J. Cryptol. (2012) 25: 723–747
DOI: 10.1007/s00145-011-9108-z

Journal of
CRYPTOLOGY

Batch Verification of Short Signatures

Jan Camenisch
IBM Research, Zürich, Switzerland
jca@zurich.ibm.com

Susan Hohenberger
Johns Hopkins University, Baltimore, USA
susan@cs.jhu.edu

Michael Østergaard Pedersen
Miracle A/S, Jaegergaardsgade 122-1, 8000 Aarhus C, Denmark
mop@miracleas.dk

Communicated by Dan Boneh.

Received 1 August 2007
Online publication 20 October 2011

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, a frequent requirement is that the communication overhead inflicted be small and that many messages be processable at the same time. In this paper, we consider the suitability of public key signatures in the latter scenario. That is, we consider (1) signatures that are short and (2) cases 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.

Key words. Digital signatures, Batch verification, Bilinear groups

1. Introduction

As the world moves toward 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 [16,50]. Raya and Hubaux suggest that vehicles will transmit safety messages every 300 ms to all other vehicles within a minimum range of 110 meters [49], which in turn may retransmit these messages.

For such pervasive systems to work properly, there are many competing constraints [16,37,49,50]. For one, 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 [37]. 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. Furthermore, different messages from many different signers may need to be verified and processed quickly (e.g., every 300 ms [49]). Another possible 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 300 ms is not a problem for current systems, but transmitting and/or verifying 100+ messages per second might pose a problem. For example, using Rivest, Shamir, and Adleman (RSA) signatures 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 Sect. 8.2 of [49]). While many new schemes based on bilinear maps can provide the same security with significantly smaller signatures, they take significantly more time to verify (e.g., [6,9,13,53]). 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 [40]. Barr and Asanović note that the wireless transmission of just a single bit can use more than 1000 times the energy required for a 32 bit computation [3].

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 or try to reduce the signature size of a fast signature scheme such as RSA. In this paper we take the first route and investigate the known batch verification techniques and to what extent they are applicable to bilinear map-based schemes, whereas, for example, Gentry provides a method for compressing Rabin signatures in [28]. We note that while these two techniques are not mutually exclusive (in fact, Gentry mentions that the compressed Rabin signatures can be aggregated [28]), compressing signatures has not been the focus of our work. More precisely, the main contributions of this paper are as follows.

1. We instantiate the general batch verification definitions of Bellare, Garay, and Rabin [4] 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 [19]. (More precisely, this is the identity-based signature (IBS) scheme implicitly defined by the Chatterjee–Sarkar hierarchical identity-based encryption (IBE) [19]; it can also be viewed as a generalized version of the Boyen–Waters IBS [11], 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 scheme. 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–Lysyanskaya signature scheme [13], 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 [9]—but users may only issue one signature per period (e.g., users might only be allowed to sign one message per 300 ms). 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 the Π-Sig signature scheme, we can issue signatures with Π-Sig and certificates with the Boneh, Lynn and Shacham scheme [9]. 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*

We start with some historical remarks and then later present the schemes relevant to this paper in more detail.

Batch cryptography was introduced in 1989 by Fiat [26] for a variant of RSA. Later, in 1994, Naccache, M'Raïhi, Vaudenay, and Raphaeli [48] gave the first efficient batch verifier for Digital Signature Algorithm (DSA) signatures; however, an interactive batch verifier presented in an early version of their paper was broken by Lim and Lee [43]. In 1995 Laih and Yen proposed a new method for batch verification of DSA and RSA signatures [39], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [10]. In 1998, Harn presented two batch verification techniques for DSA and RSA [32,33], but both were later broken [10,35,36]. The same year, Bellare, Garay, and Rabin took the first systematic look at batch verification [4] 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 [39,48]. They showed how to apply these methods to batch verification of DSA signatures and also introduced a weaker form of batch verification called *screening*. Later, Cheon and Lee introduced two new methods called the *sparse exponents test* and the *complex exponents test* [22], which they claim to be about twice as fast as the small exponents test. In 2000, Boyd and Pavlovski published some attacks against different batch verification schemes, mostly ones based on the small exponents test and related tests [10]. 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 incorrectly. However, the authors also describe methods to repair some broken schemes based on this test. In 2001, Hoshino, Masayuki, and Kobayashi [34] pointed out that the problem discovered by Boyd and Pavlovski [10] might not be critical for batch verification of signatures, but only when using batch verification to verify, for example, zero-knowledge proofs. In 2004 Yoon, Cheon, and Kim proposed a new ID-based signature scheme with batch verification [21], but their security proof is for aggregate signatures and does not meet the definition of batch verification by Bellare et al. [4]; hence their title is somewhat misleading. Other schemes for batch verification based on bilinear maps were proposed [17,54–56], but all were later broken by Cao, Lin, and Xue [15]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [42], but Stanek [52] showed that this method is flawed. Shacham and Boneh gave a practical application of batch verification by using a modified version of Fiat's batch verifier for RSA to improve the efficiency of Secure Sockets Layer (SSL) handshakes on a busy server [51]. Ferrara, Green, Hohenberger, and Pedersen gave performance measurements for the schemes herein, and also showed how to batch verify other types of signatures, such as group and ring signatures [25]. Law and Matt pointed out some IBS schemes that batch well, and also gave methods for identifying invalid signatures in a batch [41].

Bellare, Garay, and Rabin Testing Techniques Let g generate a group of prime order. In 1998, Bellare, Garay and Rabin described some tests [4] for verifying equations of the form $y_i = g^{x_i}$ for $i = 1$ to n. 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 their product 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 initial proposal.

Random Subset Test: The idea here 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 choose exponents δ_i of (a small number of) ℓ bits and compute $\prod_{i=1}^n y_i^{\delta_i} = g^{\sum_{i=1}^n x_i \delta_i}$. Bellare et al. 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; 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: This method 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 Boneh–Lynn–Shacham (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 [44] provides a number of efficient methods for doing m-term exponentiations, and Granger and Smart [31] give improvements over the naive method for computing a product of pairings, which is why we state them explicitly.

$m\text{-}\mathsf{MultPairCost}^s_{\mathbb{G},\mathbb{H}}$	s m-term pairings $\prod_{i=1}^m \mathbf{e}(g_i, h_i)$ where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$m\text{-}\mathsf{MultExpCost}^s_{\mathbb{G}}(k)$	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 $i = 1 \dots s$, where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$\mathsf{ExpCost}^s_{\mathbb{G}}(k)$	s exponentiations g^{a_i} for $i = 1 \dots s$ 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 [10]. It is different from normal RSA in 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 applies 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 [10]. 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. [4] present 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 [48] compatible with the *small exponents test* from [10]. There are two differences from 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 [19] for which we provide a batch verifier without random oracles in Sect. 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.

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

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 [9]. 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 [8] can be used. The signature is only one group element in a bilinear group and the same applies 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 a 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 the Camenisch and Lysyanskaya signatures [13] which we present in Sect. 5; it is designed specifically to enable efficient batch verification. The signature is only one bilinear group element; the same is true 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 Sect. 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 $\mathsf{Gen}(1^\ell) \to (pk, sk)$, the algorithm $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

There are two common notions of security. Goldwasser, Micali, and Rivest [30] defined a scheme to be *unforgeable* as follows: let $\mathsf{Gen}(1^\ell) \to (pk, sk)$. Suppose (m, σ) is output by a probabilistic polynomial time (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 Sect. 4 meet the GMR [30] definition, while those in Sect. 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 [4] 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, $k, n \in \text{poly}(\ell)$, and $(pk_1, sk_1), \ldots, (pk_k, sk_k)$ are generated independently according to $\mathsf{Gen}(1^\ell)$. Let $PK = \{pk_1, \ldots, pk_k\}$. Then we call probabilistic Batch a batch verification algorithm when the following conditions hold:

- If $pk_{t_i} \in PK$ and $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ 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 $pk_{t_i} \in PK$ for all $i \in [1, n]$ and $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 0$ for some $i \in [1, n]$, then $\mathsf{Batch}((pk_{t_1}, m_1, \sigma_1), \ldots, (pk_{t_n}, m_n, \sigma_n)) = 0$ except with probability negligible in ℓ, 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].

On Differences Between Batch Verification, Screening, and Aggregate Signatures As we discussed in the introduction, when doing our literature search on batch verification, we often came across works (e.g., [21,23]) which confuse the goals of aggregate signatures and batch verification or claim to do batch verification when, in fact, they often meet a weaker guarantee called *screening* [4]. Let us clarify these distinct notions.

Informally, in both batch verification and screening, the goal is an algorithm that takes as input n distinct signatures and verifies them quickly. In batch verification, the batch of signatures should verify if and only if all individual signatures are valid. In the screening security model, an honest signer is protected in the sense that an attacker cannot cause her to become bound to a message that she did not sign, even if the attacker controls all other signers; however, honest verifiers are not totally protected from dishonest signers in the sense that a dishonest signer might be able to devise a batch of signatures that pass the screening test, but do not all individually verify.

The goal in aggregate signatures is an algorithm that takes as input n distinct signatures and compresses them to save bandwidth. It happens that the security definition of aggregate signatures [8] implies screening, while neither definition implies batch verification. We first give the formal definitions of screening and aggregate signatures, and then discuss a scheme which satisfies these notions, but *not* batch verification.

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:

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

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

The above definition is generalized to the multiple-signer case from the single-signer screening definition of Bellare, Garay, and Rabin [4]. We now describe the security notion for aggregate signatures; the correctness property should be obvious.

Definition 2.3 (Aggregate Signatures Security [8]). 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 $\mathsf{AggVerify}$ an *aggregate verification algorithm* when $\mu(\ell)$ defined as follows is a negligible function:

$$\begin{aligned}&\Pr\big[(pk_0, sk_0) \leftarrow \mathsf{Gen}\big(1^\ell\big); (pk_1, \ldots, pk_n, m_0, \ldots, m_n, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot)}(pk_0) :\\ &\quad \mathsf{AggVerify}\big((pk_0, \ldots, pk_n), (m_0, \ldots, m_n), \sigma\big) = 1 \ \wedge\ m_0 \notin Q\big] = \mu(\ell),\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{sk_0}(\cdot)$.

As mentioned above, screening is the (maximum) guarantee that some 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. We now give an example of a construction which can satisfy both Definitions 2.2 and 2.3, but which provably does not satisfy Definition 2.1. Consider the aggregate signature scheme of Boneh, Gentry, Lynn, and Shacham [8] based on the BLS signatures [9]. First, we review the BLS signatures. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q that provided for a bilinear map $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. 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 a signature σ on a 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) where each message is distinct, the BGLS scheme 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 observe [8]), 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 Sect. 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.

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 e-mail scenario, where the mailserver is verifying the signatures on e-mails 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 of the security parameter 1^ℓ, outputs the parameters for a bilinear map as $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order $q \in \Theta(2^\ell)$. The efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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 often also denoted as 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* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct groups, possibly without efficient isomorphisms between them. Describing the details about how these bilinear maps are constructed is not the purpose of this paper, so we just give a 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 usually a subgroup of $E(\mathbb{F}^*_{p^k})$.

Let's look at the size of the group elements. For simplicity we will assume that we are aiming for security comparable to 1024 bit RSA. Note that although a point (x, y) on a curve consists of two elements x and y in the underlying field, the size of a group's elements is equivalent to the size of elements in the underlying field. The reason is that we only need to represent the x coordinate and the least significant bit of y in order to reconstruct y when needed.

So what is the minimum size the group elements can have? First of all, the group order q must be large enough to resist attacks on discrete logarithms, such as the Pollard-ρ attack, which means that $q \geq 160$. Second, the MOV attack states that solving the discrete logarithm problem on a curve reduces to solving it over the corresponding finite field [46], which means that the bit length of p^k must be around 1024, which has implications for the size of $\mathbb{G}_T$. The best known curves in the symmetric setting work with $|p| = 512$ and $k = 2$, and hence elements of $\mathbb{G}$ will be 512 bits, while elements of $\mathbb{G}_T$ will be 1024 bits. In the asymmetric setting one can choose $|p| = 160$ and $k = 6$, which results in elements of size 160 bits in $\mathbb{G}_1$, while elements of $\mathbb{G}_2$ and $\mathbb{G}_T$ will be 960 bits. In some cases elements of $\mathbb{G}_2$ can be represented in the subfield $E(\mathbb{F}_{p^{k/2}})$ instead, resulting in elements of 480 bits [38].

Our constructions from Sect. 5 also work in the asymmetric setting, which allows us to use a short representation of the signatures. The Π-IBS scheme from Sect. 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 Sects. 4 and 5 for a more detailed discussion.

Testing Membership in $\mathbb{G}$ In a *non-bilinear* setting, Boyd and Pavlovski [10] 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 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 [10] 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. For example, 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 [10] suggest methods for overcoming this problem through a 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. Assume 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 for which we are interested in testing membership.

If the order of $\mathbb{G}_1$ is q, one option is to verify that an element y is in $\mathbb{G}_1$ by checking that $y^q = 1$. While this might seem inefficient, it is actually not a problem in practice when working with pairing-based schemes, since the time required for a single exponentiation is considerably less than the time required for computing a pairing. This has been verified experimentally by Ferrara et al. [25]. However, one area for improvement in batching is to devise more efficient methods for membership testing in bilinear groups. Chen, Cheng, and Smart [20] provide more details on this.

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

Assumption 3.1 (Computational Diffie–Hellman [24]). *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 negligible in ℓ:*

$$\Pr\bigl[a, b, \leftarrow \mathbb{Z}_q; z \leftarrow \mathcal{A}\bigl(g, g^a, g^b\bigr) : z = g^{ab}\bigr].$$

Assumption 3.2 (Decisional Bilinear Diffie–Hellman [7]). *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:*

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

Assumption 3.3 (LRSW [45]). *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 of 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:*

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

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

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 be executed 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 $\mathrm{MIN}\{(2n + 3), (z + 3)\}$.

On the practical side, we note that as z grows there is a corresponding degradation in the concrete security of the IBS scheme (see [19] 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 [19], 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) two-level hierarchical identity-based encryption (2-HIBE) scheme [11].

In 2004, Boneh and Boyen described an efficient HIBE in the selective-ID security model [5]. In 2005, Waters described how to alter this scheme to make it fully secure [53]. The IBS scheme that can be extracted from Waters's 2-HIBE was proven secure under computational Diffie–Hellman (CDH) in the standard model by Boyen and Waters [11]. In the conference version of this paper [12], 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 from the conference version can verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly $\mathrm{MIN}\{(2n+3), (k_1+n+3), (n+k_2+3), (k_1+k_2+3)\}$. 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 [47] and Chatterjee and Sarkar [18] independently showed how to generalize the Waters IBE to optimize it for efficiency. In 2006 Chatterjee and Sarkar extended these ideas to Waters HIBE and the resulting HIBE was proven secure under decisional bilinear Diffie–Hellman (DBDH) in the standard model [19]. We call the IBS scheme implicitly defined by this generalized HIBE "Π-IBS." It is known to be secure under DBDH [19], 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 [12]. Indeed, the structure imposed by the generalization [19,47] makes 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 $y = (y_1, \dots, y_z) \in \mathbb{Z}_q^z$. The master secret key is $MK = h^\alpha$ and the public parameters are given as: $PP = (g, A, u'_1 = g^{y'_1}, u'_2 = g^{y'_2}, u_1 = g^{y_1}, \dots, u_z = g^{y_z})$.

We use the notation of Chatterjee and Sarkar [19] to define the following function. Let $v = (v_1, \dots, v_z)$, 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 $K_{ID} = (h^\alpha \cdot U_1(ID)^r,\ g^{-r})$.

Sign: To sign a message $m = (m_1, \ldots, m_z)$, where each m_i is a (k/z)-bit string, using private key $K = (K_1, K_2)$, select $s \in \mathbb{Z}_q$ and return

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

Verify: To verify a signature $S = (S_1, S_2, S_3)$ from identity $ID = \kappa_1, \ldots, \kappa_z$ on message m, parse $m = (m_1, \ldots, m_z)$, where each m_i is a (k/z)-bit string, and check that

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

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 [4] and to take advantage of the peculiarities of bilinear maps.

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 that all the identities have the correct length and that $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ 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

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

Depending on the values of z and n proceed as follows: if $n < 2z$ check whether

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

holds; otherwise verify the equation

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

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

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

Proof. Let $ID_i = (\kappa_1^i, \ldots, \kappa_z^i)$. The requirement that all public keys are valid is trivially satisfied for an identity-based scheme, once it has been verified that all identities have the correct length. First we show that $\mathsf{Verify}(ID_1, M_1, S_1) = \cdots =$

$\mathsf{Verify}(ID_n, M_n, S_n) = 1$ implies that $\mathsf{Batch}((ID_1, M_1, S_1), \ldots, (ID_n, M_n, S_n)) = 1$. This follows from the verification equation for the Π-IBS scheme:

$$\prod_{i=1}^{n} A^{\delta_i} = \prod_{i=1}^{n} \big(\mathbf{e}(S_1^i, g) \cdot \mathbf{e}(S_2^i, U_1(ID_i)) \cdot \mathbf{e}(S_3^i, U_1(M_i))\big)^{\delta_i} \tag{3}$$

$$= \mathbf{e}\left(\prod_{i=1}^{n} {S_1^i}^{\delta_i}, g\right) \cdot \prod_{i=1}^{n} \mathbf{e}\left({S_2^i}^{\delta_i}, u_1' \prod_{j=1}^{z} u_j^{\kappa_j^i}\right) \cdot \prod_{i=1}^{n} \mathbf{e}\left({S_3^i}^{\delta_i}, u_2' \prod_{j=1}^{z} u_j^{m_j^i}\right)$$

$$= P \cdot \prod_{i=1}^{n} \left(\mathbf{e}\left({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}\right) \cdot \mathbf{e}\left({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}\right)\right). \tag{4}$$

For the first part of the proof, all we need now is to show that (1) is equivalent to (2). Since for all i, $\mathsf{Verify}(ID_i, M_i, S_i) = 1$, (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 parentheses of (1) and get (2):

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

$$= \prod_{i=1}^{n} \big(\mathbf{e}(g^{b_i}, g^{\sum_{j=1}^{z} \kappa_j^i y_j}) \cdot \mathbf{e}(g^{c_i}, g^{\sum_{j=1}^{z} m_j^i y_j})\big)^{\delta_i}$$

$$= \prod_{i=1}^{n} \big(\mathbf{e}(g, g)^{\sum_{j=1}^{z} (\delta_i b_i \kappa_j^i y_j + \delta_i c_i m_j^i y_j)}\big)$$

$$= \prod_{j=1}^{z} \big(\mathbf{e}(g, g)^{y_j \sum_{i=1}^{n} (\delta_i b_i \kappa_j^i + \delta_i c_i m_j^i)}\big)$$

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

We must now show the other direction. This proof is an application of the technique for proving the small exponents test in [4]. 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 $a_i, b_i, c_i \in \mathbb{Z}_q$. Also, since $h \in \mathbb{G}$ we can write $h = g^d$ for some $d \in \mathbb{Z}_q$.

Since (3) is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as

$$\prod_{i=1}^{n} A^{\delta_i} = \prod_{i=1}^{n} \big(\mathbf{e}(g^{a_i}, g) \cdot \mathbf{e}(g^{b_i}, g^{y_1'} g^{\sum_{j=1}^{z} y_j \kappa_j}) \cdot \mathbf{e}(g^{c_i}, g^{y_2'} g^{\sum_{j=1}^{z} y_j m_j})\big)^{\delta_i}$$

$$= \prod_{i=1}^{n} \mathbf{e}(g, g)^{\delta_i (a_i + b_i y_1' + c_i y_2' + b_i \sum_{j=1}^{z} y_j \kappa_j + c_i \sum_{j=1}^{z} y_j m_j)}$$

$$= \mathbf{e}(g,h)^{\sum_{i=1}^{n}\delta_i d^{-1}(a_i+b_i y_1'+c_i y_2'+b_i\sum_{j=1}^{z} y_j\kappa_j^i+c_i\sum_{j=1}^{z} y_j m_j^i)}$$

$$\Rightarrow \sum_{i=1}^{n}\delta_i\alpha - \sum_{i=1}^{n}\delta_i d^{-1}\left(a_i + b_i y_1' + c_i y_2' + b_i\sum_{j=1}^{z} y_j\kappa_j^i + c_i\sum_{j=1}^{z} y_j m_j^i\right)$$
$$\equiv 0 \pmod q.$$

Setting $\beta_i = \alpha - d^{-1}(a_i + b_i y_1' + c_i y_2' + b_i \sum_{j=1}^{z} y_j\kappa_j^i + c_i\sum_{j=1}^{z} y_j m_j^i)$, this can be written as

$$\sum_{i=1}^{n}\delta_i\beta_i \equiv 0 \pmod q. \tag{5}$$

Assume that $\mathsf{Batch}((ID_1, M_1, S_1), \ldots, (ID_n, M_n, S_n)) = 1$, but for at least one i it is the case that $\mathsf{Verify}(ID_i, M_i, S_i) = 0$. Assume w.l.o.g. 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 (5) give us

$$\delta_1 \equiv -\gamma_1\sum_{i=2}^{n}\delta_i\beta_i \pmod q. \tag{6}$$

Given (ID_i, M_i, S_i), where $i = 1\ldots n$, let E be an event that $\mathsf{Verify}(ID_1, M_1, S_1) = 0$ holds but that $\mathsf{Batch}((ID_1, M_1, S_1), \ldots, (ID_n, M_n, S_n)) = 1$, 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 (6) 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 $\Pr[E] \le \sum_{i=1}^{|\Delta'|}(\Pr[E|\Delta'] \cdot \Pr[\Delta'])$. Plugging in the values, we get $\Pr[E] \le \sum_{i=1}^{2^{\ell_b(n-1)}}(2^{-\ell_b}\cdot 2^{-\ell_b(n-1)}) = 2^{-\ell_b}$. □

Efficiency Note The signature for Π-IBS consists of three group elements, but since it is identity-based there is 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, by switching the order of the elements in the first pairing and modifying the public parameters accordingly, the scheme also works in the asymmetric bilinear setting.

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 elements S_2 and S_3 can be represented using 160 bits, whereas S_1 needs 512 bits. So, all in all, we can represent the signature on the message and the identity using only 832 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 [9] for the certificates and a modified version of the CL signature scheme [13] 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 Sect. 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 Sect. 1.2 notation.

Signatures: We describe a new signature scheme Π-Sig with a batch verifier in Sect. 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 300 ms), and only signatures from the same ϕ can be batch verified together. To verify n Π-Sig 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 [9] described in Sect. 2, using the small exponents test [4], which requires distinct messages.

Batch Verify: Given purported signatures σ_i from n users on *distinct* messages M_i for $i = 1 \dots n$, first check that all public keys pk_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i. 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 $\mathsf{Verify}(pk_1, M_1, S_1) = \cdots = \mathsf{Verify}(pk_n, M_n, S_n) = 1$ implies that $\mathsf{Batch}((pk_1, M_1, S_1), \dots, (pk_n, M_n, S_n)) = 1$. This follows from the verification equation for the BLS scheme:

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} \Leftrightarrow \mathbf{e}\left(\prod_{i=1}^{n} \sigma_i^{\delta_i}, g\right) = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i}. \tag{7}$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [4]. 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 (7) holds, so we can rewrite it as

$$\begin{aligned} \prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} &= \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(g, g)^{\delta_i r_i x_i} \\ &\Rightarrow \mathbf{e}(g, g)^{\sum_{i=1}^{n} \delta_i c_i} = \mathbf{e}(g, g)^{\sum_{i=1}^{n} \delta_i r_i x_i} \\ &\Rightarrow \sum_{i=1}^{n} \delta_i c_i - \sum_{i=1}^{n} \delta_i r_i x_i \equiv 0 \pmod{q}. \end{aligned}$$

Setting $\beta_i = c_i - r_i x_i$, this is equivalent to

$$\sum_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod{q}.$$

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

Single Signer for BLS However, BLS [9] 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 ([9]). *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 [45], which is based on the Camenisch–Lysyanskaya signature scheme [13].

The Original CL Scheme Recall the Camenisch and Lysyanskaya signature scheme [13]. 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., they can all 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).

We note that the idea of sharing some value between the signers in order to efficiently perform some operation on the signatures is not new. Gentry and Ramzan present an identity-based aggregate signature scheme [29] in which signatures can only be aggregated if all signers agree on some dummy message that none of them have used before.

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 $\epsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$ 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 predefined. 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

$$H_1(\phi_i) = \begin{cases} g^{r_i} & \text{if } \phi_i \neq \phi'; \\ a' & \text{otherwise} \end{cases} \tag{8}$$

and

$$H_2(\phi_i) = \begin{cases} g^{s_i} & \text{if } \phi_i \neq \phi'; \\ b' & \text{otherwise.} \end{cases} \tag{9}$$

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

$$H_3(m_l||\phi_i) = \begin{cases} t_{l,i} & \text{if } \phi_i \neq \phi' \text{ or } l \neq j; \\ w' & \text{otherwise.} \end{cases} \tag{10}$$

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

$$\mathcal{O}_{sk^*}(m_l||\phi_i) = \begin{cases} \text{abort} & \text{if } \phi_i = \phi' \text{ and } l \neq j; \\ c' & \text{else if } \phi_i = \phi' \text{ and } l = j; \\ X^{r_i} X^{(s_i) t_{l,i}} & \text{otherwise.} \end{cases} \tag{11}$$

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. 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 $\epsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$. □

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 $(x_1, y_1), \dots, (x_k, y_k)$, 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 simply constrain them exactly as we programmed our random oracles.

Fortunately, Canetti, Halevi, and Katz [14] describe a method of efficiently constructing such a hash function family which allows us to map strings to bilinear map elements (or to map strings to elements in another prime-order algebraic group such as $\mathbb{Z}_q$). 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 $\sigma_1, \dots, \sigma_n$ on messages $m_1, \dots, m_n$ for the same period ϕ can be done as follows. (Recall that each signer can issue at most one signature per time period. Thus, these n signatures are all from different signers.) Assume that user i with public key X_i signed message m_i. Set $w_i = H(m_i || \phi)$. First check that all public keys X_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i. If not, output *reject*. Otherwise pick a vector $\Delta = (\delta_i, \dots, \delta_n)$ 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 $\mathsf{Verify}(X_1, M_1, S_1) = \dots = \mathsf{Verify}(X_n, M_n, S_n) = 1$ implies that $\mathsf{Batch}((X_1, M_1, S_1), \dots, (ID_n, M_n, S_n)) = 1$. This follows from the verification equation for the Π-Sig scheme if we keep in mind that

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

$$\Leftrightarrow \mathbf{e}\left(\prod_{i=1}^n \sigma_i^{\delta_i}, g\right) = \mathbf{e}\left(a, \prod_{i=1}^n X_i^{\delta_i}\right) \cdot \mathbf{e}\left(b, \prod_{i=1}^n X_i^{w_i \delta_i}\right). \tag{12}$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [4]. 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 (12) is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} \bigl(\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i}\bigr)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(g, g)^{\delta_i (r x_i + s x_i w_i)}$$

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

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

Setting $\beta_i = c_i - (r x_i + s x_i w_i)$, this is equivalent to

$$\sum_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod{q}.$$

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 $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$, and $\sigma = a^x b^{xw}$ 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

$\mathbf{e}: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, 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 512 bits for a security level comparable to RSA-1024. For Π-Sig signatures we need to hash into $\mathbb{G}_1$, which, according to Galbraith, Paterson, and Smart, can be done efficiently [27]. 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.

Second, suppose one uses the universal one-way hash functions described by Canetti, Halevi, and Katz [14] 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 requires 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 [9] provide a batch verification scheme for multiple signers on the *same* message). We extended the general batch verification definition of Bellare, Garay, and Rabin [4] 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 and 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 Sect. 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.

Acknowledgements

We thank Ivan Damgård, Anna Lisa Ferrara, Jean-Pierre Hubaux, Panos Papadimitratos, and the anonymous reviewers for their helpful input. Susan Hohenberger and Michael Østergaard Pedersen performed part of this research while at IBM Research, Zürich Research Laboratory. Also, Michael Østergaard Pedersen performed part of this research while at the University of Aarhus. Susan Hohenberger is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF CAREER CNS-1053886, a Microsoft Faculty Fellowship, and a Google Faculty Research Award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government.

References

[1] J.H. An, Y. Dodis, T. Rabin, On the security of joint signature and encryption, in *Advances in Cryptology—EUROCRYPT'02*, ed. by L.R. Knudsen. Lecture Notes in Computer Science, vol. 2332 (Springer, Berlin, 2002), pp. 83–107

[2] B. Barak, R. Canetti, J.B. Nielsen, R. Pass, Universally composable protocols with relaxed set-up assumptions, in *45th Symposium on Foundations of Computer Science (FOCS)* (IEEE Computer Society, Washington, 2004), pp. 186–195

[3] K. Barr, K. Asanović, Energy aware lossless data compression, in *MobiSys* (USENIX, Berkeley, 2003)

[4] M. Bellare, J.A. Garay, T. Rabin, Fast batch verification for modular exponentiation and digital signatures, in *Advances in Cryptology—EUROCRYPT'98*, ed. by K. Nyberg. Lecture Notes in Computer Science, vol. 1403 (Springer, Berlin, 1998), pp. 236–250

[5] D. Boneh, X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles, in *Advances in Cryptology—EUROCRYPT'04*, ed. by C. Cachin, J. Camenisch. Lecture Notes in Computer Science, vol. 3027 (Springer, Berlin, 2004), pp. 223–238

[6] D. Boneh, X. Boyen. Short signatures without random oracles, in *Advances in Cryptology—EUROCRYPT'04*, ed. by C. Cachin, J. Camenisch. Lecture Notes in Computer Science, vol. 3027 (Springer, Berlin, 2004), pp. 382–400

[7] D. Boneh, M.K. Franklin, Identity-based encryption from the Weil pairing, in *Advances in Cryptology—CRYPTO'01*, ed. by J. Kilian. Lecture Notes in Computer Science, vol. 2139 (Springer, Berlin, 2001), pp. 213–229

[8] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in *Advances in Cryptology—EUROCRYPT'03*, ed. by E. Biham. Lecture Notes in Computer Science, vol. 2656 (Springer, Berlin, 2003), pp. 416–432

[9] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing. *J. Cryptol.* **17**(4), 297–319 (2004)

[10] C. Boyd, C. Pavlovski, Attacking and repairing batch verification schemes, in *Advances in Cryptology—ASIACRYPT'00*, ed. by T. Okamoto. Lecture Notes in Computer Science, vol. 1976 (Springer, Berlin, 2000), pp. 58–71

[11] X. Boyen, B. Waters, Compact group signatures without random oracles, in *Advances in Cryptology—EUROCRYPT'06*, ed. by S. Vaudenay. Lecture Notes in Computer Science, vol. 4004 (Springer, Berlin, 2006), pp. 427–444

[12] J. Camenisch, S. Hohenberger, M.Ø. Pedersen, Batch verification of short signatures, in *Advances in Cryptology—EUROCRYPT'07*, ed. by M. Naor. Lecture Notes in Computer Science, vol. 4515 (Springer, Berlin, 2007), pp. 246–263

[13] J. Camenisch, A. Lysyanskaya, Signature schemes and anonymous credentials from bilinear maps, in *Advances in Cryptology—CRYPTO'04*, ed. by M.K. Franklin. Lecture Notes in Computer Science, vol. 3152 (Springer, Berlin, 2004), pp. 56–72

[14] R. Canetti, S. Halevi, J. Katz, A forward-secure public-key encryption scheme, in *Advances in Cryptology—EUROCRYPT'03*, ed. by E. Biham. Lecture Notes in Computer Science, vol. 2656 (Springer, Berlin, 2003), pp. 255–271

[15] T. Cao, D. Lin, R. Xue, Security analysis of some batch verifying signatures from pairings. *Int. J. Netw. Secur.* **3**(2), 138–143 (2006)

[16] Car 2 Car, Communication consortium. http://car-to-car.org

[17] J.C. Cha, J.H. Cheon, An identity-based signature from gap Diffie–Hellman groups, in *6th Public Key Cryptography (PKC)*, ed. by Y. Desmedt. Lecture Notes in Computer Science, vol. 2567 (Springer, Berlin, 2003), pp. 18–30

[18] S. Chatterjee, P. Sarkar, Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model, in *8th Information Security and Cryptology (ICISC)*, ed. by D. Won, S. Kim. Lecture Notes in Computer Science, vol. 3935 (Springer, Berlin, 2005), pp. 424–440

[19] S. Chatterjee, P. Sarkar, HIBE with short public parameters without random oracle, in *Advances in Cryptology—ASIACRYPT'06*, ed. by X. Lai. Lecture Notes in Computer Science, vol. 4284 (Springer, Berlin, 2006), pp. 145–160

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

[21] J.H. Cheon, Y. Kim, H.J. Yoon, A new ID-based signature with batch verification, 2004. Cryptology ePrint Archive: Report 2004/131

[22] J.H. Cheon, D.H. Lee, Use of sparse and/or complex exponents in batch verification of exponentiations. *IEEE Trans. Comput.* **55**(12), 1536–1542 (2006)

[23] S. Cui, P. Duan, C.W. Chan, An efficient identity-based signature scheme with batch verifications, in *1st International Conference on Scalable Information Systems (InfoScale)*, ed. by A. Chowdhury, F. Lau, F.Z. Wang (ACM Press, New York, 2006)

[24] W. Diffie, M. Hellman, New directions in cryptography. *IEEE Trans. Inf. Theory* **22**, 644–654 (1976)

[25] A.L. Ferrara, M. Green, S. Hohenberger, M.Ø. Pedersen, Practical short signature batch verification, in *CT-RSA* (2009). Cryptology ePrint Archive: Report 2008/015

[26] A. Fiat, Batch RSA, in *Advances in Cryptology—CRYPTO'89*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1989), pp. 175–185

[27] S.D. Galbraith, K.G. Paterson, N.P. Smart, Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165

[28] C. Gentry, How to compress Rabin ciphertexts and signatures (and more), in *Advances in Cryptology—CRYPTO'04*, ed. by M.K. Franklin. Lecture Notes in Computer Science, vol. 3152 (Springer, Berlin, 2004), pp. 179–200

[29] C. Gentry, Z. Ramzan, Identity-based aggregate signatures, in *9th Public Key Cryptography (PKC)*, ed. by M. Yung. Lecture Notes in Computer Science, vol. 3958 (Springer, Berlin, 2006), pp. 257–273

[30] S. Goldwasser, S. Micali, R.L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2) (1988)

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

[32] L. Harn, Batch verifying multiple DSA digital signatures. *Electron. Lett.* **34**(9), 870–871 (1998)

[33] L. Harn, Batch verifying multiple RSA digital signatures. *Electron. Lett.* **34**(12), 1219–1220 (1998)

[34] F. Hoshino, M. Abe, T. Kobayashi, Lenient/strict batch verification in several groups, in *4th Information Security*, ed. by G.I. Davida, Y. Frankel. Lecture Notes in Computer Science, vol. 2200 (Springer, Berlin, 2001), pp. 81–94

[35] M.-S. Hwang, C.-C. Lee, Y.-L. Tang, Two simple batch verifying multiple digital signatures, in *3rd Information and Communications Security (ICICS)*, ed. by S. Qing, T. Okamoto, J. Zhou. Lecture Notes in Computer Science, vol. 2229 (Springer, Berlin, 2001), pp. 233–237

[36] M.-S. Hwang, I.-C. Lin, K.-F. Hwang, Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica* **11**(1), 15–19 (2000)

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

[38] N. Koblitz, A. Menezes, Pairing-based cryptography at high security levels, 2005. Cryptology ePrint Archive: Report 2005/076

[39] C.-S. Laih, S.-M. Yen, Improved digital signature suitable for batch verification. *IEEE Trans. Comput.* **44**(7), 957–959 (1995)

[40] O. Landsiedel, K. Wehrle, S. Götz, Accurate prediction of power consumption in sensor networks, in *IEEE Workshop on Embedded Networked Sensors (EmNetS-II)* (2005)

[41] L. Law, B.J. Matt, Finding invalid signatures in pairing-based batches, in *Cryptography and Coding, 11th IMA International Conference*, ed. by S.D. Galbraith. Lecture Notes in Computer Science, vol. 4887 (Springer, Berlin, 2007), pp. 34–53

[42] S. Lee, S. Cho, J. Choi, Y. Cho, Efficient identification of bad signatures in RSA-type batch signature. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E89-A**(1), 74–80 (2006)

[43] C. Lim, P. Lee, Security of interactive DSA batch verification. *Electron. Lett.* **30**(19), 1592–1593 (1994)

[44] C.H. Lim, Efficient multi-exponentation and application to batch verification of digital signatures, 2000. http://dasan.sejong.ac.kr/chlim/english_pub.html

[45] A. Lysyanskaya, R.L. Rivest, A. Sahai, S. Wolf, Pseudonym systems, in *6th Selected Areas in Cryptography (SAC)*, ed. by C. Adams, H. Heys. Lecture Notes in Computer Science, vol. 1758 (Springer, Berlin, 1999), pp. 184–199

[46] A. Menezes, S. Vanstone, T. Okamoto, Reducing elliptic curve logarithms to logarithms in a finite field, in *23rd ACM Symposium on Theory of Computing (STOC)* (1991), pp. 80–89

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

[48] D. Naccache, D. M'Raïhi, S. Vaudenay, D. Raphaeli, Can DSA be improved? Complexity trade-offs with the digital signature standard, in *Advances in Cryptology—EUROCRYPT'94*, ed. by A. De Santis. Lecture Notes in Computer Science, vol. 950 (Springer, Berlin, 1994), pp. 77–85

[49] M. Raya, J.-P. Hubaux, Securing vehicular ad hoc networks. *J. Comput. Secur.* **15**, 39–68 (2007)

[50] SeVeCom, Security on the road. http://www.sevecom.org

[51] H. Shacham, D. Boneh, Improving SSL handshake performance via batching, in *Cryptographer's Track at RSA Conference '01*, ed. by D. Naccache. Lecture Notes in Computer Science, vol. 2020 (Springer, Berlin, 2001), pp. 28–43

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

[53] B. Waters, Efficient identity-based encryption without random oracles, in *Advances in Cryptology—EUROCRYPT'05*, ed. by R. Cramer. Lecture Notes in Computer Science, vol. 3494 (Springer, Berlin, 2005), pp. 320–329

[54] H.J. Yoon, J.H. Cheon, Y. Kim, Batch verifications with ID-based signatures, in *7th Information Security and Cryptology (ICISC)*, ed. by C. Park, S. Chee. Lecture Notes in Computer Science, vol. 3506 (Springer, Berlin, 2004), pp. 233–248

[55] F. Zhang, K. Kim, Efficient ID-based blind signature and proxy signature from bilinear pairings, in *8th Information Security and Privacy, Australasian Conference (ACISP)*, ed. by R. Safavi-Naini, J. Seberry. Lecture Notes in Computer Science, vol. 2727 (Springer, Berlin, 2003), pp. 312–323

[56] F. Zhang, R. Safavi-Naini, W. Susilo, Efficient verifiably encrypted signature and partially blind signature from bilinear pairings, in *Progress in Cryptology—INDOCRYPT'03*, ed. by T. Johansson, S. Maitra. Lecture Notes in Computer Science, vol. 2904 (Springer, Berlin, 2003), pp. 191–204