# Identification of Multiple Invalid Signatures in Pairing-Based Batched Signatures

Brian J. Matt⋆

Johns Hopkins University
Applied Physics Laboratory
Laurel, MD, 21102, USA
`brian.matt@jhuapl.edu`

**Abstract.** This paper describes new methods in pairing-based signature schemes for identifying the invalid digital signatures in a batch, after batch verification has failed. These methods efficiently identify non-trivial numbers of invalid signatures in batches of (potentially large) numbers of signatures.

Our methods use "divide-and-conquer" search to identify the invalid signatures within a batch, but prune the search tree to substantially reduce the number of pairing computations required. The methods presented in this paper require computing on average $O(w)$ products of pairings to identify $w$ invalid signatures within a batch of size $N$, compared with the $O(w(\log_2(N/w) + 1))$ [for $w < N/2$] that traditional divide-and-conquer methods require. Our methods avoid the problem of *exponential growth* in expected computational cost that affect earlier proposals which, on average, require computing $O(w)$ products of pairings.

We compare the expected performance of our batch verification methods with previously published divide-and-conquer and exponential cost methods for Cha-Cheon identity-based signatures [6]. However, our methods also apply to a number of short signature schemes and as well as to other identity-based signature schemes.

**Keywords:** Pairing-based signatures, Identity-based signatures, Batch verification, Short signatures, Wireless networks.

## 1 Introduction

Public-key digital signatures have frequently been used in proposals for securing wireless network protocols. Proposals include methods for performing the following: combating SPAM [11]; securing routing protocols [19,30]; providing secure accounting and charging for use of the wireless network, or securely giving incentives to nodes for desirable (to the network) behavior [22,4]; protecting

---

location and safety messages in vehicular networks [23,21]; and securely transporting ordinary messages in delay (or disruption) tolerant networks [7,26]. Even in wireless networks that have significant performance constraints such as sensor networks, it has been argued on efficiency grounds that signature schemes should be used for message authentication rather than symmetric cryptographic techniques [10,27].

When protocol designers need to select a bandwidth efficient signature scheme, they will be drawn to schemes based on bilinear pairings, such as the short signature schemes [2,5] or the bandwidth efficient identity-based signature schemes [6,5]. However, the computational cost of such schemes, especially the cost of their verification algorithms, can negatively impact the performance of wireless networks (e.g., increased delay, CPU utilization, energy consumption). Therefore, whenever circumstances allow, designers will employ batch verification methods for such pairing-based signature schemes [29,5,14]. Adversaries can attempt to negate the advantages of batch verification by corrupting messages or signatures within a batch. To counter such attacks, efficient methods are needed to identify the valid signatures within a batch that has failed initial batch verification.

To discover the valid signatures in an invalid batch, rather than verifying each signature individually, "divide-and-conquer" (DQ) techniques have been proposed [20,14]. These methods can be significantly faster than verifying individually whenever the ratio of the number of invalid signatures to the batch size is low. These methods require only $O(w(\log_2(N/w)+1))$ batch verifications and, for pairing-based signatures, product of pairings computations [13]. Recent methods for identification of invalid pairing-based signatures require $O(w)$ batch verifications [14]. When the ratio $w/N$ is very low these methods can provide significant performance improvements over DQ methods; however, the cost of performing the batch verifications used in these methods grows exponentially, limiting their use to very small batches, and to batches with only very few invalid signatures.

*Our contribution.* In this paper, we present two new methods for finding invalid signatures in pairing-based schemes. These methods are based on divide-and-conquer searching, but differ from previous methods in how the (sub-) batches are verified. The average number of product of pairings computations required in our methods is $O(w)$, which is a substantial improvement over previous divide-and-conquer methods when the ratio $w/N$ is low, and is the same complexity as the exponential cost methods. The expected number of multiplications in $\mathbb{F}_{q^d}$ required of the new methods is $O(w\sqrt{N})$, and $O(wN)$, compared to estimates of the cost of the two exponential cost methods, $O(N^{w-1}/(w-1)!)$ and $O(w^{w-1}N^{\frac{w-1}{2}}/(w-1)!)$ [14]. We have specified these methods and compared their performance for Cha-Cheon signatures [6]; however, these methods can be applied to several other pairing-based signature schemes, specifically the batched identity-based and batched short signature schemes discussed in [8].

## 2   Notation

In this paper we assume that pairing-based schemes use bilinear pairings on an elliptic curve $E$, defined over $\mathbb{F}_q$, where $q$ is a large prime. $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct subgroups of prime order $r$ on this curve, where $\mathbb{G}_1$ is a subset of the points on $E$ with coordinates in $\mathbb{F}_q$, and $\mathbb{G}_2$ is a subset of the points on $E$ with coordinates in $\mathbb{F}_{q^d}$, for a small integer $d$ (the embedding degree). The pairing $e$ is a map from $\mathbb{G}_1 \times \mathbb{G}_2$ into $\mathbb{G}_T$ where $\mathbb{G}_T$ is a multiplicative group of order $r$ in the field $\mathbb{F}_{q^d}$.

We use the following notation for the components of the costs of (batch) signature verification and invalid signature identification methods for Cha-Cheon signatures. CstDblPair is the cost of a double product of pairings computation [13]. $\mathsf{CstMult}\mathbb{G}_1(t_1)$ is the cost of multiplying an element of $\mathbb{G}_1$ by a scalar $s$ of size $|s|$ and $t_1 = \lceil \log_2(|s|) \rceil$; likewise $\mathsf{CstDlbMult}\mathbb{G}_1(t_1, t_2)$ is the cost of a pair of multiplications of elements of $\mathbb{G}_1$ by scalars of size $t_1$ and $t_2$ simultaneously. $\mathsf{CstAdd}\mathbb{G}_1$ is the cost of adding two elements of $\mathbb{G}_1$, and $\mathsf{CstSub}\mathbb{G}_1$ is the cost of subtracting an element of $\mathbb{G}_1$ from another element. $\mathsf{CstInv}\mathbb{G}_T$ is the cost of computing an inverse of an element in $\mathbb{G}_T$; $\mathsf{CstMult}\mathbb{G}_T$ is the cost of multiplying two elements of $\mathbb{G}_T$; and $\mathsf{CstExpt}\mathbb{G}_T(t_1)$ is the cost of raising an element of $\mathbb{G}_T$ to the power $s$.

## 3   Background

Batch cryptography was introduced by Fiat [9], and the first batch signature scheme was that of Naccache *et al.* [18] for a variant of DSA signatures. Bellare *et al.* [1] presented three generic methods for batching modular exponentiations: *the random subset test*, the *small exponents test* (SET), and the *bucket test*, which are related to techniques in [18,28].

The inputs to the small exponents test are a security parameter $l$, a generator $g$ of the group $G$ of prime order $q$, and $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with $x_i \in Z_q$ and $y_i \in G$. The verifier 1) checks that $g^{x_i} = y_i$ for all $i$, $1 \leq i \leq N$; 2) chooses $n$ random integers $r_1, \ldots, r_N$ in the range $[0, 2^l - 1]$; 3) computes $x = \sum_{i=1}^{N} x_i r_i$ and $y = \prod_{i=1}^{N} y_i^{r_i}$; and 4) tests whether $g^x = y$ and accepts the batch if true, else rejects. If the test rejects a batch, then there is at least one invalid pair $(x_i, y_i)$; the probability that the test accepts a batch containing invalid signatures is at most $2^{-l}$ [1], if the order of $G$ is a prime [3]. One of the $r$'s can be set to one without loss of security [14]. The small exponents test has appeared in pairing-based signature schemes [2,5] including, in [14], as the batch verifier for the Cha-Cheon signature scheme [6].[1]

*Cha-Cheon Identity-Based Signature scheme.* $H(m, U)$ is a cryptographic hash that maps a bit string $m$ and a point $U \in \mathbb{G}_1$ to an integer between 1 and $r$.

---

[1] For Cha-Cheon signatures with the cost parameters in Section 5, SET always verifies a valid batch more efficiently than the random subset test, and the bucket test verifies a valid batch more efficiently than SET when the batch size exceeds 512.

1. *Setup:* The system manager selects an order $r$ point $T \in \mathbb{G}_2$ and randomly selects an integer $s$ in the range $[1, r - 1]$. The manager computes $S = sT$. The public system parameters are $T$ and $S$. The system manager's secret key is $s$.
2. *Extract:* Each user is given a key pair. The user's public key, $Q$, is a point in $\mathbb{G}_1$ that is derived from the user's identity using a public algorithm. The user's private key is $C = sQ$.
3. *Sign:* To sign a message $m$, the signer randomly generates an integer $t$ in the range $[1, r - 1]$ and outputs a signature $(U, V)$ where $U = tQ$ and $V = (t + H(m, U))C$.
4. *Verify:* To verify a signature $(U, V)$ of message $m$, the verifier derives the signer's public key $Q$ from the alleged signer's identity and computes $h = H(m, U)$. If $e(U + hQ, S) = e(V, T)$ then the signature is accepted. Otherwise, the signature is rejected. This test can be rewritten as $1 = e(U + hQ, S) \cdot e(V, -T)$ which can be computed more efficiently [13].

In [14] the following batch verifier was presented. The verifier obtains $N$ messages $m_i$, for $i = 1$ to $N$, and the signatures $(U_i, V_i)$ and signer's identity for each message. The verifier derives each public key $Q_i$ from the signer's identity and checks that $U_i$ and $V_i$ are elements of $\mathbb{G}_1$. The verifier sets $r_1 = 1$ and generates random values $r_i$ from $[0, 2^l - 1]$, for $i = 2$ to $N$. The batch is valid if

$$1 = \alpha_0 = e\left(\sum_{i=1}^{N} B_i, S\right) \cdot e\left(\sum_{i=1}^{N} D_i, -T\right)$$

where $B_i = r_i(U_i + H(m_i, U_i) \cdot Q_i)$, $D_i = r_i V_i$, and 1 is the identity in $\mathbb{F}_{q^d}$.

## 3.1   Identifying Invalid Signatures

The problem of identifying invalid signatures within a batch has only recently been investigated. Work in this area generally falls into three categories: divide-and-conquer methods [20,14], identification code based methods [20], and expo-nent testing methods [15,16,25,14].

**Divide-and-Conquer Methods.** Pastuszak *et al.* [20] first investigated methods for identifying invalid signatures within a batch. They explored "divide-and-conquer" methods for generic batch verifiers, i.e., methods that work with any of the three batch verifiers studied by Ballare *et al.* In these methods the set of signatures in an invalid batch is repeatedly divided into $d \geq 2$ smaller sub-batches to verify. The most efficient of their techniques, the *Fast DC Verifier* method, exploits knowledge of the results of the first $d-1$ sub-batch verifications to determine whether the verification of the $d^{th}$ sub-batch is necessary, i.e., if a (sub-)batch *batch* is invalid and the first $d - 1$ sub-batches of *batch* are all valid, then the $d$'th sub-batch must be invalid, and the batch verifier for that sub-batch is not computed. Performance measurements of one of the methods of [20] for the Boneh, Lynn and Shacham (BLS) [2] signature scheme have been reported [8]. The authors observed that the divide-and-conquer method they

studied outperformed verifying each signature individually when $w/N < .15$ in batches of 1024 BLS signatures using 160-bit MNT curves.

In [14] a more efficient divide-and-conquer method called Binary Quick Search (BQS) was presented; BQS is applicable to small exponents test based verifiers. In this method a batch verifier that compares two quantities, $X$ and $Y$, is replaced with an equivalent test $A = XY^{-1}$, and the batch is accepted if $A = 1$.[2] The BQS algorithm is always[3] more efficient than any $d = 2^n$ary DC Verifier; When it is necessary to verify the $d^{th}$ child sub-batch, in BQS the sub-batch can be verified by simply computing a single inverse operation and a single multi-term multiplication (or $d - 1$ ordinary multiplications) rather than the much more expensive batch verification required by the Fast DC verifier. The upper bound of the number of batch verifications required by BQS is half that of the Fast DC Verifier for $d = 2$ [14].

**Identification Code Based Methods.** Pastuszak *et al.* [20] investigated using a Hamming identification code and a two-layer Hamming identification code for identifying invalid signatures in generic batch verification. The Hamming code verifier can identify a single error in a batch of size $2^n - 1$ using $n + 2$ batch verifications, and the two-layer verifier can identify 2 invalid signatures in a batch of $2^n - 2$ signatures using $3n + 3$ batch verifications.

**Exponent Testing Methods.** The first exponent testing method, developed by Lee *et al.* [15], was capable of finding a single invalid signature within a batch of "DSA-type" signatures. Signatures of this type have verification equations of the form "$g^m = s \mod p$" where $m$ is the message, $s$ is the signature, the generator $g$ has order $q$, and $p$ and $q$ are primes where $q \mid (p - 1)$. To identity an invalid signature, compute $X = \prod_{i=1}^{N} s_i/g^{\sum_{i=1}^{N} m_i}$ and $Y = \prod_{i=1}^{N} s_i^i/g^{\sum_{i=1}^{N} i \cdot m_i}$ and test whether $Y = X^z$ for $z \in [1, N]$. The Exponentiation method of Law and Matt [14], for the special case of identifying a single invalid signature, is similar to the above method.

Lee et al. [16] applied their approach for DSA-type signatures to identifying a single invalid signature in batches of RSA signatures. They addressed the problem of identifying multiple invalid RSA signatures by using their RSA method in a divide-and-conquer method that is somewhat similar to the Single Pruning Search we present in Section 4. However, Stanek showed in [25] that their approach for RSA signatures is not secure.

In [14] two exponent testing methods for pairing-based batch signatures, the Exponentiation method and the Exponentiation with Sectors method, were presented. Both methods require computing a number of batch verifications that are proportional to the number of invalid signatures $w$ in the batch. The Exponentiation method requires $w+1$ verifications (including the initial batch verification)

---

[2] For the initial batch verification, if it is more efficient to do so compute $X$ and $Y$, compare them, and compute $A = XY^{-1}$ if the comparison fails; otherwise $A$ is computed directly, e.g., in Cha-Cheon where $A = \alpha_0$.

[3] Except when $w = 1$ and the invalid signature is located in the rightmost position in the batch then Fast DC verifier and BQS have equal costs.

and the same number of product of pairings computations. Exponentiation with Sectors requires at most $2w+1$ product of pairings computations. Both methods use exhaustive search during batch verification, resulting in exponential cost.

*Exponentiation Method.* For the Cha-Cheon signature scheme, compute $\alpha_0$ and test whether $\alpha_0$ is equal to the identity. If so, the batch is valid. Otherwise compute $\alpha_j, w \geq j \geq 1$,

$$\alpha_j = e\left(\sum_{i=1}^{N} i^j B_i, S\right) e\left(\sum_{i=1}^{N} i^j D_i, -T\right), \tag{1}$$

and perform a test on the values $\alpha_j, \alpha_{j-1}, \ldots \alpha_0$. For $j = 1$, test whether $\alpha_1 = \alpha_0^{z_1}$ has a solution for $1 \leq z_1 \leq N$ using Shanks' giant-step baby-step algorithm [24]. If successful, $w = 1$ and $z_1$ is the position of the invalid signature. In general the method tests whether

$$\alpha_j = \prod_{t=1}^{j} (\alpha_{j-t})^{(-1)^{t-1} p_t} \tag{2}$$

has a solution where $p_t$ is the $t$th elementary symmetric polynomial in $1 \leq z_1 \leq \ldots \leq z_j \leq N$. The authors show that the tests can be performed in $O(\sqrt{N})$ for $j = 1$ and $O(N^{j-1}/(j-1)!)$ for $j \geq 2$ multiplications in $\mathbb{F}_{q^d}$. If a test fails increment $j$, compute $\alpha_j$, and test. When $j = w$ the test will succeed, and the values of $z_1, \ldots, z_w$ are the positions of the invalid signatures.

*Exponentiation with Sectors Method.* The Exponentiation with Sectors Method uses two stages. In the first stage, the batch is divided into approximately $\sqrt{N}$ sectors of approximately equal size and the Exponentiation method is used, where each $B_i$, and $D_i$ within a sector is multiplied by the same constant, to identify the $v$ invalid sectors. In the second stage, the Exponentiation method is used to find the invalid signatures within a batch consisting of the signatures from the $v$ invalid sectors. This method requires $w + v + 1$ product of pairings computations, including the initial verification, where $v \leq \min(w, \sqrt{N})$. During the first stage the tests can be performed in $O(N^{\frac{1}{4}})$ for $j = 1$ and $O(\sqrt{N}^{j-1}/(j-1)!)$ for $j \geq 2$ multiplications in $\mathbb{F}_{q^d}$. During the second stage the number of multiplications required for $w \leq j \geq v$ is $O(\sqrt{v\sqrt{N}})$ for $j = 1$, and $O(v\sqrt{N}^{j-1}/(j-1)!)$ for each $j \geq 2$.

## 4   An Alternate Approach to Divide-and-Conquer Methods

Divide-and-conquer methods can be viewed as operating on (for simplicity) a binary tree $T$ with $w \geq 1$ invalid signatures whose root node, $root_T$, is the batch, and each pair of child nodes represents the two nearly equal size sub-batches of their parent. Previously published methods such as Binary DC Verifier and BQS identify the invalid signatures within the initial batch by descending through the

tree, performing verifications on the sub-batches of the nodes they encounter. When one of the methods reaches a node whose sub-batch is valid, the methods do not visit its descendants, if any. The methods identify the invalid signatures by identifying those nodes that are either the ancestors of the leaf nodes of $T$ that represent invalid signatures, or the leaf nodes themselves. The difference between the published methods are 1) the degree of the tree and 2) how efficiently the nodes of the tree are verified.

The methods we propose view $T$ as consisting of a parent sub-tree $PT$ with root node $root_{PT} = root_T$, and the leaves of $PT$ are the roots of the $w$ maximal sub-trees $ST_i$, $i = 1, .., w$, of $T$ which represent sub-batches that have a single invalid signature. If the $w = 1$, then $T = ST_1$ and $PT$ is the node $root_T$. The new methods identify invalid signatures by descending through $PT$ and identifying its leaves, and concurrently identifying the single invalid signature in each of the sub-batches these leaves represent.

For signature schemes such as the Cha-Cheon, $node$ of $T$ is the root of some $ST_i$ if there exists a value $z$, $lb \leq z \leq ub$, that is a solution to $\alpha_{1,node} = \alpha_{0,node}^z$. The values $lb$ and $ub$ are the lower and upper bounds of the sub-batch represented by $node$ within $B$, and $\alpha_{j,node} = e\left(\sum_{i=lb}^{ub} i^j B_i, S\right) \cdot e\left(\sum_{i=lb}^{ub} i^j D_i, -T\right)$. Shanks' giant-step baby-step algorithm can determine if such a solution exists in time (multiplications in $\mathbb{F}_{q^d}$) proportional to the square root of the size of the sub-batch. If no solution is found, then $node$ is in $PT$ but is not a leaf; hence its children must be tested. We refer to this approach as *single pruning*.

When the children of an interior node $p$ in $PT$, $l$ and its sibling $r$, are leaves of $PT$, there exist values $z_l$ in the range of indexes of the signatures in the left sub-batch and $z_r$ in the range of signatures in the right sub-batch, such that $\alpha_{1,p} = \alpha_{0,l}^{z_l} \cdot \alpha_{0,r}^{z_r}$. The values $z_l$ and $z_r$ can be determined using an algorithm, $PairSolver(Left, Right)$, with cost proportional to the size of the (sub-)batch represented by $p$. If the algorithm fails, then at least one of the child nodes is not a leaf of $PT$ and they are tested individually using Shanks' algorithm. We refer to this approach as *paired single pruning*.

## 4.1  Single Pruning Search (SPS) Method

The recursive algorithm below describes the Single Pruning Search (SPS) method on a batch $B$ which is a list of $N = 2^h$, $h \geq 1$, randomly ordered message / signature pairs $((m_1, s_1), \ldots, (m_N, s_N))$ where the signature components for Cha-Cheon are verified elements of $\mathbb{G}_1$. On the initial call to $SPS(X, \alpha_{0,P}, \alpha_{1,P})$, $X = B$, $\alpha_{0,P} = 1$, $\alpha_{1,P} = 1$. SPS uses the following algorithms:

1. $Get_0(X)$ – checks whether $\alpha_0$ has been computed for $X$ and if so returns it; otherwise it computes $\alpha_0$ by the most efficient method available, and it may compute $\alpha_0^{-1}$ [17].
2. $Get_1(X)$ – checks whether $\alpha_1$ has been computed for $X$ and if so returns it; otherwise it computes $\alpha_1$ by the most efficient method available, and it may compute $\alpha_1^{-1}$ [17].

3. $Shanks(X)$ – if $X$ has a single invalid signature, the algorithm returns the position of the invalid signature; otherwise the algorithm returns 0 [17].
4. $Left(X)$ – returns a sub-batch with the first $len/2$ pairs in $X$, or $\varnothing$ if $X = \varnothing$.
5. $Right(X)$ – returns a sub-batch with the later $len/2$ pairs in $X$, or $\varnothing$ if $X = \varnothing$.
6. $Len(X)$ – returns the number of pairs in $X$, or 0 if $X = \varnothing$.

**Algorithm.** $SPS(X, \alpha_{0,P}, \alpha_{1,P})$   *(Single Pruning Search)*

Input: $X$ a list of message / signature pairs, $\alpha_{0,P}$ and $\alpha_{1,P}$ in $\mathbb{G}_T$.
Output: A list of the invalid pairs in the batch.
Return: A boolean.

   if $(Len(X) = 1)$ then
       output $X$
       return $(true)$
   else
       $\alpha_{0,N} \leftarrow Get_0(X)$
       if $(\alpha_{0,N} = 1)$ then
          return $(true)$
       elseif $(X = B)$ then
          $inv\alpha_{0,[B]} \leftarrow \alpha_{0,N}^{-1}$
       endif
       $\alpha_{1,N} \leftarrow Get_1(X)$
       if $(\alpha_{0,N} \neq \alpha_{0,P})$ then
          $z \leftarrow Shanks(X)$
          if $(z \neq 0)$ then
             output $(m_z, s_z)$
             return $(true)$
          elseif $(X = B)$ then
             $inv\alpha_{1,[B]} \leftarrow \alpha_{1,N}^{-1}$
          endif
       endif
       if $(SPS(Left(X), \alpha_{0,N}, \alpha_{1,N}))$ then
          $SPS(Right(X), \alpha_{0,N}, \alpha_{1,N})$
       endif
       return $(\alpha_{0,N} \neq \alpha_{0,P})$
   endif

$Get_0(X)$ computes products of pairings only for the root node and left children nodes that are tested by SPS. $Get_1(X)$ only computes products of pairings for the root and for each left child $X$ tested by SPS when the $\alpha_0$ of $X$ is not equal to $\alpha_0$ of the parent of $X$.

### 4.2   Paired Single Pruning Search Method

The recursive algorithm below describes the Paired Single Pruning Search (PSPS) method on a batch $B$, which is a list of $N = 2^h$, $h \geq 1$, randomly ordered message /

signature pairs $((m_1, s_1), \ldots, (m_N, s_N))$ where the signature components for Cha-Cheon are verified elements of $\mathbb{G}_1$. On the initial call to $PSPS(X, \alpha_{0,P}, \alpha_{1,P})$, $X = B$, $\alpha_{0,P} = 1$, $\alpha_{1,P} = 1$. PSPS also uses the following algorithms:

1. $PairSolver(Left, Right)$ – returns the positions of two invalid signatures, one in $Left$ and one in $Right$, or returns $(0,0)$ [17].
2. $Parent(X)$ – returns the parent of $X$, or $\varnothing$ if $X$ is the initial batch $B$.

**Algorithm.** $PSPS(X, \alpha_{0,P}, \alpha_{1,P})$    *(Paired Single Pruning Search)*

Input: $X$ a list of message / signature pairs, $\alpha_{0,P}$ and $\alpha_{1,P}$ in $\mathbb{G}_T$.
Output: A list of the invalid pairs in the batch.
Return: A boolean.

> if $(Len(X) = 1)$ then
> > output $X$
> > return $(true)$
>
> else
> > $\alpha_{0,N} \leftarrow Get_0(X)$
> > if $\alpha_{0,N} = 1$ then
> > > return $(true)$
> >
> > elseif $(X = B)$ then
> > > $inv\alpha_{0,[B]} \leftarrow \alpha_{0,N}^{-1}$
> >
> > endif
> > if $(\alpha_{0,N} = \alpha_{0,P})$ then
> > > $\alpha_{1,N} \leftarrow Get_1(X)$
> >
> > else
> > > if $(X \neq B$ and $X = Left(Parent(X)))$ then
> > > > $(z_l, z_r) \leftarrow PairSolver(X, Right(Parent(X)))$
> > > > if $z_l \neq 0$ then
> > > > > output $(m_{z_l}, s_{z_l}), (m_{z_r}, s_{z_r})$
> > > > > return $(false)$
> > > >
> > > > end
> > >
> > > end
> > > $\alpha_{1,N} \leftarrow Get_1(X)$
> > > $z \leftarrow Shanks(X)$
> > > if $z \neq 0$ then
> > > > output $(m_z, s_z)$
> > > > return $(true)$
> > >
> > > elseif $(X = B)$ then
> > > > $inv\alpha_{1,[B]} \leftarrow \alpha_{1,N}^{-1}$
> > >
> > > endif
> >
> > endif
> > if $(PSPS(Left(X), \alpha_{0,N}, \alpha_{1,N}))$ then
> > > $PSPS(Right(X), \alpha_{0,N}, \alpha_{1,N})$
> >
> > endif
> > return $(\alpha_{0,N} \neq \alpha_{0,P})$
>
> endif

## 5   Performance

For Cha-Cheon signatures, the divide-and-conquer methods and the exponentiation methods batch verify by first checking that the signature components are in $\mathbb{G}_1$, then computing $\alpha_0$ for $B$, and testing whether $\alpha_0 = 1$. With the exception of BQS and the DC Verifiers, they compute their $\alpha_0$s, as shown in $Get_0(B)$ in Appendix A. The cost (not including the membership tests) is $N \cdot \mathsf{CstDlbMult}\mathbb{G}_1(\mathsf{t}_1, \mathsf{t}_2) + N \cdot \mathsf{CstMult}\mathbb{G}_1(\mathsf{t}_1) + 2(N-1)\,\mathsf{CstAdd}\mathbb{G}_1 + \mathsf{CstDblPair}$ with $t_1 = \lceil \log_2(r)/2 \rceil$ and $t_2 = \lceil \log_2(r) \rceil$.[4]

### 5.1   Cost of the New Methods When $w \geq 1$

**Single Pruning Search   Performance.** If $w = 1$, the cost of SPS increases by the cost of computing $\alpha_0^{-1}$ ($\mathsf{CstInv}\mathbb{G}_T$) and $\alpha_1$ for $B$ ($2(N-1)\,\mathsf{CstAdd}\mathbb{G}_1$ $+\,\mathsf{CstDblPair}$), plus the expected cost of a successful $Shanks(B)$ call, which is approximately $\frac{4}{3}\sqrt{N}\,\mathsf{CstMult}\mathbb{G}_T$.[5] If $w \geq 2$, the average cost of SPS is the sum of the costs of computing $\alpha_0$, $\alpha_0^{-1}$, $\alpha_1$, a failed $Shanks(B)$ call ($2\sqrt{N}\,\mathsf{CstMult}\mathbb{G}_T$), $\alpha_1^{-1}$, plus the sum of the costs generated as SPS investigates the descendents of $root_T$. The following recurrence relation generates these costs:

$\mathsf{R}_{(S)}(w, M) =$

$$
\begin{cases}
0, & \begin{aligned} & w = 0, 1, \\ & w > M \text{ or} \\ & w = 2 \text{ and } M = 2; \end{aligned} \\[2em]
\dfrac{\left[ \begin{aligned} & 2\binom{M/2}{2}(\mathsf{R}_{(S)}(2, M/2) + \mathsf{C}_{(S)}(2, 0, M/2)) + \\ & \binom{M/2}{1}^2 (\mathsf{C}_{(S)}(1, 1, M/2)) \end{aligned} \right]}{\binom{M}{2}}, & w = 2 \text{ and } M > 2; \\[3em]
\dfrac{\left[ \begin{aligned} & 2\binom{M/2}{w}(\mathsf{R}_{(S)}(w, M/2) + \mathsf{C}_{(S)}(w, 0, M/2)) + \\ & 2\binom{M/2}{w-1}\binom{M/2}{1}(\mathsf{R}_{(S)}(w-1, M/2) + \mathsf{C}_{(S)}(w-1, 1, M/2)) + \\ & \sum_{i=2}^{w-2}\binom{M/2}{w-i}\binom{M/2}{i}(\mathsf{R}_{(S)}(w-i, M/2) + \mathsf{R}_{(S)}(i, M/2) + \mathsf{C}_{(S)}(w-i, i, M/2)) \end{aligned} \right]}{\binom{M}{w}}, & w \geq 3,
\end{cases}
$$

---

[4] BQS and the DC Verifiers can compute $\alpha_0$ with the same cost [17].

[5] $Shanks(X)$ tests whether the equation $\alpha_{1,n} = \alpha_{0,n}^z$ has a solution $z$ in the range of $l$ up to $u$, the bounds of the (sub-)batch $X$ within the original batch. The algorithm alternately computes a value from the series $\alpha_1 \cdot (\alpha_0^{-1})^i$ and the series $(\alpha_0^s)^j \cdot \alpha_0^l$, $s = \lfloor \sqrt{|X|} \rfloor$, stopping when a match is found (single invalid signature) or when both series have been computed. Similarly, $PairSolver(Left, Right)$ alternately computes a value from one of two series, and terminates when a newly computed value from one series is equal to one of the values already computed for the other series, or when both series have been computed.

where for Cha-Cheon:

|  | Costs | | |
| --- | --- | --- | --- |
| *Argument* | CstDblPair | CstInv$\mathbb{G}_T$ | CstMult$\mathbb{G}_T$ |
| $C_{(S)}(2, 0, M/2)$ | 1 | | |
| $C_{(S)}(1, 1, M/2)$ | 2 | 2 | $\frac{8}{3}\sqrt{M/2}$ |
| $C_{(S)}(w, 0, M/2)$ | 1 | | |
| $C_{(S)}(w - 1, 1, M/2)$ | 2 | 2 | $\frac{10}{3}\sqrt{M/2}$ |
| $C_{(S)}(w - i, i, M/2)$ | 2 | 2 | $4\sqrt{M/2}$ |

For $C_{(S)}(2, 0, M/2)$ and $C_{(S)}(w, 0, M/2)$, $Get_0(X)$ is called for the left child node and no inverse is computed, with cost CstDblPair. For $C_{(S)}(1, 1, M/2)$, both $Get_0(X)$ and $Get_1(X)$ are called for the left child, combined cost is $2\,$CstDblPair $+$ $2\,$CstInv$\mathbb{G}_T$; for the right child, cost is zero, and two successful calls are made to $Shanks(X)$ with combined cost of $\frac{8}{3}\sqrt{M/2}\,$CstMult$\mathbb{G}_T$. $C_{(S)}(w - 1, 1, M/2)$ is similar to $C_{(S)}(1, 1, M/2)$ except that one of the calls to $Shanks(X)$ fails to find a solution. Both calls to $Shanks(X)$ fail for $C_{(S)}(w - i, i, M/2)$.

**Paired Single Pruning Search Performance.** If $w \leq 1$, PSPS has the same average cost as SPS. If $w \geq 2$, the average cost of PSPS is the sum of the costs of computing $\alpha_0$, $\alpha_0^{-1}$, $\alpha_1$, $\alpha_1^{-1}$, the cost of the failed $Shanks$ test on the batch $B$, and the sum of the costs generated as PSPS investigates the descendants of $root_T$. The following recurrence relation generates these costs:

$R_{(P)}(w, M) =$

$$
\begin{cases}
0, & \begin{aligned} &w = 0, 1, \\ &w > M \text{ or} \\ &w = 2 \text{ and } M = 2; \end{aligned} \\[2em]
\dfrac{\left[ \begin{aligned} &2\binom{M/2}{2}(R_{(P)}(2, M/2) + C_{(P)}(2, 0, M/2)) + \\ &\binom{M/2}{1}^2 (C_{(P)}(1, 1, M/2)) \end{aligned} \right]}{\binom{M}{2}}, & w = 2 \text{ and } M > 2; \\[3em]
\dfrac{\left[ \begin{aligned} &2\binom{M/2}{w}(R_{(P)}(w, M/2) + C_{(P)}(w, 0, M/2)) + \\ &2\binom{M/2}{w-1}\binom{M/2}{1}(R_{(P)}(w-1, M/2) + C_{(P)}(w-1, 1, M/2)) + \\ &\sum_{i=2}^{w-2} \binom{M/2}{w-i}\binom{M/2}{i}(R_{(P)}(w-i, M/2) + R_{(P)}(i, M/2) + C_{(P)}(w-i, i, M/2)) \end{aligned} \right]}{\binom{M}{w}}, & w \geq 3,
\end{cases}
$$

where for Cha-Cheon:

| | Costs | | |
|---|---|---|---|
| Argument | CstDblPair | CstInv$\mathbb{G}_T$ | CstMult$\mathbb{G}_T$ |
| $\mathsf{C}_{(\mathsf{P})}(2, 0, M/2)$ | 1 | | |
| $\mathsf{C}_{(\mathsf{P})}(1, 1, M/2)$ | 1 | 1 | $M/2$ |
| $\mathsf{C}_{(\mathsf{P})}(w, 0, M/2)$ | 1 | | |
| $\mathsf{C}_{(\mathsf{P})}(w-1, 1, M/2)$ | 2 | 2 | $M + \frac{10}{3}\sqrt{M/2}$ |
| $\mathsf{C}_{(\mathsf{P})}(w-i, i, M/2)$ | 2 | 2 | $M + 4\sqrt{M/2}$ |

For $\mathsf{C}_{(\mathsf{P})}(2, 0, M/2)$ and $\mathsf{C}_{(\mathsf{P})}(w, 0, M/2)$, $Get_0(X)$ is called for the left child node and no inverse is computed. For $\mathsf{C}_{(\mathsf{P})}(1, 1, M/2)$, $Get_0(X)$ is called for the left child, the cost is $\mathsf{CstDblPair} + \mathsf{CstInv}\mathbb{G}_T$, and a successful call is made to $PairSolver$ $(Left, Right)$ with expected cost of $M/2\,\mathsf{CstMult}\mathbb{G}_T$. For the argument $\mathsf{C}_{(\mathsf{P})}(w-1, 1, M/2)$, both $Get_0(X)$ and $Get_1(X)$ are called for both the left and right child, one failed call is made to $PairSolver(Left, Right)$ with cost $M\,\mathsf{CstMult}\mathbb{G}_T$, and one successful and one failed call are made to $Shanks(X)$. $\mathsf{C}_{(\mathsf{P})}(w-i, i, M/2)$ is similar to $\mathsf{C}_{(\mathsf{P})}(w-1, 1, M/2)$ except that both calls to $Shanks(X)$ fail.

### 5.2   Number of Product of Pairings Computations of SPS and PSPS

Let $T$ be a perfect binary tree, and $\mathrm{PT}_{(2)}$ be the sub-tree of $PT$, where each node represents 2 or more invalid signatures. For each node in $\mathrm{PT}_{(2)}$, SPS computes an $\alpha_0$ for its left child, unless the child is a leaf node of $T$. For each node in the $\mathrm{PT}_{(2)}$ with both child nodes in $PT$, SPS also computes $\alpha_1$ for its left child, unless the child is a leaf node of $T$. This computation occurs $w-1$ times. SPS also computes a pair of $\alpha$s for the root. Therefore, including the initial batch verification, SPS requires $|\mathrm{PT}_{(2)}| + (w+1)$ product of pairings computations, if none of the leaf nodes of $PT$ are leaves of $T$. SPS requires two fewer $\alpha$ computations whenever a pair of leaves of $PT$ are leaves of $T$.

For a perfect binary tree, the number of ways $j$ pairs leaves of $PT$ can be leaves of $T$ is $\binom{N/2}{j}$, and the number of ways the remaining $w-2j$ invalid signatures can be in the remaining $N/2 - j$ distinct 3 node subtrees at the lowest level of $T$ is $\binom{N/2-j}{w-2j}2^{w-2j}$. Therefore, the expected number of occurrences of two sibling leaf nodes of $T$ both representing invalid signatures is

$$\frac{1}{\binom{N}{w}} \sum_{j=1}^{\lfloor w/2 \rfloor} \binom{N/2}{j}\binom{N/2-j}{w-2j}2^{w-2j}.$$

Since $\binom{N-2}{w-2} = \sum_{j=1}^{\lfloor w/2 \rfloor} \binom{N/2}{j}\binom{N/2-j}{w-2j}2^{w-2j}$ the expression simplifies to $\frac{w(w-1)}{2(N-1)}$, and the expected number of $\alpha$s computed by Single Pruning Search when the batch size is a power of 2 is

$$|\mathrm{PT}_{(2)}| + (w+1) - \frac{w(w-1)}{N-1} \ .$$

In Appendix B we show that $|\,\mathrm{PT}_{(2)}\,| < 2w - 1$; therefore the expected number of product of pairings computations required by SPS is less than $3w$. Since the number of product of pairings computations in the cost functions of PSPS are all less than or equal to the corresponding functions of SPS , the average number of product of pairings computations used by PSPS is also $O(w)$.

## 5.3   Number of Multiplications in $F_q$

Figure 1 and Figure 2 compare methods analyzed in Section 5.1 and in [17] for finding invalid signatures in a batch once the initial batch verification has failed
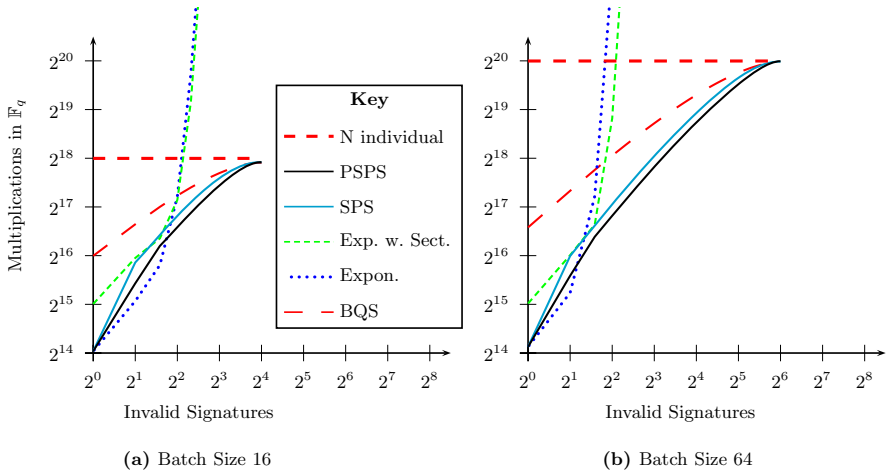


**(a)** Batch Size 16

**(b)** Batch Size 64

**Fig. 1. (a,b)** Number of multiplies in $\mathbb{F}_q$, where $r$ and $q$ are 160-bit values and $d = 6$



**(c)** Batch Size 256

**(d)** Batch Size 1024

**Fig. 1. (c,d)** Number of multiplies in $\mathbb{F}_q$, where $r$ and $q$ are 160-bit values and $d = 6$

**(a)** Batch Size 16

**(b)** Batch Size 64

**Fig. 2. (a,b)** Number of multiplies in $\mathbb{F}_q$, where $r$ and $q$ are 256-bit values and $d = 12$



**(c)** Batch Size 256

**(d)** Batch Size 1024
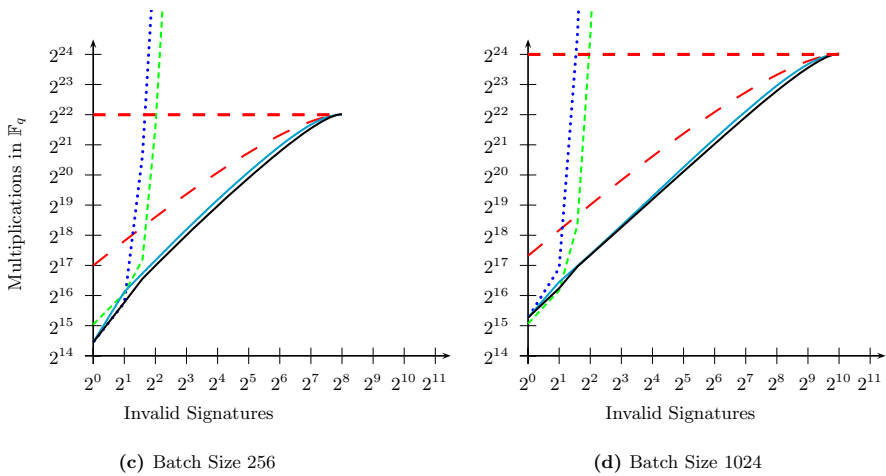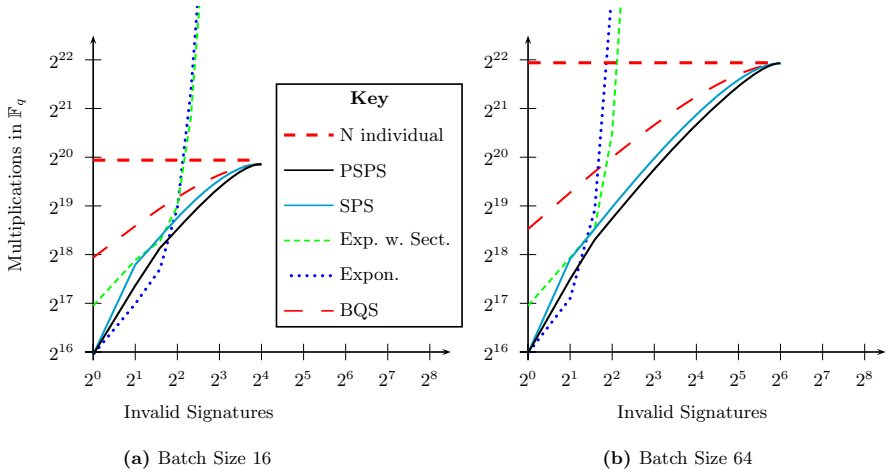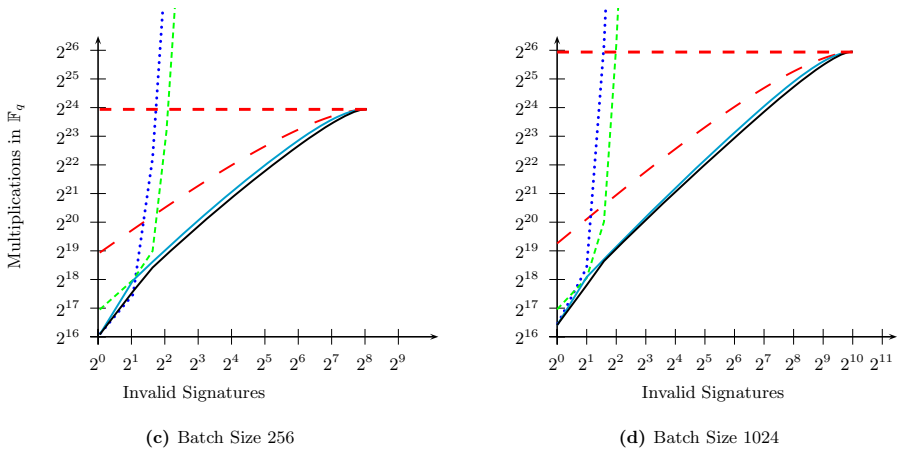
**Fig. 2. (c,d)** Number of multiplies in $F_q$, where $r$ and $q$ are 256-bit values and $d = 12$

for Cases A and C of [12]. In Case A, the group order $r$ is a 160-bit value, the elliptic curve $E$ is defined over $\mathbb{F}_q$, where $q$ is a 160-bit value, and the embedding degree $d = 6$. In Case C, the group order $r$ is a 256-bit value, $q$ is a 256-bit value, and the embedding degree $d = 12$. All costs are given in terms of the number of multiplications $(m)$ in $\mathbb{F}_q$ using the following estimates from Granger, Page and Smart [12], and Granger and Smart [13].

– For Case A, 1 double product of pairings $= 16,355m$, 1 multiplication in $F_{q^6} = 15m$, 1 inverse in $F_{q^6} = 44m$ (assuming 1 inverse in $F_q = 10m$), 1 elliptic curve addition $= 11m$, and an elliptic point multiplication by a 160-bit value is $1614m$ and by an 80-bit value is $827m$.

– For Case C, 1 double product of pairings $= 62,797m$, 1 multiplication in $F_{q^{12}} = 45m$, 1 inverse in $F_{q^{12}} = 104m$, 1 elliptic curve addition $= 11m$, and an elliptic point multiplication by a 256-bit value is $2535m$ and by an 128-bit value is $1299m$.

## 6   Conclusion

We have presented two new methods, Single Pruning Search and Paired Single Pruning Search, for identifying invalid signatures in pairing-based batch signature schemes using the small exponents test, and have analyzed their average case performance. These new methods require $O(w)$ product of pairings computations and $O(w\sqrt{N})$ and $O(wN)$ number of multiplications in $\mathbb{F}_{q^d}$. The methods are described for Cha-Cheon signatures, but are applicable to other batch verified signature schemes such as the batch verifiers presented in [8].

These new methods, like BQS and earlier divide-and-conquer methods, can be used when there is uncertainty in the number of invalid signatures in a batch. As shown in the figures in Section 5.3, the new methods significantly outperform the Binary Quick Search method when $w \ll N$, and perform as well as or better than the exponentiation methods except when $N$ and $w$ are small. Unlike the exponentiation methods, with the new methods a batch verifier is not forced to switch methods when tests for small $w$ fail.

In [14] the authors suggested that the exponentiation methods can be used with BQS to provide improved performance after tests for small values of $w$ fail. While this is certainly true, the result can be expensive. For example, with $N = 64$, a batch verifier that assumes that the number of invalid signature in the batch is small would start with the Exponentiation Method, but if the tests for $w = 1$ and $w = 2$ both fail, the verifier would switch to Exponentiation with Sectors Method to test if $w = 3$. If the test for $w = 3$ fails, then the verifier would switch to BQS. If $w = 4$, the cost of this sequence for Case A (ignoring the common cost of signature component validation and $\alpha_0$ computation) is at least $\approx 3.70 \times 10^5$ multiplications in $\mathbb{F}_q$, compared to $\approx 2.73 \times 10^5$ multiplications if only BQS was used, and $\approx 1.16 \times 10^5$ multiplications with the Paired Single Pruning Search Method.

Ideally, we would have a single efficient method for finding the invalid signatures in a batch that always has the lowest expected cost no matter how many signatures are invalid. Such a method would be especially useful when an adversary is occasionally able to inject bursts of several invalid signatures into some batches. Short of that ideal, but a practical alternative, would be a small set of methods, each of which for some range of batch sizes of interest always provides the lowest expected cost. Currently the Paired Single Pruning Search, provides the lowest expected cost when the batch size is in the range 128 to 512. For batches larger than 512, we would expect batch verifiers to utilize the bucket test for Cha-Cheon and related signature schemes rather than the small exponents test. Finding such a minimal cost method for batches smaller than 128 is as an open problem. Another open problem is to find more efficient methods

than the generic DC verifiers of [20] for identifying the invalid signatures in a batch when an initial bucket test verifier fails.

# References

1. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)
2. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
3. Boyd, C., Pavlovski, C.: Attacking and repairing batch verification schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 58–71. Springer, Heidelberg (2000)
4. Buchegger, S., Boudec, J.-Y.L.: Performance analysis of the CONFIDANT protocol (Cooperation of Nodes: Fairness In Dynamic Ad-hoc NeTworks). In: ACM/SIGMOBILE Third International Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC). ACM, New York (2002)
5. Camenisch, J., Hohenberger, S., Pedersen, M.: Batch verification of short signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007); see also Cryptology ePrint Archive, Report 2007/172, 2007, http://eprint.iacr.org/2007/172
6. Cha, J., Cheon, J.: An identity-based signature from gap diffie-hellman groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2002)
7. Fall, K.: A delay-tolerant network architecture for challenged internets. In: SIGCOMM 2003: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 27–34 (2003)
8. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.O.: On the practicality of short signature batch verification. Cryptology ePrint Archive, Report 2008/015 (2008), http://eprint.iacr.org/2008/015
9. Fiat, A.: Batch RSA. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 175–185. Springer, Heidelberg (1990)
10. Gaubatz, G., Kaps, J.-P., Sunar, B.: Public key cryptography in sensor networks—revisited. In: Castelluccia, C., Hartenstein, H., Paar, C., Westhoff, D. (eds.) ESAS 2004. LNCS, vol. 3313, pp. 2–18. Springer, Heidelberg (2005)
11. Gavidia, D., van Steen, M., Gamage, C., Jesi, G.P.: Canning spam in wireless gossip networks. In: Conference on Wireless On demand Network Systems and Services (WONS), pp. 208–220 (2007)
12. Granger, R., Page, D.L., Smart, N.P.: High security pairing-based cryptography revisited. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 480–494. Springer, Heidelberg (2006)
13. Granger, R., Smart, N.P.: On computing products of pairings. Cryptology ePrint Archive, Report 2006/172 (2006), http://eprint.iacr.org/2006/172
14. Law, L., Matt, B.J.: Finding invalid signatures in pairing based batches. In: Galbraith, S.D. (ed.) Cryptography and Coding 2007. LNCS, vol. 4887, pp. 35–53. Springer, Heidelberg (2007)

15. Lee, S.-W., Cho, S., Choi, J., Cho, Y.: Batch verification with DSA-type digital signatures for ubiquitous computing. In: Hao, Y., Liu, J., Wang, Y.-P., Cheung, Y.-m., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (eds.) CIS 2005. LNCS, vol. 3802, pp. 125–130. Springer, Heidelberg (2005)
16. Lee, S., Cho, S., Choi, J., Cho, Y.: Efficient identification of bad signatures in RSA-type batch signature. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A(1), 74–80 (2006)
17. Matt, B.J.: Identification of multiple invalid signatures in pairing-based batched signatures. Cryptology ePrint Archive (2009), http://eprint.iacr.org/2009
18. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. Be improved? In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
19. Papadimitratos, P., Haas, Z.: Secure routing for mobile ad hoc networks. In: Proceedings of SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002) (January 2002)
20. Pastuszak, J., Michałek, D., Pieprzyk, J., Seberry, J.: Identification of bad signatures in batches. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 28–45. Springer, Heidelberg (2000)
21. Raya, M., Hubaux, J.-P.: Securing vehicular ad hoc networks. Journal of Computer Security, Special Issue on Security of Ad Hoc and Sensor Networks 15(1), 39–68 (2007)
22. Salem, N.B., Buttyan, L., Hubaux, J.-P., Jakobsson, M.: A charging and rewarding scheme for packet forwarding in multi-hop cellular networks, In: ACM/SIGMOBILE 4th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC). ACM Press, New York (2003)
23. Sampigethaya, K., Mingyan, L., Leping, H., Poovendran, R.: Amoeba: Robust location privacy scheme for vanet. IEEE JSAC Special Issue on Vehicular Networks 25(8), 1569–1589 (2007)
24. Shanks, D.: Class number, a theory of factorization and genera. In: Symposium on Pure Mathematics, vol. 20, pp. 415–440. AMS (1971)
25. Stanek, M.: Attacking LCCC batch verification of RSA signatures. Cryptology ePrint Archive, Report 2006/111 (2006), http://eprint.iacr.org/2006/111
26. Symington, S., Farrell, S., Weiss, H., Lovell, P.: Bundle security protocol specification. draft-irtf-dtnrg-bundle-security-04 (work in progress) (September 2007)
27. Wagner, D.: The conventional wisdom about sensor network security.. is wrong. In: IEEE Security and Privacy 2005, and invited panelist, Security in Ad-hoc and Sensor Networks 2005 (2005)
28. Yen, S., Laih, C.: Improved digital signature suitable for batch verification. IEEE Transactions on Computers 44(7), 957–959 (1995)
29. Yoon, H., Cheon, J.H., Kim, Y.: Batch verifications with ID-based signatures. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 223–248. Springer, Heidelberg (2005)
30. Zapata, M.G., Asokan, N.: Securing ad hoc routing protocols. In: WiSE 2002: Proceedings of the 1st ACM workshop on Wireless security, pp. 1–10 (2002)

# A   Auxiliary Algorithms for SPS  and PSPS

The algorithms in Section 4 for the SPS and PSPS methods call $Get_0(X)$ to obtain $\alpha_0$ (and $\alpha_0^{-1}$) for $X$, and $Get_1(X)$ to obtain $\alpha_1$ (and $\alpha_1^{-1}$) for $X$. In this section we describe these algorithms for Cha-Cheon signatures. $Get_0(X)$ and $Get_1(X)$ use the following algorithms:

1. $Lowerindex(X)$ – returns the index within the batch $B$ of the message / signature pair in the lowest position in $X$.
2. $Upperindex(X)$ – returns the index in $B$ of the pair in the highest position in $X$.

**Algorithm.** $Get_0(X)$   (*Obtain $\alpha_0$ (and $\alpha_0^{-1}$) for Cha-Cheon*)
Input: $X$ a list of message / signature pairs.
Output: None.
Return: The value $\alpha_{0,[X]}$ for $X$.
$\quad$ $P \leftarrow Parent(X); L \leftarrow Left(P); R \leftarrow Right(P)$
$\quad$ if $(\alpha_{0,[X]})$ then
$\quad\quad\quad$ return $(\alpha_{0,[X]})$
$\quad$ elseif $(X = R)$ then
$\quad\quad\quad$ $\alpha_{0,[R]} \leftarrow \alpha_{0,[P]} \cdot inv\alpha_{0,[L]}$
$\quad\quad\quad$ $inv\alpha_{0,[R]} \leftarrow inv\alpha_{0,[P]} \cdot \alpha_{0,[L]}$
$\quad\quad\quad$ return $(\alpha_{0,[R]})$
$\quad$ elseif $(X = L)$ then
$\quad\quad\quad$ $l \leftarrow Lowerindex(X); u \leftarrow Upperindex(X)$
$\quad\quad\quad$ $\alpha_{0,[L]} \leftarrow e\left(VB_l - VB_{u+1}, S\right) \cdot e\left(VD_l - VD_{u+1}, -T\right)$
$\quad\quad\quad$ if $(\alpha_{0,[L]} \neq \alpha_{0,[P]})$ then
$\quad\quad\quad\quad$ $inv\alpha_{0,[L]} \leftarrow \alpha_{0,[L]}^{-1}$
$\quad\quad\quad$ else
$\quad\quad\quad\quad$ $inv\alpha_{0,[L]} \leftarrow inv\alpha_{0,[P]}$
$\quad\quad\quad$ endif
$\quad\quad\quad$ return $(\alpha_{0,[L]})$
$\quad$ else
$\quad\quad\quad$ $VB_{len(X)} \leftarrow B_{len(X)}$
$\quad\quad\quad$ $VD_{len(X)} \leftarrow D_{len(X)}$
$\quad\quad\quad$ for $i = Len(X) - 1$ downto 1 do
$\quad\quad\quad\quad$ $VB_i \leftarrow VB_{i+1} + B_i$
$\quad\quad\quad\quad$ $VD_i \leftarrow VD_{i+1} + D_i$
$\quad\quad\quad$ endfor
$\quad\quad\quad$ $\alpha_{0,[B]} \leftarrow e\left(VB_1, S\right) \cdot e\left(VD_1, -T\right)$
$\quad\quad\quad$ return $(\alpha_{0,[B]})$
$\quad$ endif

When $X$ is a left child, the cost is at most $2 \cdot \mathsf{CstSub}\mathbb{G}_1 + \mathsf{CstDblPair} + \mathsf{CstInv}\mathbb{G}_T$. If $X$ is a right child, the cost is at most $2\,\mathsf{CstMult}\mathbb{G}_T$. Since $\mathsf{CstDblPair} \gg \mathsf{CstSub}\mathbb{G}_1$ and $\mathsf{CstDblPair} \gg \mathsf{CstMult}\mathbb{G}_T$, we estimate the cost of $Get_0$ for pair of siblings as $\mathsf{CstDblPair} + \mathsf{CstInv}\mathbb{G}_T$ in Section 5.

**Algorithm.** $Get_1(X)$     (*Obtain $\alpha_1$ (and $\alpha_1^{-1}$) for Cha-Cheon*)

Input: $X$ a list of message / signature pairs.

Output: None.

Return: The value $\alpha_{1,[X]}$ for $X$.

   $P \leftarrow Parent(X); \ L \leftarrow Left(P); \ R \leftarrow Right(P)$

   if $(\alpha_{1,[X]})$ then

      return $(\alpha_{1,[X]})$

   elseif $(\alpha_{0,[X]} = \alpha_{0,[P]})$ then

      $\alpha_{1,[X]} \leftarrow \alpha_{1,[P]}$

      $inv\alpha_{1,[X]} \leftarrow inv\alpha_{1,[P]}$

      return $(\alpha_{1,[X]})$

   elseif $(X = R)$ then

      $\alpha_{1,[R]} \leftarrow \alpha_{1,[P]} \cdot inv\alpha_{1,[L]}$

      $inv\alpha_{1,[R]} \leftarrow inv\alpha_{1,[P]} \cdot \alpha_{1,[L]}$

      return $(\alpha_{1,[R]})$

   elseif $(X = L)$ then

      $l \leftarrow Lowerindex(X); \ u \leftarrow Upperindex(X)$

      $WB_{l,u} \leftarrow UB_u - (u \cdot VB_{u+1} + WB_{1,l-1})$

      $WD_{l,u} \leftarrow UD_u - (u \cdot VD_{u+1} + WD_{1,l-1})$

      if $(l \neq 1)$ then

         $WB_{1,u} \leftarrow WB_{1,l-1} + WB_{l,u}$

         $WD_{1,u} \leftarrow WD_{1,l-1} + WD_{l,u}$

      endif

      $\alpha_{1,[L]} \leftarrow e\left(WB_{l,u}, S\right) \cdot e\left(WD_{l,u}, -T\right)$

      if $(\ \alpha_{1,[L]} \neq \alpha_{1,[P]})$ then

         $inv\alpha_{1,[L]} \leftarrow \alpha_{1,[L]}^{-1}$

      else

         $inv\alpha_{1,[L]} \leftarrow inv\alpha_{1,[P]}$

      endif

      return $(\alpha_{1,[L]})$

   else

      $WB_{1,0} \leftarrow \infty$

      $WD_{1,0} \leftarrow \infty$

      $UB_1 \leftarrow VB_1$

      $UD_1 \leftarrow VD_1$

      for $i = 2$ upto $len(X)$ do

         $UB_i \leftarrow UB_{i-1} + VB_i$

         $UD_i \leftarrow UD_{i-1} + VD_i$

      endfor

      $\alpha_{1,[B]} \leftarrow e\left(UB_{len(X)}, S\right) \cdot e\left(UD_{len(X)}, -T\right)$

      return $(\alpha_{1,[B]})$

   endif

To compute $\alpha_1$s and its inverse for a left child node costs no more than $4 \cdot$ CstAdd$\mathbb{G}_1 + 2 \cdot$ CstSub$\mathbb{G}_1 + 2 \cdot$ CstMult$\mathbb{G}_1(t_1) +$ CstInv$\mathbb{G}_T +$ CstDblPair with $t_1 = \lceil log_2(Len(X)) \rceil < \lceil log_2(N) \rceil$. If $X$ is a right child, the cost is at most $2$ CstMult$\mathbb{G}_T$.

We estimate the cost of $Get_1$ for pair of siblings as $\mathsf{CstDblPair} + \mathsf{CstInv}\mathbb{G}_T$ in Section 5.

## B  $|\mathbf{PT_{(2)}}| < 2w - 1$

We show that $|\operatorname{PT}_{(2)}| < 2w-1$ whenever $N = 2^i$ for $i = 1, 2, \ldots$. Let $S_{(2)}(i, w) = |\operatorname{PT}_{(2)}|$ for $N = 2^i$ and $0 \leq w \leq N$. Note that $S_{(2)}(i, w) = 0$ when $w = 0, 1$, and $S_{(2)}(1, 2) = 1$. Assume that $S_{(2)}(i, w) < 2w - 1$.

For $w = 2$

$$
\begin{aligned}
S_{(2)}(i+1, 2) &= \sum_{j=0}^{2} \frac{\binom{2^i}{2-j}\binom{2^i}{j}}{\binom{2^{i+1}}{2}} (S_{(2)}(i, 2-j) + S_{(2)}(i, j) + 1) \\
&< \frac{2\binom{2^i}{2}\binom{2^i}{0}}{\binom{2^{i+1}}{2}}(4) + \frac{\binom{2^i}{1}\binom{2^i}{1}}{\binom{2^{i+1}}{2}}(1) \\
&< 2 + \frac{2^i - 2}{2^{i+1} - 1} \\
&< 3.
\end{aligned}
$$

For $w \geq 3$

$$
\begin{aligned}
S_{(2)}(i+1, w) &= \sum_{j=0}^{w} \frac{\binom{2^i}{w-j}\binom{2^i}{j}}{\binom{2^{i+1}}{w}} (S_{(2)}(i, w-j) + S_{(2)}(i, j) + 1) \\
&< \frac{2\binom{2^i}{w}\binom{2^i}{0}}{\binom{2^{i+1}}{w}}((2w)) + \frac{2\binom{2^i}{w-1}\binom{2^i}{1}}{\binom{2^{i+1}}{w}}(2w-2) + \sum_{j=2}^{w-2} \frac{\binom{2^i}{w-j}\binom{2^i}{j}}{\binom{2^{i+1}}{w}}(2w-1) \\
&< 2w - 1 - \frac{2\binom{2^i}{w-1}}{w\binom{2^{i+1}}{w}}\left((2^i + 1)(w-1)\right) \\
&< 2w - 1.
\end{aligned}
$$