Practical Short Signature Batch Verification

Anna Lisa Ferrara[1], Matthew Green[2], Susan Hohenberger[3], and Michael Østergaard Pedersen[4]

[1] University of Illinois at Urbana-Champaign
[2] Independent Security Evaluators
[3] Johns Hopkins University
[4] Lenio A/S

Abstract. In many applications, it is desirable to work with signatures that are short, and yet where *many* messages from *different* signers be verified very quickly. RSA signatures satisfy the latter condition, but are generally thousands of bits in length. Recent developments in pairing-based cryptography produced a number of "short" signatures which provide equivalent security in a fraction of the space. Unfortunately, verifying these signatures is computationally intensive due to the expensive pairing operation. Toward achieving "short and fast" signatures, Camenisch, Hohenberger and Pedersen (Eurocrypt 2007) showed how to *batch verify* two pairing-based schemes so that the total number of pairings was independent of the number of signatures to verify.

In this work, we present both theoretical and practical contributions. On the theoretical side, we introduce new batch verifiers for a wide variety of regular, identity-based, group, ring and aggregate signature schemes. These are the first constructions for batching group signatures, which answers an open problem of Camenisch et al. On the practical side, we implement each of these algorithms and compare each batching algorithm to doing individual verifications. Our goal is to test whether batching is practical; that is, whether the benefits of removing pairings significantly outweigh the cost of the additional operations required for batching, such as group membership testing, randomness generation, and additional modular exponentiations and multiplications. We experimentally verify that the theoretical results of Camenisch et al. and this work, indeed, provide an efficient, effective approach to verifying multiple signatures from (possibly) different signers.

1 Introduction

As we move into the era of pervasive computing, where computers are everywhere as an integrated part of our surroundings, there are going to be a host of devices exchanging messages with each other, e.g., sensor networks, vehicle-2-vehicle communications [1,2]. For these systems to work properly, messages must carry some form of authentication, but the system requirements on the authentication are particularly demanding. Any cryptographic solution must simultaneously be:

M. Fischlin (Ed.): CT-RSA 2009, LNCS 5473, pp. 309–324, 2009.

1. *Short*: Bandwidth is an issue. Raya and Hubaux argue that due to the limited spectrum available for vehicular communication, something shorter than RSA signatures is needed [3].
2. *Quick to verify large numbers of messages from different sources*: Raya and Hubaux also suggest that vehicles will transmit safety messages every 300ms to all other vehicles within a minimum range of 110 meters [3], which in turn may retransmit these messages. Thus, it is much more critical that authentications be quick to verify rather than to generate.
3. *Privacy-friendly*: Users should be held accountable, but not become publicly identifiable.

Due to the high overhead of using digital signatures, researchers have developed a number of alternative protocols designed to amortize signatures over many packets [4,5], or to replace them with symmetric MACs [6]. Each approach has significant drawbacks; e.g., the MAC-based protocols use time-delayed delivery so that the necessary verification keys are delivered *after* the authenticated messages arrive. This approach can be highly efficient within a restricted setting where synchronized clocks are available, but it does not provide non-repudiability of messages (to hold malicious users accountable) or privacy. Signature amortization requires verifiers to obtain many packets before verifying, and is vulnerable to denial of service. Other approaches, such as the short, undeniable signatures of Monnerat and Vaudenay [7,8] are inappropriate for the pervasive settings we consider, since verification requires interaction with the signer.

In 2001, Boneh, Lynn and Shacham developed a pairing-based signature that provides security equivalent to 1024-bit RSA at a cost of only 170 bits [9] (slightly larger than HMAC-SHA1). This was followed by many signature variants, some of them privacy-friendly, which were also relatively short, e.g., [10,11,12,13]. Unfortunately, the focus was on reducing the signature size, but less attention was paid to the verification cost which require expensive pairing operations.

Recently, Camenisch, Hohenberger and Pedersen [14] took a step toward speeding up the verification of short signatures, by showing how to *batch verify* two short pairing-based signatures so that the total number of dominant (pairing) operations was independent of the number of signatures to verify. However, their solution left open several questions which this work addresses.

First, their work was purely theoretical. To our knowledge, we are the first to provide a detailed *empirical analysis* of batch verification of short signatures. This is interesting, because our theoretical results and those of Camenisch et al. [14] reduce the total number of pairings by *adding* in other operations, such as random number generation and small modular exponentiations, so it was unclear how well these algorithms would perform in practice. Fortunately, in section 5, we verify that these algorithms do work well.

Second, Camenisch et al. [14] dealt only with batching regular and identity-based signatures. They specifically mentioned batching group signatures as an interesting open problem. Here, we present the *first* batch verifier for a group signature scheme, as well as new verifiers for many other types of regular, identity-based, ring and aggregate signatures.

Finally, Camenisch et al. [14] did not address the practical issue of what to do if the batch verification fails. How does one detect *which* signatures in the batch are invalid? Does this detection process eliminate all of the efficiency gains of batch verification? Fortunately, our empirical studies reveal good news: invalid signatures can be detected via a recursive divide-and-conquer approach, and if $< 15\%$ of the signatures are invalid, then batch verification is still more efficient than individual verification. At the time we conducted these experiments, the divide-and-conquer approach was the best method known to us. Recently, Law and Matt [15] proposed three new techniques for finding invalid signatures in a batch. One of their techniques allows to save approximately half the time needed by the simple divide-and-conquer approach, for large batch sizes. Thus, while our numbers seem good, they can be further improved.

Overall, we conclude that many interesting short signatures can be batch verified, and that batch verification is an extremely valuable tool for system implementors. As an example of our results in section 5, for the short group signatures of Boneh, Boyen and Shacham [10], we see that when batching 200 group signatures (in a 160-bit MNT curve) individual verification takes 139ms whereas batch verification reduces the cost to 25ms per signature (see Figure 3).

2 Algebraic Setting: Pairings

Let PSetup be an algorithm that, on input the security parameter 1^τ, outputs the parameters for a bilinear pairing as $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ are of prime order $q \in \Theta(2^\tau)$. The efficient mapping $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is both: (*bilinear*) for all $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ and $a, b \leftarrow \mathbb{Z}_q$, $\mathbf{e}(g^a, h^b) = \mathbf{e}(g, h)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}_1$ and h generates $\mathbb{G}_2$, then $\mathbf{e}(g, h) \neq 1$. This is called the *asymmetric* setting; in the *symmetric* setting, $\mathbb{G}_1 = \mathbb{G}_2$.

In the asymmetric setting, the best we can hope for are group elements in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of size 160, 512 and 1024 bits respectively. In the symmetric setting, it seems the best curve is a supersingular curve (with $k = 2$), where $\mathbb{G}_1 = \mathbb{G}_2$ and $\mathbb{G}_T$ will be of size 512 and 1024 bits respectively. Most of the signature schemes we discuss can be implemented in the asymmetric setting to take advantage of the smaller group sizes. We discuss this more and the case of batching composite order groups in the full version of this paper [16].

Testing Membership. Our proofs will require that elements of purported signatures are members of $\mathbb{G}_1$, but how efficiently can this fact be verified? Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. Assume that the subgroup has order q. The easy way to verify if $y \in \mathbb{G}_1$ is simply to test $y^q = 1$. Since q might be quite large this test is inefficient, but as we will see later the time required to test membership of group elements are insignificant compared to the time required to do the pairings in the applications we have in mind. Yet, in some cases, there are more efficient ways to test group membership [17].

3 Basic Tools for Pairing-Based Batch Verification

Let us begin with a formal definition of a *pairing based* batch verifier. Recall that PSetup is an algorithm that, on input the security parameter 1^τ, outputs the parameters $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order $q \in \Theta(2^\tau)$. Pairing-based verification equation are represented by a *generic pairing based claim* X corresponding to a boolean relation of the following form: $\prod_{i=1}^{k} \mathbf{e}(f_i, h_i)^{c_i} \stackrel{?}{=} A$, for $k \in \text{poly}(\tau)$ and $f_i \in \mathbb{G}_1, h_i \in \mathbb{G}_2$ and $c_i \in \mathbb{Z}_q^*$, for each $i = 1, \ldots, k$. A pairing-based verifier Verify for a generic pairing-based claim is a probabilistic poly(τ)-time algorithm which on input the representation $\langle A, f_1, \ldots, f_k, h_1, \ldots, h_k, c_1, \ldots, c_k \rangle$ of a claim X, outputs *accept* if X holds and *reject* otherwise. We define a batch verifier for pairing-based claims.

Definition 1 (Pairing-based Batch Verifier). *Let* $\mathsf{PSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$. *For each* $j \in [1, \eta]$, *where* $\eta \in \text{poly}(\tau)$, *let* $X^{(j)}$ *be a generic pairing-based claim and let* Verify *be a pairing based verifier. We define a* pairing-based batch verifier *for* Verify *as a probabilistic poly(τ)-time algorithm which outputs:*

- accept *if* $X^{(j)}$ *holds for all* $j \in [1, \eta]$*;*
- reject *if* $X^{(j)}$ *does not hold for any* $j \in [1, \eta]$ *except with negligible probability.*

3.1 Small Exponents Test Applied to Pairings

Bellare, Garay and Rabin proposed methods for verifying multiple equations of the form $y_i = g^{x_i}$ for $i = 1$ to n, where g is a generator for a group of prime order [18]. One might be tempted to just multiply these equations together and check if $\prod_{i=1}^{n} y_i = g^{\sum_{i=1}^{n} x_i}$. However, it would be easy to produce two pairs (x_1, y_1) and (x_2, y_2) such that the product of them verifies correctly, but each individual verification does not, e.g. by submitting the pairs $(x_1 - \alpha, y_1)$ and $(x_2 + \alpha, y_2)$ for any α. Instead, Bellare et al. proposed the following method, which we will later apply to pairings.

Small Exponents Test: Choose exponents δ_i of (a small number of) ℓ_b bits and compute $\prod_{i=1}^{n} y_i^{\delta_i} = g^{\sum_{i=1}^{n} x_i \delta_i}$. Then the probability of accepting a bad pair is $2^{-\ell_b}$. The size of ℓ_b is a tradeoff between efficiency and security. (In Section 5, we set $\ell_b = 80$ bits.)

Theorem 1. *Let* $\mathsf{PSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$ *where* q *is prime. For each* $j \in [1, \eta]$, *where* $\eta \in \text{poly}(\tau)$, *let* $X^{(j)}$ *corresponds to a generic claim as in Definition 1. For simplicity, assume that* $X^{(j)}$ *is of the form* $A \stackrel{?}{=} Y^{(j)}$ *where* A *is fixed for all* j *and all the input values to the claim* $X^{(j)}$ *are in the correct groups. For any random vector* $\Delta = (\delta_1, \ldots, \delta_\eta)$ *of* ℓ_b *bit elements from* $\mathbb{Z}_q$, *an algorithm* Batch *which tests the following equation* $\prod_{j=1}^{\eta} A^{\delta_j} \stackrel{?}{=} \prod_{j=1}^{\eta} {Y^{(j)}}^{\delta_j}$ *is a pairing-based batch verifier that accepts an invalid batch with probability at most* $2^{-\ell_b}$.

The proof closely follows the proof of the small exponents test by Bellare et al. [18], we include a full proof of this theorem in the full version of this paper [16]. Thus, Theorem 1 provides a *single* verification equation, which we then want to optimize.

3.2 Basic Batching Techniques

Armed with Theorem 1, let's back up for a moment to get a complete picture of how to develop an efficient batch verifier. This summarizes the ideas we used to obtain the results in Figure 1, which we believe will be useful elsewhere. Immediately after the summary, we'll explain the details.

Summary: Suppose you have η bilinear equations. Batch verify them as follows:

1. Apply Technique 1 to the individual verification equation, if applicable.
2. Apply Theorem 1 to the equations. This *combines* all equations into a single equation after checking membership in the expected algebraic groups and using the small exponents test.
3. Optimize the resulting single equation using Techniques 2, 3 and 4.
4. If batch verification fails, use the divide-and-conquer approach to identify the bad signatures.

Technique 1 *Change the verification equation.* Recall that a Σ-protocol is a three step protocol (commit, challenge, response) allowing a prover to prove various statements to a verifier. Using the Fiat-Shamir heuristic a Σ-protocol can be turned into a signature scheme, by forming the challenge as the hash of the commitment and the message to be signed. The signature is then either (commit, response) or (challenge, response). The latter is often preferred, since the challenge is usually smaller than the commitment, which results in a smaller signature. However, we observed that this often causes batch verification to become very inefficient, whereas using (commit, response) results in a much more suitable verification equation.

We use this technique to help batch the Hess IBS [19] and the group signatures of Boneh, Boyen and Shacham [10] and Boyen and Shacham [11]. Indeed, we believe that prior attempts to batch verify group signatures overlooked this idea and thus came up without efficient solutions.

Combination Step: Given η pairing-based claims, apply Theorem 1 to obtain a single equation. The combination step actually consist of two substeps:

1. *Check Membership*: Check that all elements are in the correct subgroup. Only elements that could be generated by an adversary needs to be checked (e.g., elements of a signature one wants to verify). Public parameters need not be checked, or could be checked only once.
2. *Small Exponents Test*: Combine all equations into one and apply the small exponents test.

Next, optimize this *single* equation using any of the following techniques in any order.

Technique 2 *Move the exponent into the pairing.* When a pairing of the form $\mathbf{e}(g_i, h_i)^{\delta_i}$ appears, move the exponent δ_i into $\mathbf{e}()$. Since elements of $\mathbb{G}$ are usually smaller than elements of $\mathbb{G}_T$, this gives a small speedup when computing the exponentiation.

$$\text{Replace } \mathbf{e}(g_i, h_i)^{\delta_i} \text{ with } \mathbf{e}(g_i^{\delta_i}, h_i)$$

Technique 3. *When two pairings with a common first or second element appear, they can be combined.* This can reduce η pairings to one. It will work like this:

$$\text{Replace } \prod_{i=1}^{\eta} \mathbf{e}(g_i^{\delta_i}, h) \text{ with } \mathbf{e}(\prod_{i=1}^{\eta} g_i^{\delta_i}, h)$$

In rare cases, it might be useful to apply this technique "in reverse", e.g., splitting a single pairing into two or more pairings to allow for the application of other techniques. For example, we do this when batching Boyen's ring signatures [13], so that we can apply Technique 4 below.

Technique 4 *Waters hash.* In his IBE, Waters described how hash identities to values in $\mathbb{G}_1$ [20], using a technique that was subsequently employed in several signature schemes. Assume the identity is a bit string $V = v_1 v_2 \ldots v_m$, then given public parameters $u_1, \ldots, u_m$, $u' \in \mathbb{G}_1$, the hash is $u' \prod_{i=1}^{m} u_i^{v_i}$. Following works by Naccache [21] and Chatterjee and Sarkar [22,23] documented the generalization where instead of evaluating the identity bit by bit, divide the k bit identity bit string into z blocks, and then hash. (In Section 5, we SHA1 hash our messages to a 160-bit string, and use $z = 5$ as proposed in [21].) Recently, Camenisch et al. [14] pointed out the following method:

$$\text{Replace } \prod_{j=1}^{\eta} \mathbf{e}(g_j, \prod_{i=1}^{m} u_i^{v_{ij}}) \text{ with } \prod_{i=1}^{m} \mathbf{e}(\prod_{j=1}^{\eta} {g_j}^{v_{ij}}, u_i)$$

In the full version of this paper [16], we apply this technique to schemes with structures related to the Waters hash; namely, the ring signatures of Boyen [13] and the aggregate signatures of Lu et al. [24].

3.3 Handling Invalid Signatures

If there is even a single invalid signature in the batch, then the batch verifier will reject the entire batch with high probability. In many real-world situations, a signature collection may contain invalid signatures caused by accidental data corruption, or possibly malicious activity by an adversary seeking to degrade service. In some cases, this may not be a serious concern. E.g., sensor networks with a high level of redundancy may choose to simply drop messages that cannot

be efficiently verified. Alternatively, systems may be able to cache and/or individually verify important messages when batch verification fails. Yet, in some applications, it might be critical to tolerate some percentage of invalid signatures without losing the performance advantage of batch verification.

In Section 5.2, we employ a recursive *divide-and-conquer* approach, similar to that of Pastuszak, Pieprzyk, Michalek and Seberry [25], as: First, shuffle the incoming batch of signatures, and if batch verification fails, simply divide the collection into two halves, and recurse on the halves. When this process terminates, the batch verifier outputs the index of each invalid signature. Through careful implementation and caching of intermediate results, much of the work of the batch verification (i.e., computing the product of many signature elements) can be performed once over the full signature collection, and need not be repeated when verifying each sub-collection. Thus, the cost of each recursion is dominated by the number of pairings used in the batch verification algorithm. In Section 5.2, we show that even if up to 15% of the signatures are invalid, this technique still performs faster than individual verification.

Recently, Law and Matt [15] proposed three new techniques for finding invalid signatures in a batch. One of their techniques, which is the most efficient for large batch sizes, allows to save approximately half the time needed by the simple divide-and-conquer approach. Thus, it is possible to do even better than the performance numbers we present.

4 Batch Verifiers for Short Signatures

Given the basic batching tools in the last section, it still requires creativity to figure out how best to apply them to batch any given scheme. In this section, we present new results for batch verifying a selection of existing regular, identity-based, group, ring, and aggregate signature schemes. To our knowledge, these are the first such verifiers for group, ring and aggregate signatures. After a search through the existing literature, we present the schemes with the best results.

Figure 1 shows a summary of our theoretical results, together with an indication of which batching techniques were used. Due to space limitations, we cannot describe the details of each scheme. Instead, we demonstrate one example in the Boneh, Boyen and Shacham [10] group signatures and then describe all remaining signatures and their batch verifiers in the full version of this paper [16].

4.1 Batching the Boneh-Boyen-Shacham (BBS) Group Signatures

This scheme does not appear to batch well *without* making some alterations, which increase the signature size by one group element, but where only 2 pairings are sufficient to batch an arbitrary number of signatures. A group signature scheme allows any member to sign on behalf of the group in such a way that anyone can verify a signature using the group public key while nobody, but the group manager, can identify the actual signer. A scheme consists of four algorithms: KeyGen, Sign, Verify and Open, that, respectively generate public

Scheme	Model	Individual-Verify	Batch-Verify	Reference	Techniques
Group Signatures					
BBS [10]	RO	5η	2	§4.1	1,2,3
BS [11]	RO	5η	2	[16]	1,2,3
ID-based Ring Signatures					
CYH [12]	RO	2η	2	[16]	2,3
Ring Signatures					
Boyen [13] (same ring)	plain	$\ell \cdot (\eta + 1)$	$\min\{\eta \cdot \ell + 1, 3 \cdot \ell + 1\}$	[16]	2,3,4
Signatures					
BLS [26]	RO	2η	$s + 1$	[26]	2,3
CHP [14] (time restrictions)	RO	3η	3	[14]	2,3
ID-based Signatures					
Hess [19]	RO	2η	2	[16]	1,2,3
ChCh [27]	RO	2η	2	[15]	2,3
Waters [20,21,28,23]	plain	3η	$\min\{(2\eta + 3), (z + 3)\}$	[14]	2,3,4
Aggregate Signatures					
BGLS [29] (same users)	RO	$\eta(\ell + 1)$	$\ell + 1$	[16]	2,3
Sh [30] (same users)	RO	$\eta(\ell + 2)$	$\ell + 2$	[16]	2,3
LOSSW [24] (same sequence)	plain	$\eta(\ell + 1)$	$\min\{(\eta + 2), (\ell \cdot k + 3)\}$	[16]	2,3,4

Fig. 1. Signatures with Efficient Batch Verifiers. Let η be the number of signatures to verify, s be the number of distinct signers involved and ℓ be either the size of a ring or the size of an aggregate. Boyen batch verifier requires each signature to be issued according to the same ring. Aggregate verifiers work for signatures related to the same set of users. In CHP, only signatures from the same time period can be batched and z is a (small) parameter (e.g., 8). In LOSSW, k is the message bit-length. RO stands for random oracle. The details of each scheme and its batch verifier are provided in the full version of this paper [16].

and private keys for users and the group manager, sign a message on behalf of a group, verify the signature on a message according to the group and trace a signature to a signer. For our purposes, we focus on the verification algorithm.

The Boneh-Boyen-Shacham (BBS) Group Signatures. Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$, where $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a hash function and there exists an efficiently-computable isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. Let ℓ be the number of users in a group.

Key Gen. Select a random $g_2 \in \mathbb{G}_2$ and sets $g_1 \leftarrow \psi(g_2)$. Select $h \xleftarrow{\$} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$, $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q^*$, and set u, v such that $u^{r_1} = v^{r_2} = h$. Select $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$, and set $w = g_2^\gamma$. For $i = 1$ to n, select $x_i \xleftarrow{\$} \mathbb{Z}_q^*$, and set $f_i = g_1^{1/(\gamma + x_i)}$. The public key is $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, the group manager's secret key is $\mathsf{gmsk} = (r_1, r_2)$ and the secret key of the i'th user is $\mathsf{gsk[i]} = (f_i, x_i)$.

Sign. Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a user private key (f, x) and a message $M \in \{0,1\}^*$, compute the signature σ as follows: Select $\alpha, \beta, r_\alpha, r_\beta, r_x, r_{\gamma_1}, r_{\gamma_2} \xleftarrow{\$} \mathbb{Z}_q$. Compute $T_1 = u^\alpha$; $T_2 = v^\beta$; $T_3 = f \cdot h^{\alpha+\beta}$, $\gamma_1 = x \cdot \alpha$ and $\gamma_2 = x \cdot \beta$, $R_1 = u^{r_\alpha}$; $R_2 = v^{r_\beta}$;$R_3 = \mathbf{e}(T_3, g_2)^{r_x} \cdot \mathbf{e}(h, w)^{-r_\alpha - r_\beta} \cdot \mathbf{e}(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}$; $R_4 = T_1^{r_x} \cdot u^{-r_{\gamma_1}}$; $R_5 = T_2^{r_x} \cdot v^{-r_{\gamma_2}}$. Compute $c = H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. Compute $s_\alpha = r_\alpha + c \cdot \alpha$; $s_\beta = r_\beta + c \cdot \beta$;

$s_x = r_x + c \cdot x$; $s_{\gamma_1} = r_{\gamma_1} + c \cdot \gamma_1$; $s_{\gamma_2} = r_{\gamma_2} + c \cdot \gamma_2$. The signature is $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$.

Verify. Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a message M and a group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, compute the values $R_1 = u^{s_\alpha} \cdot T_1^{-c}$, $R_2 = v^{s_\beta} \cdot T_2^{-c}$, $R_3 = \mathbf{e}(T_3, g_2)^{s_x} \cdot \mathbf{e}(h, w)^{-s_\alpha - s_\beta} \cdot \mathbf{e}(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \left(\mathbf{e}(T_3, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^c$, and $R_4 = T_1^{s_x} \cdot u^{-s_{\delta_1}}$; $R_5 = T_2^{s_x} \cdot v^{-s_{\delta_2}}$. Accept iff $c \stackrel{?}{=} H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$.

An Efficient Batch Verifier for BBS Group Signatures. Computing R_3 is the most expensive part of the verification above, but at first glance it is not clear that this can be batched, because each R_3 is hashed in the verification equation. However, as described by Technique 1, the signature and the verification algorithm can be modified at the expense of increasing the signature size by one element. Let $\sigma = (T_1, T_2, T_3, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$ be the new signature, together with:

New Individual Verify. Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a message M and a group signature $\sigma = (T_1, T_2, T_3, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, compute the values $R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c}$; $R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c}$; $R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\gamma_1}}$; $R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\gamma_2}}$, then check the following equation

$$\mathbf{e}(T_3, g_2)^{s_x} \cdot \mathbf{e}(h, w)^{-s_\alpha - s_\beta} \cdot \mathbf{e}(h, g_2)^{-s_{\gamma_1} - s_{\gamma_2}} \cdot \left(\mathbf{e}(T_3, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^c \stackrel{?}{=} R_3.$$

Finally check if $c \stackrel{?}{=} H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. Accept if all checks succeed, else reject.

Now we define a batch verifier, where the main objective is to use a *constant* number of pairings.

Batch Verify. Let $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ be the group public key, and let $\sigma_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,3}, c_j, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ be the j'th signature on the message M_j, for each $j = 1, \ldots, \eta$. For each $j = 1, \ldots, \eta$, compute the following values:

$$R_{j,1} \leftarrow u^{s_{j,\alpha}} \cdot T_{j,1}^{-c_j} \qquad R_{j,2} \leftarrow v^{s_{j,\beta}} \cdot T_{j,2}^{-c_j}$$
$$R_{j,4} \leftarrow T_{j,1}^{s_{j,x}} \cdot u^{-s_{j,\gamma_1}} \qquad R_{j,5} \leftarrow T_{j,2}^{s_{j,x}} \cdot v^{-s_{j,\gamma_2}}$$

Now for each $j = 1, \ldots, \eta$, check that $c_j \stackrel{?}{=} H(M_j, T_{j,1}, T_{j,2}, T_{j,3}, R_{j,1}, R_{j,2}, R_{j,3}, R_{j,4}, R_{j,5})$. Then check the following *single* pairing based equation

$$\mathbf{e}(\prod_{j=1}^{\eta} (T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1} - s_{j,\gamma_2}} \cdot g_1^{-c_j})^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta} (h^{-s_{j,\alpha} - s_{j,\beta}} \cdot T_3^c)^{\delta_j}, w) \stackrel{?}{=} \prod_{j=1}^{\eta} R_{j,3}^{\delta_j}.$$

where $(\delta_1, \ldots, \delta_\eta)$ is a random vector of ℓ_b bit elements from $\mathbb{Z}_q$. Accept iff all checks succeed.

Theorem 2. *For security level ℓ_b, the above algorithm is a batch verifier for the BBS group signature scheme, where the probability of accepting an invalid signature is $2^{-\ell_b}$.* (Proof of this theorem appears in the full version of this paper [16].)

5 Implementation and Performance Analysis

The previous work on batching short signatures [14] considers only asymptotic performance. Unfortunately, this "paper analysis" conceals many details that are revealed only through empirical evaluation. Additionally, the existing work does not address how to handle invalid signatures.

We seek to answer these questions by conducting the first empirical investigation into the feasibility of short signature batching. To conduct our experiments, we built concrete implementations of seven signature schemes described in this work, including two public key signature schemes (BLS, CHP), three Identity-Based Signature schemes (ChCh, Hess, Waters), a ring signature (CYH), and a short group signature scheme (BBS). *For each scheme, we measured the performance of the individual verification algorithm against that of the corresponding batch verifier.* We then turned our attention to the problem of efficiently sorting out invalid signatures.

Experimental Setup. To evaluate our batch verifiers, we implemented each signature scheme in C++ using the MIRACL library for elliptic curve operations [31]. Our timed experiments were conducted on a 3.0Ghz Pentium D 930 with 4GB of RAM running Linux Kernel 2.6. All hashing was implemented using SHA1,[1] and small exponents were of size 80 bits. For each scheme, our basic experiment followed the same outline: (1) generate a collection of η distinct signatures on 100-byte random message strings. (2) Conduct a timed verification of this collection using the batch verifier. (3) Repeat steps $(1, 2)$ four times, averaging to obtain a mean timing. To obtain a view of batching efficiency on collections of increasing size, we conducted the preceding test for values of η ranging from 1 to approximately 400 signatures in intervals of 20. Finally, to provide a baseline, we separately measured the performance of the corresponding *non-batched* verification, by verifying 1000 signatures and dividing to obtain the average verification time per signature. A high-level summary of our results is presented in Figure 3.

Curve	k	$\mathcal{R}(\mathbb{G}_1)$	$\mathcal{R}(\mathbb{G}_T)$	$\mathcal{S}_{RSA}$	Pairing Time
MNT160	6	160 bits	960 bits	960 bits	23.3 ms
MNT192	6	192 bits	1152 bits	1152 bits	33.2 ms
SS512	2	512 bits	1024 bits	957 bits	16.7 ms

Fig. 2. Description of the elliptic curve parameters used in our experiments. $\mathcal{R}(\cdot)$ describes the approximate number of bits to optimally represent a group element. $\mathcal{S}_{RSA}$ is an estimate of "RSA-equivalent" security derived via the approach of Page et al. [32].

[1] We selected SHA1 because the digest size closely matches the order of $\mathbb{G}_1$. One could use other hash functions with a similar digest size, *e.g.,* RIPEMD-160, or truncate the output of a hash function such as SHA-256 or Whirlpool. Because the hashing time is negligible in our experiments, this should not greatly impact our results.

Scheme	*Signature Size (bits)* MNT160	MNT192	SS512	*Individual Verification* MNT160	MNT192	SS512	*Batched Verification** MNT160	MNT192	SS512
Signatures									
BLS (single signer)	160	192	512	47.6 ms	77.8 ms	52.3 ms	2.28 ms	2.93 ms	32.42 ms
CHP	160	192	512	73.6 ms	119.0 ms	93.0 ms	26.16 ms	34.66 ms	34.50 ms
BLS cert + CHP sig	1280	1536	1536	121.2 ms†	196.8 ms†	145.3 ms†	28.44 ms†	37.59 ms†	66.92 ms†
Identity-Based Signatures									
ChCh	320	384	1024	49.1 ms	79.7 ms	73.3 ms	3.93 ms	5.24 ms	59.45 ms
Waters	480	576	1536	91.2 ms	138.64 ms	61.1 ms	9.44 ms	11.49 ms	59.32 ms
Hess	1120	1344	1536	49.1 ms	79.0 ms	73.1 ms	6.70 ms	8.72 ms	55.94 ms
Anonymous Signatures									
BBS (modified per §[16])	2400	2880	3008	139.0 ms	218.3 ms	193.0 ms	24.80 ms	34.18 ms	198.03 ms
CYH, 2-member ring	480	576	1536	52.0 ms	77.0 ms	113.0 ms	6.03 ms	8.30 ms	105.69 ms
CYH, 20-member ring	3360	4032	10752	86.5 ms	126.8 ms	829.3 ms	43.93 ms	61.47 ms	932.66 ms

*Average time per verification when batching 200 signatures.
†Values were derived by manually combining data from BLS and CHP tests.

Fig. 3. Summary of experimental results. Timing results indicate verification time *per signature*. With the exception of BLS, our experiments considered signatures generated by distinct signers.

Curve Parameters. The selection of elliptic curve parameters impacts both signature size and verification time. The two most important choices are the size of the underlying finite field $\mathbb{F}_p$, and the curve's embedding degree k. Due to the MOV attack, security is bounded by the size of the associated finite field $\mathbb{F}_{p^k}$. Simultaneously, the representation of elements $\mathbb{G}_1$ requires approximately $|p|$ bits. Thus, most of the literature on short signatures recommends choosing a relatively small p, and a curve with a high value of k. (For example, an MNT curve with $|p| = 192$ bits and $k = 6$ is thought to offer approximately the same level of security as 1152-bit RSA [32].) The literature on short signatures focuses mainly on signature size rather than verification time, so it is easy to miss the fact that using such high-degree curves *substantially* increases the cost of a pairing operation, and thus verification time. To incorporate these effects into our results, we implemented our schemes using two high-degree ($k = 6$) MNT curves with $|p|$ equal to 160 bits and 192 bits. For completeness, we also considered a $|p|$=512 bit supersingular curve with embeddeing degree $k = 2$, and a subgroup $\mathbb{G}_1$ of size 2^{160}. Figure 2 details the curve choices along with relevant details such as pairing time and "RSA-equivalent" security determined using the approach of Page et al. [32].

5.1 Performance Results

Public-Key signatures. Figure 4 presents the results of our timing experiments for the public-key BLS and CHP verifiers. Because the BLS signature does not batch efficiently for messages created by *distinct* signers, we studied the combination suggested in [14], where BLS is used for certificates which are created by a single master authority, and CHP is used to sign the actual messages under users' individual signing keys. Unfortunately, the CHP batch verifier appears to be quite costly in the recommended MNT curve setting. This outcome stems from the requirement that user public keys be in the $\mathbb{G}_2$ subgroup. This

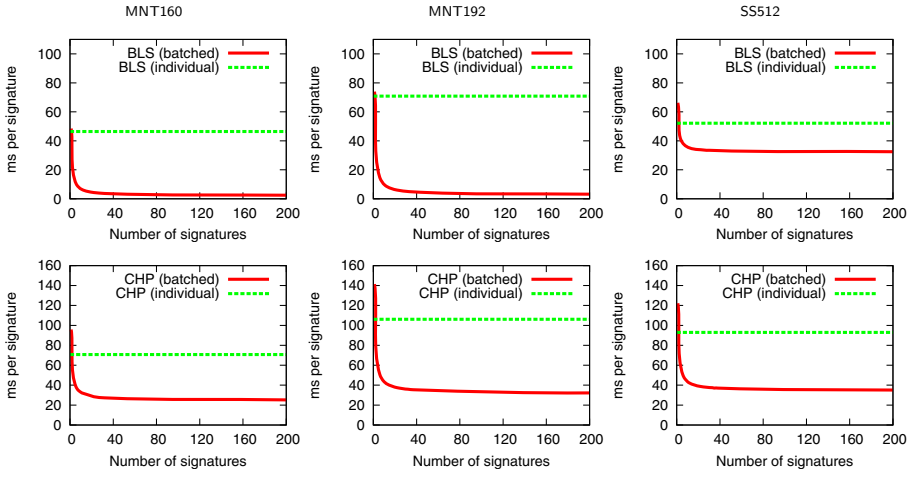

Fig. 4. Public-Key Signature Schemes. Per-signature times were computed by dividing total batch verification time by the number of signatures verified. Note that in the BLS case, all signatures are formulated by the same signer (as for certificate generation), while for CHP each signature was produced by a different signer. Individual verification times are included for comparison.

necessitates expensive point operations in the curve defined over the extension field, which undoes *some* of the advantage gained by batching. However, batching still reduces the per-signature verification cost to as little as 1/3 to 1/4 that of individual verification.

Identity-Based signatures. Figure 5 gives our measurements for three IBS schemes: ChCh, Waters and Hess. (For comparison, we also present CHP signatures with BLS-signed public-key certificates.) In all experiments, we consider signatures generated by different signers. In contrast with regular signatures, the IBSes batch quite efficiently, at least when implemented in MNT curves. The Waters scheme offers strong performance for a scheme not dependent on random oracles.[2] In our implementation of Waters, we first apply a SHA1 to the message, and use the Waters hash parameter $z = 5$ which divides the resulting 160-bit digest into blocks of 32 bits (as in [21]).

Anonymous signatures. Figure 6 gives our results for two privacy-preserving signatures: the CYH ring signature and the modified BBS group signature. As is common with ring signatures, in CYH both the signature size and verification time grow linearly with the number of members in the ring. For our experiments we arbitrarily selected two cases: (1) where all signatures are formed under a 2-member ring (useful for applications such as lightweight email signing [33]), and

[2] However, it should be noted that Waters has a somewhat loose security reduction, and may therefore require larger parameters in order to achieve security comparable to alternative schemes.

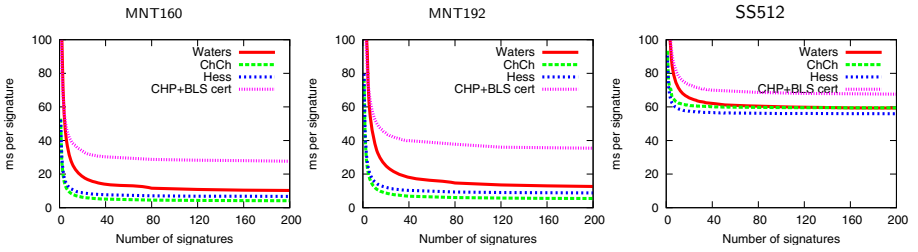

Fig. 5. Identity-Based Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. "CHP+BLS cert" represents the batched public-key alternative using certificates, and is included for comparison.

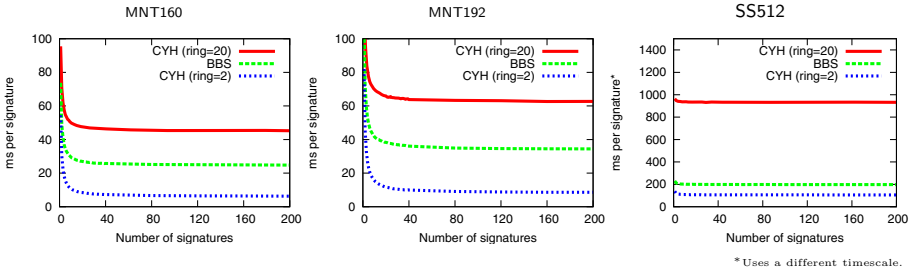

Fig. 6. Anonymous Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. For the CYH ring signature, we consider two distinct signature collections, one consisting of 2-member rings, and another with 20-member rings. The BBS signature verification is independent of the group size.

(2) where all signatures are formed using a 20-member ring.[3] In contrast, both the signature size and verification time of the BBS group signature are independent of the size of the group. This makes group signatures like BBS significantly more practical for applications such as vehicle communication networks, where the number of signers might be quite large.

5.2 Batch Verification and Invalid Signatures

In Section 3.3, we discuss techniques for dealing with invalid signatures. When batch verification fails, this *divide-and-conquer* approach recursively applies the batch verifier to individual halves of the batch, until all invalid invalid signatures have been located. To save time when recursing, we compute products of the form $\prod_{i=1}^{\eta} x_i^{\delta_i}$ so that partial products will be in place for each subset on which me might recurse. We accomplish this by placing each $x_i^{\delta_i}$ at the leaf of a binary tree and caching intermediate products at each level. This requires no additional

[3] Although the CYH batch verifier can easily batch signatures formed over differently-sized rings, our experiments use a constant ring size. Our results are representative of any signature collection where the *mean* ring size is 20.

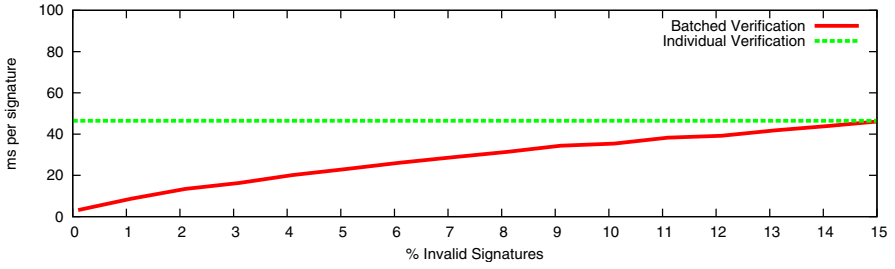

Fig. 7. BLS batch verification in the presence of invalid signatures (160-bit MNT curve). A "resilient" BLS batch verifier was applied to a collection of 1024 purported BLS signatures, where some percentage were randomly corrupted. Per-signature times were computed by dividing the total verification time (including identification of invalid signatures) by the total number of signatures (1024), and averaging over multiple experimental runs.

computation, and total storage of approximately 2η group elements for each product to be computed.

To evaluate the feasibility of this technique, we used it to implement a "resilient" batch verifier for the BLS signature scheme. This verifier accepts as input a collection of signatures where some may be invalid, and outputs the index of each invalid signature found. To evaluate batching performance, we first generated a collection of 1024 valid signatures, and then randomly corrupted an r-fraction by replacing them with random group elements. We repeated this experiment for values of r ranging from 0 to 15% of the collection, collecting multiple timings at each point, and averaging to obtain a mean verification time. The results are presented in Figure 7.

Batched verification of BLS signatures is preferable to the naïve individual verification algorithm even as the number of invalid signatures exceeds 10% of the total batch size. The random distribution of invalid signatures within the collection is nearly the worst-case for resilient verification. In practice, invalid signatures might be grouped together within the batch (e.g., if corruption is due to a burst of EM interference). In this case, the verifier might achieve better results by omitting the random shuffle step or using another re-ordering technique.

6 Conclusion and Open Problems

Our experiments provide strong evidence that batching short signatures is practical, even in a setting where an adversary can inject invalid signatures. We present new algorithms for batching a host of short signature schemes, including the first such verifiers for group, ring and aggregate signatures. At a deeper level, our results indicate that efficient batching depends heavily on the underlying design of a signature scheme, particularly on the placement of elements within the elliptic curve subgroups. For example, the CHP signature and the ChCh IBS have comparable size and security, yet the latter scheme can batch more than

250 signatures per second (each from a different signer), while our CHP implementation clocks in at fewer than 40. Designers should take these considerations into account when proposing new pairing-based signature schemes.

It remains open to batch verify a group signature scheme without random oracles. While many candidate schemes exist, it is not clear how to batch verify them. It also remains open to verify a batch of very short signatures (one group element) in *constant* pairings without the time-period restriction used by Camenisch et al. [14], even with random oracles.

Acknowledgments

Anna Lisa Ferrara and Matthew Green performed part of this work while at the Johns Hopkins University. Matthew Green and Susan Hohenberger were supported by the NSF under grant CNS-0716142 and a Microsoft New Faculty Fellowship. Michael Østergaard Pedersen performed part of this research while at the University of Aarhus.

References

1. Car 2 Car: Communication consortium, `http://car-to-car.org`
2. SeVeCom: Security on the road, `http://www.sevecom.org`
3. Raya, M., Hubaux, J.-P.: Securing vehicular ad hoc networks. J. of Computer Security 15, 39–68 (2007)
4. Gennaro, R., Rohatgi, P.: How to sign digital streams. Inf. Comput. 165(1), 100–116 (2001)
5. Lysyanskaya, A., Tamassia, R., Triandopoulos, N.: Multicast authentication in fully adversarial networks. In: IEEE Security and Privacy, pp. 241–253 (2004)
6. Perrig, A., Canetti, R., Song, D.X., Tygar, J.D.: Efficient and secure source authentication for multicast. In: NDSS 2001, The Internet Society (2001)
7. Monnerat, J., Vaudenay, S.: Undeniable signatures based on characters: How to sign with one bit. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 69–85. Springer, Heidelberg (2004)
8. Monnerat, J., Vaudenay, S.: Short 2-move undeniable signatures. In: Nguyên, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 19–36. Springer, Heidelberg (2006)
9. 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)
10. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
11. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: CCS, pp. 168–177 (2004)
12. Chow, S.S.M., Yiu, S.-M., Hui, L.C.K.: Efficient identity based ring signature. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 499–512. Springer, Heidelberg (2005)
13. Boyen, X.: Mesh signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 210–227. Springer, Heidelberg (2007)
14. Camenisch, J.L., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007), `http://eprint.iacr.org/2007/172`

15. 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. 34–53. Springer, Heidelberg (2007)
16. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification, Cryptology ePrint Archive: Report 2008/015 (2008)
17. Chen, L., Cheng, Z., Smart, N.: Identity-based key agreement protocols from pairings, Cryptology ePrint Archive: Report 2006/199 (2006)
18. 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)
19. Hess, F.: Efficient identity based signature schemes based on pairings. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 310–324. Springer, Heidelberg (2003)
20. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
21. Naccache, D.: Secure and practical identity-based encryption, Cryptology ePrint Archive: Report 2005/369 (2005)
22. Chatterjee, S., Sarkar, P.: Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 424–440. Springer, Heidelberg (2006)
23. Chatterjee, S., Sarkar, P.: HIBE with short public parameters without random oracle. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 145–160. Springer, Heidelberg (2006)
24. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
25. Pastuszak, J., Michatek, 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)
26. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
27. Cha, J.C., Cheon, J.H.: 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)
28. Boyen, X., Waters, B.: Compact group signatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 427–444. Springer, Heidelberg (2006)
29. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
30. Shao, Z.: Enhanced aggregate signatures from pairings. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 140–149. Springer, Heidelberg (2005)
31. Scott, M.: Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). Published by Shamus Software Ltd. (October 2007), `http://www.shamus.ie/`
32. Page, D., Smart, N., Vercauteren, F.: A comparison of MNT curves and supersingular curves. Applicable Algebra in Eng. Com. and Comp. 17(5), 379–392 (2006)
33. Adida, B., Chau, D., Hohenberger, S., Rivest, R.L.: Lightweight email signatures (Extended abstract). In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 288–302. Springer, Heidelberg (2006)