On the Practicality of Short Signature Batch Verification

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

[1] The Johns Hopkins University
{alferrar,mgreen,susan}@cs.jhu.edu
[2] Università di Salerno
ferrara@dia.unisa.it
[3] University of Aarhus
michael@daimi.au.dk

Abstract. As pervasive communication becomes a reality, where everything from vehicles to heart monitors constantly communicate with their environments, system designers are facing a cryptographic puzzle on how to authenticate messages. These scenarios require that : (1) cryptographic overhead remain short, and yet (2) many messages from many different signers be verified very quickly. Pairing-based signatures have property (1) but not (2), whereas schemes like RSA have property (2) but not (1). As a solution to this dilemma, in Eurocrypt 2007, Camenisch, Hohenberger and Pedersen showed how to *batch verify* two pairing-based signatures so that the total number of pairing operations was independent of the number of signatures to verify. CHP left open the task of batching *privacy-friendly* authentication, which is desirable in many pervasive communication scenarios.

In this work, we revisit this issue from a more practical standpoint and present the following results:
1. We describe a framework, consisting of general techniques, to help scheme and system designers understand how to *securely* and *efficiently* batch the verification of pairing equations.
2. We present a detailed study of when and how our framework can be applied to existing regular, identity-based, group, ring, and aggregate signature schemes. To our knowledge, these batch verifiers for group and ring signatures are the first proposals for batching privacy-friendly authentication, answering an open problem of Camenisch et al.
3. While prior work gave mostly asymptotic efficiency comparisons, we show that our framework is practical by implementing our techniques and giving detailed performance measurements. Additionally, we discuss how to deal with invalid signatures in a batch and our empirical results show that when $\leq 10\%$ of signatures are invalid, batching remains more efficient that individual verification.

Indeed, our results show that batch verification for short signatures is an effective, efficient approach.

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 [15, 38]. 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:

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 [35].
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 [35], which in turn may retransmit these messages. Thus, it is much more critical that authentications be quick to verify rather than to generate.

⋆ Supported by NSF grant CT-0716142.

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 [23, 27], or to replace them with symmetric MACs [34]. Each approach has significant drawbacks; for example, 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 other desirable features such as 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. It is interesting to note that the short, undeniable signatures of Monnerat and Vaudenay [29, 30] support a form of batch verification. However, these are inappropriate for the pervasive settings we consider, since verification is not universal and 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., [7, 11, 20, 21]. Unfortunately, the focus was on reducing the signature size, but less attention was paid to the verification cost of these schemes which require expensive pairing operations.

Recently, Camenisch, Hohenberger and Pedersen [13] 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:

1. First and foremost, their work, in common with the other cryptography literature we found on this subject, was theoretical. To our knowledge, we are the first to provide a detailed empirical analysis of batch verification of short signatures, and to answer the questions that system implementors *need* to have answered to deploy these schemes. For example, Camenisch et al. [13] stated, for one of the signature schemes they considered (referred to here as Waters), that batching requires fewer pairings once ≥ 3 signatures are collected. To reduce their total pairings, however, they [13] added other operations, such as random number generation and small modular exponentiations, so it was unclear how well their algorithm would perform in practice. Fortunately, in section 5, we verified that their complete batching algorithm *is* more efficient for ≥ 3 Waters signatures using a standard implementation.
2. Second, the existing theoretical literature contained many good ideas on batch verification, but these ideas were scattered across multiple papers, and it wasn't always clear how to safely employ these techniques from scheme to scheme. In section 3, we present a general framework for how to securely batch verify a set of pairing-based equations.
3. Third, we present a detailed study of when and how our framework can be applied to existing regular, identity-based, group, ring, and aggregate signature schemes in section 4. To our knowledge, these are the *first* known results for batch verification of group and ring signatures, answering an open problem of Camenisch et al. [13]. This is particularly exciting, because it is the first step towards making short, *privacy-friendly* authentication fast enough for deployment in real systems.
4. Finally, Camenisch et al. [13] 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 $\leq 10\%$ of the signatures are invalid, then batch verification is still more efficient than individual verification.

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 [7], 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 1).

	Approx. Signature Size (MNT160 curve)	Verification Time Standard	Batched*
Signatures			
BLS [10] (single signer)	160 bits	47.6 ms	2.28 ms
CHP [13] (many signers)	160 bits	73.6 ms	26.16 ms
Identity-Based Signatures			
ChCh [16]	320 bits	49.1 ms	3.93 ms
Waters [40]	480 bits	91.2 ms	9.44 ms
Hess [24]	1120 bits	49.1 ms	6.70 ms
Anonymous Signatures			
BBS [7] Group signature (modified per §4.1)	2880 bits	139.0 ms	24.80 ms
CYH [20] Ring signature, 2-member ring	480 bits	52.0 ms	6.03 ms
CYH [20] Ring signature, 20-member ring	3360 bits	86.5 ms	43.93 ms

*Verification time *per signature* when batching 200 signatures.

Fig. 1. Cryptographic overhead and verification time for some of the pairing-based signatures described in this work. For this summary table, all schemes were implemented in a 160-bit MNT elliptic curve. See section 4 for a description of all signature schemes considered and section 5 for full experimental results.

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 the general case, called the *asymmetric* setting. A specialized case is the *symmetric* setting, where $\mathbb{G}_1 = \mathbb{G}_2$. We will always write group elements in the multiplicative notation, although the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are actually implemented as additive groups.

Size of Group Elements. Pairings are constructed such that $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on some elliptic curve E, and $\mathbb{G}_T$ is a subgroup of a multiplicative group over a related finite field. All groups have order q. The group of points on E defined over $\mathbb{F}_p$ is written as $E(\mathbb{F}_p)$. Usually it is the case that $\mathbb{G}_1$ is a subgroup of $E(\mathbb{F}_p)$, $\mathbb{G}_2$ is a subgroup of $E(\mathbb{F}_{p^k})$ where k is the embedding degree, and $\mathbb{G}_T$ is a subgroup of $\mathbb{F}^*_{p^k}$. In the symmetric case $\mathbb{G}_1 = \mathbb{G}_2$ is usually a subgroup of $E(\mathbb{F}_p)$.

In the following, we use numbers for security comparable to 1024 bit RSA. The MOV attack by Menezes, Vanstone and Okamoto states that solving the discrete logarithm problem on a curve reduces to solving it over the corresponding finite field [28]. Hence the size of p^k must be comparable to that of an RSA modulus to provide the same level of security, so elements of $\mathbb{F}_{p^k}$ must be of size 1024. But the size of the finite field is not the only thing that matters for security. The group order

q must also be large enough to resist the Pollard-ρ attack on discrete logarithms, which means that $q \geq 160$. Now assume that $|p| = |q| = 160$, then we would need an embedding degree $k = 6$ to get the size of the corresponding field close to the required 1024 bits. However, we could also let $|q| = 160$, $|p| = 512$, and choose $k = 2$ to achieve the same. Both these options have their advantages and disadvantages as discussed by Koblitz and Menezes [25].

We have talked about how many bits are required to represent elements in the finite fields, but what about the groups $\mathbb{G}_1$ and $\mathbb{G}_2$? Since they are subgroups of a curve over the field, they are represented by their coordinates (x, y) which are elements of the field, and hence one would expect their size to be twice the size of an element in the field. However one only needs to represent x and the LSB of y in order to recompute y later. Also, in some cases (when $\mathbb{G}_2$ is the trace zero subgroup) elements of $\mathbb{G}_2$ can be represented as elements of the field $E(\mathbb{F}_{p^{k/2}})$ instead, which would only require half the space [25].

To summarize: 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 $k = 2$, which means that elements of $\mathbb{G}_1 = \mathbb{G}_2$ and $\mathbb{G}_T$ will be of size 512 and 1024 bits respectively. Finally, an important thing to keep in mind is that no matter the order of the groups, performance is dominated by the operations in the underlying finite field.

From Symmetric to Asymmetric. If one wants to go from the symmetric to the asymmetric setting to take advantage of the small group elements in $\mathbb{G}_1$, there are a few pitfalls one should be aware of. In some asymmetric groups it is not possible to hash into $\mathbb{G}_2$, but in these groups there exist a isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. In other groups there is no such isomorphism, but it is possible to hash into $\mathbb{G}_2$. So if a scheme requires both for the security proof, that scheme cannot be realized in the asymmetric setting. Galbraith, Paterson and Smart have a more detailed discussion [22].

Testing Membership. The small exponents test assumes that the input lies in a specific group, but since the input might be supplied by the adversary we cannot trust this to hold. We have to check it. Our proofs will require that elements of purported signatures are members of $\mathbb{G}_1$ and *not* $E(\mathbb{F}_p) \backslash \mathbb{G}_1$, 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. Recall that $\mathbb{G}_1$ is a subgroup of $E(\mathbb{F}_p)$ of order q, so we can use standard cofactor multiplication to test group membership. The curve has hq points over $\mathbb{F}_p$, so if an element y satisfies the curve equation and $y^h \neq 0$, then that element is in $\mathbb{G}_1$. If h is small then this test is efficient, otherwise we have to test $y^q \neq 0$ instead. Chen, Cheng and Smart [19] discuss this and ways to test membership in $\mathbb{G}_2$; unfortunately, in many asymmetric pairings, it may not be possible to efficiently test for membership of $\mathbb{G}_2$.

3 A Framework for Pairing-Based Batch Verification

We now provide some useful observations to determine when pairing equations can be batch verified. Let us begin with a formal definition of *pairing based* batch verifier.

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. Next definition describes a batch verifier for pairing-based claims.

Definition 1 (Pairing-based Batch Verifier). *Let* $\mathsf{PSetup}(1^\tau) \to (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$. *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* pairing-based batch verifier *for* Verify *a probabilistic poly(τ)-time algorithm which outputs* accept *if* $X^{(j)}$ *holds for all* $j \in [1, \eta]$ *whereas it outputs* reject *if* $X^{(j)}$ *does not hold for any* $j \in [1, \eta]$ *except with negligible probability.*

Note that equations from many different pairing-based schemes can be batched together as long as they all share the same setting defined by $\mathsf{PSetup}(1^\tau)$.

Small Exponents Test Applied to Pairing Based Schemes. 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 [4]. 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 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 $2^{-\ell}$. The size of ℓ is a tradeoff between efficiency and security. (In Section 5, we set $\ell = 80$ bits.)

Theorem 1. *Let* $\mathsf{PSetup}(1^\tau) \to (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$ *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* $2^{-\ell_b}$.

Proof. The proof closely follows the proof of the small exponents test by Bellare et al. [4], but for completeness we include a full proof of this theorem in Appendix A.[4]

Theorem 1 provides a *single* verification equation, which we then want to optimize.

3.1 Techniques to Speed Up Batch Verification

Armed with Theorem 1, let's back up for a moment to get a complete picture of how to develop an efficient batch verifier. Immediately following the summary, we'll explain the details.
Framework Summary: Suppose you have η bilinear equations, to batch verify them, do the following:

[4] A natural question to ask is if this batch verifier also works for composite order groups. Unfortunately the answer is not straightforward. The reason for requiring a prime order group, is that for the proof of the small exponents test to go through, we need an element β_1 to have an inverse in $\mathbb{Z}_q$, which is the case if $gcd(\beta_1, q) = 1$. If q is prime this is always the case, but what if q is composite? If $q = p_1 p_2$, where p_1, p_2 are primes, then this is the case except when β_1 is a multiple of p_1, p_2 or q. If β_1 is chosen at random it is very unlikely that an inverse does not exist, and the small exponents test will work in almost all cases. However, this really depends on the signature scheme, so if one wants to apply this method to a scheme set in a composite order group, one should examine the proof in Appendix A and make sure that it still applies to the chosen scheme.

1. Apply Technique 1 to the individual verification equation, if applicable.
2. Apply Theorem 1 to the equations, this involves checking membership in the expected algebraic groups and using the small exponents test.
3. Optimize the resulting equation using Techniques 2, 3 and 4.
4. If batch verification fails, use the divide-and-conquer approach to identity the bad signatures.

Technique 1 *Change the verification equation.* 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 any 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 scheme [24] and the group signatures of Boneh, Boyen and Shacham [7] 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 once and for all.
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.

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

Remember that it is also possible to move an exponent out of the pairing, or move it between the two elements of the pairing. In some instances, this allows for further optimizations.

Technique 3 *When two pairings with a common first or second element appear, they can be combined.* A simple example could be the following:

$$\text{Replace } \mathbf{e}(a, g) \cdot \mathbf{e}(b, g) \text{ with } \mathbf{e}(ab, g)$$

When applying the batching technique from Theorem 1 to verify η equations, one will often end up with an equation that can be optimized using this technique. It will work like this:

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

When batching η instances using Theorem 1 this will reduce η pairings to one. This is also worth keeping in mind when designing schemes, or picking schemes that one wants to batch verify. Pick a scheme so that when $\mathbf{e}(g,h)$ appears in the verification equation, g or h is fixed. In rare cases it might even be useful to apply this technique "in reverse", e.g. splitting a single pairing into two pairings, to allow for more efficient batch verification. An example is the ring signature scheme by Boyen [21] where this is needed to apply Technique 4 below.

Technique 4 *Waters hash.* In his IBE, Waters described how hash identities to values in $\mathbb{G}_1$ [40], using a technique that was subsequently employed in several signature schemes. Assume the identity is a bit string $V = v_1v_2 \ldots v_m$, then given public parameters $u', u_1, \ldots, u_m \in \mathbb{G}_1$, the hash is $u' \prod_{i=1}^{m} u_i^{v_i}$. Following works by Naccache [31] and Chatterjee and Sarkar [17, 18] documented the generalization where instead of evaluating the identity bit by bit, divide the k bit identity bit string into z blocks, and use the Waters hash as before. (In section 5, we SHA1 hash our messages to a 160-bit string, and use $z = 5$ as proposed in [31].) Recently, Camenisch et al. [13] pointed out the following method for faster batching of Waters hashes.

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

In this work, we apply this technique to schemes with structures related to the Waters hash; namely, the ring signatures of Boyen [21] and the aggregate signatures of Lu et al. [26].

Handling Invalid Signatures. If there is even a single invalid signature in the batch, then the batch verifier will reject with high probability, but 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 many cases, the ratio of invalid signatures to valid could be quite small, and yet a standard batch verifier will reject the entire collection.

In some cases this may not be a serious concern. For example, 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. However, in some applications, it might be critical to tolerate some percentage of invalid signatures without losing the performance advantage of batch verification.

To our knowledge, the best known solution to this problem is to use a recursive *divide-and-conquer* approach, similar to that of Pastuszak, Pieprzyk, Michalek and Seberry [33], 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 10% of the signatures are invalid, this technique can still be faster than individual verification.

4 Applying the Framework to Signature Schemes

Next, we apply our framework to a (non-exhaustive) collection of existing regular, identity-based, group, ring, and aggregate signature schemes. After a careful literature search, we are presenting only the schemes with the best results (although we often make a note in the particular sections about common schemes that do not seem to batch well.) To our knowledge, our batch verifiers for the group and ring signatures are the first proposals for batching privacy-friendly authentication. Figure 2 shows a summary of our results.

Scheme	Model	Individual-Verify	Batch-Verify	Reference
Group Signatures				
BBS [7]	RO	5η	2	§4.1
BS [11]	RO	5η	2	§4.1
ID-based Ring Signatures				
CYH [20]	RO	2η	2	§4.2
Ring Signatures				
Boyen [21] (same ring)	plain	$\ell \cdot (\eta + 1)$	$\min\{\eta \cdot \ell + 1, 3 \cdot \ell + 1\}$	§4.2
Signatures				
BLS [10]	RO	2η	$s + 1$	[10]
CHP [13] (time restrictions)	RO	3η	3	[13]
ID-based Signatures				
Hess [24]	RO	2η	2	§4.2
ChCh [16]	RO	2η	2	§4.2
Waters [40, 31, 12, 18]	plain	3η	$\min\{(2\eta + 3), (z + 3)\}$	[13]
Aggregate Signatures				
BGLS [8] (same users)	RO	$\eta(\ell + 1)$	$\ell + 1$	§4.2
Sh [39] (same users)	RO	$\eta(\ell + 2)$	$\ell + 2$	§4.2
LOSSW [26] (same sequence)	plain	$\eta(\ell + 1)$	$\min\{(\eta + 2), (\ell \cdot k + 3)\}$	§4.2

Fig. 2. Summary of signatures schemes for which our framework applies. 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.

4.1 Short Group Signatures

In this section, we show how to modify the short group signatures of Boneh, Boyen and Shacham (BBS) [7] and Boneh and Shacham (BS) [11] in order to allow for a batch verifier which requires only 2 pairings at the expense of an increase in the signature size. Fortunately, however, this increase in size still keeps the signatures shorter than their corresponding RSA-based counterparts. To our knowledge, these are the first known results for batch verification of group signatures.

Recall that 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 group signature scheme consists in four algorithm: KeyGen, Sign, Verify and Open, that, respectively generate public 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.

The 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. Let ℓ be the number of users in a group. Note that the BBS scheme requires a computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ since their definition of the SDH assumption is based on it, but unfortunately such an isomorphism does not exist for the MNT curves we use in Section 5. Boneh and Boyen recently gave a cleaner definition which doesn't require said isomorphism [6].

Key Gen. Select a generator $g_2 \in \mathbb{G}_2$ at random and set $g_1 \leftarrow \psi(g_2)$. 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 each $i = 1, \ldots, n$, select $x_i \xleftarrow{\$} \mathbb{Z}_q^*$, and set $f_i \leftarrow g_1^{\frac{1}{\gamma + x_i}}$. 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: (1) Select $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ and compute $T_1 \leftarrow u^{\alpha}; T_2 \leftarrow v^{\beta}; T_3 \leftarrow f \cdot h^{\alpha+\beta}$. (2) Compute $\gamma_1 \leftarrow x \cdot \alpha$ and $\gamma_2 \leftarrow x \cdot \beta$. (3) Select $r_\alpha, r_\beta, r_x, r_{\gamma_1}, r_{\gamma_2} \xleftarrow{\$} \mathbb{Z}_q$ and compute $R_1 \leftarrow u^{r_\alpha}$; $R_2 \leftarrow v^{r_\beta}$; $R_3 \leftarrow \mathbf{e}(T_3, g_2)^{r_x} \cdot \mathbf{e}(h, w)^{-r_\alpha - r_\beta} \cdot \mathbf{e}(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}$; $R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\gamma_1}}$; $R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\gamma_2}}$. (4) Compute $c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. (5) Compute $s_\alpha \leftarrow r_\alpha + c \cdot \alpha$; $s_\beta \leftarrow r_\beta + c \cdot \beta$; $s_x \leftarrow r_x + c \cdot x$; $s_{\gamma_1} \leftarrow r_{\gamma_1} + c \cdot \gamma_1$; $s_{\gamma_2} \leftarrow r_{\gamma_2} + c \cdot \gamma_2$. 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 $R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c}$; $R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c}$; $R_3 \leftarrow \mathbf{e}(T_3, g_2)^{s_x} \cdot \mathbf{e}(h, w)^{-s_\alpha - s_\beta} \cdot \mathbf{e}(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \left(\mathbf{e}(T_3, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^c$; $R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}}$; $R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}}$. Accept if and only if $c \stackrel{?}{=} H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$.

An Efficient Batch Verifier for the BBS *Group Signature Scheme.* Computing R_3 is the most expensive part of the verification above, but at first glance it isn't clear this can be batched, because each R_3 is individually hashed. However, as described by technique 1, the signature and the verification algorithm can be modified in order to efficiently apply our framework without comprise the security properties of the scheme at the expense of an increase in the signature size.

In particular, the signature is replaced by $\sigma = (T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$ and the verification algorithm becomes as follows:

New Verify. Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a message M and a group signature $\sigma = (T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, first compute $c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, then verify the validity of the following four (non-pairing) equations:

$$u^{s_\alpha} \stackrel{?}{=} T_1^c \cdot R_1, \quad v^{s_\beta} \stackrel{?}{=} T_2^c \cdot R_2, \quad T_1^{s_x} \cdot u^{-s_{\gamma_1}} \stackrel{?}{=} R_4, \quad T_2^{s_x} \cdot v^{-s_{\gamma_2}} \stackrel{?}{=} R_5.$$

and finally check the following pairing based equation

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

Accept if all checks succeed and reject otherwise.

Now we are ready to define a batch verifier for η BBS purported group signatures, where the main objective is to cut down on the number of pairings required.

BBS Batch Verify. Let $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ be the group public key, and let $\sigma_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,1}, R_{j,2}, R_{j,3}, R_{j,4}, R_{j,5}, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ be the j'th signature on the message M_j, for each $j = 1, \ldots, \eta$. For each $j = 1, \ldots, \eta$, first compute $c_j \leftarrow H(M_j, T_{j,1}, T_{j,2}, T_{j,3}, R_{j,1}, R_{j,2}, R_{j,3}, R_{j,4}, R_{j,5})$ and verify the validity of the following non-pairing equations:

$$u^{s_{j,x}} \stackrel{?}{=} T_{j,1}^{c_j} \cdot R_{j,1}, \quad v^{s_{j,\beta}} \stackrel{?}{=} T_{j,2}^{c_j} \cdot R_{j,2}, \quad T_{j,1}^{s_{j,x}} \cdot u^{-s_{j,\gamma_1}} \stackrel{?}{=} R_{j,4}, \quad T_{j,2}^{s_{j,x}} \cdot v^{-s_{j,\gamma_2}} \stackrel{?}{=} R_{j,5}.$$

Then check the following *single* pairing based equation

$$\mathbf{e}(\prod_{j=1}^{\eta}(T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1}-s_{j,\gamma_2}} \cdot g_1^{-c_j})^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta}(h^{-s_{j,\alpha}-s_{j,\beta}} \cdot T_3^{c})^{\delta_j}, w) \stackrel{?}{=} \prod_{j=1}^{\eta} R_{j,3}^{\delta_j}. \tag{2}$$

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 $2^{-\ell_b}$.*

A proof sketch of this theorem is in Appendix B.

The BS *Group Signatures.* As we point out in Figure 2, the (even shorter) group signatures of Boneh and Shacham [11] can also be batch verified using techniques similar to those above. The BS scheme includes an additional feature known as *verifier local revocation* (VLR), which allows verifiers to discard signatures from revoked signers. Unfortunately, the checks required to test for revoked signers cannot easily be batched. Thus, our batch verifier omits them. Since VLR is not a "traditional" property of a group signature (e.g., [2, 7]), we believe that the resulting batch verifier is still quite useful for applications where only a standard group signature is needed. Note that verifiers may still perform revocation checks using the non-batched Verify algorithm. Due to space limitations, we omit this description and move instead to an efficiency evaluation of both schemes.

Performance and Signature Length. The BBS batch verifier is suitable to verify many signatures issued by many group members on different messages. The original BBS signature consists of three elements of $\mathbb{G}_1$ and six elements of $\mathbb{Z}_q$ while its modified version, needed to construct the BBS batch verifier, requires seven elements of $\mathbb{G}_1$, one element of $\mathbb{G}_T$, and five elements of $\mathbb{Z}_q$. When implemented in the 170-bit MNT curve proposed by Boneh et al., this results in a signature representation of approximately 3,067 bits with security approximately equivalent to 1024-bit RSA. This is still shorter than the comparable (non-pairing) scheme of Ateniese, Camenisch, Joye and Tsudik [2] which achieves a similar security level at a cost of at least 3,872 bits. For applications where bandwidth is at a premium, it is desirable to use the extremely short group signature of Boneh and Shacham [11] which is suitable to construct a batch verifier that requires only two pairing operations. With appropriate modifications to permit batching, a BS signature results in four elements of $\mathbb{G}_1$, one element of $\mathbb{G}_T$, and four elements of $\mathbb{Z}_q$ which can be represented in 2,384 bits.

4.2 Other Types of Short Signatures

Ring and Identity-based Ring Signatures. In Appendix C, we show how to batch verify:

- The standard model ring signatures of Boyen (Boyen) [21] with the restriction that we can only batch ring signatures which have the same ring of ℓ signers using $\leq 3\ell + 1$ pairings.

- The random oracle model, identity-based ring signatures of Chow, Yiu and Hui (CYH) [20], where even rings of different sizes involving different ring members on different messages, can be batched using only 2 pairings.

Recall that a ring signature scheme allows a signer to sign a message on behalf of a set of users which include the signer itself in such a way that a verifier is convinced that the signer is one of the ring members, but he cannot tell which member is the actual signer. In an identity-based ring signature, a user can choose an arbitrary string, for example her email address, as her public key.

The CYH scheme is fairly straightforward to batch, while the Boyen scheme required more creativity, especially in the application of techniques 3 and 4.

Signature Schemes and Identity-based Signature Schemes. In Appendix D, we review the known batch verifiers [10, 13] for:

- The short, random oracle model signatures of Boneh, Lynn and Shacham (BLS) [10] for signatures by the *same* signer, which require 2 pairings to batch. (In section 5, we'll use this scheme to batch certificates.)
- The short, random oracle model signatures of Camenisch, Hohenberger and Pedersen (CHP) [13] for signatures by different signers within the *same* time period, which require 3 pairings to batch.
- The standard model, identity-based signatures, called Waters, which were implicitly defined by Waters [40] and then generalized by subsequent works [31, 12, 18]. These signatures can be batched using $\leq z + 3$ pairings, where z is a small security parameter (e.g., $z = 5$.)

We then present new results on batch verifiers, requiring only two pairings, for:

- The random oracle model, identity-based signatures of Cha and Cheon (ChCh) [16].
- The random oracle model, identity-based signatures of Hess (Hess) [24].

Interestingly, the ID-based signature due to Sakai, Ohgishi and Kasahara [36] is very similar to those above, and yet its subtle differences make it a poor candidate for batching.

Aggregate and Sequentially Aggregate Signatures. In Appendix E, we show batch verifiers for the aggregate signature scheme by Boneh, Gentry, Lynn and Shacham (BGLS) [8] (same users) and Shao (Shao) [39] (same users), and for the sequential aggregate scheme by Lu, Ostrovsky, Sahai, Shacham and Waters (LOSSW) [26] (same sequence).

5 Implementation and Performance Analysis

The previous work on batching short signatures [13] considers the asymptotic performance of several batch verifiers. Unfortunately, this "paper analysis" conceals many details that are revealed only through empirical evaluation. Additionally, the existing work does not address the most important practical issue facing system implementors, namely: how a batch verifier will perform in the face of adversarial behavior such as deliberate injection of 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 standard signature verification algorithm against that of the corresponding batch verifier. We then turned our attention to the problem of invalid signatures, constructing a "resilient" *divide-and-conquer* batch verifier which efficiently locates invalid signatures in a batch.

Our results lead to several surprising conclusions. First, we note that our batched Identity-Based signatures provide substantially better performance than standard (public-key) signatures, in the case where signatures are produced by different signers. This is due to a fluke of scheme construction, one that appears to stem from the related nature of IBS signing keys. Secondly, we observe that the "ideal" high-degree elliptic curve setting for short signatures (see section below) simultaneously implies both costly individual signature verification, *as well as* highly-efficient batch verifiers. Finally, we gather evidence indicating that "resilient" batch verification appears to be practical even in the presence of a substantial number of invalid signatures. The latter two results provide strong evidence for the practicality of batch verification in applications where short signatures and verification times are necessary.

Experimental Setup. To evaluate our batch verifiers, we implemented each signature scheme in C++ using the MIRACL library for elliptic curve operations [37]. 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,[5] 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 4.

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. 3. 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].

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

[5] We selected SHA1 because the digest size closely matches the order of $\mathbb{G}_1$. It would be possible to use alternative hash functions with a similar digest size, *e.g.,* RIPEMD-160, or to 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 §4.1)	2880	3456	5408	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. 4. Summary of experimental results. Timing results indicate verification time *per signature*. With the exception of BLS, our experiments considered signatures generated by distinct signers. The composite scheme "BLS cert + CHP sig" describes a BLS-signed certificate on a CHP public key, along with a CHP signature.

$|p|$ 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 3 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

We now present the results of our timing experiments. We first consider the two standard (public-key) signature schemes, followed by three Identity-Based alternatives. We then turn our attention to anonymous ring and group signatures. Finally, we evaluate the performance of a "resilient" batch verifier designed to verify efficiently in the presence of invalid signatures.

Public-Key signatures. Figure 5 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 considered these schemes in the combination suggested by Camenisch et al. [13], 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. Surprisingly, the CHP batch verifier appears to be quite costly in the recommended MNT curve setting. This result, which is not obvious from the high-level analysis of Camenisch et al., stems from the requirement that user public keys be in the $\mathbb{G}_2$ subgroup. This necessitates expensive point operations in the curve defined over the extension field, which undoes some of the advantage gained by batching. However, even with this limitation, batching reduces the per-signature verification cost to as little as 1/3 to 1/4 that of individual verification.

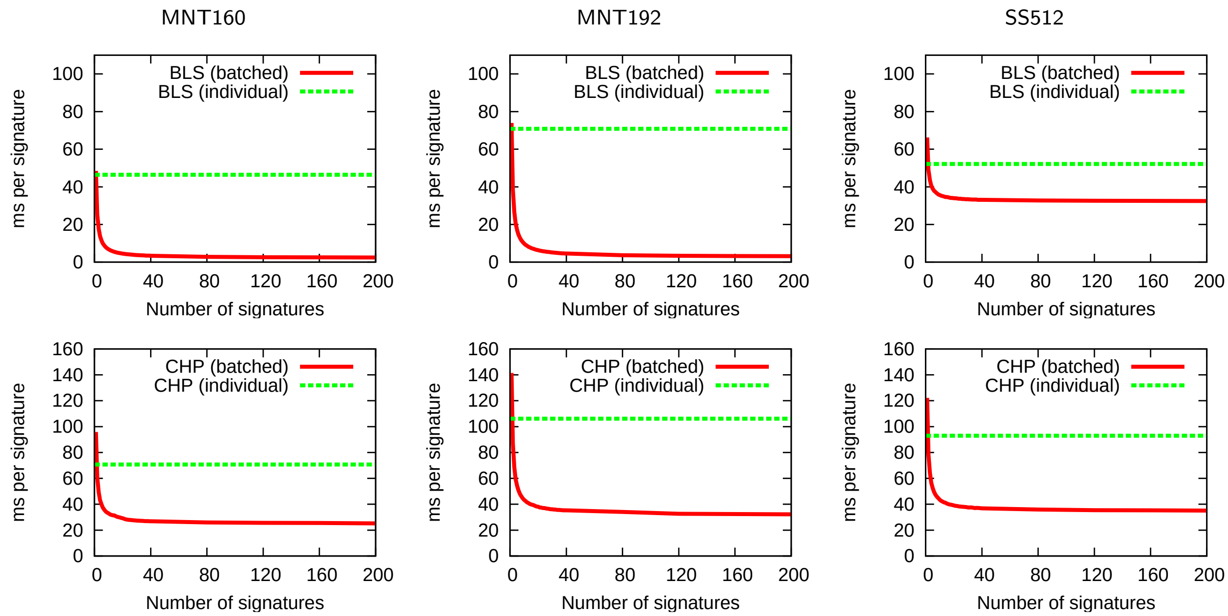

Fig. 5. Public-Key Signature Schemes. Per-signature times were computed by dividing total batch verification time by the number of signatures verified. Note that in the BLS case, all signatures are formulated by the same signer (as for certificate generation), while for CHP each signature was produced by a different signer. Individual verification times are included for comparison.

Identity-Based signatures. Figure 6 details the results of our timing experiments for three Identity-Based signature schemes, ChCh, Waters and Hess. (For comparison, our graphs also present the non-IBS approach employing CHP signatures with BLS-signed public-key certificates.) In all experiments we consider signatures generated by different signers. We observe that in contrast with the public-key schemes, the IBSes batch quite efficiently in this case, at least when implemented in MNT curves. In particular, the Waters scheme offers surprisingly strong performance for a scheme not dependent on random oracles.[6] Note that in 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 proposed in [31]). Because we selected these parameters, we did not bother to implement the first case of the batch verifier, since the appropriate condition applies only for batches of size $\eta \leq 3$.

Anonymous signatures. Figure 7 details the results of our timing experiments for two privacy-preserving signature schemes: 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

[6] 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.

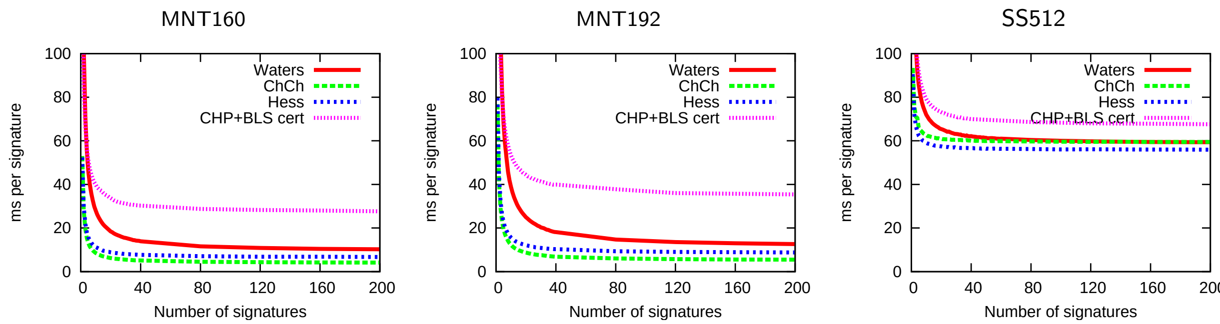

Fig. 6. Identity-Based Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. "CHP+BLS cert" represents the batched public-key alternative using certificates, and is included for comparison.

email signing [1]), and (2) where all signatures are formed using a 20-member ring.[7] 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.

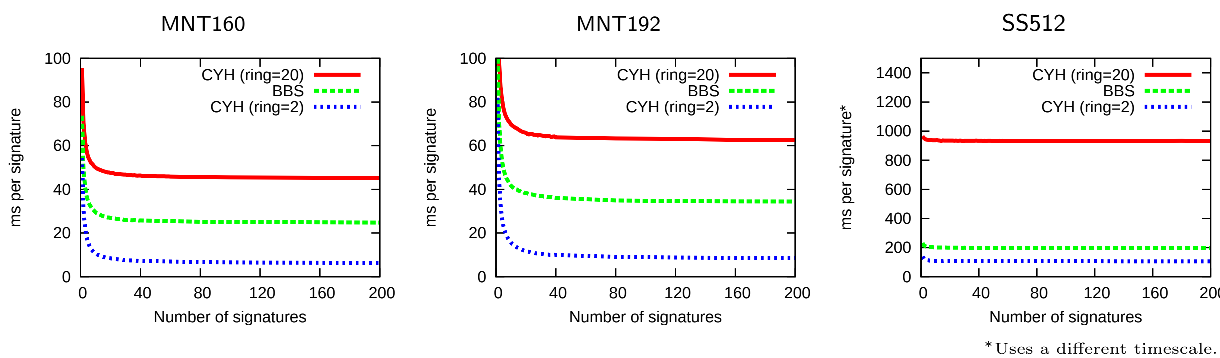

Fig. 7. Anonymous Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. For the CYH ring signature, we consider two distinct signature collections, one consisting of 2-member rings, and another with only 20-member rings. The BBS group signature verification is independent of the group size.

5.2 Batch Verification and Invalid Signatures

In section 3.1, we discuss a general technique for dealing with invalid signatures encountered when batching. When batch verification fails, this *divide-and-conquer* approach recursively applies the batch verifier to individual halves of the signature collection, 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

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

placing each $x_i^{\delta_i}$ at the leaf of a binary tree and caching intermediate products at each level. This requires no additional 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.[8] The results of the experiment are presented in figure 8.

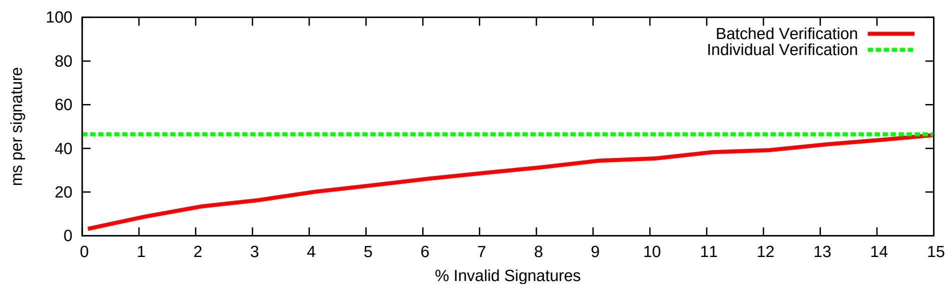

Fig. 8. BLS batch verification in the presence of invalid signatures (160-bit MNT curve). A "resilient" BLS batch verifier was applied to a collection of 1024 purported BLS signatures, where some percentage were randomly corrupted. Per-signature times were computed by dividing the total verification time (including identification of invalid signatures) by the total number of signatures (1024), and averaging over multiple experimental runs.

Our results indicate that 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 size of the batch. Note also that the random distribution of invalid signatures within the collection is nearly the worst-case for resilient verification. In many practical scenarios, 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 by using an alternative re-ordering that is more appropriate for the setting.

6 Conclusion

Our experiments provide strong evidence that batching short signatures is practical, even in a setting where an adversary can inject invalid signatures. Our results apply to standard and Identity-Based signatures, as well as to the more exotic short ring and group signatures which are increasingly being considered for privacy-critical applications. 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

[8] Although our experiment does not re-order the signature collection, such a re-ordering need not involve memory copies and could be performed at minimal additional cost.

at fewer than 40. We believe that scheme designers should begin taking these considerations into account when proposing new pairing-based signature schemes.

References

1. Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest. Lightweight email signatures (extended abstract). In *SCN '06*, volume 4116 of LNCS, pages 288–302. Springer, 2006.
2. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO '00*, volume 1880 of LNCS, pages 255–270, 2000.
3. Man Ho Au, Joseph K. Liu, Tsz Hon Yuen, and Duncan S. Wong. ID-based ring signature scheme secure in the standard model. In *IWSEC '06*, volume 4266 of LNCS, pages 1–16. Springer, 2006.
4. Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT '98*, volume 1403 of LNCS, pages 236–250. Springer, 1998.
5. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT '04*, volume 3027 of LNCS, pages 223–238. Springer, 2004.
6. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *To appear in Journal of Cryptology*, 2007. Available at `http://www.cs.stanford.edu/~xb/joc07/`.
7. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of LNCS, pages 45–55, 2004.
8. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT '03*, volume 2656 of LNCS, pages 416–432. Springer, 2003.
9. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01*, volume 2248 of LNCS, pages 514–532, 2001.
10. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
11. Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *CCS '04*, pages 168–177, New York, NY, USA, 2004. ACM Press.
12. Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 427–444. Springer, 2006.
13. Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In *EUROCRYPT '07*, volume 4515 of LNCS, pages 246–263. Springer, 2007. Full version at `http://eprint.iacr.org/2007/172`.
14. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04*, volume 3152 of LNCS, pages 56–72. Springer, 2004.
15. Car 2 Car. Communication consortium. `http://car-to-car.org`.
16. Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In *PKC '03*, volume 2567 of LNCS, pages 18–30. Springer, 2003.
17. Sanjit Chatterjee and Palash Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In *8th Information Security and Cryptology (ICISC)*, volume 3935 of LNCS, pages 424–440. Springer, 2005.
18. Sanjit Chatterjee and Palash Sarkar. HIBE with short public parameters without random oracle. In *ASIACRYPT '06*, volume 4284 of LNCS, pages 145–160. Springer, 2006.
19. L. Chen, Z. Cheng, and N.P. Smart. Identity-based key agreement protocols from pairings, 2006. Cryptology ePrint Archive: Report 2006/199.
20. Sherman S. M. Chow, Siu-Ming Yiu, and Lucas C.K. Hui. Efficient identity based ring signature. In *Applied Cryptography and Network Security – ACNS '05*, volume 3531 of LNCS, pages 499–512. Springer, 2005.
21. Andreas Enge and Pierrick Gaudry. An L (1/3 + *epsilon*) algorithm for the discrete logarithm problem for low degree curves. In *EUROCRYPT '07*, volume 4515 of LNCS, pages 379–393. Springer, 2007.
22. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.
23. Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. *Inf. Comput.*, 165(1):100–116, 2001.
24. Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, volume 2595 of LNCS, pages 310–324. Springer, 2002.
25. Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels, 2005. Cryptology ePrint Archive: Report 2005/076.

26. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 465–485. Springer, 2006.
27. Anna Lysyanskaya, Roberto Tamassia, and Nikos Triandopoulos. Multicast authentication in fully adversarial networks. In *IEEE Symposium on Security and Privacy*, pages 241–253. IEEE Computer Society, 2004.
28. Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *23rd ACM Symposium on Theory of Computing (STOC)*, pages 80–89. ACM Press, 1991.
29. Jean Monnerat and Serge Vaudenay. Undeniable signatures based on characters: How to sign with one bit. In *PKC '04*, volume 2947 of LNCS, pages 69–85, 2004.
30. Jean Monnerat and Serge Vaudenay. Short 2-move undeniable signatures. In *Vietcrypt '06*, volume 4341 of LNCS, pages 19–36, 2006.
31. D. Naccache. Secure and *practical* identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.
32. Dan Page, Nigel Smart, and Fre Vercauteren. A comparison of MNT curves and supersingular curves. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):379–392, October 2006.
33. Jaroslaw Pastuszak, Dariusz Michatek, Josef Pieprzyk, and Jennifer Seberry. Identification of bad signatures in batches. In *PKC '00*, volume 3958 of LNCS, pages 28–45. Springer-Verlag, 2000.
34. Adrian Perrig, Ran Canetti, Dawn Xiaodong Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *NDSS '01*. The Internet Society, 2001.
35. Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15:39–68, 2007.
36. Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.
37. Michael Scott. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). Published by Shamus Software Ltd., `http://www.shamus.ie/`.
38. SeVeCom. Security on the road. `http://www.sevecom.org`.
39. Zuhua Shao. Enhanced aggregate signatures from pairings. In *CISC '05*, volume 3822 of LNCS, pages 140–149, 2005.
40. Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05*, volume 3494 of LNCS, pages 320–329. Springer, 2005.

A Proof of the Small Exponents Test

Proof. It is easy to see that if $A = Y^{(j)}$ holds for all $j \in [1, \eta]$, then $\prod_{j=1}^{\eta} A^{\delta_j} = \prod_{j=1}^{\eta} {Y^{(j)}}^{\delta_j}$ holds for any random vector $\Delta = (\delta_1, \ldots, \delta_\eta)$. We must now show the other direction, that if Batch outputs *accept*, then $A = Y^{(j)}$ holds for all $j \in [1, \eta]$ except with probability at most $2^{-\ell_b}$. Since A and $Y^{(j)}$ are in $\mathbb{G}_T$, we can write $A = \mathbf{e}(g, g)^a$ and $Y^{(j)} = \mathbf{e}(g, g)^{y^{(j)}}$ for some $a, y^{(j)} \in \mathbb{Z}_q$. The batch verification equation can then be written as $\prod_{j=1}^{\eta} \mathbf{e}(g, g)^a = \prod_{j=1}^{\eta} \mathbf{e}(g, g)^{y^{(j)}} \Rightarrow \mathbf{e}(g, g)^{\sum_{j=1}^{\eta} a} = \mathbf{e}(g, g)^{\sum_{j=1}^{\eta} y^{(j)}}$. Now define $\beta_j = a - y^{(j)}$. Since Batch accepts it must be true that

$$\sum_{j=1}^{\eta} \beta_j \delta_j \equiv 0 \pmod q \tag{3}$$

Now assume that at least one of the individual equations do not hold. We assume without loss of generality that this is true for equation $j = 1$. This means that $\beta_1 \neq 0$. Since q is a prime then β_1 has an inverse γ_1 such that $\beta_1 \gamma_1 \equiv 1 \pmod q$. This and Equation 3 gives us

$$\delta_1 \equiv -\gamma_1 \sum_{j=2}^{\eta} \delta_j \beta_j \pmod q \tag{4}$$

Let event E occurs if $A \neq Y^{(1)}$, but Batch accepts. Note that we do not make any assumptions about the remaining values. Let $\Delta' = \delta_2, \ldots, \delta_\eta$ denote the last $\eta - 1$ values of Δ and let $|\Delta'|$ be the number of possible values for this vector. Equation 4 says that given a fixed vector Δ' there is exactly one value of δ_1 that will make event E happen, or in other words that the probability of E given a randomly chosen δ_1 is $\Pr[E|\Delta'] = 2^{-\ell_b}$. So if we pick δ_1 at random and sum over all possible choices of Δ' we get $\Pr[E] \leq \sum_{i=1}^{|\Delta'|} (\Pr[E|\Delta'] \cdot \Pr[\Delta'])$. Plugging in the values, we get: $\Pr[E] \leq \sum_{i=1}^{2^{\ell_b(\eta-1)}} \left(2^{-\ell_b} \cdot 2^{-\ell_b(\eta-1)}\right) = 2^{-\ell_b}$.

B Proof Sketch for Theorem 2 (Batch Verification of BBS Group Signatures)

We show how to apply Theorem 1 in the following proof of Theorem 2.

Proof sketch. Let $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ be the group public key, and let $\sigma_j = (T_{j,1},\ T_{j,2},\ T_{j,3},\ R_{j,1},\ R_{j,2},\ R_{j,3},\ R_{j,4},\ R_{j,5}, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ be the j'th signature on the message M_j, for each $j = 1, \ldots, \eta$. Since BBS Batch Verify algorithm performs the first four non-pairing tests of the New Verify algorithm for each signature separately, we just need to prove that equation 2 is a batch verifier for the pairing based equation 1. From Theorem 1, for any random vector $(\delta_1, \ldots, \delta_\eta)$ of ℓ_b bit elements from $\mathbb{Z}_q$, the following pairing based equation

$$\prod_{j=1}^{\eta} \left(\mathbf{e}(T_{j,3}, g_2)^{s_{j,x}} \cdot \mathbf{e}(h, w)^{-s_{j,\alpha} - s_{j,\beta}} \cdot \mathbf{e}(h, g_2)^{-s_{j,\gamma_1} - s_{j,\gamma_2}} \cdot \left(\mathbf{e}(T_{j,3}, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^{c_j}\right)^{\delta_j} \stackrel{?}{=} \prod_{j=1}^{\eta} R_{j,3}^{\delta_j} \tag{5}$$

is a batch verifier for the pairing based equation 1. It is easy to see that equation 5 is equivalent to equation 2. Indeed, equation 2 is an optimized version of equation 5 obtained by applying techniques 2 and 3. □

C Ring and Identity-based Ring Signature Schemes

Recall that a ring signature scheme allows a signer to sign a message on behalf of a set of users which include the signer itself in such a way that a verifier is convinced that the signer is one of the ring members, but he cannot tell which member is the actual signer. A ring signature is a triple of algorithms KeyGen, Sign and Verify, that, respectively generate public and private keys for a user, sign a message on behalf of the ring and verify the signature on a message according to the ring. In an identity-based ring signature, a user can choose an arbitrary string, for example her email address, as her public key.

In an identity-based ring signature, a user can choose an arbitrary string, for example her email address, as her public key. The corresponding private key is then created by binding such a string which represents the user's identity with the master key of a trusted party called private key generator (PKG). Such a scheme consists of four algorithms: Setup, KeyGen, Sign and Verify. During Setup, the PKG sets the system parameters P_{pub} and chooses a master secret key msk. During KeyGen, the PKG gives the user a secret key based on her identity string. Then the signing and verification algorithms and verification algorithms operate as before, except that only P_{pub} and the ring members identities are needed in place of their public keys.

Figures 9 and 10 summarizes the scheme we consider and how to batch them, respectively.[9] The CYH scheme is fairly straightforward to batch, while the Boyen scheme required more creativity, especially in the application of techniques 3 and 4.

Scheme	Setup ――― Key Generation	Signature	Verify
CYH	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \to \mathbb{G}_1$ $H_2 : \{0,1\}^* \to \mathbb{Z}_q^*$ $\alpha \xleftarrow{\$} \mathbb{Z}_q^*$ $msk \leftarrow \alpha$ $P_{pub} \leftarrow g^\alpha$ ――― $sk \leftarrow H_1(ID)^\alpha$ $pk \leftarrow H_1(ID)$	Let $L = \{ID_1, ID_2, \ldots, ID_\ell\}$ Let ID_s be the signer $\forall i \in [1, \ell]$ s.t. $i \neq s$ $u_i \xleftarrow{\$} \mathbb{G}_1$ $h_i \leftarrow H_2(M \Vert L \Vert u_i)$ $r \xleftarrow{\$} \mathbb{Z}_q$ $u_s \leftarrow pk_s^r \cdot \left(\prod_{i \neq s} u_i \cdot pk_i^{h_i}\right)^{-1}$ $h_s \leftarrow H_2(M \Vert L \Vert u_s)$ $S = sk_s^{h_s + r}$ $\sigma \leftarrow (u_1, \ldots, u_\ell, S)$	Let $\sigma = (u_1, \ldots, u_\ell, S)$ $\forall i \in [1, \ell]$ $h_i \leftarrow H_2(M \Vert L \Vert u_i)$ ――― $\mathbf{e}(\prod_{i=1}^{\ell} u_i \cdot pk_i^{h_i}, P_{pub}) \stackrel{?}{=} \mathbf{e}(S, g_2)$
Boyen	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H : \{0,1\}^* \to \mathbb{Z}_q^*$ $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ $\hat{A}_0, \hat{B}_0, \hat{C}_0 \xleftarrow{\$} \mathbb{G}_2$ ――― $a, b, c \xleftarrow{\$} \mathbb{Z}_q^*$ $A \leftarrow g_1^a;\ B \leftarrow g_1^b;\ C \leftarrow g_1^c$ $\hat{A} \leftarrow g_2^a;\ \hat{B} \leftarrow g_2^b;\ \hat{C} \leftarrow g_2^c$ $sk \leftarrow (a, b, c)$ $pk \leftarrow (A, B, C, \hat{A}, \hat{B}, \hat{C})$	Let $L = \{pk_1, pk_2, \ldots, pk_\ell\}$ where $pk_i = (A_i, B_i, C_i, \hat{A}_i, \hat{B}_i, \hat{C}_i)$ W.l.o.g., let pk_ℓ be the signer $s_0, s_1, \ldots, s_{\ell-1}, t_0, t_1, \ldots, t_\ell \xleftarrow{\$} \mathbb{Z}_q$ $\forall i \in [0, \ell-1]$, $S_i \leftarrow g_1^{s_i}$ $d \leftarrow \frac{1}{a_\ell + b_\ell \cdot M + c_\ell \cdot t_\ell}$ $S_\ell \leftarrow \left(g \cdot \prod_{i=0}^{\ell-1} (A_i \cdot B_i^M \cdot C_i^{t_i})^{-s_i}\right)^d$ $\sigma = (S_0, \ldots, S_\ell, t_0, \ldots, t_\ell)$	Let $\sigma = (S_0, \ldots, S_\ell, t_0, \ldots, t_\ell)$ Let $D = \mathbf{e}(g_1, g_2)$ ――― $\prod_{i=0}^{\ell} \mathbf{e}(S_i, \hat{A}_i \cdot \hat{B}_i^{M} \cdot \hat{C}_i^{t_i}) \stackrel{?}{=} D$

Fig. 9. Ring signature schemes that we consider. We denote by P_{pub}, sk and pk the system parameters, user private key and user public key, respectively. Moreover, we denote with pk_i and sk_i the public and private keys of the i-th user in the ring. A ring signature on a message M is denoted by σ and ℓ represents the ring size.

D Signature and Identity-based Signature Schemes

In this section we briefly review some short signature schemes and the corresponding batch verifier. In 2001 Boneh et al. [10] proposed the first short pairing-based signature scheme which is secure against existential forgery under adaptive chosen message attack in the random oracle model. In the following we refer to such a scheme as BLS. As also noticed by the authors, BLS is suitable to verify a bunch of purported signatures either issued from the same signer on different messages or by different public keys on the same message in a faster way than simply verifying each signature separately. Indeed, consider η BLS signatures $\sigma_1, \ldots, \sigma_\eta$ issued by means of the BLS signature

[9] In the course of the study about schemes suitable to apply our framework, we noticed that the identity-based ring signature scheme proposed in [3] is a very nice candidate. Unfortunately, we found that, for ring size greater than two, the security proof has a flaw. After hearing of this proof flaw, Brent Waters translated it into an attack on the scheme (personal communication). It is still open to see if such a scheme is indeed secure for rings of size two.

Scheme	Batch Verification Precomputation / Batch Verification Equation	Techniques
CYH	Let $\sigma_j = (u_{j,1}, \ldots, u_{j,\ell}, S_j)$ and $L_j = \{ID_{j,1}, \ldots, ID_{j,\ell_j}\}$; $\forall i, j\ h_{j,i} \leftarrow H_2(M_j \| L_j \| u_{j,i})$	2,3
	$\mathbf{e}(\prod_{j=1}^{\eta} \prod_{i=1}^{\ell_j} pk_{j,i}^{(h_{j,i}+u_{j,i})\cdot\delta_j}, P_{pub})$	
Boyen	Let $\sigma_j = (S_{j,0}, \ldots, S_{j,\ell}, t_{j,0}, \ldots, t_{j,\ell})$, $pk_i \leftarrow (A_i, B_i, C_i, \hat{A}_i, \hat{B}_i, \hat{C}_i)$ and $D = \mathbf{e}(g_1, g_2)$	2,3,4
	If $\eta < 3$, $\prod_{j=1}^{\eta} \prod_{i=0}^{\ell} \mathbf{e}(S_{j,i}^{\delta_j}, \hat{A}_i \cdot \hat{B}_i^{m_{j,i}} \cdot \hat{C}_i^{t_{j,i}}) = \prod_{j=1}^{\eta} D^{\delta_j}$	
	Otherwise, $\prod_{i=0}^{\ell} \Big(\mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j}, \hat{A}_i) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j m_{j,i}}, \hat{B}_i) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j t_{j,i}}, \hat{C}_i)\Big) = \prod_{j=1}^{\eta} D^{\delta_j}$	

Fig. 10. Batch verifier for the ring signature and ID-based ring signature schemes we consider. Let η be the number of signatures to verify and M_j be the message corresponding to the j'th signature σ_j. With $pk_{j,i}$ and ℓ_j we denote the public key of the i'th ring member and the size of the ring associated to the j'th signature, respectively. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test.

Scheme	Setup / Key Generation	Signature	Verification Precomputation / Verification Equation
BLS	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H : \{0,1\}^* \to \mathbb{G}$	$\sigma \leftarrow H(M)^{sk}$	
	$\alpha \xleftarrow{\$} \mathbb{Z}_q$ $sk \leftarrow \alpha$; $pk \leftarrow g_2^{\alpha}$		$\mathbf{e}(H(M), pk) \stackrel{?}{=} \mathbf{e}(\sigma, g_2)$
CHP	Let Φ be the set of time periods. $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \Phi \to \mathbb{G}_1$, $H_2 : \Phi \to \mathbb{G}_1$ $H_3 : \{0,1\}^* \times \Phi \to \mathbb{Z}_q$	$a \leftarrow H_1(\phi)$ $h \leftarrow H_2(\phi)$ $b \leftarrow H_3(M\|\phi)$ $\sigma \leftarrow a^{sk} \cdot h^{sk \cdot b}$	$a \leftarrow H_1(\phi); h \leftarrow H_2(\phi); b \leftarrow H_3(M\|\phi)$
	$\alpha \xleftarrow{\$} \mathbb{Z}_q$ $sk \leftarrow \alpha$; $pk \leftarrow g_2^{\alpha}$		$\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \mathbf{e}(a, pk) \cdot \mathbf{e}(h, pk)^b$

Fig. 11. Signature Schemes that we consider. We denote by pk and sk the public key and the private key of a user, respectively. We denote by σ a signature on a message M. In CHP, ϕ is a time period in the set of time periods Φ.

algorithm (see Figure 11) under the same public key pk on different messages $M_1 \ldots, M_\eta$. According to the BLS verification equation (see Figure 11), 2η pairing evaluations are needed to verify each equation separately, while applying techniques 2 and 3, only two pairing evaluations suffice: $\mathbf{e}(\prod_{j=1}^{\eta} H(M_j)^{\delta_j}, pk) = \mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2)$, for some vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$. A similar approach can be used to batch verify signatures issued by the same public key with only two pairing evaluations. In Figure 12 we describe a more general batch verification equation, where we consider a bunch of signatures issued by s different signers. Applying technique 3, it is easy to see that the pairings on the left hand side of the BLS verification equation corresponding to signatures issued by the same public key can be grouped in a single pairing. This yields to the BLS batch verification equation of Figure 12, where each signer i, for $i = 1, \ldots, s$, is responsible of the n_i out of the η signatures identified by the indices $i_1, \ldots, i_{n_i}$. The BLS batch verification equation of Figure 12 requires $s+1$ pairing evaluations. A similar approach can be used to quickly batch verify signatures on m different messages with $m+1$ pairing evaluations.

Scheme	Batch Verification Precomputation	Batch Verification Equation	Techniques
BLS		$\prod_{i=1}^{s} \mathbf{e}(\prod_{\ell=i_1}^{i_{n_i}} H(M_\ell)^{\delta_\ell}, pk_i) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2)$	2,3
CHP	$a \leftarrow H_1(\phi); h \leftarrow H_2(\phi); \forall j \in [1, \eta],\ b_j \leftarrow H_3(M_j \vert\vert \phi)$	$\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(a, \prod_{j=1}^{\eta} pk_j^{\delta_j}) \cdot \mathbf{e}(h, \prod_{j=1}^{\eta} pk_j^{b_j \cdot \delta_j})$	2,3

Fig. 12. Batch verifiers for the signature schemes we consider. Let η be the number of signatures to verify. With pk_j we denote the public key of the user who issued the j'th signature. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In BLS, s is the number of different signer and n_i is the number of signatures issued by the i'th signer (for details see the text). In CHP, ϕ is a time period in the set of time periods Φ.

In [13], Camenisch et al., proposed a signature scheme secure in the random oracle model which is derived from the Camenisch and Lysyanskaya signature scheme (CL in brief) [14]. The scheme in [13] which we refer to as CHP (see Figure 11) allows efficient batch verification of signatures made by different signers provided that all signatures have been issued during the same period of time. Since the values g_2, a and h are the same for all signatures, from techniques 2 and 3, the CHP batch verification equation shown in Figure 12 requires only three pairings, instead of the 5η pairings required to verify η original CL signatures. In the following we focus on batch verification for identity-based signature schemes. An identity based signature scheme consists of four algorithms: Setup, Key Generation, Sign and Verify. The public key generator PKG initializes the system during the Setup phase by choosing the system parameters P_{pub} which are made public. Moreover, the PKG chooses a master key msk and keeps it secret. The master key is used in the key generation phase along with the identity of a user to compute the user's private key. A user can sign a message by using the Sign algorithm. Finally, a verifier can check a signature on a message by using the Verify algorithm on input the signature, the public parameters and the identity of the signer. In Figure 13 we summarize the identity-based signature schemes we consider.

As shown in Figure 14, techniques 2 and 3 allow to construct a batch verifier which requires only two pairing evaluations for the schemes ChCh and Hess. Both ChCh and Hess schemes are proved secure in the random oracle model. By following the lines of Theorem 2 it is easy to see that the batch verification equations shown in Figure 10 are batch verifier for the corresponding schemes. In [13] Camenisch et al. showed a batch verifier for an identity-based signature scheme secure in the standard model. This scheme, which we refer to as Waters, is derived from a number of contributions [40, 31, 18] which improve upon the Boneh and Boyen identity-based encryption [5]. This scheme, which we refer to as Waters, is derived from a number of contributions [40, 31, 18] which improve upon the Boneh and Boyen identity-based encryption [5]. In particular, Waters described how to modify Boneh and Boyen identity-based encryption to make it fully-secure [40]. The difference between these two IBEs is the way the identity is evaluated. Assume the identity is a bit string $V = v_1 v_2 \ldots v_m$. Instead of evaluating it as $u' g_1^V$ [5] then evaluating it as $u' \prod_{i=1}^{m} u_i^{v_i}$ [40] makes the scheme fully secure. In 2005 Naccache [31] and Chatterjee and Sarkar [17] independently showed how to generalize the Waters IBE to optimize it for efficiency. These ideas were extended in 2006 by Chatterjee and Sarkar to Waters HIBE and the resulting HIBE was proven secure in the standard model [18]. Finally, Waters is the identity-based scheme implicitly defined by Chatterjee and Sarkar's HIBE [18]. In Waters identities and messages are parsed as sequences of z chunks of ℓ-bit integers. As remarked in Figure 14, by using techniques 2, 3 and 4, Waters allows to define a batch verifier where the number of pairing evaluations is proportional to the minimum between the number of signatures η and the number of chunks z.

E Aggregate Signatures

Aggregate signatures were introduced by Boneh et al. in [9]. An aggregate signature is a shorter representation of n signatures provided by different users on different messages. In particular, consider n signatures $\sigma_1, \ldots, \sigma_n$ on messages $M_1, \ldots, M_n$ issued by n users with public keys $pk_1, \ldots, pk_n$. An aggregate signature scheme provides an aggregation algorithm, which can be run by anyone and outputs a compressed short signature σ on input all σ_i, for $i = 1, \ldots, n$. Moreover, there is a verification algorithm that on inputs the signature σ the public keys $pk_1, \ldots, pk_n$ and the messages $M_1, \ldots, M_n$ decides if σ is a valid aggregate signature. Figure E reviews the aggregate signatures we consider. Sh scheme [39] requires the existence of a third party named *aggregator* who is responsible of aggregating signatures. LOSSW scheme [26], proved to be secure in the standard model, is a sequential aggregate signature scheme. The aggregate signature must be constructed sequentially, with each signer adding its signature in turn. Figure 16 shows the corresponding batch verifier obtained by using our framework. Following the line of Theorem 2 it is easy to see that the pairing based equations in Figure 16 are batch verifiers for the corresponding schemes when all aggregate signatures are issued by the same set of users.

Scheme	Setup Key Generation	Sign	Verification Precomputation Verification Equation
ChCh	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ $H_2 : \{0,1\}^* \times \mathbb{G}_1 \rightarrow \mathbb{Z}_q$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $msk \leftarrow \alpha$; $P_{pub} \leftarrow g_2^\alpha$ $sk \leftarrow H_1(ID)^\alpha$; $pk \leftarrow H_1(ID)$	$s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow pk^s$ $a \leftarrow H_2(M\|\|S_1)$ $S_2 \leftarrow sk^{s+a}$ $\sigma \leftarrow (S_1, S_2)$	Let $\sigma = (S_1, S_2)$, $a \leftarrow H_2(M\|\|S_1)$ $\mathbf{e}(S_2, g_2) \stackrel{?}{=} \mathbf{e}(S_1 \cdot pk^a, P_{pub})$
Hess	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ $H_2 : \{0,1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_q$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $msk \leftarrow \alpha$; $P_{pub} \leftarrow g_2^\alpha$ $sk \leftarrow H_1(ID)^\alpha$; $pk \leftarrow H_1(ID)$	$h \xleftarrow{\$} \mathbb{G}$ $s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow \mathbf{e}(h, g_2)^s$ $a \leftarrow H_2(M\|\|S_1)$ $S_2 \leftarrow sk^a \cdot h^s$ $\sigma \leftarrow (S_1, S_2)$	Let $\sigma = (S_1, S_2)$, $a \leftarrow H_2(M\|\|S_1)$ $\mathbf{e}(S_2, g_2) \stackrel{?}{=} \mathbf{e}(pk, P_{pub})^a \cdot S_1$
Waters	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $h \xleftarrow{\$} \mathbb{G}_1$ $A \leftarrow \mathbf{e}(h, g_2)^\alpha$ $y_1', y_2', y_1, y_2, \ldots, y_z \xleftarrow{\$} \mathbb{Z}_q$ $u_1' \leftarrow g_1^{y_1'}$; $u_2' \leftarrow g_1^{y_2'}$ $\forall \ell \in [1, z]$, $u_\ell \leftarrow g_1^{y_\ell}$ $\hat{u_1}' \leftarrow g_2^{y_1'}$; $\hat{u_2}' \leftarrow g_2^{y_2'}$ $\forall \ell \in [1, z]$, $\hat{u_\ell} \leftarrow g_2^{y_\ell}$ $msk \leftarrow h^\alpha$ $P_{pub} \leftarrow (A, u_1', u_2', u_1, \ldots, u_z,$ $\hat{u_1}', \hat{u_2}', \hat{u_1}, \ldots, \hat{u_z})$ $r \xleftarrow{\$} \mathbb{Z}_q$ $k_1 \leftarrow h^\alpha \cdot (u_1' \cdot \prod_{i=1}^{z} u_i^{\kappa_i})^r$ $k_2 \leftarrow g_1^{-r}$ $sk \leftarrow (k_1, k_2)$	$s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow k_1 \cdot (u_2' \cdot \prod_{i=1}^{z} u_i^{m_i})^s$ $S_2 \leftarrow k_2$ $S_3 \leftarrow g_1^{-s}$ $\sigma \leftarrow (S_1, S_2, S_3)$	Let $\sigma = (S_1, S_2, S_3)$ and $A = e(h, g_2)^\alpha$ $\mathbf{e}(S_1, g_2) \cdot \mathbf{e}(S_2, \hat{u_1}' \cdot \prod_{i=1}^{z} \hat{u_i}^{\kappa_i}) \cdot \mathbf{e}(S_3, \hat{u_2}' \cdot \prod_{i=1}^{z} \hat{u_i}^{m_i}) \stackrel{?}{=} A$

Fig. 13. Identity-based signature schemes that we consider. We denote by msk, P_{pub}, sk and pk the master key, the system parameters, user private key and user public key, respectively. We denote by σ a signature on a message M. In Waters, z is the number of ℓ-bit chunks. Moreover, the identity ID and the message M are parsed as $\kappa_1, \ldots, \kappa_z$ and $m_1, \ldots, m_z$, respectively.

Scheme	Batch Verification Precomputation / Batch Verification Equation	Techniques
ChCh	Let $\sigma_j = (S_{j,1}, S_{j,2})$. $\forall j \in [1, \eta]$, $a_j \leftarrow H_2(M_j \Vert S_{j,1})$	2,3
	$\mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta}(S_{j,1} \cdot pk_j^{a_j})^{\delta_j}, P_{pub})$	
Hess	Let $\sigma_j = (S_{j,1}, S_{j,2})$. $\forall j \in [1, \eta]$, $a_j \leftarrow H_2(M_j \Vert S_{j,1})$	2,3
	$\mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} pk_j^{a_j \cdot \delta_j}, P_{pub}) \cdot \prod_{j=1}^{\eta} S_{j,1}^{\delta_j}$	
Waters	Let $\sigma_j = (S_{j,1}, S_{j,2}, S_{j,3})$ and $P_{pub} = (A, u_1', u_2', u_1, \ldots, u_z, \hat{u_1}', \hat{u_2}', \hat{u}_1, \ldots, \hat{u_z})$	2,3, 4
	If $z > 2\eta - 2$, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}, g_2) \cdot \prod_{j=1}^{\eta} \left(\mathbf{e}(S_{j,1}^{\delta_j}, \hat{u_1}' \prod_{i=1}^{z} \hat{u_j}^{k_{j,i}}) \cdot \mathbf{e}(S_{j,3}^{\delta_j}, \hat{u_2}' \prod_{i=1}^{z} \hat{u_j}^{m_{j,i}})\right) \stackrel{?}{=} A^{\sum_{j=1}^{\eta} \delta_j}$ Otherwise, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, \hat{u_1}') \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,3}^{\delta_j}, \hat{u_2}') \cdot \prod_{i=1}^{z} \mathbf{e}(\prod_{j=1}^{\eta}(S_{j,2}^{k_{j,i}} \cdot S_{j,3}^{m_{j,i}})^{\delta_j}, \hat{u}_i) \stackrel{?}{=} A^{\sum_{j=1}^{\eta} \delta_j}$	

Fig. 14. Batch verifiers for the id-based signature schemes we consider. Let η be the number of signatures to verify. With pk_j we denote the public key of the user who issued the j'th signature σ_j on message M_j. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In CHP-2, z is the number of ℓ-bit chunks. Moreover, the identity ID_j and the message M_j corresponding to the j-th signature are parsed as $\kappa_{j,1}, \ldots, \kappa_{j,z}$ and $m_{j,1}, \ldots, m_{j,z}$, respectively.

Scheme	Setup / Key Generation	Aggregate Signature	Verification
BGLS	Same as BLS ——— Same as BLS	Let σ_i be a BLS signature on message M_i under private key pk_i $\sigma \leftarrow \prod_{i=1}^{\ell} \sigma_i$	$\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \prod_{i=1}^{\ell} \mathbf{e}(H(M_i), pk_i)$
Sh	Same as BLS ——— For users and aggregator, same as BLS	Let σ_i be a BLS signature on message M_i under private key pk_i If all BLS signatures are valid, the aggregator use its secret key sk_{ag} to compute $\sigma \leftarrow H(M_1\|\|\ldots\|\|M_\ell)^{sk_{ag}} \cdot \prod_{i=1}^{\ell} \sigma_i$	$\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \mathbf{e}(H(M_1,\ldots,M_\ell), pk_{ag}) \cdot \prod_{i=1}^{\ell} \mathbf{e}(H(M_i), pk_i)$
LOSSW	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ ——— $\alpha, y' \xleftarrow{\$} \mathbb{Z}_q$ $(y_1, \ldots, y_k) \xleftarrow{\$} \mathbb{Z}_q^k$ $\mathbf{y} = (y_1, \ldots, y_k)$ $u' \leftarrow g_1^{y'}$ $\hat{u}' \leftarrow g_2^{y'}$ $\forall i = 1, \ldots, k,$ $u_i \leftarrow g_1^{y_i}$ $\hat{u}_i \leftarrow g_2^{y_i}$ $\mathbf{u} = (u_1, \ldots, u_k, \hat{u}_1, \ldots, \hat{u}_k)$ $A \leftarrow \mathbf{e}(g_1, g_2)^\alpha$ $sk \leftarrow (\alpha, y', \mathbf{y})$ $pk \leftarrow (A, u', \hat{u}', \mathbf{u})$	Let $\sigma' = (\prod_{i}^{\ell-1} g_1^{\alpha_i} \cdot \prod_{i=1}^{\ell-1} (u'_i \prod_{t=1}^{k} u_{i,t}^{m_{i,t}})^{r'}, g_1^{r'}) = (S'_1, S'_2)$ be an aggregate so far on a set of messages $\{M_1, \ldots, M_{\ell-1}\}$ under public keys $\{pk_1, \ldots, pk_{\ell-1}\}$. Let M_ℓ be the message to sign under public key pk_ℓ and corresponding secret key sk_ℓ. We denote $pk_i = (A_i, u'_i, \hat{u_i}', u_{i,1}, \ldots, u_{i,k}, \hat{u_{i,1}}, \ldots, \hat{u_{i,k}})$, $sk_i = (\alpha_i, y'_i, y_{i,1}, \ldots, y_{i,k})$ and $M_i = m_{i,1}, \ldots, m_{i,k}$[10]. $w_1 \leftarrow S'_1 \cdot g_1^{\alpha} \cdot (S'_2)^{(y'_\ell + \sum_{t=1}^{k} y_{\ell,t} \cdot m_{\ell,t})}$ $w_2 \leftarrow S'_2$ $r \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow w_1 (u'_\ell \prod_{t=1}^{k} u_{\ell,t}^{m_{\ell,t}})^r \prod_{i=1}^{t} (u'_i \prod_{t=1}^{k} u_{i,t}^{m_{i,t}})^r$ $S_2 \leftarrow w_2 \cdot g_1^r$ $\sigma = (S_1, S_2)$	$\prod_{i=1}^{\ell} A_i \stackrel{?}{=} \mathbf{e}(S_1, g_2)/\mathbf{e}(S_2, \prod_{i=1}^{\ell} (\hat{u_i}' \prod_{t=1}^{k} \hat{u_{i,t}}^{m_{i,t}}))$

Fig. 15. For setup, key generation and signature of BGLS and Sh see Figure 11. We denote by σ an aggregate signature on a set of ℓ messages $M_1, \ldots, M_\ell$. In LOSSW a message M_i is processed as a k-bit string denoted by $m_{i,1}, \ldots, m_{i,k}$.

Scheme	Batch Verification Equation	Techniques
BGLS	$\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \prod_{i=1}^{\ell} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,i})^{\delta_j}, pk_i)$	2,3
Sh	$\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,1}, \ldots, M_{j,\ell})^{\delta_j}, pk_{ag}) \cdot \prod_{i=1}^{\ell} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,i}), pk_i)$	2,3
LOSSW	Let $\sigma_j = (S_{j,1}, S_{j,2})$ and $pk_i = (A_i, u'_i, \hat{u_i}', u_{i,1}, \ldots, u_{i,k}, \hat{u_{i,1}}, \ldots, \hat{u_{i,k}})$ If $\eta < \ell \cdot k + 1$, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \prod_{j=1}^{\eta} \mathbf{e}(S_{j,2}^{-\delta_j}, \prod_{i=1}^{\ell} (\hat{u_i}' \prod_{t=1}^{k} \hat{u_{i,t}}^{m_{j,i,t}})) \stackrel{?}{=} \prod_{j=1}^{\eta} \prod_{i=1}^{\ell} A_i^{\delta_j}$ Otherwise, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{-\delta_j}, \prod_{i=1}^{\ell} \hat{u_i}') \cdot \prod_{i=1}^{\ell} \prod_{t=1}^{k} \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{-\delta_j m_{j,i,t}}, \hat{u_{i,t}}) \stackrel{?}{=} \prod_{j=1}^{\eta} \prod_{i=1}^{\ell} A_i^{\delta_j}$	2,3,4

Fig. 16. Let η be the number of signatures to verify. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In LOSSW a message $M_{j,i}$ provided by pk_i in the j'th aggregate is processed as a k-bit string denoted by $m_{j,i,1}, \ldots, m_{j,i,k}$.