasons.

In this algorithm, if U or V are even, multi-bit shifting occurs in Steps 4–9. Note that the maximum size of the right shift in Algorithm 6 is three in Steps 4 and 7. Although the maximum shift increases, performance will increase slightly because the probability of maximum shifting is small. Furthermore, for an U or V, this algorithm works similarly to Kaliski's Montgomery inverse algorithm [8]. Therefore, the effect of multi-bit shifting is restricted and this algorithm is slightly faster than the original algorithm. This tendency is demonstrated in Section V.

It is easy to see that one of output k in Algorithm 6 is equal to the output k in Algorithm 5 when the same A and P are applied because this algorithm simply accelerates consecutive shift operations.

Algorithm 6 Gutub-Tenca-Savaş-Koç [17] (Multi-Bit Shifting MonInv Algorithm)

Phase 1 (Algomost Montgomery Inverse Phase)
input : $A2^r \in [1, P-1]$ where $r \ge n$ $2^{n-1} \le P < 2^n$
output: $D \in [1, P-1]$ and k where $D = A^{-1}2^{k-r} \bmod P$ and $n \le k \le 2n$

begin
  $U = P$, $V = A2^r$, $R = 0$, $S = 1$, and $k = 0$;
  while $V \ne 0$ **do**
    if $u_2\ u_1\ u_0 = 000$ **then** $U = U/8$, $S = 8S$, $k = k+3$;
    else if $u_2\ u_1\ u_0 = 100$ **then** $U = U/4$, $S = 4S$, $k = k+2$;
    else if $u_1\ u_0 = 10$ **then** $U = U/2$, $S = 2S$, $k = k+1$;
    else if $v_2\ v_1\ v_0 = 000$ **then** $V = V/8$, $R = 8R$, $k = k+3$;
    else if $v_2\ v_1\ v_0 = 100$ **then** $V = V/4$, $R = 4R$, $k = k+2$;
    else if $v_1\ v_0 = 10$ **then** $V = V/2$, $R = 2R$, $k = k+1$;
    else if $U > V$ **then** $U = (U-V)/2$, $R = R+S$, $S = 2S$, $k = k+1$;
    else if $V \le U$ **then** $V = (V-U)/2$, $S = R+S$, $R = 2R$, $k = k+1$;
  end
  if $P > R$ **then** $D = P - R$;
  return $D = 2P - R$ *and* k
end

Phase 2 (Correction)
input : $D \in [1, P-1]$ and k where $D \equiv A^{-1}2^{k-r} \bmod P$ and $n < k < 2n$
output: $X \in [1, P-1]$, where $X \equiv A^{-1}2^n \bmod P$

begin
  $j = 2r - k$;
  while $j > 1$ **do**
    if $j = 1$ **then** $D = 2D$, $j = j-1$;
    else $D = 4D$, $j = j-2$;
    if $D > 3P$ **then** $D = D - 3P$;
    else if $D > 2P$ **then** $D = D - 2P$;
    else if $D > P$ **then** $D = D - P$;
  end
  return $X = D$
end

C. NEW MULTI-BIT MONTGOMERY INVERSE ALGORITHM

In this subsection, we propose a multi-bit Montgomery inverse algorithm with not only multi-bit shifting for the intermediate parameters U and V, but also multi-bit reduction by exploiting Lemma 1 to calculate the modulo 2^k. Also note that even in the case where one of U or V is not odd, multi-bit reduction $(U + qV)/16$ or $(V + qU)/16$ can be used, whereas previous algorithms requires that both U and V are odd to compute $(U - V)/2$ or $(V - U)/2$, which can significantly reduce U or V. This property accelerates the proposed algorithm.

We will separately consider Phases 1 and 2 of the proposed algorithm. Algorithm 7 described Phase 1 of the proposed multi-bit Montgomery inverse algorithm. In this algorithm, the function QUOT(X, Y) is defined as:

$$\text{QUOT}(X, Y) \equiv -\frac{X}{Y} \bmod 2^k$$

which can be decomposed into the following two stages.

$$\text{QUOT}(X, Y) \equiv \left[\left(-\frac{1}{Y}\right) \bmod 2^k\right] \cdot X \bmod 2^k$$

The value of $q = \text{QUOT}(X, Y)$ will be interpreted as a 2's complement number which is in the range $-2^{k-1} \leq q < 2^{k-1}$. Note that the first stage can be computed using Lemma 1 up to $k = 8$. Although it is possible to compute 8 bits (or more), Algorithm 7 here is based on a 4-bit modular inverse because of the simple calculation of such an inverse. The second stage uses the lower k bits of $k \times k$ multiplication, which account for nearly half the computation of $k \times k$ multiplication. Note that the function QUOT(U, V) generates a multiple of V that satisfies $U + qV \equiv 0 \pmod{2^4}$. Therefore, U or V can be divided by 2^4 in Steps 11 or 18, respectively. Additionally, it must be emphasized that in order to compute the inverse modulo 2^4, V must be odd, while U can be even.

In Algorithm 7, LSD_U and LSD_V represent the 4 LSBs of the intermediate values U and V as:

$$\begin{aligned}\text{LSD}_U &= u_3 2^3 + u_2 2^2 + u_1 2^1 + u_0 \\ \text{LSD}_V &= v_3 2^3 + v_2 2^2 + v_1 2^1 + v_0.\end{aligned}$$

Note that we must correct the results $D \equiv A^{-1}2^k \pmod{P}$ of Algorithm 7 in Phase 2 of Algorithm 8. If we wish to obtain $X \equiv A^{-1}2^{2n} \pmod{P}$, we can use Algorithm 8. To achieve a greater speedup, Phase 2 of Algorithm 5 processes 4 bits at a time for $k > 2n$ by using the QUOT(X, Y) function.

Now, we derive some properties of Phase 1 of Algorithm 7. Because Algorithm 7 is derived from Algorithm 5, it also satisfies the same invariants, as shown in the following lemma.

Lemma 4: The parameters of Algorithm 7 also satisfy the following invariants.

$$\begin{aligned}AR &\equiv -U2^k \bmod P \\ AS &\equiv V2^k \bmod P \\ US + VR &= P.\end{aligned}$$

Furthermore, for the existence of a Montgomery inverse, A and P must be relatively prime. Based on this property, we can prove the following lemma, which indicates that the intermediate values U and V are always relatively prime. Let U_i and V_i be the intermediate values of U and V in the ith loop, respectively.

Lemma 5: The parameters U_i and V_i of Algorithm 7 are relatively prime, that is:

$$\gcd(U, V) = \gcd(A, P) = 1.$$

Proof: For the first step, because $U_0 = P$ and $V_0 = A$, the property is clearly satisfied. For an odd V_{i-1}, without loss

Algorithm 7 Main Result 2 (Multi-Bit Montgomery Inverse Algorithm (Phase 1))

input : $A \in [1, P-1]$ where $2^{n-1} \leq P < 2^n$
output: $D \in [1, P-1]$ and k where $D = A^{-1}2^{k-r} \bmod P$

1 **begin**
2 $U = P$, $V = A$, $R = 0$, $S = 1$, and $k = 0$.;
3 **while** *($|U| \neq 1$) and ($|V| \neq 1$)* **do**
4 **if** $U\%16 = 0$ **then** $U = U/16$, $S = 16S$, $k = k + 4$;
5 **else if** $V\%16 = 0$ **then** $V = V/16$, $R = 16R$, $k = k + 4$;
6 **else if** $|U| > |V|$ **then**
7 **if** $v_0 = 1$ **then** $k = k + 4$;
8 **else if** $v_1 = 1$ **then** $V = V/2$, $R = 2R$, $k = k + 5$;
9 **else if** $v_2 = 1$ **then** $V = V/4$, $R = 4R$, $k = k + 6$;
10 **else if** $v_3 = 1$ **then** $V = V/8$, $R = 8R$, $k = k + 7$;
11 $q = \text{QUOT}(\text{LSD}_U, \text{LSD}_V)$, $U = (U + qV)/16$, $R = R - qS$, $S = 16S$;
12 **end**
13 **else if** $|V| > |U|$ **then**
14 **if** $u_0 = 1$ **then** $k = k + 4$;
15 **else if** $u_1 = 1$ **then** $U = U/2$, $S = 2S$, $k = k + 5$;
16 **else if** $u_2 = 1$ **then** $U = U/4$, $S = 4S$, $k = k + 6$;
17 **else if** $u_3 = 1$ **then** $U = U/8$, $S = 8S$, $k = k + 7$;
18 $q = \text{QUOT}(\text{LSD}_V, \text{LSD}_U)$, $V = (V + qU)/16$, $S = S - qR$, $R = 16R$;
19 **end**
20 **end**
21 **if** $U = 1$ **then return** $D = P - R$ *and* k;
22 **else if** $U = -1$ **then return** $D = R$ *and* k;
23 **else if** $V = 1$ **then return** $D = S$ *and* k;
24 **else if** $V = -1$ **then return** $D = P - S$ *and* k;
25 **end**

Algorithm 8 Main Result 2 (Multi-Bit Montgomery Inverse Algorithm (Phase 2))

input : $D \equiv A^{-1}2^k \bmod P$, k from Phase 1 and P
output: $X \equiv A^{-1}2^{2n} \bmod P$

```
begin
    if k < 2n then
        j = 2n − k;
        while j > 0 do
            if j = 1 then D = 2D, j = j − 1;
            else D = 4D, j = j − 2;
            if D > 3P then D = D − 3P;
            else if D > 2P then D = D − 2P;
            else if D > P then D = D − P;
        end
    end
    else
        r ≡ (k − 2n) mod 4  // (r ∈ {0, 1, 2, 3});
        l = (k − 2n − r)/4;
        for i = 1 to l do
            q = QUOT(LSB_D, LSB_P);
            D = (D + qP)/16;
        end
        for i = 1 to r do
            if D is even then D = D/2;
            else D = (D + P)/2;
        end
    end
    return X = D
end
```

of generality, suppose that $U > V$. Then, we have:

$$\begin{aligned}\gcd(U_i, V_i) &= \gcd(\frac{U_{i-1} + qV_{i-1}}{16}, V_{i-1}) \\ &= \gcd(U_{i-1} + qV_{i-1}, V_{i-1}) \\ &= \gcd(U_{i-1}, V_{i-1}) = 1.\end{aligned}$$

If U_{i-1} (or V_{i-1}) is a multiple of 16, then its counterpart V_{i-1} (or U_{i-1}) must be odd because of $\gcd(U_{i-1}, V_{i-1}) = 1$. Thus, we have:

$$\gcd(U_i, V_i) = \gcd(\frac{U_{i-1}}{16}, V_{i-1}) = \gcd(U_{i-1}, V_{i-1}) = 1.$$

For an even V_{i-1}, U_{i-1} should be odd. Therefore, we have:

$$\begin{aligned}\gcd(U_i, V_i) &= \gcd(\frac{U_{i-1} + q\frac{V_{i-1}}{2^s}}{16}, \frac{V_{i-1}}{2^s}) \\ &= \gcd(U_{i-1}, \frac{V_{i-1}}{2^s}) = \gcd(U_{i-1}, V_{i-1}) = 1\end{aligned}$$

where s is the index of the first nonzero bit from the LSB of V_{i-1}. Therefore, we can easily prove the statement by induction. □

The following theorem shows that the algorithm will converge to zero within a finite number of iterations.

Theorem 6 (Proof of Termination): Suppose that $\gcd(A, P) = 1$. Then, $|U|$ and $|V|$ in Algorithm 7 are equal to 1 for some k.

Proof: We will show that the values of $|U|$ and $|V|$ are monotonically decreasing over the iterations. That is, we will show that $|U_i| \le |U_{i-1}|$ and $|V_i| \le |V_{i-1}|$. If U_{i-1} or V_{i-1} is a multiple of 16, then it is clear that:

$$\begin{aligned}|U_i| &= |U_{i-1}/16| < |U_{i-1}|, \text{ or} \\ |V_i| &= |V_{i-1}/16| < |V_{i-1}|\end{aligned}$$

If $|U_{i-1}| > |V_{i-1}|$, using Algorithm 7, we have:

$$16|U_i| = |U_{i-1} + qV'_{i-1}|$$

where $V'_{i-1} = V_{i-1}/2^s$ and s is the index of the first nonzero bit of the LSB of V_{i-1}. From the Cauchy-Schwarz inequality, we can obtain:

$$\begin{aligned}16|U_i| &= |U_{i-1} + qV'_{i-1}| \le |U_{i-1}| + |q||V'_{i-1}| \\ &\le |U_{i-1}| + |q||V_{i-1}|.\end{aligned}$$

Because $|U_{i-1}| > |V_{i-1}|$, we have:

$$16|U_i| \le |U_{i-1}| + |q||V_{i-1}| < (1 + |q|)|U_{i-1}|.$$

Because $|q| < 8$, we have:

$$|U_{i-1}| > \frac{16}{8}|U_i| > |U_i|.$$

If $|V_{i-1}| \ge |U_{i-1}|$, we can similarly show that:

$$|V_i| < \frac{16}{8}|V_i| \le \frac{1}{8}(|V_{i-1}| + |q||U_{i-1}|) < |V_{i-1}|.$$

Therefore, $|U_i|$ and $|V_i|$ converge to zero by the well-ordering principle as the number of iterations increases. Suppose that $|U_i|$ or $|V_i|$ is equal to zero for some i. Let $|U_i| = 0$ without loss of generality. This means that $|U_{i-1}|$ is a multiple of $|V_{i-1}|$ because $U_{i-1} + qV_{i-1} = 0$. Therefore, from Lemma 5, $\gcd(A, P) = \gcd(|U_{i-1}|, |V_{i-1}|) = |V_{i-1}| = 1$. Therefore, $|U_{i-1}| = 1$ or $|V_{i-1}| = 1$. □

From Theorem 6, we can conclude that Algorithm 7 finishes after a finite number of iterations.

Proof of Correctness of Algorithm 7: From Lemma 4, Algorithm 7 always satisfies $AR \equiv -U2^k \bmod P$ and $AS \equiv V2^k \bmod P$. Since the termination condition is both $|U| = 1$ and $|V| = 1$, finally we can have $AR \equiv \pm 2^k \bmod P$ and $AS \equiv \pm 2^k \bmod P$, that is, $R \equiv \pm A^{-1}2^k \bmod P$ and $S \equiv \pm A^{-1}2^k \bmod P$, respectively. From Theorem 6, Algorithm 7 will produce the result after a finite number of steps. □.

Suppose that the number of iterations is c. Let k_i be the difference between $(i-1)$th k and ith k in Algorithm 7. Then, the output value k is the sum of each k, $k = \sum_{j=1}^{c} k_j$. As stated in [19], U (and V) can be maintained as positive values by reversing the signs of U and R (or V and S) simultaneously. In such a case, the invariants (8) and (9) can be conserved while the invariant (10) changes slightly when $US + VR = -P$. Therefore, it is reasonable to assume that U and V are always positive values. Now, we can obtain

the bounds of the output k in Algorithm 7, as shown in the following theorem.

Theorem 7: Suppose that $\gcd(A, P) = 1$, P is odd, $2^{n-1} \leq P < 2^n$, and $P > A > 0$. Suppose that one of $|U_c|$ and $|V_c|$ is equal to 1 and the other is equal to $2^d - 1$, where $d \geq 0$. Then, we have:

$$n - d - 1 \leq k \leq 2n + 3c - d + 1.$$

Proof: In this proof, we assume that U_i and V_i are positive based on the above information [19]. First, we concentrate on the lower bound. From the assumptions in this theorem, we have:

$$|U_c| + |V_c| \geq \frac{|U_{c-1}| + |V_{c-1}|}{2^{k_c}} \geq \cdots \geq \frac{|U_0| + |V_0|}{2^{\sum_{j=1}^{c} k_j}} = \frac{A + P}{2^k} > \frac{P}{2^k}.$$

If $|U_c| = 1$ and $|V_c| = 2^d - 1$, we have:

$$2^{k+d} \geq P \geq 2^{n-1}.$$

Therefore, the lower bound of k is given as $k \geq n - d - 1$. If $|V_c| = 1$ and $|U_c| = 2^d - 1$, we have the same bound $k \geq n - d - 1$. Note that in the worst case, it is possible to have $c = 1$. Therefore, we obtain the lower bound of k as 4, $k \geq 4$.

Next, we consider the upper bound. If $U > V$ and $q > 0$, we have:

$$|U_c| = \frac{|U_{c-1} + qV_{c-1}|}{16} \leq \frac{|1 + q||U_{c-1}|}{16} \leq \frac{8|U_{c-1}|}{16}$$
$$|V_c| = \frac{|V_{c-1}|}{2^{s_{c-1}}}$$

where s_{c-1} is the number of right shift bits for V_{c-1}. If $U > V$ and $q \leq 0$, we have:

$$|U_c| = \frac{|U_{c-1} + qV_{c-1}|}{16} \leq \frac{|U_{c-1}|}{16} < \frac{8|U_{c-1}|}{16}.$$

In this case, we always have:

$$|U_c||V_c| < \frac{8}{2^{k_c}}|U_{c-1}||V_{c-1}|$$

where $k_c = 4 + s_{c-1}$. The converse case also yields the same upper bound. By repeating this process until reaching the initial values, we have:

$$|U_c||V_c| \leq \frac{8^c}{2^k}|U_0||V_0|. \tag{11}$$

If $|U_c| = 1$ and $|V_c| = 2^d - 1$, we have:

$$2^{d-1} < 2^d - 1 \leq \frac{8^c}{2^k}AP < \frac{8^c}{2^k}P^2 < 2^{3c-k+2n}.$$

Thus, we obtain the upper bound:

$$k + d - 1 < 2n + 3c.$$

□

By comparing the average values from Section II, we note that the upper bound is not tight. On average, according to the simulation results, the actual upper bound is closer to $2n + 1.5c - d + 1$ than $2n + 3c - d + 1$.

V. PERFORMANCE COMPARISON

In this section, we compare the expected performances of Phase 1 of Algorithms 5, 6, and 7. We will also attempt to derive the probabilistic behaviors of the proposed multi-bit Montgomery inverse algorithm. The simulation results for the average cycles required to reach $|U| = 1$ or $|V| = 1$ are presented for three algorithms. Finally, we interpret the average values using the probabilistic results.

A. EXPECTED NUMBER OF PROCESSED BITS FOR A LOOP

In order to measure the speed of the proposed algorithm, because P and A are initially assigned to U or V, respectively, we should estimate the average number of processed bits of U and V in a loop of Algorithm 7. In this estimation, we assume that ones and zeros in the binary representations of U and V are sufficiently random.

We will estimate the size of the division factor of U or V for each loop. Note that some steps among the 25 total steps in Algorithm 7 do not reduce the intermediate values of U or V.

First, we estimate the average reduction factor of U. In Algorithm 7, the following steps reduce U with corresponding probabilities based on the lower four bits of the intermediate value U.

- In Step 4, U is reduced by a factor of 16 with a probability of $\frac{1}{16}$.
- In Step 5, U is not reduced with a probability of $\frac{15}{16} \cdot \frac{1}{16}$.
- In Steps 7–10, U is reduced by a factor of 16 on average with a probability of $(\frac{15}{16})^2 \cdot \frac{1}{2}$.
- In Step 14, U is not reduced with a probability of $\frac{15}{16} \cdot \frac{1}{4}$.
- In Step 15, U is reduced by a factor of 2 with a probability of $\frac{15}{16} \cdot \frac{1}{8}$.
- In Step 16, U is reduced by a factor of 4 with a probability of $\frac{15}{16} \cdot \frac{1}{16}$.
- In Step 17, U is reduced by a factor of 8 with a probability of $\frac{15}{16} \cdot \frac{1}{32}$.

Note that in Steps 11 and 18, q can be any integer between -8 and 7. Therefore, roughly speaking, it can be assumed that the addition of qV in those steps is nearly zero on average. According to the simulation, the average value of q is approximately -0.11 because of the unequal probabilities of even and odd qs. However, because q and $-q$ occur with the same probability, each pair of q and $-q$ will be canceled out and only $q = -8$ will remain. Because the probability of an even q is less than that of an odd q, the quotient is less than -0.5. Therefore, it is reasonable to assume that the reduction factor is 16 in those steps.

The expected division factor of U in each loop is calculated to be 9.027. In other words, on average, the magnitude of $|U_i|$ will be 9.027 times smaller than the magnitude of $|U_{i-1}|$ in the previous loop. Thus, we can say that 3.174 bits of U are processed in each loop. Therefore, if P is an n-bit value, we can conclude that we need at least $n/3.174$ cycles in order to reach $|U| = 1$.

Similarly, we can calculate the average number of processed bits of V in each loop of Algorithm 7 as 3.164 bits. Because $A < P < 2^n$, we can say that in order to

reach $|V| = 1$, we need approximately $n/3.164$ cycles on average. Note that $|U|$ and $|V|$ are processed simultaneously. Therefore, in the worst case scenario, the number of cycles required to obtain the Montgomery inverse is less than $n/3.164 = 0.316n$. It should be emphasized that there is no need for both $|U|$ and $|V|$ to reach 1. When one of $|U|$ or $|V|$ is equal to 1, looping can be stopped.

B. EXPECTED INCREMENT OF K FOR EACH LOOP

In Algorithm 5, the parameter k is the number of cycles for Phase 1, as well as the exponent of two, which is multiplied by the inverse $A^{-1}2^k \pmod P$. In Algorithm 7, even though the number of cycles required to finish Phase 1 is less than that in Algorithms 5 and 6, the parameter k is greater than that in previous algorithms. In this subsection, we estimate the expected value of k in the proposed algorithm for a random U and V.

Let k_i and Pr(Step i) be the number of increments of k in Step i and the probability of executing Step i for a given U and V in Algorithm 7, respectively. Then, we can evaluate each probability of Pr(Step i) and value k_i as:

- In Step 4, $\Pr(\text{Step } 4) \cdot k_4 = \frac{1}{16} \cdot 4$.
- In Step 5, $\Pr(\text{Step } 5) \cdot k_5 = \frac{1}{16} \cdot 4$.
- In Steps 7–10, $\sum_{i=7}^{10} \Pr(\text{Step } i) \cdot k_i = (\frac{15}{16})^2 \cdot \frac{1}{2} \cdot [\frac{4}{2} + \frac{5}{4} + \frac{6}{8} + \frac{7}{16}]$.
- In Steps 14–17, $\sum_{i=14}^{17} \Pr(\text{Step } i) \cdot k_i = (\frac{15}{16})^2 \cdot \frac{1}{2} \cdot [\frac{4}{2} + \frac{5}{4} + \frac{6}{8} + \frac{7}{16}]$.

Thus, we obtain that the parameter k increases by 4.38 on average in each loop of Algorithm 7. In other words, the value of k will increase by 4.38 in each loop. However, it was noted in Section V-A that the number of processed bits in a loop is approximately 3.174 in the case of $|U| = 1$, while it is one bit in the case of Kaliski's original algorithm. Therefore, on average, the value of k will be almost 1.38 times greater than that in Algorithm 5.

TABLE 2. The average number of cycles for each algorithm over 20,000 samples.

Number	Kaliski	[17]		Proposed	
		k	# of cycles	k	# of cycles
100,053,193	38.3	38.3	30.6	53.1	11.6
102,500,551	38.4	38.4	30.7	53.2	11.6
117,950,089	38.7	38.7	30.9	53.5	11.7
100,053,193	39.0	39.0	31.0	53.8	11.7
100,053,193	39.2	39.2	31.2	54.2	11.8

Table 2 lists the average numbers of cycles in Phase 1 of Algorithms 5–7. For this simulation, we run an straightforward implementation on a PC with AMD Ryzen^{TM} Threadripper^{TM} 1950X running at 3.40GHz (Turbo Boost is disabled). This machine has 32GB of RAM and runs Ubuntu 16.04.5 LTS.

From the table, we can see that the size of k in the proposed algorithm is approximately 1.382 times greater than that in Algorithms 5 and 6. It should also be noted that the proposed algorithm is at least three times faster than Phase 1 of Algorithm 5.

C. DISTRIBUTION OF K AND ITS BOUNDS

Fig. 2 presents the distributions of the parameter k for Algorithm 5 at the end of Phase 1 and Algorithm 7. For comparison, we include an additional distribution of k for a modification of Algorithm 7, which continues looping until both $|U| \leq 1$ $|V| \leq 1$ are satisfied instead of either $|U| = 1$ or $|V| = 1$. In this simulation, we calculated 2,000,000 distinct inverses for the modulus $P = 100,053,193$. As a result, we obtained that the expected of k in Algorithm 7 is approximately $53 \cong 2n$, while that in Phase 1 of Algorithm 5 is approximately $38.3 \cong 1.42n$. Therefore, in Phase 2 of the Montgomery inverse algorithm, the proposed algorithm is expected to achieve much better performance than Kaliski's algorithm because the parameter k in Algorithm 7 is already very close to $2n$ on average. It should also be noted that the expected value of k for the modified algorithm is 57.1. Therefore, it requires more steps to achieve both $|U| \leq 1$ and $|V| \leq 1$.

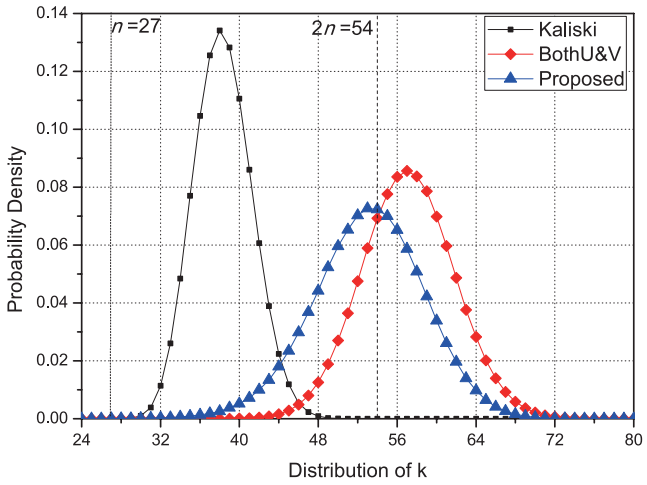

FIGURE 2. Distribution of k where $P = 100,053,193$ (27-bit digit) with 2,000,000 samples.

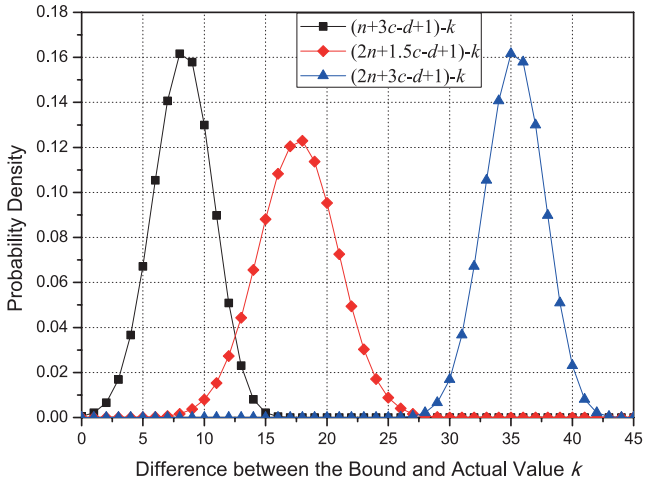

FIGURE 3. Difference between the upper bound and actual value of k.

Fig. 3 presents the difference between the upper bound from Theorem 7 and the actual value k from a simulation using the same parameters as those in Fig. 2. From the

simulation, the expected difference is approximately 35.2, which is even greater than the bit size $n = 27$ of P. Therefore, the upper bound is closer to $n+3c-d+1$ than $2n+3c-d+1$. However, there are a few cases in which k is greater than $n + 3c - d + 1$.

Based on the simulation, we can say that the bound $2n + 1.5c-d+1$ is a relatively tight bound of k. The expected value of the difference $(2n+1.5c-d+1)-k$ is approximately 17.6. This means that it is possible to use a factor of three instead of a factor of eight in (11).

D. COMPLEXITY COMPARISON

In this subsection, we will compare the complexity of Algorithm 6 proposed by Gutub, Tenca, Savaş, and Koç and that of Algorithm 7 proposed in this paper. Let M, A, and S be the n-bit $\times$ 4-bit multiplication, n-bit addition, and l-bit left/right shifts operations, respectively. In Algorithm 6, the computation is $3A + 2S$ with high probability in an iteration, while the computation is $2M + 3A + 3S$ and a 4-bit quotient calculation, Q, in an iteration of Algorithm 7. Notice that the 4-bit quotient calculation is equivalent a 3-input XOR operation according to Proposition 1. Also note that M corresponds approximately to at most $3A$ because $15B = 8B + 4B + 2B + B$ is the worst case. However, the number of iterations of the proposed algorithm is about 1/3 of that of Algorithm 6. If the number of iteration is R, then expected computations are $3R \times (3A+2S) = 9RA+6RS$ and $R\times(2M+3A+3S+Q) = 9RA+3RS+RQ$ in Algorithm 6 and Algorithm 7, respectively. Therefore, the overall complexity of each of the two algorithms is almost the same.

VI. CONCLUSION

We have proposed an algorithm for calculating an inverse modulo 2^k without using integer division by modifying the Arazi-Qi algorithm. This algorithm has been generalized to a scalable form to compute the inverse modulo 2^{kl} by iteratively applying lower k-bit inverses. Because of this scalability, it is possible to cost-effectively implement the generalized algorithm in hardware. By using a relatively small processing unit, we can calculate a larger inverse. Furthermore, we have derived the direct relationship between an odd integer P and its inverse P^{-1}. Therefore, we can calculate the lower k-bit inverses for the generalized algorithm in a short time by using this direct relationship. This makes the generalized algorithm more feasible.

As a second major contribution, by using this direct relationship, we proposed a multi-bit Montgomery inverse algorithm that is at least three times faster on average than Kaliski's original Montgomery inverse algorithm. The faster calculation of a Montgomery inverse is important for the key cryptosystems, such as elliptic curve cryptosystems, because the choice of coordinate systems for cryptosystems is heavily dependent on the relative speed of the modular inverse operation. For future work, we will derive tighter bounds for the parameter k. Although we derived a loose bound for k in this paper, we have encountered the possibility of much tighter bounds, which should be theoretically proved in future work.

APPENDIX A
DERIVATION OF DIRECT RELATIONSHIP

Proof: The first output x_0 is clearly equal to 1. We also have $S_0 = (\times, p_{k-1}, p_{k-2}, \cdots, p_1)_2$. Here $\times$ indicates an arbitrary irrelevant bit. If we add p_1 P to S_0, then the result has a zero LSB. That is to say, the second bit is given as $x_1 = p_1$ and we have:

$$\begin{aligned}
S_1 &= S_0 + p_1P \\
&= [\cdots + (p_7 2^6 + p_6 2^5 + \cdots + p_3 2^2 + p_2 2 + p_1)] \\
&\quad + [\cdots + (p_6 p_1 2^6 + \cdots + p_2 p_1 2^2 + p_1^2 2 + p_1)] \\
&= \cdots + (p_7 + p_6 p_1)2^6 + \cdots + (p_3 + p_2 p_1)2^2 \\
&\quad + (p_2 + p_1)2 + 2p_1 \\
&= \cdots + (p^7 + p_6 p_1)2^6 + \cdots + (p_3 + p_2 p_1 + p_1)2^2 \\
&\quad + p_2 2 + 0.
\end{aligned}$$

Therefore, it is clear that the third bit is determined as $x_2 = p_2$. Similarly, we have:

$$\begin{aligned}
S_2 &= S_1/2 + p_2P \\
&= [\cdots + (p_7 + p_6 p_1)2^5 + \cdots + (p_3 + p_2 p_1 + p_1)2 + p_2] \\
&\quad + [\cdots + p_5 p_2 2^5 + \cdots + p_3 p_2 2^3 + p_2^2 2^2 + p_1 p_2 2 + p_2] \\
&= \cdots + (p_7 + p_6 p_1 + p_5 p_2)2^5 + \cdots + (p_3 + p_2 + p_1)2 \\
&\quad + 0.
\end{aligned}$$

The fourth bit can be determined as $x_3 = p_3 + p_2 + p_1$. In the next round, we have:

$$\begin{aligned}
S_3 &= S_2/2 + (p_3 + p_2 + p_1)P \\
&= [\cdots + (p_7 + p_6 p_1 + p_5 p_2)2^4 + \cdots + (p_3 + p_2 + p_1)] \\
&\quad + [p_4(p_3 + p_2 + p_1)2^4 + \cdots + (p_3 + p_2 + p_1)] \\
&= \cdots + (p_7 + p_6 p_1 + p_5 p_2 + p_4 p_3 + p_4 p_2 + p_4 p_1 + p_3 p_2)2^4 \\
&\quad + (p_6 + p_5 p_1 + p_4 p_2 + p_3 + p_3 p_1 + p_2 p_1 + p_2)2^3 \\
&\quad + (p_5 + p_4\, p_1 + p_3 p_1 + p_1)2^2 + (p_4 + p_3)2 + 0.
\end{aligned}$$

The fifth bit is given as $x_4 = p_4 + p_3$. Similarly we have:

$$\begin{aligned}
S_4 &= S_3/2 + (p_4 + p_3)P \\
&= [\cdots + (p_7 + p_6 p_1 + p_5 p_2 + p_4 p_3 + p_4 p_2 + p_4 p_1 + p_3 p_2)2^3 \\
&\quad + \cdots + (p_5 + p_4 p_1 + p_3 p_1 + p_1)2 + (p_4 + p_3)] \\
&\quad + [\cdots + p_2(p_4 + p_3)2^2 + p_1(p_4 + p_3)2 + (p_4 + p_3)] \\
&= \cdots + (p_7 + p_6 p_1 + p_5 p_2 + p_4 p_1 + p_3 p_2 + p_3 p_1 + p_3)2^3 \\
&\quad + (p_6 + p_5 p_1 + p_3 + p_2 + p_2 p_1 + p_3 p_2 + p_4 p_1)2^2 \\
&\quad + (p_5 + p_4 + p_3 + p_1)2 + 0.
\end{aligned}$$

The sixth bit is given as $x_5 = p_5 + p_4 + p_3 + p_1$. Again, the next round is given as:

$$\begin{aligned}
S_5 &= S_4/2 + (p_5 + p_4 + p_3 + p_1)P \\
&= [\cdots + (p_7 + p_6 p_1 + p_5 p_2 + p_4 p_1 + p_3 p_2 + p_3 p_1 + p_3)2^2 \\
&\quad + (p_6 + p_5 p_1 + p_3 + p_2 + p_2 p_1 + p_3 p_2 + p_4 p_1)2 \\
&\quad + l(p_5 + p_4 + p_3 + p_1)] + [\cdots + p_2(p_5 + p_4 + p_3 + p_1)2^2 \\
&\quad + p_1(p_5 + p_4 + p_3 + p_1)2 + (p_5 + p_4 + p_3 + p_1)] \\
&= \cdots + (p_7 + p_6 p_1 + p_5 p_1 + p_4 p_2 + p_3 p_1 + p_2 p_1 + p_1)2^2 \\
&\quad + (p_6 + p_3 p_2 + p_3 p_1 + p_2 p_1 + p_5 + p_4 + p_2)2 + 0.
\end{aligned}$$

The seventh bit is given as $x_6 = p_6 + p_3\,p_2 + p_3\,p_1 + p_2\,p_1 + p_5 + p_4 + p_2$. Finally, the eighth bit can be determined in the next round as:

$$\begin{aligned} S_6 &= S_5/2 + (p_6 + p_3p_2 + p_3p_1 + p_2p_1 + p_5 + p_4 + p_2)P \\ &= [\cdots + (p_7 + p_6p_1 + p_5p_1 + p_4p_2 + p_3p_1 + p_2p_1 + p_1)2 \\ &\quad + (p_6 + p_3p_2 + p_3p_1 + p_2p_1 + p_5 + p_4 + p_2)] \\ &\quad + [\cdots + p_1(p_6 + p_3p_2 + p_3p_1 + p_2p_1 + p_5 + p_4 + p_2)2 \\ &\quad + (p_6 + p_3p_2 + p_3p_1 + p_2p_1 + p_5 + p_4 + p_2)] \\ &= \cdots + (p_7 + p_6 + p_5 + p_4 + p_3p_2p_1 \\ &\quad + (p_4 + p_3 + 1)(p_2 + p_1))2 + 0. \end{aligned}$$

This results in $x_7 = p_7 + p_6 + p_5 + p_4 + p_3p_2p_1 + (p_4 + p_3 + 1)(p_2 + p_1)$.

□

APPENDIX B
DIRECT INVERSE RELATIONSHIP FOR THE BINARY EXTENSION FIELD

It is possible to derive the lower 8-bit modular inverse relationship for the binary extension field $GF(2^m)$ as shown in the following corollary.

Corollary 8: The lower 8-bit modular inverse $Y(x) = (y_7, y_6, \ldots, y_1, y_0)$ relationship for the primitive irreducible polynomial $P(x)$ of $GF(2^m)$ is given as:

$$\begin{aligned} &y_0 = 1, \quad y_1 = p_1,\ y_2 = p_2 \oplus p_1, \\ &y_3 = p_3 \oplus p_1, \quad y_4 = p_4 \oplus (p_1 + p_2), \\ &y_5 = p_5 \oplus p_1(p_3 \oplus \overline{p_2}), \\ &y_6 = p_6 \oplus p_3 \oplus p_1p_4 \oplus (p_1 + p_2), \\ &y_7 = p_7 \oplus p_7 \oplus p_1\overline{p_5} \oplus p_2\,p_3 \end{aligned}$$

where $\oplus$ and $+$ denote exclusive-OR (XOR) and OR operations, respectively.

REFERENCES

[1] Ç. K. Koç, "High-speed RSA implementation," RSA Laboratories, Bedford, MA, USA, Tech. Rep. TR 201, Nov. 1994.
[2] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, Jun. 1996.
[3] E. Savas and Ç. K. Koç, "Montgomery inversion," *J. Cryptograph. Eng.*, vol. 8, no. 3, pp. 201–210, Sep. 2018.
[4] C. K. Koc, "A new algorithm for inversion mod p^k," Cryptol. ePrint Arch., Tech. Rep. 2017/411, 2017. [Online]. Available: https://eprint.iacr.org/2017/411
[5] S. Vollala, K. Geetha, and N. Ramasubramanian, "Efficient modular exponential algorithms compatible with hardware implementation of public-key cryptography," *Secur. Commun. Netw.*, vol. 9, no. 16, pp. 3105–3115, Nov. 2016.
[6] M. Joye and P. Paillier, "GCD-free algorithms for computing modular inverse," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 2779, C. D. Walter, Ç. K. Koç, and C. Paar, Eds. Berling, Germany: Springer, 2003, pp. 243–253.
[7] A. F. Tenca, G. Todorov, and Ç. K. Koç, "High-radix design of a scalable modular multiplier," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 2162, Ç. K. Koç, D. Naccache, C. Paar, Eds. Berling, Germany: Springer, 2001, pp. 185–201.
[8] B. S. Kaliski, "The Montgomery inverse and its applications," *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 1064–1065, Aug. 1995.
[9] E. Savas and Ç. K. Koç, "The Montgomery modular inverse-revisited," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 763–766, Jul. 2000.
[10] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 1514. Berling, Germany: Springer, 1998, pp. 51–65.
[11] H. Aigner, H. Bock, M. Hütter, and J. Wolkerstorfer, "A low-cost ECC coprocessor for smartcards," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 3156. Berling, Germany: Springer, 2004, pp. 107–118.
[12] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery, "Trading inversions for multiplications in elliptic curve cryptography," *Designs, Codes Cryptogr.*, vol. 39, no. 2, pp. 189–206, May 2006.
[13] D. W. Matula, A. Fit-Florea, and M. A. Thornton, "Table lookup structures for multiplicative inverses modulo 2^k," in *Proc. 17th IEEE Symp. Comput. Arithmetic (ARITH)*, Jun. 2005, pp. 156–163.
[14] S. R. Dussé and B. S. Kaliski, Jr., "A cryptographic library for the motorola DSP56000," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 473. Berlin, Germany: Springer, 1990, pp. 230–244.
[15] O. Arazi and H. Qi, "On calculating multiplicative inverses modulo 2^m," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1435–1438, Oct. 2008.
[16] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Apr. 1985.
[17] A. A.-A. Gutub, A. F. Tenca, E. Savaş, and R. K. Koç, "Scalable and unified hardware to compute Montgomery inverse in GF(p) and GF(2^n)," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 2779, B. S. Kaliski, Jr., Ed. 2003, pp. 484–499.
[18] S. S. Erdem, T. Yanık, and A. Çelebi, "A general digit-serial architecture for Montgomery modular multiplication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1658–1668, May 2017.
[19] E. Savas, M. Naseer, A. A. A. Gutub, and C. K. Koc, "Efficient unified Montgomery inversion with multibit shifting," *IEE Proc.-Comput. Digit. Techn.*, vol. 152, no. 4, pp. 489–498, Jul. 2005.

YOUNG-SIK KIM (M'10) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University in 2001, 2003, and 2007, respectively. He joined the Semiconductor Division, Samsung Electronics, where he carried out research and development of security hardware IPs for various embedded systems, including modular exponentiation hardware accelerator (called *Tornado 2MX2*) for RSA and elliptic curve cryptography in smart card products and mobile application processors of Samsung Electronics, until 2010. He is currently an Associate Professor with Chosun University, Gwangju, South Korea. He is also a submitter for two candidate algorithms (McNie and pqsigRM) in the first round for the NIST Post Quantum Cryptography Standardization. His research interests include post-quantum cryptography, IoT security, physical layer security, data hiding, channel coding, and signal design. He selected as one of the 2025's 100 Best Technology Leaders (for Crypto-Systems) by the National Academy of Engineering of Korea.