Batch Verification of Short Signatures

Jan Camenisch[1], Susan Hohenberger[2⋆], and Michael Østergaard Pedersen[3∗]

[1] IBM Research, Zürich Research Laboratory
jca@zurich.ibm.com
[2] The Johns Hopkins University
susan@cs.jhu.edu
[3] University of Aarhus
michael@daimi.au.dk

Abstract. With computer networks spreading into a variety of new environments, the need to authenticate and secure communication grows. Many of these new environments have particular requirements on the applicable cryptographic primitives. For instance, several applications require that communication overhead be small and that many messages be processed at the same time. In this paper we consider the suitability of public key signatures in the latter scenario. That is, we consider signatures that are 1) short and 2) where many signatures from (possibly) different signers on (possibly) different messages can be verified quickly.

We propose the first batch verifier for messages from many (certified) signers without random oracles and with a verification time where the dominant operation is independent of the number of signatures to verify. We further propose a new signature scheme with very short signatures, for which batch verification for *many* signers is also highly efficient. Prior work focused almost exclusively on batching signatures from the same signer. Combining our new signatures with the best known techniques for batching certificates from the *same* authority, we get a fast batch verifier for certificates and messages combined. Although our new signature scheme has some restrictions, it is the only solution, to our knowledge, that is a candidate for some pervasive communication applications.

1 Introduction

As the world moves towards pervasive computing and communication, devices from vehicles to dog collars will soon be expected to communicate with their environments. For example, many governments and industry consortia are currently planning for the future of *intelligent cars* that constantly communicate with each other and the transportation infrastructure to prevent accidents and to help alleviate traffic congestion [11, 37]. Raya and Hubaux suggest that vehicles will transmit safety messages every 300ms to all other vehicles within a minimum range of 110 meters [36], which in turn may retransmit these messages.

⋆ Research performed while at IBM Research, Zürich Research Laboratory.

For such pervasive systems to work properly, there are many competing constraints [11, 37, 27, 36]. First, there are physical limitations, such as a limited spectrum allocation for specific types of communications and the potential roaming nature of devices, that require that messages be kept very short and (security) overhead be minimal [27]. Yet for messages to be trusted by their recipients, they need to be authenticated in some fashion, so that entities spreading false information can be held accountable. Thus, some short form of authentication must be added. Third, different messages from many different signers may need to be verified and processed quickly (e.g., every 300ms [36]). A possible fourth constraint that these authentications remain anonymous or pseudonymous, we leave as an exciting open problem.

In this work, we consider the suitability of public key signatures to the needs of pervasive communication applications. Generating one signature every 300ms is not a problem for current systems, but transmitting and/or verifying 100+ messages per second might pose a problem. Using RSA signatures for example seems attractive as they are verified quickly, however, one would need approximately 3000 bits to represent a signature on a message plus the certificate (i.e., the public key and signature on that public key) which might be too much (see Section 8.2 of [36]). While many new schemes based on bilinear maps can provide the same security with significantly smaller signatures, they take significantly more time to verify.

1.1 Our Contributions

Now, if one wants both, short signatures and short verification times, it seems that one needs to improve on the verification of the bilinear-map based schemes. In this paper we take this route and investigate the known batch-verification techniques and to what extent they are applicable to such schemes. More precisely, the main contributions of this paper are:

1. We instantiate the general batch verification definitions of Bellare, Garay, and Rabin [2] to the case of signatures from many signers. We also do this for a weaker notion of batch verification called *screening* and show the relation of these notions to the one of aggregate signatures. Surprisingly, for most known aggregate signature schemes a batching algorithm is provably *not* obtained by aggregating many signatures and then verifying the aggregate.
2. We present a batch verifier for the Waters IBS scheme [39, 7]. To our knowledge, this is the *first* batch verifier for a signature scheme without random oracles. When identities are k_1 bits and messages are k_2 bits, our algorithm verifies n Waters IBS signatures using only (k_1+k_2+3) pairings. Individually verifying n signatures would cost $3n$ pairings.
3. We present a new signature scheme, CL*, derived from the Camenisch and Lysyanskaya signature scheme [8]. We show that CL* can be realized without random oracles when the message space is polynomial. CL* signatures require only one-third the space of the original CL signatures– on par with the shortest signatures known [5] –, but users may only issue one signature

per period (e.g., users might only be allowed to sign one message per 300ms). We present a batch verifier for these signatures from many different signers that verifies n signatures using only three total pairings, instead of the $5n$ pairings required by n original CL signatures. Yet, our batch verifier has the restriction that it can only batch verify signatures made during the same period. CL* signatures form the core of the only public key authentication, known to us, that is extremely short and highly efficient to verify in bulk.

4. Often signatures and certificates need to be verified together. This happens implicitly in IBS schemes, such as Waters. To achieve this functionality with CL* signatures, we use a known batch verifier for the Boneh, Lynn, and Shacham signatures in the random oracle model [5, 4] that can batch verify n signatures from the *same* signer using only two pairings.

1.2 Batch Verification Overview

Batch cryptography was introduced in 1989 by Fiat [17] for a variant of RSA. Later, in 1994, Naccache, M'Raïhi, Vaudenay and Raphaeli [35] gave the first efficient batch verifier for DSA signatures, however an interactive batch verifier presented in an early version of their paper was broken by Lim and Lee [31]. In 1995 Laih and Yen proposed a new method for batch verification of DSA and RSA signatures [29], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [6]. In 1998 Harn presented two batch verification techniques for DSA and RSA [22, 23] but both were later broken [6, 25, 26]. The same year, Bellare, Garay and Rabin took the first systematic look at batch verification [2] and presented three generic methods for batching modular exponentiations, called the *random subset test*, the *small exponents test* and the *bucket test* which are similar to the ideas from [35, 29]. They showed how to apply these methods to batch verification of DSA signatures and also introduced a weaker form of batch verification called *screening*. In 2000 some attacks against different batch verification schemes, mostly ones based on the small exponents test and related tests, were published [6]. These attacks do not invalidate the proof of security for the small exponents test, but rather show how the small exponents test is often used in a wrong way. However, they also describe methods to repair some broken schemes based on this test. In 2001 Hoshino, Masayuki and Kobayashi [24] pointed out that the problem discovered in [6] might not be critical for batch verification of signatures, but when using batch verification to verify for example zero-knowledge proofs, it would be. In 2004 Yoon, Cheon and Kim proposed a new ID-based signature scheme with batch verification [15], but their security proof is for aggregate signatures and does not meet the definition of batch verification from [2]; hence their title is somewhat misleading. Of course not all aggregate signature schemes claim to do batch verification. For example Gentry and Ramzan present a nice aggregate signature scheme in [19] that does not claim to be, nor is, a batch verification scheme. Other schemes for batch verification based on bilinear maps were proposed [12, 40–42] but all were later broken by Cao, Lin and Xue [10]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [30], but Stanek [38] showed that this method is flawed.

1.3 Efficiency of Prior Work and our Contributions

Efficiency will be given as an abstract cost for computing different functions. We begin by discussing prior work on RSA, DSA, and BLS signatures mostly for single signers, and then discuss our new work on Waters, BLS, and CL signatures for many signers. Note that Lim [32] provides a number of efficient methods for doing m-term exponentiations and Granger and Smart [21] give improvements over the naive method for computing a product of pairings, which is why we state them explicitly.

m-$\mathsf{MultPairCost}^s_{\mathbb{G},\mathbb{H}}$	s m-term pairings $\prod_{i=1}^{m} \mathbf{e}(g_i, h_i)$ where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
m-$\mathsf{MultExpCost}^s_{\mathbb{G}}(k)$	s m-term exponentiations $\prod_{i=1}^{m} g^{a_i}$ where $g \in \mathbb{G}$, $\lvert a_i \rvert = k$.
$\mathsf{PairCost}^s_{\mathbb{G},\mathbb{H}}$	s pairings $\mathbf{e}(g_i, h_i)$ for $i = 1 \ldots s$, where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$\mathsf{ExpCost}^s_{\mathbb{G}}(k)$	s exponentiations g^{a_i} for $i = 1 \ldots s$ where $g \in \mathbb{G}$, $\lvert a_i \rvert = k$.
$\mathsf{GroupTestCost}^s_{\mathbb{G}}$	Testing whether or not s elements are in the group $\mathbb{G}$.
$\mathsf{HashCost}^s_{\mathbb{G}}$	Hashing s values into the group $\mathbb{G}$.
$\mathsf{MultCost}^s$	s multiplications in one or more groups.

If $s = 1$ we will omit it. Throughout this paper we assume that n is the number of message/signature pairs and ℓ_b is a security parameter such that the probability of accepting a batch that contains an invalid signature is at most $2^{-\ell_b}$.

RSA* is a modified version of RSA by Boyd and Pavlovski [6]. The difference to normal RSA is that the verification equation accepts a signature σ as valid if $\alpha\sigma^e = m$ for some element $\alpha \in \mathbb{Z}^*_m$ of order no more than 2, where m is the product of two primes. The signatures are usually between $1024 - 2048$ bits and the same for the public key. A single signer batch verifier for this signature scheme with cost n-$\mathsf{MultExpCost}^2_{\mathbb{Z}_m}(\ell_b) + \mathsf{ExpCost}_{\mathbb{Z}_m}(k)$, where k is the number of bits in the public exponent e, can be found in [6]. Note that verifying n signatures by verifying each signature individually only costs $\mathsf{ExpCost}^n_{\mathbb{Z}_m}(k)$, so for small values of e ($\lvert e \rvert < 2\ell_b/3$) the naive method is a faster way to verify RSA signatures and it can also handle signatures from multiple signers. Bellare et al. [2] presents a screening algorithm for RSA that assumes distinct messages from the same signer and costs $2n + \mathsf{ExpCost}_{\mathbb{Z}_m}(k)$.

DSA** is a modified version of DSA from [35] compatible with the *small exponents test* from [6]. There are two differences to normal DSA. First there is no reduction modulo q, so the signatures are 672 bits instead of 320 bits and second, individual verification should check both a signature σ and $-\sigma$ and accept if one of them holds. Messages and public keys are both 160 bits long. Using the small exponents test the cost is n-$\mathsf{MultExpCost}_{\mathbb{G}}(\ell_b) + \mathsf{ExpCost}^2_{\mathbb{G}}(160) + \mathsf{HashCost}^n_{\mathbb{G}} + \mathsf{MultCost}^{2n+1}$ multiplications. This method works for a single signer only.

Waters IBS is the Waters 2-level hierarchical signature scheme from [7] for which we provide a batch verifier without random oracles in Section 4. An interesting property of this scheme is that the identity does not need to be verified separately. Identities are k_1 bits, messages are k_2 bits and a signature is three group elements in a bilinear group. The computational effort required depends on the number of messages and the security parameters. Let $\mathsf{M} = n\text{-}\mathsf{MultExpCost}_{\mathbb{G}_T}(\ell_b) + n\text{-}\mathsf{MultExpCost}^3_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^{3n}_{\mathbb{G}} + \mathsf{MultCost}^4$ and refer to the table below for efficiency of the scheme. We assume that $k_1 < k_2$.

$$\begin{aligned}
n < k_1 &: \mathsf{M} + n\text{-}\mathsf{MultPairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{ExpCost}^{2n}_{\mathbb{G}}(\ell_b) + \mathsf{MultCost}^{k_1+k_2}\\
k_1 \le n \le k_2 &: \mathsf{M} + k_1\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + n\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}}\\
&\quad + n\text{-}\mathsf{MultExpCost}^{k_1}_{\mathbb{G}}(\ell_b) + \mathsf{ExpCost}^{n}_{\mathbb{G}}(\ell_b) + \mathsf{MultCost}^{k_2}\\
n > k_2 &: \mathsf{M} + k_1\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + k_2\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}}\\
&\quad + n\text{-}\mathsf{MultExpCost}^{2}_{\mathbb{G}}(\ell_b)
\end{aligned}$$

The naive application of Waters IBS to verify n signatures costs $\mathsf{PairCost}^{3n}_{\mathbb{G},\mathbb{G}} + \mathsf{MultCost}^{n(k_1+k_2+3)}$. Also note that in many applications we do not need to transmit the identity as a separate parameter, as it is given to us for free. For example as the hardware address of the network interface card.

BLS is the signature scheme by Boneh, Lynn and Shacham [5, 4]. We discuss batch verifiers for BLS signatures based on the small exponents test. For a screening algorithm, aggregate signatures by Boneh, Gentry, Lynn and Shacham [3] can be used. The signature is only one group element in a bilinear group and the same for the public key. For different signers the cost of batch verification is $n\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}_{\mathbb{G},\mathbb{G}} + \mathsf{ExpCost}^n_{\mathbb{G}_T}(\ell_b) + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, but for single signer it is only $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$.

CL* is a new variant of Camenisch and Lysyanskaya signatures [8] presented in Section 5. The signature is only one bilinear group element and the same for the public key. Batch verification costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, where w is the output of a hash function. However, the scheme has some additional restrictions.

Bucket Test. Bellare, Garay and Rabin [2] provide a method called the *bucket test* which is even more efficient than the small exponents test for large values of n. We note that one can use the tests we outline in this paper as subroutines to the *bucket test* to further speed up verification.

2 Definitions

Recall that a *digital signature scheme* is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ that also is *correct* and *secure*. The correctness property states that for all $\mathsf{Gen}(1^\ell) \to (pk, sk)$, the algorithm $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

There are two common notions of security. Goldwasser, Micali, and Rivest [20] defined a scheme to be *unforgeable* as follows: Let $\mathsf{Gen}(1^\ell) \rightarrow (pk, sk)$. Suppose (m, σ) is output by a p.p.t. adversary $\mathcal{A}$ with access to a signing oracle $\mathcal{O}_{sk}(\cdot)$ and input pk. Then the probability that m was *not* queried to $\mathcal{O}_{sk}(\cdot)$ and yet $\mathsf{Verify}(pk, m, \sigma) = 1$ is negligible in ℓ. An, Dodis, and Rabin [1] proposed the notion of *strong unforgeability*, where if $\mathcal{A}$ outputs a pair (m, σ) such that $\mathsf{Verify}(pk, m, \sigma) = 1$, then except with negligible probability at some point oracle $\mathcal{O}_{sk}(\cdot)$ was queried on m and outputted signature σ exactly. In other words, an adversary cannot create a new signature even for a previously signed message. Our batch verification definitions work with either notion. The signatures used in Section 4 meet the GMR [20] definition, while those in Section 5 meet the ADR [1] definition.

Now, we consider the case where we want to quickly verify a set of signatures on (possibly) different messages by (possibly) different signers. The input is $\{(t_1, m_1, \sigma_1), \ldots, (t_n, m_n, \sigma_n)\}$, where t_i specifies the verification key against which σ_i is purported to be a signature on message m_i. We extend the definitions of Bellare, Garay and Rabin [2] to deal with multiple signers. And this is an important point that wasn't a concern with only a single signer: *one or more of the signers may be maliciously colluding.*

Definition 1 (Batch Verification of Signatures). *Let ℓ be the security parameter. Suppose* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a signature scheme, $n \in \text{poly}(\ell)$, and $(pk_1, sk_1), \ldots, (pk_n, sk_n)$ are generated independently according to* $\mathsf{Gen}(1^\ell)$*. Then we call probabilistic* Batch *a batch verification algorithm when the following conditions hold:*

- *If* $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ *for all $i \in [1, n]$, then* $\mathsf{Batch}((pk_{t_1}, m_1, \sigma_1), \ldots, (pk_{t_n}, m_n, \sigma_n)) = 1$.
- *If* $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 0$ *for any $i \in [1, n]$, then* $\mathsf{Batch}((pk_{t_1}, m_1, \sigma_1), \ldots, (pk_{t_n}, m_n, \sigma_n)) = 0$ *except with probability negligible in k, taken over the randomness of* Batch.

Note that Definition 1 does not require verification keys *to belong* to honest users, only to keys that were *generated* honestly (and are perhaps now held by an adversary). In practice, users could register their keys and prove some necessary properties of the keys at registration time.

Confusion between Batch Verification, Aggregate Signatures, and Screening. As we discussed in the introduction, several works (e.g., [15, 16]) claim to do batch verification when, in fact, they often meet a weaker guarantee called *screening* [2]. However, in most cases the confusion is about words, e.g. when the words *batch verification* are used to describe an aggregate signature scheme.

Definition 2 (Screening of Signatures). *Let ℓ be the security parameter. Suppose* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a signature scheme, $n \in \text{poly}(\ell)$ and $(pk^*, sk^*) \leftarrow$* $\mathsf{Gen}(1^\ell)$*. Let $\mathcal{O}_{sk^*}(\cdot)$ be an oracle that on input m outputs $\sigma =$* $\mathsf{Sign}(sk^*, m)$*. Then* *for all p.p.t. adversaries $\mathcal{A}$, we call probabilistic* Screen *a* screening algorithm *when $\mu(\ell)$ defined as follows is a negligible function:*

$$\Pr[(pk^*, sk^*) \leftarrow \mathsf{Gen}(1^\ell), (pk_1, sk_1) \leftarrow \mathsf{Gen}(1^\ell), \ldots, (pk_n, sk_n) \leftarrow \mathsf{Gen}(1^\ell),$$
$$D \leftarrow \mathcal{A}^{\mathcal{O}_{sk^*}(\cdot)}(pk^*, (pk_1, sk_1), \ldots, (pk_n, sk_n)) :$$
$$\mathsf{Screen}(D) = 1 \ \wedge \ (pk^*, m_i, \sigma_i) \in D \ \wedge \ m_i \notin Q] = \mu(\ell),$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}$.

The above definition is generalized to the multiple-signer case from the single-signer screening definition of Bellare et al. [2].

Interestingly, screening is the (maximum) guarantee that most aggregate signatures offer if one were to attempt to batch verify a group of signatures by first aggregating them together and then executing the aggregate-verification algorithm. Consider the aggregate signature scheme of Boneh, Gentry, Lynn and Shacham [3] based on the BLS signatures [5, 4]. First, we describe the BLS signatures. Let $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where g generates the group $\mathbb{G}$ of prime order q. Gen chooses a random $sk \in \mathbb{Z}_q$ and outputs $pk = g^{sk}$. A signature on message m is $\sigma = H(m)^{sk}$, where H is a hash function. To verify signature σ on message m, one checks that $\mathbf{e}(\sigma, g) = \mathbf{e}(H(m), pk)$. Given a group of message-signature pairs $(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$ (all purportedly from the same signer), BGLS aggregates them as $A = \prod_{i=1}^n \sigma_i$. Then all signatures can be verified in aggregate (i.e., screened) by testing that $\mathbf{e}(A, g) = \mathbf{e}(\prod_{i=1}^n H(m_i), pk)$. This scheme is *not*, however, a batch verification scheme since, for any $a \neq 1 \in \mathbb{G}$, the two *invalid* message-signature pairs $P_1 = (m_1, a \cdot H(m_1)^{sk})$ and $P_2 = (m_2, a^{-1} \cdot H(m_2)^{sk})$ will verify under Definition 2 (as BGLS prove [3]), but will not verify under Definition 1. Indeed, for some pervasive computing applications only guaranteeing screening would be disastrous, because only P_1 may be relevant information to forward to the next entity – and it won't verify once it arrives! To be fair, batch verification is not what aggregate schemes were designed to do, but it is a common misuse of them.

Let $D = \{(t_1, m_1, \sigma_1), \ldots, (t_n, m_n, \sigma_n)\}$. We note that while $\mathsf{Screen}(D) = 1$ does not guarantee that $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i)$ for all i; it does guarantee that the holder of sk_{t_i} authenticated m_i. Thus, we can always prove this by first creating a *new* signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}')$ where the verification algorithm Verify' is modified w.r.t. the original scheme as follows. Apart from the original signatures, it also accepts signatures σ_i' derived from D such that if and only if for all $(t_i, m_i, \sigma_i) \in D$, $\mathsf{Verify}'(pk_{t_i}, m_i, \sigma_i') = 1$ we have $\mathsf{Screen}(D) = 1$. One method to construct σ_i' would be to give a zero-knowledge proof of knowledge of D such that $\mathsf{Screen}(D) = 1$, although (using the naive solution) these new signatures σ_i' will require $O(n)$ space and Verify' will run in $O(n)$ time.

3 Algebraic Setting and Group Membership

Bilinear Groups. Let BSetup be an algorithm that, on input the security parameter 1^ℓ, outputs the parameters for a bilinear map as $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where

$\mathbb{G}, \mathbb{G}_T$ are of prime order $q \in \Theta(2^\ell)$. The efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is both: (*Bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_q$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$; and (*Non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$. Following prior work, we write $\mathbb{G}$ and $\mathbb{G}_T$ in multiplicative notation, although $\mathbb{G}$ is actually an additive group. Our constructions from Section 5 also work in the setting $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct groups, possibly without efficient isomorphisms between them, but it is more tedious to write. However, this later implementation allows for the shortest group elements. We note that if the Waters IBS scheme also works in this setting, so will our proposed batch verifier in Section 4.

Testing Membership in $\mathbb{G}$. In a *non-bilinear* setting, Boyd and Pavlovski [6] observed that the proofs of security for many previous batch verification or screening schemes *assumed* that the signatures (potentially submitted by a malicious adversary) were elements of an appropriate subgroup. For example, it was common place to assume that signatures submitted for batch DSA verification contained an element in a subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order q. Boyd and Pavlovski [6] pointed out efficient attacks on many batching algorithms via exploiting this issue. Of course, group membership cannot be *assumed*, it must be *tested* and the work required by this test might well obliterate all batching efficiency gains. E.g., verifying that an element y is in $\mathbb{G}$ by testing if $y^q \mod q = 1$; easily obliterates the gain of batching DSA signatures. Boyd and Pavlovski [6] suggest methods for overcoming this problem through careful choice of q.

In this paper, we will work in a bilinear setting, and we must be careful to avoid this common mistake in batch verification. To do so, we must say more about the groups in which we are working. Let E be an elliptic curve over a finite field $\mathbb{F}_p$ and let $\mathcal{O}$ denote the point at infinity. We denote the group of points on E defined over $\mathbb{F}_p$ as $E(\mathbb{F}_p)$. Then, a prime subgroup $\mathbb{G} \subseteq E(\mathbb{F}_p)$ of order q is chosen appropriately for our mapping. Our proofs will require that elements of purported signatures are members of $\mathbb{G}$ and *not* $E(\mathbb{F}_p) \setminus \mathbb{G}$. The question is: 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. Assuming we have a bilinear map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. In all the schemes we use, signatures are in $\mathbb{G}_1$, so this is the group we are interested in testing membership of. Elements in $\mathbb{G}_1$ will always be in $\mathbb{F}_p$ and have order q, so we can use cofactor multiplication: The curve has hq points over $\mathbb{F}_p$, so if an element y satisfies the curve equation and $hy \neq \mathcal{O}$ (here $\mathbb{G}_1$ is expressed in additive notation), then that element is in $\mathbb{G}_1$. If one chooses a curve with $h = 1$ then this test is trivial, but even if $h > 1$, but still much smaller than q, this test is efficient. Chen, Cheng and Smart discuss this and ways to test membership in $\mathbb{G}_2$ in [14].

4 Batch Verification without Random Oracles

In this section, we present a method for batch verifying signatures together with their accompanying certificates. We propose using Waters Two-Level Hierarchical Signatures [39, 7] with the first level corresponding to the certificate and the second level used for signing messages. We assume all certificates originate from the same authority. The scheme is secure under the Computational Diffie-Hellman assumption in the plain model. This batch verification method can execute in different modes, optimizing for the lowest runtime. Let n be the number of certificate/signature pairs, let 2^{k_1} be the number of users, and let k_2 be the bits per message. Then our batch verifier will verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly $\text{MIN}\{(2n+3)\ ,\ \ (k_1+n+3)\ ,\ \ (n+k_2+3)\ ,\ \ (k_1+k_2+3)\}$. Suppose there are one billion users ($k_1 = 30$) and RIPEMD-160 is used to hash all the messages ($k_2 = 160$), then when $n \geq 64$ batching becomes faster than individual verification. However, one can imagine many usage scenarios where devices send predefined messages requiring less than 160 bits, e.g., ISO defined error messages. For example, if $k_1 = 30$ and $k_2 = 32$, then when $n \geq 22$ batching should be used.

4.1 Batch Verification for Waters IBS

We describe a batch verification algorithm for the Waters IBS scheme from [7], where the number of pairings depends on the security parameter and not on the number of signatures. We assume that the identities are bit strings of length k_1 and the messages are bit strings of length k_2. First we describe the signature scheme. Let $\mathsf{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$.

Setup: First choose a secret $\alpha \in \mathbb{Z}_q$ and calculate $A = \mathbf{e}(g, g)^\alpha$. Then pick two random integers $y', z' \in \mathbb{Z}_q$ and two random vectors $y = (y_1, \ldots, y_{k_1}) \in \mathbb{Z}_q^{k_1}$ and $z = (z_1, \ldots, z_{k_2}) \in \mathbb{Z}_q^{k_2}$. The master secret key is $MK = g^\alpha$ and the public parameters are given as: $PP = g, A, u' = g^{y'}, u_1 = g^{y_1}, \ldots, u_{k_1} = g^{y_{k_1}}, v' = g^{z'}, v_1 = g^{z_1}, \ldots, v_{k_2} = g^{z_{k_2}}$

Extract: To create a private key for a user with identity $ID = (\kappa_1, \ldots, \kappa_{k_1}) \in \{0,1\}^{k_1}$, select $r \in \mathbb{Z}_q$ and return $K_{ID} = \left(g^\alpha (u' \prod_{i=1}^{k_1} u_i^{\kappa_i})^r, g^{-r}\right)$.

Sign: To sign a message $m = (m_1, \ldots, m_{k_2}) \in \{0,1\}^{k_2}$ using private key $K = (K_1, K_2)$, select $s \in \mathbb{Z}_q$ and return $S = \left(K_1 (v' \prod_{j=1}^{k_2} v_j^{m_j})^s, K_2, g^{-s}\right)$.

Verify: To verify a signature $S = (S_1, S_2, S_3)$ from identity $ID = (\kappa_1, \ldots, \kappa_{k_1})$ on message $M = (m_1, \ldots, m_{k_2})$, check that $\mathbf{e}(S_1, g) \cdot \mathbf{e}(S_2, u' \prod_{j=1}^{k_1} u_j^{\kappa_j}) \cdot \mathbf{e}(S_3, v' \prod_{j=1}^{k_2} v_j^{m_j}) = A$. If this equation holds, output *accept*; otherwise output *reject*.

We now introduce a batch verifier for this signature scheme. The basic idea is to adopt the small exponents test from [2] and to take advantage of the peculiarities of bilinear maps. Let *KeyGen*, *Sign* and *Verify* be as before.

Batch Verify: Let κ_j^i and m_k^i denote the j'th bit of the identity of the i'th signer respectively the k'th bit in the message signed by the i'th signer, and let $S^i = (S_1^i, S_2^i, S_3^i)$ denote the signature from the i'th signer. First check if $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ for all i. If not; output *reject*. Otherwise generate a vector

$\Delta = (\delta_1, \ldots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$ and set $P = \mathbf{e}(\prod_{i=1}^n {S_1^i}^{\delta_i}, g) \cdot \mathbf{e}(\prod_{i=1}^n {S_2^i}^{\delta_i}, u') \cdot \mathbf{e}(\prod_{i=1}^n {S_3^i}^{\delta_i}, v')$. Depending on the values of k_1, k_2 and n (c.f. below), pick and check one of the following equations:

$$\prod_{i=1}^n A^{\delta_i} = P \cdot \prod_{i=1}^n \mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{k_1} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^n \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{k_2} v_j^{m_j^i}) \tag{1}$$

$$\prod_{i=1}^n A^{\delta_i} = P \cdot \prod_{j=1}^{k_1} \mathbf{e}(\prod_{i=1}^n {S_2^i}^{\delta_i \kappa_j^i}, u_j) \cdot \prod_{i=1}^n \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{k_2} v_j^{m_j^i}) \tag{2}$$

$$\prod_{i=1}^n A^{\delta_i} = P \cdot \prod_{j=1}^{k_1} \mathbf{e}(\prod_{i=1}^n {S_2^i}^{\delta_i \kappa_j^i}, u_j) \cdot \prod_{j=1}^{k_2} \mathbf{e}(\prod_{i=1}^n {S_3^i}^{\delta_i m_j^i}, v_j) \tag{3}$$

Output *accept* if the chosen equation holds; otherwise output *reject*.

Let us discuss which equation should be picked. Assume that $k_1 < k_2$ (i.e., that fewer bits are used for the identities of the users than for the messages). If $n < k_1$ use equation 1, if $k_1 \leq n \leq k_2$ use equation 2; otherwise use equation 3.

Theorem 1. *The above algorithm is a batch verifier for the Waters IBS.*

Proof. First we prove that $\mathsf{Verify}(ID_{t_1}, M_1, S_1) = \cdots = \mathsf{Verify}(ID_{t_n}, M_n, S_n) = 1 \Rightarrow \mathsf{Batch}((ID_{t_1}, M_1, S_1), \ldots, (ID_{t_n}, M_n, S_n)) = 1$. This follows from the verification equation for the Waters IBS scheme:

$$\begin{aligned}
\prod_{i=1}^n A^{\delta_i} &= \prod_{i=1}^n \left(\mathbf{e}(S_1^i, g) \cdot \mathbf{e}(S_2^i, u' \prod_{j=1}^{k_1} u_j^{\kappa_j^i}) \cdot \mathbf{e}(S_3^i, v' \prod_{j=1}^{k_2} v_j^{m_j^i}) \right)^{\delta_i} \\
&= \mathbf{e}(\prod_{i=1}^n {S_1^i}^{\delta_i}, g) \cdot \prod_{i=1}^n \mathbf{e}({S_2^i}^{\delta_i}, u' \prod_{j=1}^{k_1} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^n \mathbf{e}({S_3^i}^{\delta_i}, v' \prod_{j=1}^{k_2} v_j^{m_j^i}) \\
&= P \cdot \prod_{i=1}^n \mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{k_1} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^n \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{k_2} v_j^{m_j^i}) \qquad (4)
\end{aligned}$$

Since for all i, $\mathsf{Verify}(ID_{t_i}, M_i, S_i) = 1$, S_2^i is a valid part of a signature, and hence $S_2^i \in \mathbb{G}$. This means that we can write $S_2^i = g^{b_i}$ for some $b_i \in \mathbb{Z}_q$ and get $\prod_{i=1}^n \mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{k_1} u_j^{\kappa_j^i}) = \prod_{i=1}^n \mathbf{e}(g^{\delta_i b_i}, g^{\sum_{j=1}^{k_1} y_j \kappa_j^i}) = \mathbf{e}(g,g)^{\sum_{i=1}^n \sum_{j=1}^{k_1} \delta_i b_i y_j \kappa_j^i} = \prod_{j=1}^{k_1} \mathbf{e}(g^{\sum_{i=1}^n \delta_i b_i \kappa_j^i}, g^{y_j}) = \prod_{j=1}^{k_1} \mathbf{e}(\prod_{i=1}^n {S_2^i}^{\delta_i \kappa_j^i}, u_j)$. We can do the same with $\prod_{i=1}^n \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{k_2} v_j^{m_j^i})$, so correctness of the different verification equations follows from this and equation 4.

We must now show the other direction. This proof is an application of the technique for proving the small exponents test in [2]. Since $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ we can write $S_1^i = g^{a_i}, S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some $a_i, b_i, c_i \in \mathbb{Z}_q$. This means that the verification equation for Waters IBS can be rewritten as:

$$
\begin{aligned}
\mathbf{e}(g,g)^{\alpha} &= \mathbf{e}(g^a,g)\cdot \mathbf{e}(g^b, g^{y'} g^{\sum_{j=1}^{k_1} y_j \kappa_j})\cdot \mathbf{e}(g^c, g^{z'} g^{\sum_{j=1}^{k_2} z_j m_j}) \\
&= \mathbf{e}(g,g)^{a+y'b+z'c+b\sum_{j=1}^{k_1} y_j\kappa_j + c\sum_{j=1}^{k_2} z_j m_j} \qquad (5)
\end{aligned}
$$

Using 5 we can rewrite equation 1 as:

$$
\mathbf{e}(g,g)^{\sum_{i=1}^{n}\delta_i\alpha} = \mathbf{e}(g,g)^{\sum_{i=1}^{n}\delta_i\left(a_i+y'b_i+z'c_i+b_i\sum_{j=1}^{k_1} y_j\kappa_j + c_i\sum_{j=1}^{k_2} z_j m_j\right)} \qquad (6)
$$

Setting $\beta_i = \alpha - \left(a_i + y'b_i + z'c_i + b_i\sum_{j=1}^{k_1} y_j\kappa_j^i + c_i\sum_{j=1}^{k_2} z_j m_j^i\right)$ and rewriting equation 6 we get:

$$
\begin{aligned}
&\mathbf{e}(g,g)^{\sum_{i=1}^{n}\delta_i\alpha - \sum_{i=1}^{n}\delta_i\left(a_i+y'b_i+z'c_i+b_i\sum_{j=1}^{k_1} y_j\kappa_j + c_i\sum_{j=1}^{k_2} z_j m_j\right)} = 1 \\
&\Rightarrow \sum_{i=1}^{n}\delta_i\alpha - \sum_{i=1}^{n}\delta_i\left(a_i + y'b_i + z'c_i + b_i\sum_{j=1}^{k_1} y_j\kappa_j + c_i\sum_{j=1}^{k_2} z_j m_j\right) \equiv 0 \pmod q \\
&\Rightarrow \sum_{i=1}^{n}\delta_i\beta_i \equiv 0 \pmod q \qquad (7)
\end{aligned}
$$

Assume that $\mathsf{Batch}((ID_{t_1}, M_1, S_1), \ldots, (ID_{t_n}, M_n, S_n)) = 1$, but for at least one i it is the case that $\mathsf{Verify}(ID_{t_i}, M_i, S_i) = 0$. Assume wlog that this is true for $i = 1$, which 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 7 gives us:

$$
\delta_1 \equiv -\gamma_1 \sum_{i=2}^{n}\delta_i\beta_i \pmod q \qquad (8)
$$

Given (ID_{t_i}, M_i, S_i) where $i = 1 \ldots n$, let E be an event that occurs if $\mathsf{Verify}(ID_{t_1}, M_1, S_1) = 0$ but $\mathsf{Batch}((ID_{t_1}, M_1, S_1), \ldots, (ID_{t_n}, M_n, S_n)) = 1$, or in other words that we break batch verification. Note that we do not make any assumptions about the remaining values. Let $\Delta' = \delta_2, \ldots, \delta_n$ denote the last $n-1$ values of Δ and let $|\Delta'|$ be the number of possible values for this vector. Equation 8 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: $\Pr[E] \leq \sum_{i=1}^{2^{\ell_b(n-1)}}\left(2^{-\ell_b}\cdot 2^{-\ell_b(n-1)}\right) = 2^{-\ell_b}$. □

5 Faster Batch Verification with Restrictions

In this section, we present a second method for batch verifying signatures together with their accompanying certificates. We propose using the BLS signature scheme [5] for the certificates and a modified version of the CL signature scheme [8] for signing messages. This method requires only two pairings to verify n certificates (from the same authority) and three pairings to verify n signatures (from possibly different signers). The cost for this significant efficiency gain is some usage restrictions, although as we will discuss, these restrictions may not be a problem for some of the applications we have in mind.

Certificates: We use a batch verifier for BLS signatures from the same authority as described in Section 5.1. The scheme is secure under CDH in the random oracle model. To verify n BLS certificates costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, using the Section 1.2 notation.

Signatures: We describe a new signature scheme CL* with a batch verifier in Section 5.2. The scheme is secure under the LRSW assumption in the plain model when the message space is a polynomial and in the random oracle model when the message space is super-polynomial. We assume that there are discrete time or location identifiers $\phi \in \Phi$. A user can issue at most one signature per ϕ (e.g., this might correspond to a device being allowed to broadcast at most one message every 300ms) and only signatures from the same ϕ can be batch verified together. To verify n CL* signatures, costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, where w is the output of a hash function.

5.1 Batch Verification of BLS Signatures

We describe a batch verifier for *many signers* for the Boneh, Lynn, and Shacham signatures [5, 4] described in Section 2, using the small exponents test [2].

Batch Verify: Given purported signatures σ_i from n users on messages M_i for $i = 1 \dots n$, first check that $\sigma_i \in \mathbb{G}$ for all i and if not; output *reject*. Otherwise compute $h_i = H(M_i)$ and generate a vector $\delta = (\delta_1, \dots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$. Check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i}$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 2. *The algorithm above is a batch verifier for BLS signatures.*

Proof. The proof is similar to proof 4.1 and omitted for space reasons.

Single Singer for BLS. However, BLS [5, 4] previously observed that if we have a single signer with public key v, the verification equation can be written as $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(\prod_{i=1}^n h_i^{\delta_i}, v)$ which reduces the load to only two pairings.

Theorem 3 ([5, 4]). *The algorithm above is a single-signer, batch verifier for BLS signatures.*

5.2 A New Signature Scheme CL*

In this section we introduce a new signature scheme secure under the LRSW assumption [33], which is based on the Camenisch and Lysyanskaya signatures [8].

Assumption 4 (LRSW Assumption) *Let* $\mathsf{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$*. Let* $X, Y \in \mathbb{G}$*,* $X = g^x$*, and* $Y = g^y$*. Let* $\mathcal{O}_{X,Y}(\cdot)$ *be an oracle that, on input a value* $m \in \mathbb{Z}_q^*$*, outputs a triple* $A = (a, a^y, a^{x+mxy})$ *for a randomly chosen* $a \in \mathbb{G}$*. Then for all probabilistic polynomial time adversaries* $\mathcal{A}^{(\cdot)}$*,* $\nu(\ell)$ *defined as follows is a negligible function:*

$$\Pr[(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}) \leftarrow \mathsf{BSetup}(1^\ell); x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q; X = g^x; Y = g^y;$$
$$(m, a, b, c) \leftarrow \mathcal{A}^{\mathcal{O}_{X,Y}}(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y) \ : \ m \notin Q \ \wedge \ m \in \mathbb{Z}_q^* \ \wedge$$
$$a \in \mathbb{G} \wedge \ b = a^y \ \wedge \ c = a^{x+mxy}] = \nu(\ell) \ ,$$

where Q *is the set of queries that* $\mathcal{A}$ *made to* $\mathcal{O}_{X,Y}(\cdot)$*.*

The Original CL Scheme. Recall the Camenisch and Lysyanskaya signatures [8]. Let $\mathsf{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Choose the secret key $(x, y) \in \mathbb{Z}_q^2$ at random and set $X = g^x$ and $Y = g^y$. The public key is $pk = (X, Y)$. To sign a message $m \in \mathbb{Z}_q^*$, choose a random $a \in \mathbb{G}$ and compute $b = a^y$, $c = a^x b^{xm}$. Output the signature (a, b, c). To verify, check whether $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$ holds.

CL*: A version of the CL Scheme Allowing Batch Verification. Our goal is to batch-verify CL signatures made by different signers. That is we need to consider how to verify equations of the form $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$. The fact that the values X, a, b, and c are different for each signature seems to prevent efficient batch verification. Thus, we need to find a way such that many different signers share some of these values. Obviously, X and c need to be different. Now, depending on the application, all the signers can use the same value a by choosing a as the output of some hash function applied to, e.g., the current time period or location. We then note that all signers can use the same b in principle, i.e., have all of them share the same Y as it is sufficient for each signer to hold only one secret value (i.e., $sk = x$). Indeed, the only reason that the signer needs to know Y is to compute b. However, it turns out that if we define b such that $\log_a b$ is not known, the signature scheme is still secure. So, for instance, we can derive b in a similar way to a using a second hash function. Thus, all signers will virtually sign using the same Y per time period (but a different one for each period).

Let us now describe the resulting scheme. Let $\mathsf{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Let $\phi \in \Phi$ denote the current time period or location, where $|\Phi|$ is polynomial. Let $\mathcal{M}$ be the message space, for now let $\mathcal{M} = \{0, 1\}^*$. Let $H_1 : \Phi \rightarrow \mathbb{G}$, $H_2 : \Phi \rightarrow \mathbb{G}$, and $H_3 : \mathcal{M} \times \Phi \rightarrow \mathbb{Z}_q$ be different hash functions.

KeyGen: Choose a random $x \in \mathbb{Z}_q$ and set $X = g^x$. Set $sk = x$ and $pk = X$.

Sign: If this is the first call to Sign during period $\phi \in \Phi$, then on input message $m \in \mathcal{M}$, set $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$ and output the signature $\sigma = a^x b^{xw}$. Otherwise, abort.

Verify: On input message-period pair (m, ϕ) and purported signature σ, compute $w = H_3(m||\phi)$, $a = H_1(\phi)$ and $b = H_2(\phi)$, and check that $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$. If true, output *accept*; otherwise output *reject*.

Theorem 5. *Under the LRSW assumption in $\mathbb{G}$, the CL* signature scheme is existentially unforgeable in the random oracle model for message space $\mathcal{M} = \{0,1\}^*$.*

Proof. We show that if there exists a p.p.t. adversary $\mathcal{A}$ that succeeds with probability ε in forging CL* signatures, then we can construct a p.p.t. adversary $\mathcal{B}$ that solves the LRSW problem with probability $\varepsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$ in the random oracle model, where q_H is the maximum number of oracle queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$. Recall that $|\Phi|$ is a polynomial. Adversary $\mathcal{B}^{\mathcal{O}_{X,Y}(\cdot)}$ against LRSW operates as follows on input $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y)$. Let ℓ be the security parameter. We assume that Φ is pre-defined. Let q_H be the maximum number of queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$.

1. *Setup:* Send the bilinear parameters $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ to $\mathcal{A}$. Choose a random $w' \in \mathcal{M}$ and query $\mathcal{O}_{X,Y}(w')$ to obtain an LRSW instance (w', a', b', c'). Choose a random $\phi' \in \Phi$. Treat H_1, H_2, H_3 as random oracles. Allow $\mathcal{A}$ access to the hash functions H_1, H_2, H_3.
2. *Key Generation:* Set $pk^* = X$. For $i = 1$ to n, choose a random $sk_i \in \mathbb{Z}_q$ and set $pk_i = g^{sk_i}$. Output to $\mathcal{A}$ the keys pk^* and all (pk_i, sk_i) pairs.
3. *Oracle queries:* $\mathcal{B}$ responds to $\mathcal{A}$'s hash and signing queries as follows. Choose random r_i and s_i in $\mathbb{Z}_q$ for each time period (except ϕ'). Set up H_1 and H_2 such that:
$$H_1(\phi_i) = \begin{cases} g^{r_i} & \text{if } \phi_i \neq \phi' \\ a' & \text{otherwise} \end{cases} \tag{9}$$
and
$$H_2(\phi_i) = \begin{cases} g^{s_i} & \text{if } \phi_i \neq \phi' \\ b' & \text{otherwise} \end{cases} \tag{10}$$
Pick a random j in the range $[1, q_H]$. Choose random $t_{l,i} \in \mathbb{Z}_q$, such that $t_{l,i} \neq w'$, for $l \in [1, q_H]$ and $i \in [1, |\Phi|]$. Set up H_3 such that:
$$H_3(m_l||\phi_i) = \begin{cases} t_{l,i} & \text{if } \phi_i \neq \phi' \text{ or } l \neq j \\ w' & \text{otherwise} \end{cases} \tag{11}$$
$\mathcal{B}$ records $m^* := m_j$. Finally, set the signing query oracle such that on the lth query involving period ϕ_i:
$$\mathcal{O}_{sk^*}(m_l||\phi_i) = \begin{cases} \text{abort} & \text{if } \phi_i = \phi' \text{ and } l \neq j \\ c' & \text{else if } \phi_i = \phi' \text{ and } l = j \\ X^{r_i} X^{(s_i) t_{l,i}} & \text{otherwise} \end{cases} \tag{12}$$
4. *Output:* At some point $\mathcal{A}$ stops and outputs a purported forgery $\sigma \in \mathbb{G}$ for some (m_l, ϕ_i). If $\phi_i \neq \phi'$, $\mathcal{B}$ did not guess the correct period and thus $\mathcal{B}$ outputs a random guess for the LRSW game. If $m_l = m^*$ or the CL* signature does not verify, $\mathcal{A}$'s output is not a valid forgery and thus $\mathcal{B}$ outputs a random guess for the LRSW game. Otherwise, $\mathcal{B}$ outputs $(t_{l,i}, a', b', \sigma)$ as the solution to the LRSW game.

We now analyze $\mathcal{B}$'s success. If $\mathcal{B}$ is not forced to abort or issue a random guess, then we note that $\sigma = H_1(\phi_i)^x H_2(\phi_i)^{xH_3(m_l||\phi_i)}$. In this scenario $\phi_i = \phi'$ and $t_{l,i} \neq w'$. We can substitute as $\sigma = (a')^x(b')^{x(t_{l,i})}$. Thus, we see that $(t_{l,i}, a', b', \sigma)$ is indeed a valid LRSW instance. Thus, $\mathcal{B}$ succeeds at LRSW whenever $\mathcal{A}$ succeeds in forging CL* signatures, except when $\mathcal{B}$ is forced to abort or issue a random guess. First, when simulating the signing oracle, $\mathcal{B}$ is forced to abort whenever it incorrectly guesses which query to H_3, during period ϕ', $\mathcal{A}$ will eventually query to $\mathcal{O}_{sk^*}(\cdot,\cdot)$. Since all outputs of H_3 are independently random, $\mathcal{B}$ will be forced to abort at most q_H^{-1} probability. Next, provided that $\mathcal{A}$ issued a valid forgery, then $\mathcal{B}$ is only forced to issue a random guess when it incorrectly guesses which period $\phi \in \Phi$ that $\mathcal{A}$ will choose to issue its forgery. Since, from the view of $\mathcal{A}$ conditioned on the event that $\mathcal{B}$ has not yet aborted, all outputs of the oracles are perfectly distributed as either random oracles (H_1, H_2, H_3) or as a valid CL* signer $(\mathcal{O}_{sk^*})$. Thus, this random guess is forced with probability at most $|\Phi|^{-1}$. Thus, if $\mathcal{A}$ succeeds with ε probability, then $\mathcal{B}$ succeeds with probability $\varepsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$. □

Theorem 6. *Under the LRSW assumption in $\mathbb{G}$, the CL* signature scheme is existentially unforgeable in the plain model when $|\mathcal{M}|$ is polynomial.*

Proof sketch. If there exists a p.p.t. adversary $\mathcal{A}$ that succeeds with probability ε in forging CL* signatures when $|\mathcal{M}| = \text{poly}(\ell)$, then we can construct a p.p.t. adversary $\mathcal{B}$ that solves the LRSW problem with probability $\varepsilon \cdot |\Phi|^{-1} \cdot |\mathcal{M}|^{-1}$. Canetti, Halevi, and Katz [9] described one method of constructing a universal one-way hash function that satisfies a polynomial number of input/output constraints, i.e., pairs (x_i, y_i) such that $H(x_i) = y_i$. Furthermore, we note that H_1, H_2 and H_3 have $|\Phi|$, $|\Phi|$, and $|\Phi| \cdot |\mathcal{M}|$ constraints, respectively. Since these are all polynomials, $\mathcal{B}$ can efficiently construct the appropriate hash functions. The analysis follows the proof with random oracles.

Efficiency Note. First, we observe that the CL* signatures are *very* short, requiring only one element in $\mathbb{G}$. Since the BLS signatures also require only one element in $\mathbb{G}$, and since a public key for the CL* scheme is also only one group element, the entire signature plus certificate could be transmitted in three $\mathbb{G}$ elements. In order to get the shortest representation for these elements, we need to use asymmetric bilinear maps $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1 \neq \mathbb{G}_2$, which will allow elements in $\mathbb{G}_1$ to be 160 bits and elements of $\mathbb{G}_2$ to be 1024 bits for a security level comparable to RSA-1024 [28, 18]. For BLS this means that the public key will be around 1024 bits, but since we use it for single signer, the public key of the certifying authority is probably embedded in the systems at production time. For CL* signatures we need to hash into $\mathbb{G}_1$ which according to Galbraith, Paterson and Smart [18] can be done efficiently. To summarize; using BLS and CL* we can represent the signature plus certificate using approximately 1344 bits with security comparable to RSA-1024, compared to around 3072 bits for actually using RSA-1024. We note that this is based on current state of the art for pairings, and might improve in the future.

Second, suppose one uses the universal one-way hash functions described by Canetti, Halevi, and Katz [9] to remove the random oracles from CL*. These hash functions require one exponentiation per constraint. In our case, we may require as many as $|\Phi| \cdot |\mathcal{M}|$ constraints. Thus, the cost to compute the hashes may dampen the efficiency gains of batch verification. However, our scheme will benefit from improvements in the construction of universal one-way hash functions with constraints. To keep $|\Phi|$ small in practice, users might need to periodically change their keys.

Batch Verification of CL* Signatures. Batch verification of n signatures $\sigma_1, \ldots, \sigma_n$ on messages $m_1, \ldots, m_n$ for the same period ϕ can be done as follows. Assume that user i with public key X_i signed message m_i. Set $w_i = H(m_i || \phi)$. First check if $\sigma_i \in \mathbb{G}$ for all i. If not; output *reject*. Otherwise pick a vector $\Delta = (\delta_i, \ldots, \delta_n)$ with each element being a random ℓ_b-bit number and check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(a, \prod_{i=1}^n X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^n X_i^{w_i \delta_i})$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 7. *The algorithm above is a batch verifier for CL* signatures.*

Proof. The proof is similar to proof 4.1 and omitted for space reasons.

6 Conclusions and Open Problems

In this paper we focused on batch verification of signatures. We overviewed the large body of existing work, almost exclusively dealing with single signers. We extended the general batch verification definition of Bellare, Garay and Rabin [2] to the case of multiple signers. We then presented, to our knowledge, the first efficient and practical batch verification scheme for signatures without random oracles. We focused on solutions that comprehended the time to verify the signature *and* the corresponding certificate for the verification key. First, we presented a batch verifier for the Waters IBS that can verify n signatures using only $(k_1 + k_2 + 3)$ pairings (the dominant operation), where identities are k_1 bits and messages are k_2 bits. This is a significant improvement over the $3n$ pairings required by individual verification. Second, we presented a solution in the random oracle model that batch verifies n certificates and n CL* signatures using only 5 pairings. Here, CL* is a variant of the Camenisch-Lysyanskaya signatures that is much shorter, allows for efficient batch verification from many signers, but where only one signature can be safely issued per period.

It is an open problem to find a fast batch verification scheme for short signatures without the period restrictions from Section 5. Perhaps this can be achieved by improving the efficiency of our scheme in Section 4, using some of the techniques applied to the Waters IBE by Naccache [34] and Chatterjee and Sarkar [13]. Another exciting open problem is to develop fast batching schemes for various forms of anonymous authentication such as group signatures, e-cash, and anonymous credentials.

Acknowledgments

We thank Jean-Pierre Hubaux and Panos Papadimitratos for helpful discussions about the practical challenges of vehicular networks. We are also grateful to Paulo Barreto, Steven Galbraith, Hovav Shacham, and Nigel P. Smart for their comments on testing membership in bilinear groups. Finally we thank Ivan Damgård and the anonymous reviewers for their valuable input. Jan Camenisch was supported by the EU projects ECRYPT and PRIME, contracts IST-2002-507932 and IST-2002-507591. Michael Østergaard Pedersen was supported by the eu-DOMAIN IST EU project, contract no. IST-004420.

References

1. J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EUROCRYPT '02*, volume 2332, p. 83–107, 2002.
2. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT '98*, LNCS vol. 1403.
3. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT '03*, LNCS vol. 2656.
4. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
5. D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. In *ASIACRYPT '01*, volume 2248 of LNCS, p. 514–532, 2001.
6. C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. In *ASIACRYPT '00*, volume 1976 of LNCS, p. 58–71, 2000.
7. X. Boyen and B. Waters. Compact group signatures without random oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, p. 427–444, 2006.
8. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04*, volume 3152 of LNCS, p. 56–72, 2004.
9. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT '03*, volume 2656 of LNCS, p. 255–271, 2003.
10. T. Cao, D. Lin, and R. Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.
11. Car 2 Car. Communication consortium. `http://car-to-car.org`.
12. J. C. Cha and J. H. Cheon. An identity-based signature from gap Diffie-Hellman groups. In *PKC '03*, p. 18–30, 2003.
13. S. Chatterjee and P. Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In *ICISC '05*, volume 3935 of LNCS, p. 257–273, 2005.
14. L. Chen, Z. Cheng, and N. Smart. Identity-based key agreement protocols from pairings, 2006. Cryptology ePrint Archive: Report 2006/199.
15. J. H. Cheon, Y. Kim, and H. J. Yoon. A new ID-based signature with batch verification, 2004. Cryptology ePrint Archive: Report 2004/131.
16. S. Cui, P. Duan, and C. W. Chan. An efficient identity-based signature scheme with batch verifications. In *InfoScale '06*, p. 22, 2006.
17. A. Fiat. Batch RSA. In *CRYPTO '89*, volume 435, p. 175–185, 1989.
18. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.

19. C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In *Public Key Cryptography 2005*, volume 3958 of LNCS, p. 257–273, 2006.
20. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.
21. R. Granger and N. Smart. On computing products of pairings, 2006. Cryptology ePrint Archive: Report 2006/172.
22. L. Harn. Batch verifying multiple DSA digital signatures. In *Electronics Letters*, volume 34(9), p. 870–871, 1998.
23. L. Harn. Batch verifying multiple RSA digital signatures. In *Electronics Letters*, volume 34(12), p. 1219–1220, 1998.
24. F. Hoshino, M. Abe, and T. Kobayashi. Lenient/strict batch verification in several groups. In *ISC '01*, p. 81–94, 2001.
25. M.-S. Hwang, C.-C. Lee, and Y.-L. Tang. Two simple batch verifying multiple digital signatures. In *ICICS '01*, p. 233–237, 2001.
26. M.-S. Hwang, I.-C. Lin, and K.-F. Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lith. Acad. Sci.*, 11(1):15–19, 2000.
27. IEEE. 5.9 GHz Dedicated Short Range Communications. `http://grouper.ieee.org/groups/scc32/dsrc`.
28. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels, 2005. Cryptology ePrint Archive: Report 2005/076.
29. C.-S. Laih and S.-M. Yen. Improved digital signature suitable for batch verification. *IEEE Trans. Comput.*, 44(7):957–959, 1995.
30. S. Lee, S. Cho, J. Choi, and Y. Cho. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(1):74–80, 2006.
31. C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), p. 1592–1593, 1994.
32. C. H. Lim. Efficient multi-exponentation and application to batch verification of digital signatures, 2000. `http://dasan.sejong.ac.kr/~chlim/english_pub.html`.
33. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *SAC*, volume 1758 of *LNCS*, p. 184–199, 1999.
34. D. Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.
35. D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be improved? complexity trade-offs with the digital signature standard. In *EUROCRYPT '94*, volume 0950 of LNCS, p. 77–85, 1994.
36. M. Raya and J.-P. Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15:39–68, 2007.
37. SeVeCom. Security on the road. `http://www.sevecom.org`.
38. M. Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.
39. B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05*, volume 3494 of LNCS, p. 320–329, 2005.
40. H. Yoon, J. H. Cheon, and Y. Kim. Batch verifications with ID-based signatures. In *ICISC*, p. 233–248, 2004.
41. F. Zhang and K. Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In *ACISP '03*, volume 2727 of LNCS, p. 312–323, 2003.
42. F. Zhang, R. Safavi-Naini, and W. Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *Indocrypt 2003*, volume 2904 of LNCS, p. 191–204, 2003.