Batch Verification of Short Signatures

Jan Camenisch[*] Susan Hohenberger[†] Michael Østergaard Pedersen[‡]

August 1, 2007

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. Prior work focused almost exclusively on batching signatures from the same signer.

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. 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 very efficient and still practical for some 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 [14, 44]. Raya and Hubaux suggest that vehicles will transmit safety messages every 300ms to all other vehicles within a minimum range of 110 meters [42], which in turn may retransmit these messages.

For such pervasive systems to work properly, there are many competing constraints [14, 44, 31, 42]. 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 [31]. 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.

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

Third, different messages from many different signers may need to be verified and processed quickly (e.g., every 300ms [42]). 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 for some applications (see Section 8.2 of [42]). While many new schemes based on bilinear maps can provide the same security with significantly smaller signatures, they take significantly more time to verify. Thus, it is not immediately clear what the proper tradeoff between message length and verification time is for many pervasive communication applications. However, in some applications, there is evidence that doing a *small* amount of additional computation is more advantageous than sending longer messages. For example, Landsiedel, Wehrle, and Götz showed that for applications using Mica2 sensors transmitting data consumes significantly more battery power than keeping the CPU active [34].

Fast verification of many signatures are an interesting problem in other scenarios as well. Consider a scenario where a mail server receives a lot of signed e-mails. To handle a variety of different e-mail clients on the internal network, it is easier to let the server do signature verification and insert a message into the body of the e-mail about who signed it. Assuming the internal network and the mail server are secure, clients can rely on the signature being correct without having to verify it themselves. However, the actual digital signature can still be attached to the e-mail should a dispute about the authenticity of the message later arise. To keep resource usage on the server to a minimum, signature verification should be fast, but we can take advantage of the fact that the server can buffer messages for a short period before verifying all of them.

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 [3] 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 Π-IBS scheme [16]. (More precisely, this is the IBS scheme implicitly defined by the Chatterjee-Sarkar hierarchical IBE [16] and it can also be viewed as a generalized version of the Boyen-Waters IBS [9] as we will discuss later.) To our knowledge, this is the first batch verifier for a signature scheme without random oracles. Let z be the additional security parameter required by the Π-IBS. When identities and messages are k bits, viewed as z chunks of k/z bits each, our algorithm verifies n Π-IBS signatures using only $(z + 3)$ pairings. Individually verifying n signatures would cost $3n$ pairings.

3. We present a new signature scheme, Π-Sig, derived from the Camenisch and Lysyanskaya signature scheme [11], which is secure in the random oracle model. Π-Sig signatures require only one-third the space of the original CL signatures– on par with the shortest signatures known [7] –, 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. Π-Sig 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. To achieve this functionality with Π-Sig signatures, we can issue signatures with Π-Sig and certificates with the Boneh, Lynn and Shacham signatures [7]. Then we can batch the Π-Sig signatures (on any message from any signer) using a new batch verifier proposed herein; and we can batch the BLS certificates (on any public key from the same authority) using a known batch verifier 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 [21] for a variant of RSA. Later, in 1994, Naccache, M'Raïhi, Vaudenay and Raphaeli [41] 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 [36]. In 1995 Laih and Yen proposed a new method for batch verification of DSA and RSA signatures [33], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [8]. In 1998 Harn presented two batch verification techniques for DSA and RSA [26, 27] but both were later broken [8, 29, 30]. The same year, Bellare, Garay and Rabin took the first systematic look at batch verification [3] 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 [41, 33]. 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 [8]. 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 [28] pointed out that the problem discovered in [8] 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 [18], but their security proof is for aggregate signatures and does not meet the definition of batch verification from [3]; 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 [23] that does not claim to be, nor is, a batch verification scheme. Other schemes for batch verification based on bilinear maps were proposed [15, 47, 48, 49] but all were later broken by Cao, Lin and Xue [13]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [35], but Stanek [45] showed that this method is flawed.

Bellare, Garay and Rabin Testing Techniques. Let g generate a group of prime order. In 1998, Bellare, Garay and Rabin described some tests [3], for verifying equations of the form $y_i = g^{x_i}$ for $i = 1$ to n, which we will use again. Obviously if one just multiplies these equations together and checks if $\prod_{i=1}^{n} y_i = g^{\sum_{i=1}^{n} x_i}$, it is 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 $(x_2 + \alpha, y_2)$ instead. Let us review three fixes to this broken proposal.

Random Subset Test: The first idea is to pick a random subset of these pairs (x_i, y_i) and multiply them together, hoping to split up the pairs that were specifically crafted to cancel each other out. Repeating this test ℓ times, picking a new random subset every time, results in the probability of accepting invalid pairs being $2^{-\ell}$.

Small Exponents Test: Instead of picking a random subset every time, one can instead choose exponents δ_i of (a small number of) ℓ bits and compute $\prod_{i=1}^{n} y_i^{\delta_i} = g^{\sum_{i=1}^{n} x_i \delta_i}$. They also prove that this test results in the probability of accepting a bad pair being $2^{-\ell}$. The size of ℓ is a tradeoff between efficiency and security and hence it is difficult to give an exact recommendation for it. It all depends on the application and how critical it is not to accept even a single invalid signature. For just a rough check that all signatures are correct 20 bits seems reasonable. In a higher security setting we should probably be using around 64 bits.

Bucket Test: Finally, a method called the *bucket test* is even more efficient than the small exponents test for large values of n. The idea is to repeat a test called the *atomic bucket test* m times. The atomic bucket test works by first putting the n instances one wants to verify into M buckets at random. This results in M new instances of the same problem, which are then checked using the small exponents test with security parameter m. After repeating the atomic bucket test m times, the probability of accepting a bad pair in the original n instances is at most 2^{-m}.

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 Π-IBS, Π-Sig and BLS signatures for many signers. Note that Lim [37] provides a number of efficient methods for doing m-term exponentiations and Granger and Smart [25] 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 [8]. 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 $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{Z}_m}(\ell_b) + \mathsf{ExpCost}_{\mathbb{Z}_m}(k)$, where k is the number of bits in the public exponent e, can be found in [8]. 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 ($|e| < 2\ell_b/3$) the naive method is a faster way to verify RSA signatures and it can also handle signatures from multiple signers. Bellare et al. [3] 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 [41] compatible with the *small exponents test* from [8]. 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 $n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(\ell_b) + \mathsf{ExpCost}^2_{\mathbb{G}}(160) + \mathsf{HashCost}^n_{\mathbb{G}} + \mathsf{MultCost}^{2n+1}$ multiplications. This method works for a single signer only.

Π-IBS is an IBS scheme derived from the Chatterjee and Sarkar HIBE scheme [16] 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 and messages are k bits divided into z logical chunks, each of k/z bits, where z is a security parameter, and a signature is three bilinear group elements. The computational effort required depends on the number of messages and the security parameters. Let $\mathsf{M} = n\text{-}\mathsf{MultExpCost}_{\mathbb{G}_T}(\ell_b) + n\text{-}\mathsf{MultExpCost}^3_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^{3n}_{\mathbb{G}} + \mathsf{MultCost}^3$ and refer to the table below for efficiency of the scheme.

$$\begin{array}{lll} n \leq 2z: & \mathsf{M} & +2n\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + z\text{-}\mathsf{MultExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z}) + \mathsf{ExpCost}^{2n}_{\mathbb{G}}(\ell_b) \\ n > 2z: & \mathsf{M} & +z\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + \mathsf{ExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z} + \ell_b) + \mathsf{MultCost}^{zn} \end{array}$$

The naive application of Π-IBS to verify n signatures costs $\mathsf{PairCost}^{3n}_{\mathbb{G},\mathbb{G}} + z\text{-}\mathsf{MultExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z}) + \mathsf{MultCost}^{4n}$. Also note that in many security applications we do not need to transmit the identity as a separate parameter, as it is already included in the larger protocol. For example, the identity may be the hardware address of the network interface card.

BLS is the signature scheme by Boneh, Lynn and Shacham [7]. 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 [6] 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 $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$.

Π-Sig is a new variant of Camenisch and Lysyanskaya signatures [11] presented in Section 5 designed specifically to enable efficient batch verification. 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.

Small Exponents and Bucket Tests. Recall the various testing techniques covered in Section 1.2. Our batch verifiers in this paper make use of the small exponents test, but since the bucket test uses the small exponents test as a subroutine, we note that we can also use the bucket test to further speed up verification of many signatures.

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 $\mathsf{Gen}(1^\ell) \rightarrow (pk, sk)$, the algorithm $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

There are two common notions of security. Goldwasser, Micali and Rivest [24] 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 the signing 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 [24] definition, while those in Section 5 meet the stronger 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 [3] 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 2.1 (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 2.1 requires that signing keys be generated honestly, but then they can be later held by an adversary. In practice, users could register their keys and prove some necessary properties of the keys at registration time [2].

Confusion between Batch Verification, Aggregate Signatures and Screening. As we discussed in the introduction, several works (e.g., [18, 19]) claim to do batch verification when, in fact, they often meet a weaker guarantee called *screening* [3]. In most cases the confusion is about words, i.e., when the words *batch verification* are used to describe an aggregate signature scheme.

Definition 2.2 (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 $(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^\ell)$. Let $\mathcal{O}_{sk_0}(\cdot)$ be an oracle that on input m outputs $\sigma = \mathsf{Sign}(sk_0, m)$. 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:*

$$\begin{aligned}\Pr[&(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^\ell), (pk_1, sk_1) \leftarrow \mathsf{Gen}(1^\ell), \ldots, (pk_n, sk_n) \leftarrow \mathsf{Gen}(1^\ell),\\ &\qquad D \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot)}(pk_0, (pk_1, sk_1), \ldots, (pk_n, sk_n)) :\\ &\qquad\qquad \mathsf{Screen}(D) = 1 \;\wedge\; (pk_0, m, \sigma) \in D \;\wedge\; m \notin Q] = \mu(\ell),\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{sk_0}(\cdot)$ and for all $(a, b, c) \in D$, $a \in \{0, \ldots, n\}$.

The above definition is generalized to the multiple-signer case from the single-signer screening definition of Bellare, Garay and Rabin [3].

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 [6] based on the BLS signatures [7]. First, we review 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. To generate a key pair, choose a random $sk \in \mathbb{Z}_q$ and set $pk = g^{sk}$. A signature on message m is $\sigma = H(m)^{sk}$, where $H : \{0,1\}^* \rightarrow \mathbb{G}$ 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 2.2 (as BGLS prove [6]), but will not verify under Definition 2.1. 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! Also recall the e-mail scenario from section 1. If we only did screening on the server, a user could send n messages with invalid signatures (to different receivers) that would screen correctly. The sender could then later claim that he did not send one of the messages and indeed the signature will not verify unless one can get hold of *all* n messages! To be fair, batch verification is not what aggregate schemes were designed to do, but it is a common misuse of them.

Let's make one final observation about the relationship between batch verification and screening. 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. That is, for all i, the holder of sk_{t_i} helped to create σ_i, which may or may not be a valid signature for m_i. Thus, a screening scheme can be employed to hold users accountable for the messages they "sign" in a set D such that $\mathsf{Screen}(D) = 1$, but to do this the entire set D must be recorded or retransmitted to a third party. In the authenticated email scenario, where the mailserver is verifying the signatures on emails for many different users, releasing D (in the event of disputes) raises serious privacy issues.

One could consider releasing a non-interactive zero-knowledge proof of knowledge of D such that $\mathsf{Screen}(D) = 1$, although the naive approach will require $O(|D|)$ space and $O(|D|)$ time to verify.

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. This bilinear map is called a *symmetric bilinear map.* A more general version of the bilinear map is the *asymmetric bilinear map* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct groups, possibly without efficient isomorphisms between them. Getting into details about how these bilinear maps are constructed is not the purpose of this paper, so we just give a very brief overview required for reasoning about the efficiency of our schemes.

$\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on some curve and $\mathbb{G}_T$ is a subgroup of a multiplicative group over a related finite field. All groups have the same order q. Let E be an elliptic curve. We denote the group of points on E defined over $\mathbb{F}_p$ as $E(\mathbb{F}_p)$. $\mathbb{G}_1$ (or $\mathbb{G}$ in the symmetric setting) is a subgroup of $E(\mathbb{F}_p)$, $\mathbb{G}_2$ is usually defined as a subgroup of $E(\mathbb{F}_{p^k})$ where k is the embedding degree and $\mathbb{G}_T$ is a subgroup of $E(\mathbb{F}^*_{p^k})$. Let F be the number of bits in the representation of p and let G be the number of bits in the representation of q. In the asymmetric setting, one can choose $G = F$ which means that elements of $\mathbb{G}_1$ can be over the smallest possible field, however this is not possible for the symmetric case due to constraints on the possible choices of p, q and k. Hence in the symmetric case $G < F$ and hence we can not represent element of $\mathbb{G}$ using only G bits [43].

So far it seems that the asymmetric setting allows for the shortest group elements, but this is only half the truth. The MOV attack states that solving the discrete logarithm problem on a curve, reduces to solving it over the corresponding finite field [39], which means that the bitlength of p^k must be comparable to that of an RSA modulus to provide the same level of security. This has implications for the size of elements in $\mathbb{G}_2$. For the asymmetric case, if we aim for security level comparable to 1024 bits RSA, one can choose F to be approximately 160 bits and hence elements in $\mathbb{G}_1$ will be 160 bits [32, 22]. However, elements in $\mathbb{F}_{p^k}$ (and hence $\mathbb{G}_2$) must be of size 1024 bits. In the symmetric case, F must be at least 512 bits for curves with embedding degree $k = 2$, and hence we need 512 bits to represent an element of $\mathbb{G}$.

Our constructions from Section 5 also work in the asymmetric setting which allows us to use a short representation of the signatures. The Π-IBS scheme from Section 4 can be modified to work in the asymmetric setting, but some parts of the signature will end up in the large group. We refer to the *efficiency note* paragraphs in Section 4 and 5 for a more detailed discussion.

Complexity Assumptions. In the coming sections, we will refer to the following complexity assumptions.

Assumption 3.1 (Computational Diffie-Hellman [20]) *Let g generate a group $\mathbb{G}$ of prime order $q \in \Theta(2^\ell)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is at most 1/2 plus a negligible function in ℓ:*

$$\Pr[a, b, c \leftarrow \mathbb{Z}_q;\ x_0 \leftarrow g^{ab};\ x_1 \leftarrow g^c;\ z \leftarrow \{0, 1\}; z' \leftarrow \mathcal{A}(g, g^a, g^b, x_z) : z = z'].$$

Assumption 3.2 (Decisional Bilinear Diffie-Hellman [5]) *Let* $\mathsf{BSetup}(1^\ell) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, *where g generates $\mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is at most 1/2 plus a negligible function in ℓ:*

$$\Pr[a, b, c, d \leftarrow \mathbb{Z}_q;\ x_0 \leftarrow \mathbf{e}(g,g)^{abc};\ x_1 \leftarrow \mathbf{e}(g,g)^d;\ z \leftarrow \{0,1\}; z' \leftarrow \mathcal{A}(g, g^a, g^b, g^c, x_z) : z = z'].$$

Assumption 3.3 (LRSW [38]) *Let* $\mathsf{BSetup}(1^\ell) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. *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 $a \in \mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}^{(\cdot)}$, the following probability is negligible in ℓ:*

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

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

Testing Membership in $\mathbb{G}$. In a *non-bilinear* setting, Boyd and Pavlovski [8] 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 [8] 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 $y^q \bmod q = 1$; easily obliterates the gain of batching DSA signatures. Boyd and Pavlovski [8] 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. 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 $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. 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 $\mathcal{O}$ is the point at infinity and $\mathbb{G}_1$ is expressed in additive notation), then that element is in $\mathbb{G}_1$. If h is small then this test is efficient. Chen, Cheng and Smart [17] discuss this and ways to test membership in $\mathbb{G}_2$.

4 Batch Verification without Random Oracles

In this section, we present a method for batch verifying an identity-based signature scheme Π-IBS. 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 be the number of users and let there be k bits per message. Let z be the additional security parameter required by the Π-IBS. Furthermore assume that the k bits are divided into z elements of k/z bits each. Then our batch verifier will verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly $\text{MIN}\{(2n+3)\ ,\ (z+3)\}$.

On the practical side, we note that as z grows there is a corresponding degradation in the concrete security of the IBS scheme (see [16] for a detailed discussion of these tradeoffs.) Setting $z = k/32$, however, seems a reasonable choice. Suppose we use SHA256 to hash all the messages ($k = 256$) and we choose the elements to be 32 bits ($k/z = 32$), then roughly when $n \geq 3$ batch verification becomes faster than individual verification.

4.1 Batch Verification for Π-IBS

We describe a batch verification algorithm for the Π-IBS scheme [16], where the number of pairings depends on the security parameter and not on the number of signatures and where no random oracles are necessary. The underlying Π-IBS signature scheme appears only implicitly in prior work, so let us clearly explain its origin. We begin with the observation by Boyen and Waters that an IBS scheme is realized by the key issuing algorithm of any (fully-secure) 2-level hierarchical identity-based encryption (HIBE) scheme [9].

In 2004, Boneh and Boyen described an efficient HIBE in the selective-ID security model [4]. In 2005, Waters described how to alter this scheme to make it fully-secure [46]. The IBS scheme that can be extracted from Waters 2-HIBE was proven secure under CDH in the standard model by Boyen and Waters [9]. In the conference version of this paper [10], we presented a batch verifier for this IBS scheme. 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 can 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 SHA256 is used to hash all the messages ($k_2 = 256$), then when $n \geq 31$ batching becomes faster than individual verification and at most 289 dominant operations will have to be performed regardless of n.

Fortunately, we are able to significantly improve the efficiency of these prior results. We begin by recalling that in 2005 Naccache showed how to generalize the Waters IBE to optimize it for efficiency [40]. These ideas were extended in 2006 by Chatterjee and Sarkar to Waters HIBE and the resulting HIBE was proven secure under DBDH in the standard model [16]. We call the IBS scheme implicitly defined by this generalized HIBE as Π-IBS. It is known to be secure under DBDH [16] and we conjecture that its security can be shown under CDH.

The Π-IBS scheme and its batch verification algorithm are both considerably more practical than the non-generalized version presented in our conference paper [10]. Indeed, the structure imposed by the generalization [40, 16] make the Π-IBS scheme particularly well-suited for batch verification. We now explicitly describe the Π-IBS and then show how to batch verify these signatures.

We assume that the identities and messages are both bit strings of length k represented by z blocks of k/z bits each. (If this is not the case, then let k be the larger bit length and then pre-pad the shorter string with zeros.) Let $\mathsf{BSetup}(1^\ell) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$.

Setup: First choose a secret $\alpha \in \mathbb{Z}_q$ and $h \in \mathbb{G}$ and calculate $A = \mathbf{e}(g,h)^\alpha$. Then pick two random integers $y'_1, y'_2 \in \mathbb{Z}_q$ and a random vector $y = (y_1, \ldots, y_z) \in \mathbb{Z}_q^z$. The master secret key is $MK = h^\alpha$ and the public parameters are given as: $PP = g, A, u'_1 = g^{y'_1}, u'_2 = g^{y'_2}, u_1 = g^{y_1}, \ldots, u_k = g^{y_z}$.

We use the notation of Chatterjee and Sarkar [16] to define the following function. Let

$v = (v_1, \ldots, v_z)$, where each v_i is a (k/z)-bit string. For $i \in \{1, 2\}$, let:

$$U_i(v) = u_i' \prod_{j=1}^{z} u_j^{v_j}.$$

Extract: To create a private key for a user with identity $ID = \kappa_1, \ldots, \kappa_z$, select $r \in \mathbb{Z}_q$ and return $K_{ID} = (g^\alpha \cdot U_1(ID)^r,\ g^{-r})$.

Sign: To sign a message $m = m_1, \ldots, m_z$ using private key $K = (K_1, K_2)$, select $s \in \mathbb{Z}_q$ and return

$$S = \left(K_1 \cdot U_2(m)^s,\ K_2,\ g^{-s}\right).$$

Verify: To verify a signature $S = (S_1, S_2, S_3)$ from identity $ID = \kappa_1, \ldots, \kappa_z$ on message $m = m_1, \ldots, m_z$, check that:

$$A = \mathbf{e}(S_1, g) \cdot \mathbf{e}(S_2, U_1(ID)) \cdot \mathbf{e}(S_3, U_2(m)).$$

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 [3] and to take advantage of the peculiarities of bilinear maps. Let *KeyGen*, *Sign* and *Verify* be as before.

Batch Verify: Suppose we want to batch verify n purported signatures. Let κ_j^i and m_j^i denote the j'th (k/z)-bit block of the identity of the i'th signer and the message signed by the i'th signer, respectively. 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

$$P = \mathbf{e}(\prod_{i=1}^{n} {S_1^i}^{\delta_i}, g) \cdot \mathbf{e}(\prod_{i=1}^{n} {S_2^i}^{\delta_i}, u_1') \cdot \mathbf{e}(\prod_{i=1}^{n} {S_3^i}^{\delta_i}, u_2').$$

Depending on the values of z and n (c.f. below), pick and check one of the following equations:

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

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

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

Let us discuss which equation should be picked. If $n < 2z$, use equation 1; otherwise, use equation 2.

Theorem 4.1 *The above algorithm is a batch verifier for the* Π*-IBS.*

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

$$\begin{aligned}
\prod_{i=1}^{n} A^{\delta_i} &= \prod_{i=1}^{n} \left(\mathbf{e}(S_1^i, g) \cdot \mathbf{e}(S_2^i, U_1(ID_{t_i})) \cdot \mathbf{e}(S_3^i, U_1(M_i))\right)^{\delta_i} && (3)\\
&= \mathbf{e}(\prod_{i=1}^{n} {S_1^i}^{\delta_i}, g) \cdot \prod_{i=1}^{n} \mathbf{e}({S_2^i}^{\delta_i}, u_1' \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^{n} \mathbf{e}({S_3^i}^{\delta_i}, u_2' \prod_{j=1}^{z} u_j^{m_j^i}) \\
&= P \cdot \prod_{i=1}^{n} \left(\mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}) \right) && (4)
\end{aligned}$$

For the first part of the proof, all we need now is to show that equation 1 is equivalent to equation 2. Since for all i, $\mathsf{Verify}(ID_{t_i}, M_i, S_i) = 1$, (S_1^i, S_2^i, S_3^i) are valid signatures and hence we can write $S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some elements $b_i, c_i \in \mathbb{Z}_q$. Now we rewrite the part inside the parenthesis of equation 1 and get equation 2:

$$\begin{aligned}
\prod_{i=1}^{n} \mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^{n} \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}) &= \prod_{i=1}^{n} \left(\mathbf{e}(g^{b_i}, g^{\sum_{j=1}^{z} \kappa_j^i y_j}) \cdot \mathbf{e}(g^{c_i}, g^{\sum_{j=1}^{z} m_j^i y_j}) \right)^{\delta_i} \\
&= \prod_{i=1}^{n} \left(\mathbf{e}(g, g)^{\sum_{j=1}^{z} (\delta_i b_i \kappa_j^i y_j + \delta_i c_i m_j^i y_j)} \right) \\
&= \prod_{j=1}^{z} \left(\mathbf{e}(g, g)^{y_j \sum_{i=1}^{n} (\delta_i b_i \kappa_j^i + \delta_i c_i m_j^i)} \right) \\
&= \prod_{j=1}^{z} \mathbf{e}(\prod_{i=1}^{n} ({S_2^i}^{\kappa_j^i} \cdot {S_3^i}^{m_j^i})^{\delta_i}, u_j).
\end{aligned}$$

We must now show the other direction. This proof is an application of the technique for proving the small exponents test in [3]. Batch verification accepts so we know that $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ and hence we can write $S_1^i = g^{a_i}, S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some $a_i, b_i, c_i \in \mathbb{Z}_q$. Since equation 3 is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as:

$$\begin{aligned}
\prod_{i=1}^{n} A^{\delta_i} &= \prod_{i=1}^{n} \left(\mathbf{e}(g^a, g) \cdot \mathbf{e}(g^b, g^{y_1'} g^{\sum_{j=1}^{z} y_j \kappa_j}) \cdot \mathbf{e}(g^c, g^{y_2'} g^{\sum_{j=1}^{z} y_j m_j}) \right)^{\delta_i} \\
&= \prod_{i=1}^{n} \mathbf{e}(g, g)^{\delta_i \left(a + b y_1' + c y_2' + b \sum_{j=1}^{z} y_j \kappa_j + c \sum_{j=1}^{z} y_j m_j\right)} \\
&= \mathbf{e}(g, g)^{\sum_{i=1}^{n} \delta_i \left(a_i + b_i y_1' + c_i y_2' + b_i \sum_{j=1}^{z} y_j \kappa_j^i + c_i \sum_{j=1}^{z} y_j m_j^i\right)} \\
&\Rightarrow \mathbf{e}(g, g)^{\sum_{i=1}^{n} \delta_i \alpha} && (5)
\end{aligned}$$

Setting $\beta_i = \alpha - \left(a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^{z} y_j \kappa_j^i + c_i \sum_{j=1}^{z} y_j m_j^i\right)$ and rewriting equation 5 we get:

$$\begin{aligned}
&\mathbf{e}(g,g)^{\sum_{i=1}^{n}\delta_i\alpha - \sum_{i=1}^{n}\delta_i\left(a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^{z} y_j\kappa_j^i + c_i\sum_{j=1}^{z} y_j m_j^i\right)} = 1 \\
&\Rightarrow \sum_{i=1}^{n}\delta_i\alpha - \sum_{i=1}^{n}\delta_i\left(a_i + b_i y'_1 + c_i y'_2 + b_i\sum_{j=1}^{z} y_j\kappa_j^i + c_i\sum_{j=1}^{z} y_j m_j^i\right) \equiv 0 \pmod{q} \\
&\Rightarrow \sum_{i=1}^{n}\delta_i\beta_i \equiv 0 \pmod{q} \qquad (6)
\end{aligned}$$

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 6 gives us:

$$\delta_1 \equiv -\gamma_1 \sum_{i=2}^{n} \delta_i\beta_i \pmod{q} \qquad (7)$$

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 7 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: $\Pr[E] \leq \sum_{i=1}^{2^{\ell_b(n-1)}}\left(2^{-\ell_b} \cdot 2^{-\ell_b(n-1)}\right) = 2^{-\ell_b}$. □

Efficiency Note. The signature for Π-IBS consists of three group elements, but since it is identity-based there are no public key, and we assume that the identity is given "for free" e.g. it could be the hardware address of the network interface card. Hence the size of the signature that verifies both the message and the identity depends only on the size of these group elements. We have described the scheme in the symmetric bilinear setting $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ because the original scheme does not work in the asymmetric bilinear setting $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. However, it can be verified that the scheme (and hence also our batch verifier) also work in the asymmetric bilinear setting by switching the order of the elements in the second and the third pairing.

In the symmetric bilinear setting elements must be around 512 bits for security comparable to 1024 bits RSA, which gives us a total signature size of 1536 bits. In the asymmetric bilinear setting the element S_1 can be represented using 160 bits, whereas S_2 and S_3 need 1024 bits. However Koblitz and Menezes [32] note that if the embedding degree k is even and $k \geq 2$ one only needs to represent the x coordinate as the y coordinate can be computed from it. Hence we only need 512 bits for S_2 and S_3 at the cost of two additional multiplications per signature we need to verify. So all in all we can represent the signature on the message and the identity using only 1184 bits. However, it might not be efficient to test membership of the group $\mathbb{G}_2$, which is needed for batch verification.

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 [7] for the certificates and a modified version of the CL signature scheme [11] 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-$\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, using the Section 1.2 notation.

Signatures: We describe a new signature scheme Π-Sig with a batch verifier in Section 5.2. The scheme is secure under the LRSW assumption in the plain model when the size of the message space is a polynomial and in the random oracle model when the size of 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 Π-Sig signatures, costs n-$\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) +$ n-$\mathsf{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, 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 [7] described in Section 2, using the small exponents test [3].

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

Proof. First we show that $\mathsf{Verify}(ID_{t_1}, M_1, S_1) = \cdots = \mathsf{Verify}(ID_{t_n}, M_n, S_n) = 1$ implies that $\mathsf{Batch}((ID_{t_1}, M_1, S_1), \dots, (ID_{t_n}, M_n, S_n)) = 1$. This follows from the verification equation for the BLS scheme:

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} \Leftrightarrow \mathbf{e}(\prod_{i=1}^{n} \sigma_i^{\delta_i}, g) = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} \tag{8}$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [3]. Batch verification accepts so we know that $\sigma_i \in \mathbb{G}$ and hence we can write $\sigma_i = g^{c_i}$ for some $c_i \in \mathbb{Z}_q$. We also know that $h_i \in \mathbb{G}$ so we write it as $h_i = g^{r_i}$. Recall that $pk_i = g^{x_i}$. We know that equation 8 holds, so we can rewrite it as:

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(g,g)^{\delta_i r_i x_i}$$
$$\Rightarrow \mathbf{e}(g,g)^{\sum_{i=1}^{n} \delta_i c_i} = \mathbf{e}(g,g)^{\sum_{i=1}^{n} \delta_i r_i x_i} \qquad (9)$$

Setting $\beta_i = c_i - r_i x_i$ and rewriting equation 9 we get:

$$\mathbf{e}(g,g)^{\sum_{i=1}^{n} \delta_i c_i - \delta_i r_i x_i} = 1$$
$$\Rightarrow \sum_{i=1}^{n} \delta_i c_i - \delta_i r_i x_i \equiv 0 \pmod q \Rightarrow \sum_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod q$$

The rest of the proof follows from the last part of the proof of Theorem 4.1. □

Single Singer for BLS. However, BLS [7] 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 5.2 ([7]) *The algorithm above is a single-signer, batch verifier for BLS signatures.*

5.2 A New Signature Scheme Π-Sig

In this section we introduce a new signature scheme secure under the LRSW assumption [38], which is based on the Camenisch and Lysyanskaya signatures [11].

The Original CL Scheme. Recall the Camenisch and Lysyanskaya signature scheme [11]. Let $\mathsf{BSetup}(1^\ell) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Choose the secret key $sk = (x, y) \in \mathbb{Z}_q^2$ 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.

Π-Sig: 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 5.3 *Under the LRSW assumption in $\mathbb{G}$, the Π-Sig 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 Π-Sig 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:

$$H_1(\phi_i) = \begin{cases} g^{r_i} & \text{if } \phi_i \neq \phi' \\ a' & \text{otherwise} \end{cases} \tag{10}$$

   and

$$H_2(\phi_i) = \begin{cases} g^{s_i} & \text{if } \phi_i \neq \phi' \\ b' & \text{otherwise} \end{cases} \tag{11}$$

   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:

$$H_3(m_l||\phi_i) = \begin{cases} t_{l,i} & \text{if } \phi_i \neq \phi' \text{ or } l \neq j \\ w' & \text{otherwise} \end{cases} \tag{12}$$

$\mathcal{B}$ records $m^* := m_j$. Finally, set the signing query oracle such that on the lth query involving period ϕ_i:

$$\mathcal{O}_{sk^*}(m_l||\phi_i) = \begin{cases} \text{abort} & \text{if } \phi_i = \phi' \text{ and } l \neq j \\ c' & \text{else if } \phi_i = \phi' \text{ and } l = j \\ X^{r_i} X^{(s_i) t_{l,i}} & \text{otherwise} \end{cases} \tag{13}$$

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 Π-Sig 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 $\sigma = H_1(\phi_i)^x H_2(\phi_i)^{x \cdot H_3(m_l||\phi_i)}$. In this scenario $\phi_i = \phi'$ and $t_{l,i} \neq w'$. We can substitute as $\sigma = (a')^x (b')^{x \cdot (t_{l,i})}$. 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 Π-Sig 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 Π-Sig 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 $\varepsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$. □

On Removing the Random Oracles. In the previous proof, notice that we treated hash functions H_1, H_2 and H_3 as independent random oracles which were (statically) programmed in $|\Phi|$, $|\Phi|$, and $|\Phi| \cdot |\mathcal{M}|$ points, respectively, where Φ is the set of time period identifiers and $\mathcal{M}$ is the signing message space. Recall that, as before, $|\Phi|$ is restricted to be polynomial in the security parameter. Now, for sufficiently short message spaces, e.g., ISO defined error messages, we can replace all three random oracles in the security proof of Π-Sig by concrete hash functions. Suppose that given a set of pairs $(x_1, y_1), \ldots, (x_k, y_k)$, it is possible to efficiently sample a function $H : \{0,1\}^\ell \to \mathbb{G}$ (where $k < 2\ell + 1$) from a $(2\ell + 1)$-independent function family $\mathcal{H}$ such that for each $H \in \mathcal{H}$, we have $H(x_i) = y_i$ for $i = 1$ to k. If such types of hash function families exist then we could simple constrain them exactly as we programmed our random oracles.

Fortunately, Canetti, Halevi, and Katz [12] describe a method of efficiently constructing such a hash function family which allows to map strings to bilinear map elements (or to map strings to elements in another prime-order algebraic group such as $\mathbb{Z}_q$). Boneh and Boyen describe another such family [4]. Any family satisfying the constraints above will work for our purposes, where H_1 and H_2 map into bilinear group $\mathbb{G}$ and H_3 maps into $\mathbb{Z}_q$. The construction remains as before and the new security proof simply uses concrete functions with constraints mirroring the points (statically) programmed in the oracles.

Lemma 5.4 *Under the LRSW assumption in $\mathbb{G}$, the Π-Sig signature scheme is existentially unforgeable in the plain model when $|\mathcal{M}|$ are polynomial in the security parameter.*

Batch Verification of Π-Sig 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 5.5 *The algorithm above is a batch verifier for* Π*-Sig signatures.*

Proof. First we show that $\mathsf{Verify}(ID_{t_1}, M_1, S_1) = \cdots = \mathsf{Verify}(ID_{t_n}, M_n, S_n) = 1$ implies that $\mathsf{Batch}((ID_{t_1}, M_1, S_1), \ldots, (ID_{t_n}, M_n, S_n)) = 1$. This follows from the verification equation for the Π-Sig scheme if we keep in mind that

$$\prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^n (\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i})^{\delta_i} = \prod_{i=1}^n \mathbf{e}(a, X_i)^{\delta_i} \cdot \prod_{i=1}^n \mathbf{e}(b, X_i)^{w_i \delta_i} \tag{14}$$
$$\Leftrightarrow \mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(a, \prod_{i=1}^n X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^n X_i^{w_i \delta_i})$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [3]. Batch verification accepts so we know that $\sigma_i \in \mathbb{G}$ and hence we can write $\sigma_i = g^{c_i}$ for some $c_i \in \mathbb{Z}_q$. We also know that a and b are in $\mathbb{G}$ so we write them as $a = g^r$ and $b = g^s$. Since equation 14 is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as:

$$\prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^n (\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i})^{\delta_i} = \prod_{i=1}^n \mathbf{e}(g, g)^{\delta_i (r x_i + s x_i w_i)}$$
$$\Rightarrow \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i c_i} = \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i (r x_i + s x_i w_i)} \tag{15}$$

Setting $\beta_i = c_i - (r x_i + s x_i w_i)$ and rewriting equation 15 we get:

$$\mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i c_i - \delta_i (r x_i + s x_i w_i)} = 1$$
$$\Rightarrow \sum_{i=1}^n \delta_i c_i - \delta_i (r x_i + s x_i w_i) \equiv 0 \pmod{q} \Rightarrow \sum_{i=1}^n \delta_i \beta_i \equiv 0 \pmod{q}$$

The rest of the proof follows from the last part of the proof of Theorem 4.1. □

Π-Sig Without Batch Verification. So far we have described Π-Sig only as an efficient signature scheme to batch verify, but for completeness we note that if we are not interested in batch verification, Π-Sig is still a fairly efficient regular signature scheme without any restrictions.

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

Sign: Generate a value $\phi \in \Phi$ that has never been used by the signer before. Then on input message $m \in \mathcal{M}$, set $w = H_3(m || \phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$, $\sigma = a^x b^{xw}$ and output the signature (σ, ϕ).

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

This is very similar to the original scheme. Note that the only change is that ϕ is now generated independently from all other signers and included as part of the signature, which makes the scheme unsuitable for batch verification (since the probability that many signers will share the same value of ϕ is small). However, now that we are only interested in individual verification, we can rewrite the original verification equation $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$ as $\mathbf{e}(\sigma, g) = \mathbf{e}(ab^w, X)$ which requires only two pairings to verify. Finally note that this variant of the verification equation does not depend on how ϕ was generated, and can always be used for individual verification if needed.

Efficiency Note. First, we observe that the Π-Sig 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 Π-Sig 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 $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, 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 [32, 22]. However, as noted before, if the embedding degree k is even and $k \geq 2$ one only needs to represent the x coordinate and hence elements of $\mathbb{G}_2$ can be represented using 512 bits. For both BLS and Π-Sig this means that the public key will be around 512 bits. For Π-Sig signatures we need to hash into $\mathbb{G}_1$ which according to Galbraith, Paterson and Smart [22] can be done efficiently. To summarize; using BLS and Π-Sig we can represent the signature plus certificate using approximately 832 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 [12] to remove the random oracles from Π-Sig. 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.

If Π-Sig is used as a signatures scheme without an efficient batch verifier, the signature require one group element in $\mathbb{G}$ and one element in Φ where the size of Φ only needs to be large enough to represent the number of times a user might want to sign with the same private key. Verification of a single Π-Sig signature requires two pairings.

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 (Boneh, Lynn and Shacham [7] provide a batch verification scheme for multiple signers on the *same* message). We extended the general batch verification definition of Bellare, Garay and Rabin [3] 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 Π-IBS that can verify n signatures using only $z + 3$ pairings (the dominant operation), where identities are k bits divided into z elements, each of k/z 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 BLS certificates and n Π-Sig signatures using only 5 pairings. Here, Π-Sig 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. Another exciting open problem is to develop fast batch verifiers 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 project eu-DOMAIN, contract no. IST-004420. Susan Hohenberger and Michael Østergaard Pedersen performed part of this research while at IBM Research, Zürich Research Laboratory.

References

[1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.

[2] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 186–195. IEEE Computer Society, 2004.

[3] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[4] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[5] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[6] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[8] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2000.

[9] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2006.

[10] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT '07*, volume 4515 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[11] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[12] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[13] Tianjie Cao, Dongdai Lin, and Rui Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.

[14] Car 2 Car. Communication consortium. `http://car-to-car.org`.

[15] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In Yvo Desmedt, editor, *6th Public Key Cryptography (PKC)*, Lecture Notes in Computer Science, pages 18–30. Springer, 2003.

[16] Sanjit Chatterjee and Palash Sarkar. HIBE with short public parameters without random oracle. In Xuejia Lai, editor, *Advances in Cryptology – ASIACRYPT '06*, volume 4284 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2006.

[17] L. Chen, Z. Cheng, and N.P. Smart. Identity-based key agreement protocols from pairings, 2006. Cryptology ePrint Archive: Report 2006/199.

[18] Jung Hee Cheon, Yongdae Kim, and Hyo Jin Yoon. A new ID-based signature with batch verification, 2004. Cryptology ePrint Archive: Report 2004/131.

[19] Shi Cui, Pu Duan, and Choong Wah Chan. An efficient identity-based signature scheme with batch verifications. In Abdur Chowdhury, Francis Lau, and Frank Zhigang Wang, editors, *1st International Conference on Scalable Information Systems (InfoScale)*. ACM Press, 2006.

[20] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[21] Amos Fiat. Batch RSA. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 1989.

[22] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.

[23] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, editor, *9th Public Key Cryptography (PKC)*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.

[24] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.

[25] R. Granger and N.P. Smart. On computing products of pairings, 2006. Cryptology ePrint Archive: Report 2006/172.

[26] Lein Harn. Batch verifying multiple DSA digital signatures. *Electronics Letters*, 34(9):870–871, 1998.

[27] Lein Harn. Batch verifying multiple RSA digital signatures. *Electronics Letters*, 34(12):1219–1220, 1998.

[28] Fumitaka Hoshino, Masayuki Abe, and Tetsutaro Kobayashi. Lenient/strict batch verification in several groups. In George I. Davida and Yair Frankel, editors, *4th Information Security*, Lecture Notes in Computer Science, pages 81–94. Springer, 2001.

[29] Min-Shiang Hwang, Cheng-Chi Lee, and Yuan-Liang Tang. Two simple batch verifying multiple digital signatures. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *3rd Information and Communications Security (ICICS)*, Lecture Notes in Computer Science, pages 233–237. Springer, 2001.

[30] Min-Shiang Hwang, Iuon-Chang Lin, and Kuo-Feng Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences*, 11(1):15–19, 2000.

[31] IEEE. 5.9 GHz Dedicated Short Range Communications. `http://grouper.ieee.org/groups/scc32/dsrc`.

[32] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels, 2005. Cryptology ePrint Archive: Report 2005/076.

[33] Chi-Sung Laih and Sung-Ming Yen. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.

[34] Olaf Landsiedel, Klaus Wehrle, and Stefan Götz. Accurate prediction of power consumption in sensor networks. In *IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, 2005.

[35] Seungwon Lee, Seongje Cho, Jongmoo Choi, and Yookun Cho. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(1):74–80, 2006.

[36] C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), pages 1592–1593, 1994.

[37] Chae Hoon Lim. Efficient multi-exponentation and application to batch verification of digital signatures, 2000. `http://dasan.sejong.ac.kr/~chlim/english_pub.html`.

[38] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Carlisle Adams and Howard Heys, editors, *6th Selected Areas in Cryptography (SAC)*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[39] 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, 1991.

[40] D. Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.

[41] David Naccache, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli. Can dsa be improved? complexity trade-offs with the digital signature standard. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 77–85. Springer, 1994.

[42] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15:39–68, 2007.

[43] Michael Scott. Scaling security in pairing-based protocols, 2005. Cryptology ePrint Archive: Report 2005/139.

[44] SeVeCom. Security on the road. `http://www.sevecom.org`.

[45] Martin Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.

[46] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 320–329. Springer, 2005.

[47] HyoJin Yoon, Jung Hee Cheon, and Yongdae Kim. Batch verifications with ID-based signatures. In Choonsik Park and Seongtaek Chee, editors, *7th Information Security and Cryptology (ICISC)*, Lecture Notes in Computer Science, pages 233–248. Springer, 2004.

[48] Fangguo Zhang and Kwangjo Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *8th Information Security and Privacy, Australasian Conference (ACISP)*, volume 2727 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2003.

[49] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology – INDOCRYPT '03*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2003.