PAPER

# Opcount: A Pseudo-Code Performance Estimation System for Pairing-Based Cryptography*

Masayuki ABE[†,††], *Senior Member*, Fumitaka HOSHINO[†,†††], *Nonmember*, *and* Miyako OHKUBO[††††a)], *Member*

**SUMMARY** We propose a simple framework for evaluating the performance of pairing-based cryptographic schemes for various types of curves and parameter settings. The framework, which we call 'Opcount', enables the selection of an appropriate curve and parameters by estimating the performance of a cryptographic scheme from a pseudo-code describing the cryptographic scheme and an implementation-information database that records the performance of basic operations in curves targeted for evaluation. We apply Opcount to evaluate and compare the computational efficiency of several structure-preserving signature schemes that involve tens of pairing products in their signature verification. In addition to showing the usefulness of Opcount, our experiments also reveal the overlooked importance of taking account of the properties of underlying curves when optimizing computations and demonstrate the impact of tight security reductions.

***key words:*** *pairing, performance estimation, benchmark, tight security*

## 1. Introduction

### (1) Background

The security and efficiency of cryptographic schemes over pairing groups depend on the choice of the parameters for the underlying elliptic curves. The most widely used curves are so-called Barreto-Naehrig (BN) curves, which balance the security and efficiency of the groups associated with the curves at the 128-bit security level. Owing to recent progress in the analysis of the discrete logarithm problem (DLP) over finite fields [11] proposed by Kim and Basbulescu in 2016, the balance of the BN curves is no longer optimal and other types of curves have again come under scrutiny. Appropriate curve types and parameters can differ according to the security and performance required for the cryptographic scheme under consideration. Since group expressions, the speed of operations, and other factors in relation to pairing groups differ greatly according to specific parameters, the platform, and the implemented high-speed technique, the only way of selecting a security level and parameters to optimize the performance of a certain cryptographic scheme or maintain the performance within an allowable range at present is to actually implement the scheme and make comparisons. In reality, however, implementations of pairing libraries are often optimized and hence limited to specific curves and parameters, and the evaluation of the implementation efficiency of an individual cryptographic scheme for a variety of curves and parameters might be difficult.

Papers on cryptographic theory often make rough estimates of performance either in terms of the number of time dominant operations are performed or by presenting the results of an implementation for only specific parameters and platforms, which makes it difficult to compare the performance of different schemes. In addition, since there are a variety of algorithms making up each scheme, the dominant operations in these algorithms may differ. Ultimately, these algorithms must be compared with sufficient accuracy taking into account the cost of a variety of operations. For example, in the verification of Groth-Sahai proofs [6], tens of pairing-product equations are evaluated, and whether such verification can be speeded up by introducing batch verification depends on the parameters and variables included in the equations. The algorithm for converting to a batch, and the number and speed ratios of all related operations must be determined. The determination of these factors is therefore not a trivial task. When comparing various algorithms in the development process of a cryptographic scheme or investigating whether the introduction of a new algorithm has any effect, it would be desirable if the performance of the cryptographic scheme could be estimated easily and accurately.

It will be even more difficult to compare schemes when the security loss in their security proofs is rigorously taken into consideration. Suppose that we compare the performance of cryptographic schemes A and B based on the same hardness assumption. Scheme A, although compact with a small number of operations, has a 60-bit reduction loss, and scheme B, although requiring more operations, exhibits a tight reduction. To ensure the same degree of security, we will have to implement scheme A with a 192-bit security parameter and scheme B with a 128-bit security parameter. Successively implementing these cryptographic schemes for different parameters, security levels, or platforms and comparing the performance after maximum tuning requires a high level of skill in the implementation of cryptographic schemes as well as considerable.

### (2) Our Contribution

In this paper, we propose a simple framework, which we call

Manuscript received December 11, 2018.
Manuscript revised April 19, 2019.

[†] The authors are with Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

[††] The author is with Graduate School of Informatics, Kyoto University, Kyoto-shi, 606-8501 Japan.

[†††] The author is with School of Computing, Tokyo Institute of Technology, Tokyo, 152-8550 Japan.

[††††] The author is with Security Fundamentals Laboratory, CSR, NICT, Koganei-shi, 184-8795 Japan.

[*] This work was supported by JSPS KAKENHI Grant Number JP16K00027.

a) E-mail: m.ohkubo@nict.go.jp

DOI: 10.1587/transfun.E102.A.1285

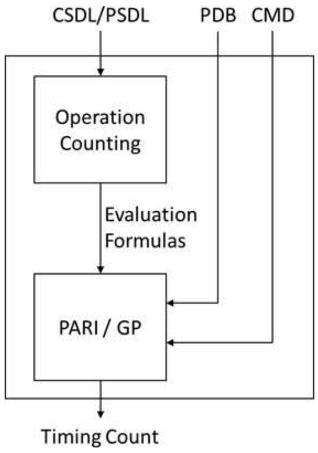

**Fig. 1** Data flow of performance estimaition with Opcount.

'Opcount', for evaluating the performance of cryptographic schemes over pairing groups and assess its effectiveness and usefulness. Opcount takes three objects: a description of a cryptographic scheme written in a proprietary pseudo-code (CSDL/PSDL), a performance database (PDB) that provides the timing related to basic pairing-group operations for specific parameters on a specific platform, and a script describing the output format (CMD). It evaluates and outputs values of specified parameters, function timing results, and so forth. Opcount is divided into two steps: an "operation-counting step" that counts the number of basic operations included in the considered cryptographic scheme, and an "evaluation step" that substitutes timing values obtained from the PDB into evaluation formulas, calculates specific values, and outputs results according to the specified script. The evaluation step uses PARI/GP systems. Figure 1 shows a schematic diagram of Opcount.

Since the scheme description and basic operation performance are handled separately, they can be used in different combinations to compare and investigate the performance for a variety of schemes and parameter settings. In addition, the CMD script enables the flexible selection of evaluation targets from multiple structured functions and variable values described in a pseudo-code. The PDB is based on existing pairing-library benchmarks, but it may also be based on theoretical estimations.

Opcount system supports two similar languages for describing a scheme in consideration. One is called C-like Scheme Description Language (CSDL) which mostly follows C as its name suggests. CSDL allows one to describe a scheme in a simple fashion by focusing on the main part of the algorithm in the scheme. One can, for instance, omit details of a low-level function if it is known in advance that its impact to overall performance is negligible. While the simplicity of CSDL is an advantage, it is useful if correctness of the scheme description is assured when all details matter. For that purpose, we provide an alternative way to describe schemes that we call Python-like Scheme Description Language (PSDL). It follows the language for Sage [13]. By providing concrete parameters, one can run the code described in PSDL over Sage to check the correctness.

In the operation counting step, an actual program that processes the given pseudo-code is different depending on the language the code is written in. While pseudo-codes written in CSDL are processed directly by a proprietary software, those written in PSDL are first given to Sage that outputs internal codes in Python. Then a proprietary program written also by Python transforms the internal codes into evaluation formulas. Support of two description languages should meet variety of demands while balancing simplicity and accuracy. Nevertheless, CSDL and PSDL are very similar, and whichever language is used the output from opcount will be the same for the same scheme. Thus, in the rest of this paper, we focus on CSDL and give notes about PSDL if necessary.

We perform the following evaluation experiments on the CSDL version to assess the effectiveness and usefulness of Opcount.

**Accuracy test:** This experiment compares timing-evaluation values obtained by Opcount with measured timing values in an actual implementation. As a measurement target, we implement a structure-preserving signature scheme in [1] that includes many pairing operations using the pairing curve library of Kiyomura et al. [12] at the 256-bit security level. We then compare measured timing values for the functions of key generation, signature generation, and signature verification with evaluation values estimated by Opcount.

**Comparison of different computation methods:** In this experiment, we use Opcount to compare different strategies of batch verification of pairing-product equations. Verifying several pairing product equations at the same time can be done efficiently by combining the equations into a single equation in such a way that the original equations hold if the combined equation holds. A typical example is a verification of a structure-preserving signature that evaluates a handful of pairing-product equations. The way of combining multiple pairing-product equations into one equation is not unique. Depending on the form of the equations and the involved variables the optimal strategy will vary. It is indeed not a trivial task to determine which strategy is the most efficient. We describe two typical strategies applied to two structure-preserving signature schemes using CSDL, and observe how their computational cost differs depends on the underlying curves by using Opcount.

**Comparison of schemes with different parameters:** In this experiment, we compare several structure-preserving signature schemes in two ways. First we compare six recent schemes in the literature under the same 128-bit security parameter. Their signatures consist of different numbers of group elements and it is expected that the smaller the signature size, the more efficient their generation and verification of the signatures will be. To examine whether this hypothesis is correct, we describe these schemes in CSDL and compare their timings by using Opcount. We then increase the rigor of this comparison by taking the reduction cost into account. For the case of $Q$ signing queries, we estimate the timings of schemes having a reduction cost of $O(Q^2)$ or $O(Q \log Q)$ at a higher, 192-bit, security level to offset the large security loss. This is carried out simply by replacing the PDB given to Opcount. We then compare the results between these two security parameters.

In short, we evaluate the effectiveness of this framework by the above accuracy test and demonstrate the flexibility and usefulness of the framework by the above effect-measurement and efficiency-comparison experiments, which reflect the ease of comparing the effects of changing the algorithms.

Furthermore, the results of the experiments are of independent interest. The second experiment reveals the overlooked importance of taking account of properties of underlying curves when optimizing computations via batch verification. The third experiment demonstrates the impact of tight security reductions by a direct comparison of schemes with different security parameters.

All documentations and data including source codes of Opcount, performance data files, and scripts for experiments are available at:
`https://github.com/security-kouza/opcount/`.

## 2. Pseudo-Code

We design CSDL to allow simple and easily understandable descriptions of cryptographic schemes over bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ of order $p$. Variables that represent group elements are assigned to types 'group0', 'group1', and 'target', respectively. The security parameter and group order are expressed as global values and specific values are used in the evaluation step. (Consequently, cryptographic schemes that can be described are limited to those that handle one pairing group.) There is also an 'integer' type for expressing integers and a 'list' type for specifying a sequence of variables of any type. Integers, whether they be literals or variables, are treated as $\mathbb{Z}_p$ elements. Group operations conform to the notation of multiplicative groups; Multiplication $\star$, exponentiation $\cdot$, and pairing $e(\cdot,\cdot)$ can be described. Our scheme description languages support simple fixed number of 'for' loops and even data-dependent branches such as 'if-then-else' when their average occurrence can be evaluated in advance. Data dependent branches whose average behaviour is hard to predict in advance cannot be described. To give a simple example, Fig. 2 shows the decoding function of Linear Encryption [4] described in CSDL.

This pseudo-code is processed by a program giving the evaluation formula shown in Fig. 3.

Each term on the right side of this formula expresses the cost (timing) incurred by a basic operation (or action). Variables in each term are assigned specific values in the evaluation step and the total cost of the entire function is determined by call_Linear_Dec$_0$. GROUP0_batch$(a, b)$ is a variable expressing the combined cost of multi-base exponentiation consisting of 'a' bases and 'b' multiplication operations. The PDB described in the next section includes information on the type of algorithm used to calculate this variable by other basic operations plus information on actual timing values.

```
group0 Linear_Dec_0(list sk, list ct)
{
    integer r1, r2;
    integer x1, x2;
    group0  u1, u2, u3, M;

    (x1, x2) = sk;
    (u1, u2, u3) = ct;
    M = u3 * u1^(-x1) * u2^(-x2);

    return M;
}
```

**Fig. 2** Pseudo-code of the decryption function of Linear Encrypiton in CSDL.

```
call_Linear_Dec_0 = (1)*GROUP0_assign+(1)*GROUP0_batch
    (2*1,1*1)+(2)*INTEGER_uminus+(2)*LIST_LITERAL_assign
    +(1)*LIST_VARIABLE_assign ;
```

**Fig. 3** Internal evaluation formula.

## 3. Performance Database

The PDB is a database of information on the timing of basic operations related to pairing groups. While it is assumed that timing values are benchmark values found in existing pairing libraries, they may also be theoretically determined evaluation values. The PDB gives specific values for parameters such as the pairing group order and embedding degree, and for each group (e.g., $\mathbb{G}_1$), it gives specific values or evaluation formulas for the multiplication cost 'GROUP0_mul', exponentiation cost 'GROUP0_pow', and low-order operations. For higher-order functions, it also gives the Miller-Loop cost 'Miller_Loop' and final-exponentiation cost 'Final_Exponentiation'. The PDB also describes the 'pairing batch', the cost of the aforementioned pairing product, in the form of an evaluation formula that reflects the provided APIs and calculation methods. An example of a PDB description is shown in Fig. 4.

For this study, we created three PDBs with different properties.

(1) Kiyo17 PDB:

This PDB is based on timing data [12] presented by Kiyomura et al. at ACNS2017. Their study implemented the BLS-24, KSS-32, KSS-36, BLS-42, and BLS-48 curves, selecting the 256-bit security level parameter for each while taking exTNFS into account. To measure low-order operation costs not presented in their study and improve the accuracy of Opcount, we conducted futher for the same library as in [12] on a different platform (Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20 GHz stepping 01, Linux 2.6 x86_64 (CentOS 6.8), gcc version 6.3.0 (GCC)) and converted the timing data that we obtained into a PDB. The measured data is listed in Table 1. The relative behavior of this timing data

```
\\ Common Parameters
CURVE  = "KSS-32";
Sep    = 256;  \\ security parameter
Order  = 738;  \\ Group order in bits
EmbDeg = 32;   \\ Embedding degree
ExtDeg = 8;    \\ Extension Degree

\\ Measured functions
GROUP0_pow           = 7.399618 * Mclk;
GROUP1_pow           = 41.328962 * Mclk;
TARGET_pow           = 49.085291 * Mclk;
Miller_Loop          = 26.057327 * Mclk;
Final_Exponentiation = 156.147854 * Mclk;
GROUP0_mul           = 0.017542 * Mclk;
GROUP0_square        = 0.009950 * Mclk;
GROUP1_mul           = 0.129000 * Mclk;
GROUP1_square        = 0.147782 * Mclk;
TARGET_mul           = 0.158917 * Mclk;
TARGET_square        = 0.116364 * Mclk;
TARGET_inv           = 0.314343 * Mclk;
GROUP0_batch(n,m) = GROUP0_pow * n + GROUP0_mul * (max(0,n-1) + m);
GROUP1_batch(n,m) = GROUP1_pow * n + GROUP1_mul * (max(0,n-1) + m);
pairing_batch(n)  = n * Miller_Loop + Final_Exponentiation + (n-1) * TARGET_mul;
```

**Fig. 4** Example of performance database (PDB).

**Table 1** Timings on Xeon with Kiyo17 implementation.

| Operation | | Timing (in Mclk) | | | | |
|---|---|---|---|---|---|---|
| | | BLS-24 | KSS-32 | KSS-36 | BLS-42 | BLS-48 |
| Pairing | ML | 42.00 | 26.05 | 29.97 | 30.52 | 16.78 |
| | FE | 68.01 | 156.14 | 114.79 | 82.15 | 77.59 |
| | Total | 110.01 | 182.19 | 144.76 | 112.67 | 94.37 |
| Scalar mult. in $\mathbb{G}_1$ | | 10.43 | 7.39 | 5.78 | 3.70 | 3.34 |
| Scalar mult. in $\mathbb{G}_2$ | | 30.92 | 41.32 | 28.73 | 39.34 | 21.18 |
| Exponentiation in $\mathbb{G}_t$ | | 52.58 | 49.08 | 58.28 | 43.94 | 49.56 |

among these curves and operations is the same as that of the benchmark data of [12], but the speed was improved by about 20% simply due to the use of a different environment.

(2) BD17 PDB:

This PDB estimates the timing of pairing-related operations in terms of the number of 16-bit × 16-bit multiplications at the 128- and 192-bit security levels in accordance with Barbulescu and Doquesme [3]. It selects the four curve parameters of BN, BLS-12, KSS-16, and KSS-18 at the 128-bit security level and the two curve parameters of KSS-18 and BLS-24 at the 192-bit security level. Khandaker et al. conducted implementations for the above BN, BLS-12, and KSS-16 curves in [9] and showed that nearly the same comparison results as in [3] could be obtained. In particular they introduced, a new technique for KSS-16 and speeded up the Miller-Loop, and concluded that KSS-16 is superior at the 128-bit security level similarly to in [3].

(3) RV15 PDB:

This PDB is based on benchmarks for the BN curves obtained on ARM v7 by Verma [14]. It measures the timing at the 128-, 192-, and 256-bit security levels using conventional parameter settings *without* taking exTNFS into account. Although not used in the experiments presented here, this PDB can be used for currently implemented applications running on BN curves to measure the effect of simply increasing the security parameter to improve the security level on the performance.

While benchmarks in a variety of environments have been presented in the literature, there are few based on parameters that take exTNFS into account. In particular, there are no reports on implementations at the 192-bit security level, so progress in this area is anticipated.

## 4. Experiments

### 4.1 Accuracy Test

In this section, we compare timing evaluation values obtained by Opcount with measured timing values in an actual implementation. As the targets of measurement, we implemented a structure-preserving signature scheme in [1] that includes many pairing operations for five types of curves at the 256-bit security level based on the Kiyo17 library of Kiyomura et al. [12]. We then measured timing values for the functions of key generation, signature generation, and signature verification and compared them with the evaluation values obtained by Opcount. Results are listed in Table 2.

Measured values were obtained by averaging 10 random trials. The standard deviation values were 1.93 – 2.96 Mclock for key generation, 2.24 – 3.81 Mclock for signature generation, and 0.35 – 5.28 Mclock for verification. There are small negative errors in some of the results for signature generation, which mean that the measured values were faster. The reason for this is thought to be that the cache effect differed between the benchmark measurement in the PDB referenced for the evaluation value and the actually

**Table 2** Real vs. estimated timings of AHNOP signature scheme.

| Curve | Timings: Real/Estd (in Mclk) | | |
|---|---|---|---|
| | KeyGen | Sign | Verify |
| BLS-24 | 944 / 941 (0.2%) | 903 / 901 (0.1%) | 3934 / 3927 (0.1%) |
| KSS-32 | 1098 / 1094 (0.3%) | 1005 / 1008 (-0.3%) | 4152 / 4149 (0.07 %) |
| KSS-36 | 782 / 777 (0.7%) | 723 / 720 (0.4%) | 3827 / 3802 (0.6%) |
| BLS-42 | 1005 / 958 (4.6%) | 890 / 860 (3.3%) | 3356 / 3349 (0.1 %) |
| BLS-48 | 553 / 550 (0.6%) | 503 / 503 (-0.05%) | 2337 / 2334 (0.1%) |

Error(%) = (real - estd)/ real.

measured value in the signature-scheme implementation.

## 4.2 Comparison of Different Computation Methods

Batch verification of pairing-product equations is a method for efficiently verifying $n$ pairing-product equations of the form

$$1 = \prod_i e(X_{1i}, Y_{1i}), \cdots, \ 1 = \prod_i e(X_{ni}, Y_{ni})$$

at the same time by verifying a combined equation of the form

$$1 = \prod_j \prod_i e(X_{ji}, Y_{ji})^{r_j}$$

using a random $r_j$. The calculation can be made more efficient by incorporating $r_j$ further within the pairings and by merging multiple pairing products having the same variables. The computational cost of evaluating merged pairings depends on the selected form and the common variable the merge is based on. We consider two strategies for merging pairings:

- **Full batch:** A merge that minimizes the number of pairing operations. As an example, $e(X, Y_i)^{r_i}\, e(X, Y_j)^{r_j}$ are merged to $e(X, Y_i^{r_i} Y_j^{r_j})$, which can be computed by a two-base scalar multiplication in $\mathbb{G}_2$, one Miller loop, and one final exponentiation.
- $\mathbb{G}_2$ **batch:** A merge that moves the random power to the $\mathbb{G}_1$ side having a comparatively small operation cost, giving the $\mathbb{G}_2$ side a form that has as much commonality as possible. The same pairings as above are transformed to $e(X^{r_i}, Y_i)\, e(X^{r_j}, Y_j)$ in this case. This can be computed by two scalar multiplications in $\mathbb{G}_1$, two Miller loops, one multiplication in $\mathbb{G}_t$, and one final exponentiation.

In either case, we consider that the effect will depend on the variables and parameters included in the original validation equations, the calculation algorithm, and the speed ratios of all related operations.

Targeting the structure-preserving signature scheme AHNOP [1], which uses many pairing-product equations in signature verification, we used Opcount to evaluate its performance for various curve parameters and for three signature-verification algorithms, no-batch, $\mathbb{G}_2$-batch, and full-batch methods. Figures A·1, A·2, and A·3 respectively illustrate the CSDL pseudo-code for the original verification equations and the $\mathbb{G}_2$-batch, and full-batched equations, which are part of the whole description of the AHNOP scheme. Despite the numerous variables involved, the descriptions are straightforward interpretation of the algorithms that are not difficult to work with. This allows us to test several merge methods and instantly see the estimated timing. On the other hand, a drawback of working with pseudo-codes is that it is difficult to find bugs since they are never compiled or executed. Verifying correctness is, however, a general concern even for real implementations.

**Table 3** Timings of batch verification for an AHNOP signature.

| | Timing (in Mclk) | | | |
|---|---|---|---|---|
| Batch algorithm | KSS-18 | KSS-16 | BLS-12 | BN |
| None | 111.2 | 96.5 | 141.1 | 206.7 |
| $\mathbb{G}_2$-batch | 46.1 | 38.4 | 70.1 | 103.3 |
| Full-batch | 39.9 | 35.4 | 56.7 | 82.1 |

The result is of interest beyond the use of Opcount and we give a futher analysis. The main operations for each algorithm and the number of times they are performed in the case of a message consisting of 10 $\mathbb{G}_1$ group elements are as follows: for no-batch verification, 69 pairing operations, for the $\mathbb{G}_2$-batch method, 56 $\mathbb{G}_1$ group scalar multiplications and 34 pairing operations, and for the full-batch method, 38 $\mathbb{G}_1$ group scalar multiplications, 18 $\mathbb{G}_2$ group scalar multiplications, and 26 pairing operations[†]. The size of the random number $r_j$ used in the above batch methods was made uniform at 128 bits. (Depending on the curve, group order can greatly exceed $2^{128}$; thus, selecting $r_j$ in a uniformly random manner from $\mathbb{Z}_p$ significantly degrades, the efficiency. In terms of security reduction, the size of $r_j$ is a statistical parameter independent of the ability of an attacker, so $r_j$ can be as short as $\lambda$ bits for $\lambda$-bit security.) Table 3 lists the performance evaluation results for curves KSS-18, KSS-16, BLS-12, and BN, all at a 128-bit security level, as registered in BV17 PDB. These results show that the full-batch method, which merges validation equations so as to reduce the number of pairings as much as possible, is the most efficient for all curves.

We applied the same experiment to the other structure-preserving signature schemes mentioned in the next section and observed the same result; the full batch is the most

[†] We assume a naive method for computing $n$-base scalar multiplications that simply computes scalar multiplications $n$ times. BV17 PDB assumes algorithms that decrease the number of squaring operations in multibase scalar multiplications.

efficient, except for the Jutla-Ohkubo-Roy (JOR) [7] scheme. The result is shown in Table 4. Its verification is faster, with the $\mathbb{G}_2$ batch on the KSS-16 curve while the full batch is better for all other curves. The reason is that operations in the extended group $\mathbb{G}_2$ are relatively less efficient on KSS-16 than on other curves due to the larger extension degree. Hence, it is reasonable to merge pairings having the same $\mathbb{G}_2$ elements so that multibase scalar multiplication takes place in $\mathbb{G}_1$, which indeed what the $\mathbb{G}_2$-batch does.

### 4.3 Comparison of Schemes with Different Parameters

Structure-preserving signatures are often combined with a proof system that proves one's possession of a correct signature. Such a proof tends to be more efficient for smaller signatures. Moreover, smaller signatures should be generated and verified more efficiently. Considerable effort has been made to construct structure-preserving signature schemes having smaller signature size such that the Kiltz-Pan-Wee (KPW) scheme [10], Jutla-Roy (JR) scheme [8], Abe-Hofheinz-Nishimaki-Ohkubo-Pan (AHNOP) scheme [1], Jutla-Ohkubo-Roy (JOR) scheme [7], Gay-Hofheinz-Kohl-Pan (GHKP) scheme [5], and Abe-Jutla-Ohkubo-Roy (AJOR) scheme [2] are those based on the symmetric external Diffie-Hellman (SXDH) assumption. They have different signature sizes and verification equations, and if they were to be compared in terms of efficiency for the same security parameter, it is expected that the JR scheme, having the smallest signature size, will be the most efficient while the AHNOP scheme, with the largest signature size, will be the least efficient. To verify the validity of this intuition, we evaluate the key generation, signature generation, and signature verification timings of all of these schemes on the KSS-16 curve at the 128-bit security level using the BD17 PDB. The message size is fixed to ten group elements in $\mathbb{G}_1$, and the full-batch verification is applied in each scheme except for the JOR scheme, where the $\mathbb{G}_2$-batch is applied instead. The results of evaluating the performance by Opcount are shown in the upper portion of Table 5.

**Table 4** Timings of full and $\mathbb{G}_2$-batch verifications of a JOR signature.

| | Timing (in Mclk) | | | |
|---|---|---|---|---|
| Batch algorithm | KSS-18 | KSS-16 | BLS-12 | BN |
| None | 77.6 | 67.1 | 99.3 | 146.0 |
| $\mathbb{G}_2$-batch | 45.2 | 37.8 | 70.5 | 107.9 |
| Full-batch | 45.2 | 41.4 | 67.6 | 102.4 |

We next consider the reduction cost in each scheme. While AHNOP, JOR, GHKP, and AJOR feature almost tight security, i.e., they have a small reduction cost, two schemes, KPW and JR, having the smallest signature size have a larger reduction cost. Concretely, for $Q$ signature generation queries, the reduction cost of KPW is $O(Q^2)$ and that of JR is $O(Q \log Q)$. Assuming $Q = 2^{30}$, if KPW and JR, both having a reduction loss greater than 30 bits, were to ensure an equivalent or better security margin than the other schemes in 128-bit security, they would have to be implemented with a 192-bit security parameter, the next higher security level. Although 192-bit security is an excessive setting to compensate for reduction loss, the selection or implementation of a suitable curve at a security parameter between the 128-bit and 192-bit levels has not presently observed. The performance of KPW and JR at a 192-bit security level is evaluated by opcount by replacing the BLS-24 curve in the BD17 PDB with the other parameters intact. The results are listed in the lower portion of Table 5.

It can be seen from these results that even the AHNOP scheme at the 128-bit security level is two to three times faster than the JR and KPW schemes at the 192-bit security level for the operations of key generation, signature generation, and signature verification. This result demonstrates the effectiveness of tight reduction in practice, where Opcount helped clarify the impact with convincing accuracy without fully implementing every scheme.

## 5. Conclusion

We proposed Opcount as a simple framework for evaluating the performance of a pairing-based cryptographic scheme based on a pseudo-code describing the scheme and verified the effectiveness and usefulness of this framework. The estimated timings were only 5% (and even < 1% in most cases) different from the actual values. Hence, timings estimated by Opcount are sufficiently reliable for rapid performance evaluation and comparison.

We have shown that the suitability of batch verification methods can depend on the target equations and underlying curves. Opcount helps find the most suitable scheme before implementing it in full. We have also shown that tightly secure schemes having more operations in signature generation and verification can be more efficient in reality than non-tight schemes having fewer operations implemented in a higher security level to offset the security loss.

**Table 5** Estimated timings of structure-preserving signature schemes.

| Curve/Security | Scheme | Reduction Cost | $\lvert\sigma\rvert$ | Timing (in Mclk) | | |
|---|---|---|---|---|---|---|
| | | | $(\mathbb{G}_1,\mathbb{G}_2)$ | KeyGen | Sign | Verify |
| KSS-16/128 | AHNOP | $O(\lambda)$ | (13,12) | 18.1 | 14.8 | 35.4 |
| | JOR | $O(\lambda)$ | (11,6) | 23.3 | 12.5 | 37.8 |
| | GHKP | $O(\log Q)$ | (8,6) | 18.8 | 8.2 | 25.0 |
| | AJOR | $O(\log Q)$ | (6,6) | 214.8 | 10.3 | 29.1 |
| | KPW | $O(Q^2)$ | (6,1) | 13.1 | 7.0 | 20.8 |
| | JR | $O(Q \log Q)$ | (5,1) | 14.1 | 6.3 | 19.0 |
| BLS-24/192 | KPW | $O(Q^2)$ | (6,1) | 55.0 | 30.6 | 120.1 |
| | JR | $O(Q \log Q)$ | (5,1) | 59.6 | 27.9 | 109.0 |

As future works, we plan to improve the descriptive power of CSDL/PSDL and enhance the PDB by adding more diverse benchmark results.

## Acknowledgements

We would like to extend our deep appreciation to Yutaro Kiyomura of NTT WEST, Yuto Kawahara of NTT Secure Platform Laboratories and Shigeo Mitsunari of Cybozu Labs, Inc. for their cooperation in creating the PDB and implementing the scheme for comparison purposes.

### References

[1] M. Abe, D. Hofheinz, R. Nishimaki, M. Ohkubo, and J. Pan, "Compact structure-preserving signatures with almost tight security," Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, J. Katz and H. Shacham, eds., Santa Barbara, CA, USA, Aug. 2017, Proceedings, Part II, vol.10402 of Lecture Notes in Computer Science, pp.548–580, Springer, 2017.

[2] M. Abe, C.S. Jutla, M. Ohkubo, and A. Roy, "Improved (almost) tightly-secure simulation-sound QA-NIZK with applications," Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, T. Peyrin and S. D. Galbraith, eds., Brisbane, QLD, Australia, Dec. 2018, Proceedings, Part I, vol.11272 of Lecture Notes in Computer Science, pp.627–656, Springer, 2018.

[3] R. Barbulescu and S. Duquesne, "Updating key size estimations for pairings," IACR Cryptology ePrint Archive, 2017:334, 2017.

[4] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, M. K. Franklin, ed., Santa Barbara, California, USA, Aug. 2004, Proceedings, vol.3152 of Lecture Notes in Computer Science, pp.41–55, Springer, 2004.

[5] R. Gay, D. Hofheinz, L. Kohl, and J. Pan, "More efficient (almost) tightly secure structure-preserving signatures," Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April-May 2018, Proceedings, Part II, vol.10821 of Lecture Notes in Computer Science, pp.230–258, Springer, 2018.

[6] J. Groth and A. Sahai, "Efficient noninteractive proof systems for bilinear groups," SIAM J. Comput., vol.41, no.5, pp.1193–1232, 2012.

[7] C.S. Jutla, M. Ohkubo, and A. Roy, "Improved (almost) tightly-secure structure-preserving signatures," Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 2018, Proceedings, Part II, vol.10770 of Lecture Notes in Computer Science, pp.123–152, Springer, 2018.

[8] C.S. Jutla and A. Roy, "Improved structure preserving signatures under standard bilinear assumptions," Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, S. Fehr, ed., Amsterdam, The Netherlands, March 2017, Proceedings, Part II, vol.10175 of Lecture Notes in Computer Science, pp.183–209, Springer, 2017.

[9] M.A. Khandaker, Y. Nanjo, L. Ghammam, S. Duquesne, Y. Nogami, and Y. Kodera, "Efficient optimal ate pairing at 128-bit security level," Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, A. Patra and N. P. Smart, eds., Chennai, India, Dec. 2017, Proceedings, vol.10698 of Lecture Notes in Computer Science, pp.186–205, Springer, 2017.

[10] E. Kiltz, J. Pan, and H. Wee, "Structure-preserving signatures from standard assumptions, revisited," Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, R. Gennaro and M. Robshaw, eds., Santa Barbara, CA, USA, Aug. 2015, Proceedings, Part II, vol.9216 of Lecture Notes in Computer Science, pp.275–295, Springer, 2015.

[11] T. Kim and R. Barbulescu, "Extended tower number field sieve: A new complexity for the medium prime case," Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, M. Robshaw and J. Katz, eds., Santa Barbara, CA, USA, Aug. 2016, Proceedings, Part I, vol.9814 of Lecture Notes in Computer Science, pp.543–571, Springer, 2016.

[12] Y. Kiyomura, A. Inoue, Y. Kawahara, M. Yasuda, T. Takagi, and T. Kobayashi, "Secure and efficient pairing at 256-bit security level," Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, D. Gollmann, A. Miyaji, and H. Kikuchi, eds., Kanazawa, Japan, July 2017, Proceedings, vol.10355 of Lecture Notes in Computer Science, pp.59–79, Springer, 2017.

[13] T.S.D. Team, Sage tutorial release8.7, http://doc.sagemath.org/pdf/en/tutorial/SageTutorial.pdf, 2019.

[14] R. Verma, "Efficient implementations of pairing-based cryptography on embedded systems," Master's thesis, Rochester Institute of Tehcnology, Rochester New York, Dec. 2015.

## Appendix: Pseudo-Code of Batched Verification Equations

```
1==e(G, Ah) * e(Z, Gh) * e(R,Ghr) * product(10, e(Mi,Ghi)) &&
1==e(C1z00^(-1), Gh) * e(C1x00, Gh) * e(C1z10, Ah) * e(G, rhoh00) &&
1==e(C2z00^(-1), Gh) * e(C2x00, Gh) * e(C2z10, Ah) * e(Q0, rhoh00) &&
1==e(C110^(-1), Ehz0) * e(C1z00, Gh) * e(C1y00, Ehs) * e(G, rhoh01) &&
1==e(C210^(-1), Ehz0) * e(C2z00, Gh) * e(C2y00, Ehs) * e(Q0, rhoh01) &&
1==e(C1z01^(-1)*(C1z11), D1x21) * e(C1z01*C1z11^(-1), D1z21)
   * e(G, pih101) * e(thet101, Gh) &&
1==e(C2z01^(-1)*C2z11, D1x21) * e(C2z01*C2z11^(-1), D1z21)
   * e(Q1, pih101) * e(thet102, Gh) &&
1==e(C1z01^(-1)*C1z11, D2x21) * e(C1z01*C1z11^(-1), D2z21)
   * e(G, pih102) * e(thet101, Qh1) &&
1==e(C2z01^(-1)*C2z11, D2x21) * e(C2z01*C2z11^(-1), D2z21)
   * e(Q1, pih102) * e(thet102, Qh1) &&
1==e(C111^(-1), Ehz0) * e(C1z01, Gh) * e(C1y01, Ehs) * e(G, rhoh11) &&
1==e(C211^(-1), Ehz0) * e(C2z01, Gh) * e(C2y01, Ehs) * e(Q1, rhoh11) &&
1==e(C111^(-1), Ehz1) * e(C1z11, Gh) * e(C1y11, Ehs) * e(G, rhoh12) &&
1==e(C211^(-1), Ehz1) * e(C2z11, Gh) * e(C2y11, Ehs) * e(Q1, rhoh12) &&
1==e(Ez2^(-1), D111) * e(G, D1z21) * e(Et, D1y21) * e(rho13, Gh) &&
1==e(Ez2^(-1), D211) * e(G, D2z21) * e(Et, D2y21) * e(rho13, Qh1)
```

**Fig. A·1** CSDL description of original verification equations of AHNOP scheme.

```
e(bitexp(batch_quality, C1x00^(-b)*C2x00^(-c)*C1z00^(-d)*C2z00^(-f)
  *C1z01^(-p)*C2z01^(-q)*C1z11^(-r)*C2z11^(-s)*C1z00^b*C1z00^c*thet101^(-h)
  *thet102^(-j)*Z*rho13^(-t)), Gh) *
e(G,Ah) *
e(bitexp(batch_quality, G^(-w)), D2z21) *
e(bitexp(batch_quality, G^(-t)), D1z21) *
e(bitexp(batch_quality, G^(-b)*Q0^(-c)), rhoh00) *
e(bitexp(batch_quality, G^(-d)*Q0^(-f)), rhoh01) *
e(bitexp(batch_quality, G^(-h)*Q1^(-j)), pih101) *
e(bitexp(batch_quality, Q1^(-L)*G^(-k)), pih102) *
e(bitexp(batch_quality, G^(-p)*Q1^(-q)), rhoh11) *
e(bitexp(batch_quality, G^(-r)*Q1^(-s)), rhoh12) *
e(bitexp(batch_quality, thet101^(-k)*thet102^(-L)*rho13^(-w)), Qh1) *
e(bitexp(batch_quality, C1y00^(-d)*C2y00^(-f)*C1y01^(-p)*C1y11^(-r)
  *C2y01^(-q)*C2y11^(-s)), Ehs) *
e(bitexp(batch_quality, C110^d*C111^p*C211^q*C210^f), Ehz0) *
e(bitexp(batch_quality, C111^r*C211^s), Ehz1) *
e(bitexp(batch_quality, Et^(-t)), D1y21) *
e(bitexp(batch_quality, Et^(-w)), D2y21) *
e(bitexp(batch_quality, Ez2^t), D111) *
e(bitexp(batch_quality, Ez2^w), D211) *
e(bitexp(batch_quality, (_C1z01_C1z11)^h*(_C2z01_C2z11)^j), D1x21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^k*(_C2z01_C2z11)^L), D2x21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^(-h)*(_C2z01_C2z11)^(-j)), D1z21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^(-k)*(_C2z01_C2z11)^(-L)), D2z21) *
e(bitexp(batch_quality, C1x10^(-b)*C2x10^(-c)), Ah) *
e(R, Ghr) *
product(10, e(Mi,Ghi))
== 1
```

**Fig. A·2** CSDL description of the $\mathbb{G}_2$-batch verification equation of AHNOP scheme.

```
e(bitexp(batch_quality, C1x00^(-b)*C2x00^(-c)*C1z00^(-d)*C2z00^(-f)*C1z01^(-p)
  *C2z01^(-q)*C1z11^(-r)*C2z11^(-s)*C1z00^b*C1z00^c*thet101^(-h)*thet102^(-j)
  *Z*rho13^(-t)), Gh) *
e(bitexp(batch_quality, thet101^(-k)*thet102^(-L)*rho13^(-w)), Qh1) *
e(bitexp(batch_quality, C1y00^(-d)*C2y00^(-f)*C1y01^(-p)*C1y11^(-r)*C2y01^(-q)
  *C2y11^(-s)), Ehs) *
e(bitexp(batch_quality, C110^d*C111^p*C211^q*C210^f), Ehz0) *
e(bitexp(batch_quality, C111^r*C211^s), Ehz1) *
e(bitexp(batch_quality, (_C1z01_C1z11)^h*(_C2z01_C2z11)^j), D1x21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^k*(_C2z01_C2z11)^L), D2x21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^(-h)*(_C2z01_C2z11)^(-j)), D1z21) *
e(bitexp(batch_quality, (_C1z01_C1z11)^(-k)*(_C2z01_C2z11)^(-L)), D2z21) *
e(bitexp(batch_quality, C1x10^(-b)*C2x10^(-c)), Ah) *
e(G, bitexp(batch_quality, Ah*rhoh00^(-b)*rhoh01^(-d)*pih101^(-h)*rhoh11^(-p)
     *pih102^(-k)*rhoh12^(-r)*D2z21^(-w)*D1z21^(-t))) *
e(Q0, bitexp(batch_quality, rhoh00^(-c)*rhoh01^(-f))) *
e(Q1, bitexp(batch_quality, pih101^(-j)*pih102^(-L)*rhoh11^(-q)*rhoh12^(-s))) *
e(Et^(-1), bitexp(batch_quality, D1y21^t*D2y21^w)) *
e(Ez2, bitexp(batch_quality, D111^t*D211^w)) *
e(R, Ghr) *
product(10, e(Mi,Ghi))
== 1
```

**Fig. A·3** CSDL description of the full-batch verification equation of AHNOP scheme.

**Masayuki Abe** has been working for NTT (Nippon Telegraph and Telephone Corporation, Japan) since 1992. He received Ph.D. from University of Tokyo in 2002. Currently, he is a senior distinguished research scientist in NTT Secure Platform Laboratories. He served as a program chair for CT-RSA'07, ACM ASIACCS'08, and Asiacrypt'10. His research interest includes digital signatures, public-key encryption, and efficient instantiation of cryptographic protocols.

**Fumitaka Hoshino** received the B.Eng. and M.Eng. degrees from University of Tokyo, Japan, in 1996 and 1998, respectively. He is a senior scientist in NTT Secure Platform Laboratories, and a doctoral student at Tokyo Institute of Technology. His current research interests cover a wide range of topics between applied mathematics and computer science e.g. algorithmic number theory, combinatorial optimization, and cryptology.

**Miyako Ohkubo** received the B.E., and M.E. degrees from Shinshu University in 1995 and 1997, respectively, and Ph.D. degree from the Chuo University in 2004. She had worked at NTT from 1997 to 2010. She is currently a senior researcher of Security Fundamentals Laboratory in National Institute of Information and Communications Technology. She received the SCIS Paper Award in 2000. She is a member of the International Association for Cryptologic Research (IACR).