Locally Verifiable
Signature and Key Aggregation

Rishab Goyal
MIT[*]

Vinod Vaikuntanathan
MIT[†]

Abstract

Aggregate signatures (Boneh, Gentry, Lynn, Shacham, Eurocrypt 2003) enable compressing a set of N signatures on N different messages into a short aggregate signature. This reduces the space complexity of storing the signatures from linear in N to a fixed constant (that depends only on the security parameter). However, verifying the aggregate signature requires access to all N messages, resulting in the complexity of verification being at least $\Omega(N)$.

In this work, we introduce the notion of *locally verifiable* aggregate signatures that enable *efficient verification*: given a short aggregate signature σ (corresponding to a set $\mathcal{M}$ of N messages), the verifier can check whether a particular message m is in the set, in time independent of N. Verification does *not* require knowledge of the entire set $\mathcal{M}$. We demonstrate many natural applications of locally verifiable aggregate signature schemes: in the context of certificate transparency logs; in blockchains; and for redacting signatures, even when all the original signatures are produced by a single user.

We provide two constructions of single-signer locally verifiable aggregate signatures, the first based on the RSA assumption and the second on the bilinear Diffie-Hellman inversion assumption, both in the random oracle model.

As an additional contribution, we introduce the notion of compressing cryptographic keys in identity-based encryption (IBE) schemes, show applications of this notion, and construct an IBE scheme where the secret keys for N identities can be compressed into a single aggregate key, which can then be used to decrypt ciphertexts sent to any of the N identities.

[*] Email: goyal@utexas.edu. Research supported by grants listed under the second author.

[†] Email: vinodv@mit.edu. Research supported in part by DARPA under Agreement No. HR00112020023, a grant from the MIT-IBM Watson AI, a grant from Analog Devices and a Microsoft Trustworthy AI grant. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

Contents

1 Introduction

The notion of aggregate signatures, introduced by Boneh, Gentry, Lynn, and Shacham [BGLS03], enables the compression of several signatures σ_i of messages m_i with respect to public keys vk_i, into a single, short signature $\widehat{\sigma}$ which authenticates the entire tuple of messages with respect to the tuple of public keys. While the original motivation for aggregate signatures was the compression of certificate chains and the aggregation of signatures in secure BGP, the notion has found a great deal of practical interest recently in the context of blockchains [Gor18].

While the aggregate signatures are short, verifying them requires access to *all* the messages. In many practical scenarios, as we describe below, the verifier is merely interested in checking if $\widehat{\sigma}$ is an aggregated signature of *some* set that contains a particular message m. It may be infeasible or undesirable to download the entire list of messages, and perform a verification computation whose runtime scales with the number of messages N. This leads us to the central question that motivates this work: *Can we construct* locally verifiable *aggregate signatures?*

Locality in access and computation is a central theme in computer science, in areas ranging from coding theory [Yek12] to proof systems [Sud09] to sub-linear algorithms [Gol17]. Thus, the question of local verifiability is both practically motivated, and also conceptually very natural.

1.1 Locally Verifiable Aggregate Signatures: Definition and Applications

Our first contribution is a definition of the notion of *locally verifiable aggregate signatures*, which turns out to require some care.

A natural formalization asks for two algorithms: an aggregation algorithm Aggregate, that takes a set of tuples $\{(m_i, \mathsf{vk}_i, \sigma_i)\}_{i=1}^{N}$ and produces a short aggregate signature $\widehat{\sigma}$ of size, say, $\mathsf{poly}(\lambda)$ bits and a local verification algorithm LocalAggVerify, that takes the aggregate signature $\widehat{\sigma}$, a public key vk, and a message m, and outputs accept or reject. It seems natural to require that LocalAggVerify runs in time independent of N, and accepts $(m, \mathsf{vk}, \widehat{\sigma})$ if and only if $(m, \mathsf{vk}) \in \{(m_i, \mathsf{vk}_i)\}_{i=1}^{N}$.

It is not hard to see that this notion is *impossible* to achieve, even in the *single-signer setting* where all signatures are produced w.r.t. a single public key vk, due to a simple incompressibility argument. Indeed, such a pair of algorithms can be used to recover all the messages given just the aggregate signature, violating incompressibility. In more detail, assume that the messages are of the form (i, b_i) where $b_i \in \{0, 1\}$ is a bit. To recover all the bits b_i given $\widehat{\sigma}$, one simply runs the LocalAggVerify algorithm with both $(i, 0)$ and $(i, 1)$ for every i.

In this work, we define the notion of locally verifiable aggregate signatures, overcoming the above incompressibility barrier. We focus on the single-signer setting, and show several applications of our notion.

Our Definition. To circumvent the incompressibility barrier, we include a hint generation algorithm LocalOpen that computes a short hint to aid local verification. Formally, in addition to the key generation, signing, and verification algorithms, a locally verifiable aggregate signature scheme consists of three additional algorithms. For the sake of concreteness, the reader should imagine three types of parties: signers who run KeyGen and Sign, storage servers (or aggregators) who run Aggregate and LocalOpen, and verifiers who run Verify and LocalAggVerify.

Aggregate is the (single-signer) signature aggregation algorithm which takes as input a sequence of pairs (m_i, σ_i) under a public key vk and produces an aggregate signature $\widehat{\sigma}$;

LocalOpen is the hint generator (also called the opening algorithm) that takes as input the aggregate signature $\widehat{\sigma}$ and the set of messages $\mathbf{m} = \{m_i\}_{i=1}^N$, and a target message $m \in \mathbf{m}$, and produces a *short hint* h;

(crucially, LocalOpen does *not* have access to the original signatures σ_i as they have been *forgotten* at this point.)

LocalAggVerify is the local verification algorithm that verifies the aggregate signature $\widehat{\sigma}$ and the short hint h for a message m.

(importantly, the run-time of LocalAggVerify is independent of N.)

The first thing to note is that our formalization circumvents the incompressibility barrier as local verification uses a message-dependent hint, and the hint generation depends on the set of all messages $\mathbf{m}$ (and not just the target message). Secondly, we will shortly describe how our definition fits into several practical applications of aggregate signatures.

For security, we propose an enhanced unforgeability property which protects from both a *malicious* aggregator and a *malicious* hint generator. It is defined against an adversary who obtains signatures for a set $\mathbf{m}$ and tries to produce a "fake" aggregate signature and a "fake" hint that makes the aggregate verifier accept a message $m \notin \mathbf{m}$. For more details, we refer the reader to Section 4.1.

How to use Locally Verifiable Aggregate Signatures in Applications. Local verifiability is an extremely desirable feature as it leads to many applications in certificate transparency logs and blockchains, generic implications to signature redactability, and provides a robust time-space tradeoff that can smoothly interpolate between aggregate signatures and plain signatures.

Certificate Transparency Logs. Certificate transparency (CT) [BLK13] is an internet security standard that creates public logs which record all certificates issued by certificate authorities (CAs). The log is audited periodically to identify mistakenly or maliciously issued certificates. A user's browser receives a certificate σ from a website, say on the message (`domain-name,IP`), and checks whether the entry exists in the CT log before proceeding to accept the connection. (This simplified description is sufficient for our purposes; however, for more details on how CT logs work, we refer the reader to [CTg]).

Aggregate signatures can *ease the burden of storage* on the CT log. Without aggregate signatures, the CT log has to store all the signatures (certificates) explicitly. With aggregate signatures, the CT log can store a short aggregate signature together with an arbitrary compressed data structure that compactly stores the list of messages (namely, domain names and IP addresses). However, even if the user's browser stores or downloads an aggregate signature, the only way to verify whether a particular entry exists in the log is to download all entries. Locally verifiable aggregate signatures allow the CT log to compress the certificates into a short aggregate signature while allowing the user to verify the existence of an entry by downloading just a few additional kilobytes (in the form of a short hint) and performing a fast computation (using the LocalAggVerify algorithm). Furthermore, our enhanced unforgeability property guarantees that this is secure against even a malicious CT log who may try to convince the user that a message $m \in \mathbf{m}$ when it isn't.

We note that *even single-signer* locally verifiable aggregate signatures are a very meaningful solution in this scenario given that the certificate authorities number in the *hundreds* while the number of certificates generated number in the *billions*. The hints need not be explicitly stored, and can be computed on-the-fly by the CT log enabling natural forms of space-time tradeoffs and caching mechanisms (for the hints) for frequently accessed websites. Jumping ahead, we note that one of our constructions (in particular, the RSA-based construction) has the surprising additional feature of being able to *reconstruct* the original signature of any particular message $m \in \mathbf{m}$ given only the aggregate signature and the set of messages $\mathbf{m}$ — this could come in handy during the auditing of the CT log.

Blockchains. Another application scenario arises in the context of blockchains where a user or an organization wants to aggregate the signatures on the set of all transactions *originating from a single payer*, and later wishes to quickly and with little communication convince a third party (e.g. an auditor) of the existence of a particular transaction. Again, the above problem can be elegantly solved by using locally verifiable aggregate signatures as the user/organization can compute the short hint to prove the existence of the appropriate transaction.

We note that local verification implicitly provides a useful privacy feature. The user/organization can prove knowledge of a single transaction without revealing the remaining transactions. This follows from the succinctness requirements, as neither the aggregate signature nor the hint grow with the number of transactions; thus, the signature and the hint jointly cannot leak too much information about the other transactions. In addition, some of our constructions satisfy properties such as *dynamic aggregation* which could be very useful in this scenario.

Time-Space Tradeoffs from Local Verifiability. Consider a server that stores a collection of N messages $\{m_i\}_{i=1}^N$ along with signatures $\{\sigma_i\}_{i=1}^N$, and several possible clients who wish to download single messages and check that they indeed belong to the collection. While this can be solved by using vanilla signatures, the server must dedicate large space for storing all N signatures. Traditional aggregate signatures can handle the server space issue, but they incur (huge) linear runtime cost for each individual client. As summarized in Table 1, the run-time for individual clients can be lowered to $O(1)$ by using locally verifiable aggregate signatures.

We can also obtain a smooth time-space tradeoff that interpolates between locally verifiable aggregate signatures and vanilla signatures. For example, the server could split the collection of N messages into blocks of length L and aggregate each block of L signatures, reducing the server run-time to $O(L)$ at the cost of increasing the server storage to $O(N/L)$. This mechanism can be further generalized to obtain a three-way time-space tradeoff that interpolates between vanilla signatures, aggregate signatures, and locally verifiable aggregate signatures. In this hybrid mode of local verification, the signer signs blocks of L_2 messages by hashing the block first and then signing it. The server stores N messages by splitting them into $N/(L_1 L_2)$ super-blocks, each of which contains L_1 blocks, where each block, in turn, contains L_2 messages (as above). The server aggregates the L_1 (locally verifiable aggregate) signatures in each super-block and stores them.

Type of Signatures	Server space (for signatures)	Server time	Per-client space (for signatures)	Per-client time
Vanilla Signatures	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Aggregate Signatures	$O(1)$	$O(1)$	$O(1)$	$O(N)$
L.V. Aggregate	$O(1)$	$O(N)$	$O(1)$	$O(1)$
Hybrid (with L_2 batch and $L_1 L_2$ block size)	$O\left(\frac{N}{L_1 L_2}\right)$	$O(L_1)$	$O(1)$	$O(L_2)$

Table 1: Time-Space Tradeoffs with Locally Verifiable (L.V.) Aggregate Signatures.

The server thus stores N/L_1L_2 signatures. To access a message, the client retrieves an entire block containing the message, spending $O(L_2)$ time. To answer the client query, the server runs in time $O(L_1)$ to generate the hint corresponding to the hash of the block queried by the client. In short, the new notion of local verification provides a *robust time-space tradeoff* for the parties involved.

Given that most data in the real world is compressible, locally verifiable aggregate signatures give the server the ability to fully leverage compression and reduce the *total* storage (including the messages) and communication to sublinear in N. This is possible neither with vanilla signatures (where one cannot compress the signatures) nor with regular aggregate signatures (where a client cannot avoid downloading all messages). Although the hint generation is expensive, it is done once by the (potentially untrusted) server as opposed to imposing a heavy verification cost per client as in regular aggregate signatures. Furthermore, the hints for the most frequently accessed messages can be cached for better performance. In a nutshell, locally verifiable aggregate signatures open up a rich space of tradeoffs in storage, communication and verification of signatures.

Redactable Signatures. Redactable signature schemes [JMSW02, SBZ01] allow a signature holder to publicly censor parts of a signed document such that the corresponding signature σ can be efficiently updated without the secret signing key, and the updated signature can still be verified given only the redacted document. These signatures have many real-world applications in privacy-preserving authentication as they can be used to sanitize digital signatures. (See [DPSS15, DKS16] for a detailed overview.)

Locally verifiable aggregate signatures provide a fresh approach to redactability and sanitization. Briefly, using a locally verifiable aggregate signature, we can sign the large sensitive document in three steps: first, split the document into small message blocks; second, sign the message blocks individually, together with their index; and third, aggregate these individual signatures and output the aggregated signature as the final signature for the full document. To verify the full (unredacted) document, one could use the regular verification algorithm that takes the entire document as input. For redaction, the redacting party can generate short hints for each of the unredacted portions of the document, and include these as part of the redacted signature. Note that the redacted signature can be verified by running local verification. At a very informal level, since the redacted signature is shorter than the total number of message blocks, this seems to guarantee some form of privacy.

While this general outline is problematic for several reasons: first, the redacted signatures are long; and secondly, the above argument does not guarantee true privacy, namely that the signature on the redacted document does not reveal *any* information about the redacted messages. However, it turns out that our RSA-based construction and a slight modification of our pairing-based construction give a complete solution to the problem, ensuring privacy of the original (unredacted) message, enabling multi-hop redaction as well as constant-size redacted signatures, improving on the construction in [JMSW02]. We refer the reader to Section 2 for more details.

1.2 Locally Verifiable Aggregate Signatures: Our Results

Our main result constructs a single-signer locally verifiable aggregate signature scheme secure under the strong RSA assumption [BP97].

Theorem 1.1 (Informal)**.** Assuming strong RSA, there is a locally verifiable aggregate signature scheme. In the random oracle model, it is fully secure; and in the standard model, it is statically secure.

Our second result shows a weaker scheme under the bilinear Diffie-Hellman inversion (BDHI) assumption [MSK02, BB04a, BB04b]. The scheme requires a long common reference string (CRS) of size equal to the number of aggregated messages. The verifier, however, only needs access to a fixed constant size portion of the CRS and is, therefore, still efficient.

Theorem 1.2 (Informal)**.** Under the BDHI assumption, there is a locally verifiable aggregate signature scheme in the long CRS model. With random oracles, the scheme is fully secure; and in the standard model, it is statically secure.

Finally, we show an initial feasibility result for a *multi-signer* locally verifiable aggregate signatures using the machinery of succinct non-interactive arguments of knowledge (SNARKs). We note that single-signer aggregate signature schemes, without locality, have several (folklore) instantiations based on the RSA assumption, the SIS assumption, and so on. This is in contrast to the multi-signer setting where bilinear maps seem to dominate. Our work generalizes single-signer aggregate signatures in a different direction, requiring locality, and exposing a new, challenging, and practically motivated facet of the problem.

1.3 Compressing Cryptographic Keys

As an independently interesting contribution, we introduce a novel generalization of signature aggregation to the setting of compressing the keys in identity-based and, more generally, attribute-based encryption schemes (IBE, ABE). This enables the decryption key holders to compress multiple keys into a short key such that the aggregated key can be used to decrypt all ciphertexts that any of individual (unaggregated) decryption keys are authorized to decrypt. Since one of the main motivations behind designing advanced encryption systems is to have the ability to generate separate keys for different users, thus it might feel counterintuitive to study compression of keys. However, there are two main reasons to study aggregation in encryption systems.

First, this immediately can be used to reduce storage space in many simple applications. For example, consider the classical application of using IBE to delegate access over time. In particular, there is a user who has an IBE master secret key msk, and generates temporary keys sk_{date} for other devices (such as mobile phones) that are more easily stolen. The messages encrypted are tagged with different dates, so the temporary keys can decrypt only the corresponding ciphertexts. *Aggregatable IBE* allows to compress any subset of these temporary keys into one short key that can decrypt ciphertexts encrypted to any of underlying dates. While one could use heavyweight tools (such as ABE) to solve this problem, our observation is that IBE constructions with such great aggregation properties can lead to a simpler and relatively lightweight solution. This directly leads to the second (and broader) reason for studying aggregatable encryption systems which is that they can enable simpler solutions to problems that otherwise needed more advanced objects. We also provide a simple construction for an aggregatable IBE scheme from the BDHI assumption.

Theorem 1.3 (Informal)**.** Under the BDHI assumption, there is an aggregatable IBE scheme in the random oracle model.

1.4 Other Related Work

The concept of aggregate signatures was first put forth by Boneh, Gentry, Lynn, and Shacham [BGLS03] to allow a third party to compress an arbitrary group of signatures into a short aggregated signature that jointly authenticates all the compressed signatures. Aggregate signatures are related to, but significantly different from, multisignatures [IN83, Oka88, OO99, MOR01, Bol03] which were introduced in 1983 [IN83], but received a formal treatment much later by Ohta and Okamoto [Oka88, OO99] and Micali, Ohta, and Reyzin [MOR01]. They differ in terms of functionality and applications since in multisignatures, a set of users all sign the same message and the result is a single signature; while aggregate signatures are used to compress a group of signatures, where each signature might be signing a distinct message. In addition to the differing functionalities, multisignatures can have the group of signers or verifiers cooperate interactively, while aggregate signatures are more commonly studied in non-interactive settings.

Variants of aggregate signatures have been studied in the sequential [LMRS04] and synchronized [GR06] settings. In the sequential mode of aggregation, the signers are required to interact either by signing in a sequential chain; while in the synchronized setting, the signing algorithm takes as input a (time) period t, and the security of the scheme is conditioned on a signer signing at most once for each period t.

Numerous works have constructed (single- and multi-signer) aggregate signatures from pairing based assumptions [BGLS03, BGLS, Bol03, GR06, LOS+06, BNN07, BGOY07, MT07, RS09, AGH10], factoring based assumptions [LMRS04, BN07, Nev08, BJ10, FLS12, LLY13a, LLY13b, BGR14, BMP16, HW18], and multilinear maps (and obfuscation) [FHPS13, HSW13, HKW15].

Another concept, loosely related to the notion of single-signer aggregate signatures, is that of batch verification which has been very well studied since the foundational work of Fiat [Fia89]. The main motivation behind batch verification of signatures (generated by a single signer) is to improve the concrete performance of the verifier checking a large sequence of messages. Thus, batch verification of signatures is not designed to produce a shorter aggregated signature which is our main goal.

2 Technical Overview

In this technical overview, we describe our RSA-based construction of locally verifiable aggregate signature in detail (proving Theorem 1.1), and briefly describe our pairing-based construction which uses similar high-level ideas but different algebraic tricks. At the end of the technical overview, we also discuss a SNARK-based construction of multi-signer locally verifiable aggregate signatures.

RSA-based Locally Verifiable Aggregate Signature. Our starting point is the classical RSA-based single-signer[1] aggregate signature scheme where the signature of a message m with respect to an RSA public key (N, e) is $\sigma = H(m)^d \pmod{N}$, where $ed = 1 \pmod{\varphi(N)}$ and H is a hash function modeled as a random oracle in the security analysis. Given L message-signature pairs $\{(m_i, \sigma_i)\}_{i=1}^{L}$, the aggregate signature is simply their product $\widehat{\sigma} = \prod_{i=1}^{L} \sigma_i \pmod{N}$. Verification proceeds by checking that

$$\widehat{\sigma}^e = \prod_{i=1}^{L} H(m_i) \pmod{N}.$$

Unfortunately, it is completely unclear how to "locally" verify a single message m_i given $\widehat{\sigma}$ and some hint h_i related to the message vector $\mathbf{m}$. Concretely, deducing how to even define the message-dependent hint is unclear. One may attempt to define the hint h_i to be the product of all hash values $H(m_j)$ for $j \neq i$, and let the local verifier check that $\widehat{\sigma}^e = h_i \cdot H(m_i)$. However, a malicious hint generator can easily fool the verifier: the hint is adversarially generated and the verifier has no mechanism to check that the hint is well-formed without recomputing it which, in turn, seems to require the verifier to know all the underlying messages, in direct conflict with the requirement of local verification. In a nutshell, the accumulator-style aggregation and the presence of a random oracle seems to make local verification challenging.

To avoid this issue, we look at other RSA-based signature schemes [GMR88, DN94, CD96, GHR99, CS00, Fis03] for adding local verifiability. While this seems like a plausible approach, it quickly gets stuck at a much earlier point. Namely, for all these schemes, the notion of single-signer aggregation has not even been studied (to the best of our knowledge). A closer inspection shows that, unlike the classical RSA-based signature scheme, most of these schemes do not support aggregation. A notable exception is the Gennaro, Halevi, and Rabin [GHR99] scheme which works as follows. Suppose H is a collision-resistant function that maps messages into large (λ-bit) *prime* numbers. The signature of a message m is $\sigma = g^{1/H(m)} \pmod{N}$ where $g \in \mathbb{Z}_N^*$ is random and (N, g) is in the public key. Letting e_{m_i} denote $H(m_i)$ and $\sigma_i = g^{1/e_{m_i}} \pmod{N}$ denote the signature of m_i, the aggregation algorithm can simply compute $\widehat{\sigma} = \prod_i \sigma_i \pmod{N}$ as the aggregated signature. Regular (non-local) verification can be performed by the following equation:

$$(\widehat{\sigma})^{\prod_i e_{m_i}} \overset{?}{=} \prod_i g^{\prod_{j \neq i} e_{m_j}} \pmod{N}.$$

[1] Incidentally, we mention that the problem of constructing a *multi-signer* aggregate signature scheme from RSA has been a long-standing open problem, although constructions of relaxed variants such as sequential or synchronized (multi-signer) aggregate signature schemes based on RSA exist [LMRS04, HW18].

A correctly generated aggregate signature passes the check because

$$(\widehat{\sigma})^{\prod_i e_{m_i}} = \left(\prod_i \sigma_i\right)^{\prod_i e_{m_i}} = \prod_i g^{\prod_{j \neq i} e_{m_j}} \pmod{N}. \tag{1}$$

We now show that the aggregate signatures $\widehat{\sigma}$ can also be *locally verified* w.r.t a message $m_j \in \mathbf{m}$ (the latter being the set of all messages whose signatures have been aggregated into $\widehat{\sigma}$) without knowing $\mathbf{m}$ but given only a short verification hint that depends on $\mathbf{m}$ and $\widehat{\sigma}$. Our first idea is to generate the following two whole numbers as the hint:

$$e_{\mathbf{m}\setminus m_j} = \prod_{i \neq j} e_{m_i}, \quad f_j = \sum_{i \neq j} \prod_{k \notin \{i,j\}} e_{m_k}.$$

Our key observation is the following equation (which is exactly the same as Equation 1 except it uses a different exponent for $\widehat{\sigma}$)

$$(\widehat{\sigma})^{e_{\mathbf{m}\setminus m_j}} = g^{f_j} \cdot g^{e_{\mathbf{m}\setminus m_j}/e_{m_j}} \pmod{N}. \tag{2}$$

This can be translated into the following verification equation:

$$\left((\widehat{\sigma})^{e_{\mathbf{m}\setminus m_j}}/g^{f_j}\right)^{e_{m_j}} \overset{?}{=} g^{e_{\mathbf{m}\setminus m_j}} \pmod{N}. \tag{3}$$

Since e_{m_j} can be computed from just the target message m_j, releasing $e_{\mathbf{m}\setminus m_j}$ and f_j as the hint enables local verification of the aggregate signature $\widehat{\sigma}$ via the above equation. It can also be proven secure in the presence of malicious hint generators as long as the local verification algorithm also checks that the numbers $e_{\mathbf{m}\setminus m_j}$ and e_{m_j} are co-prime (that is, $\gcd(e_{\mathbf{m}\setminus m_j}, e_{m_j}) = 1$).)

At a first glance, the above scheme seems to solve the problem of locally verifiable single-signer aggregate signatures from RSA; however, unfortunately, this is not the case. The hints $e_{\mathbf{m}\setminus m_j}$ and f_j have to be computed modulo $\phi(N)$, but the hint generator does not (and must not) know $\phi(N)$. The only way out seems to be to compute them over the integers which again does not work as they could be large $O(L)$-bit numbers, which is decidedly not short. These together seem like an unfortunate limitation to obtaining local verifiability. Luckily, this conundrum can be resolved in a rather simple, yet elegant, way using the surprising power of Shamir's trick [Sha84].

Our central observation is that the hint generator can completely *re-compute* the (unique) signature of *every* message in the set, starting from just the aggregate signature $\widehat{\sigma}$. In more detail, the hint generator first computes

$$z_j := (\widehat{\sigma})^{e_{\mathbf{m}\setminus m_j}}/g^{f_j} := g^{e_{\mathbf{m}\setminus m_j}/e_{m_j}} \pmod{N}.$$

Note that $e_{\mathbf{m}\setminus m_j}$ and e_{m_j} are co-prime, thus there exist efficently computable integers α and β such that $\alpha \cdot e_{\mathbf{m}\setminus m_j} + \beta \cdot e_{m_j} = 1$. The hint generator next *re*-computes the signature $g^{1/e_{m_j}}$ of m_j as

$$g^{1/e_{m_j}} = g^{(\alpha \cdot e_{\mathbf{m}\setminus m_j} + \beta \cdot e_{m_j})/e_{m_j}} = (g^{e_{\mathbf{m}\setminus m_j}/e_{m_j}})^{\alpha} \cdot g^{\beta} = z_j^{\alpha} \cdot g^{\beta} \pmod{N}.$$

It then outputs $g^{1/e_{m_j}}$ as the hint, and the local verification algorithm simply checks it by running the plain (non-aggregated) verification algorithm interpreting the hint as a signature on the message

m_j. (In fact, the local verification algorithm is independent of the aggregated signature $\widehat{\sigma}$, and only needs the hint for verification. A detailed discussion is provided in Section 5.2.)

This summarizes our RSA-based locally verifiable aggregate signature scheme, and the final remaining detail is to figure out how the function H is selected. To that end, we present two choices — the first is to let H employ a prime sequence generator based on a random oracle, which gives us a scheme that is fully secure in the random oracle model; and the second is to employ a technique similar to Micali, Rabin, and Vadhan [MRV99] (who used a t-wise independent hash function, but we use PRFs; see Section 5.1 for more details) to instantiate the scheme in the standard model. We point out that we could prove our standard model instantiation to be statically secure (in the sense that the adversary must query all messages before it sees the verification key). We leave the problem of constructing a fully secure scheme without random oracles as an interesting open problem.

In addition to the surprising (in our mind) property of allowing exact *re-computation* of individual signatures from aggregate signatures, our RSA-based scheme satisfies several additional properties such as support for multi-hop aggregation as well as unordered sequential aggregation. We also point out that the aforementioned exact re-computation property of our aggregate signatures is very useful for obtaining a redactable signature scheme which has constant-size redacted as well as unredacted signatures. In a nutshell, the redaction algorithm can first compute the individual signatures of all message blocks whose signature it wants to release, and can aggregate them again to create a shorter signature.

Pairing-based Locally Verifiable Aggregation. Our pairing-based signature scheme relies on similar core ideas, but very different details due to differing algebraic structures.

Our starting point is to translate the above process of RSA-based signature generation to bilinear maps as follows. Recall the signature of a message m is computed as $\sigma = g^{1/H(m)}$, where H is a collision-resistant function that maps messages into large *prime* numbers and the inverse in the exponent, $1/H(m)$, is computed using the factorization of the RSA modulus. To port this over to bilinear maps, we substitute $H(m)$ with $\alpha + m$, where α is a secret exponent from the master key. Basically, the signature is set as $\sigma = g^{1/(\alpha+m)}$, where g is a random public source group generator. The signature verification performs a bilinear pairing to check that $e(\sigma, g^\alpha g^m) = e(g, g)$, where g^α is part of the public key as well.

Coincidentally, this is exactly the weakly secure short signature scheme of Boneh and Boyen (BB) [BB04b, §4.3], and can be visualized as a bilinear analog of the RSA-based Gennaro-Halevi-Rabin scheme. Unfortunately, the BB scheme is also not known to be aggregatable, and while there exist pairing-based (multi-signer) aggregate signature schemes [BGLS03], they are algebraically similar to the classical RSA-based schemes, thus do not appear to support local verifiability.

Our first main observation is that the BB scheme can actually be shown to be a single-signer aggregatable scheme. Although the signature aggregation is not as simple as multiplying the signatures (as in the RSA setting), we observe that, by Lagrange's inverse polynomial interpolation technique, we can aggregate a sequence of signatures $\sigma_i = g^{1/(\alpha+m_i)}$ into $\widehat{\sigma} = g^{\prod_i 1/(\alpha+m_i)}$. Simply put, Lagrange's inverse polynomial interpolation allows the following computation without the knowledge of the secret exponent α:

$$\prod_{i=1}^{L} \frac{1}{\alpha + m_i} = \frac{\gamma_1}{\alpha + m_1} + \cdots + \frac{\gamma_L}{\alpha + m_L},$$

where the coefficients γ_i can be publicly computed given only the sequence of messages $\{m_i\}_{i=1}^{L}$. Thus, the aggregate signature $\widehat{\sigma}$ can be computed as

$$\widehat{\sigma} = \prod_{i=1}^{L} \sigma_i^{\gamma_i}.$$

In a different context of attribute-based encryption (ABE), this idea was used by Delerablée, Paillier, and Pointcheval [DPP07, DP08], except that they employed Newton's iterative algorithm instead of Lagrange's technique. More details about aggregating the group elements is provided in detail later in Section 6.

Note that while the above allows aggregation of signatures, for regular (non-local) verification, the verifier requires higher degree monomials of the secret exponent α. Concretely, the aggregate verifier symbolically evaluates the polynomial $\prod_{i=1}^{L}(X + m_i)$ to simplify it as $\sum_{i=0}^{L} \delta_i X^i$, and using bilinear maps it can verify the aggregate signature as $e(\widehat{\sigma}, \prod_i (g^{\alpha^i})^{\delta_i}) = e(g, g)$, but this needs the monomials g^{α^i} as part of the public key. This is precisely why our pairing-based scheme requires a long CRS/public key.

Unlike the [BGLS03] aggregate signature scheme, we can show that this scheme is locally verifiable. Our main observation here is that the non-local verification algorithm works in two phases. First, it pre-processes the public key, given only the set of messages, to compute $\prod_i (g^{\alpha^i})^{\delta_i} = g^{\prod_i(\alpha+m_i)}$ in the source group; second, it uses the bilinear map to pair this with the aggregate signature $\widehat{\sigma}$ and compare with $e(g, g)$. We note that the first step in the verification is inefficient, but a hint generator can speed it up for any target message m_j by generating the following two group elements as part of the short hint:

$$h_1 = g^{\prod_{i \neq j}(\alpha+m_i)}, \quad h_2 = g^{\alpha \prod_{i \neq j}(\alpha+m_i)} = h_1^{\alpha}.$$

Note that both h_1 and h_2 can also be publicly computed given only the public key, and set of messages contained in the aggregated signature. (This follows from the same symbolic execution of appropriate polynomials.)

And, given the hints h_1, h_2, a verifier can locally verify the aggregate signature as

$$e(\widehat{\sigma}, h_1^{m_j} h_2) = e(g, g).$$

However, the above verification check alone is insufficient as a malicious hint generator can very easily fool the verifier. To address malicious hint generators, we also include a simple well-formedness check of the hint as follows:

$$e(g^{\alpha}, h_1) = e(g, h_2).$$

Putting these ideas together, we construct the pairing-based locally verification single-signer aggregate signature scheme in the long CRS setting. We prove this to be statically secure in the standard model, and adaptively secure in the random oracle model by replacing $\alpha + m$ terms with $\alpha + H(m)$. For more details, see Section 6. We leave the problems of reducing the CRS size and removing the random oracle an interesting open problems.

We also note that while the above construction needs a long CRS, it satisfies a very interesting property, namely that the hint generation algorithm is *fully public*, and does not even depend on the aggregate signature. Such fully public hint generation will be useful in applications where the hint generator is unaware of the underlying aggregate signature, or the user wants to generate the hint even before the aggregate signature has been generated or made available.

Lastly, our aggregatable IBE scheme builds on the above ideas. For details, we refer the reader to Section 7.

Multi-Signer Scheme from SNARKs. In the "folklore" construction of aggregate signatures from succinct non-interactive arguments of knowledge (SNARKs), the aggregation algorithm simply proves, w.r.t. a sequence of verification key-message pairs $\{(\mathsf{vk}_i, m_i)\}_i$, that it knows a sequence of signatures $(\sigma_1, \ldots, \sigma_N)$ such that σ_i is an accepting signature for (vk_i, m_i). This results in short (aggregate) signatures and fast verification, while also ensuring that from an accepting proof, the extractor can extract an accepting signature for every verification key-message pair.

This outline can be extended in a simple way to give us a locally verifiable aggregate signature. To generate the short hint, the hint generator creates another SNARK proof, w.r.t. a target key-message pair (vk, m), that proves knowledge of a sequence of key-message pairs $\{(\mathsf{vk}_i, m_i)\}_i$ and an aggregate signature $\widehat{\sigma}$ such that (vk, m) is one of the tuples in the sequence, and $\widehat{\sigma}$ is an accepting signature for that sequence of key-message pairs. Clearly, the hint generator has the witness (i.e., sequence of key-message pairs and an accepting aggregated signature) available, thus by correctness and efficiency of SNARKs we get that the resulting proof is short and efficiently verifiable. The enhanced local unforgeability of the resulting construction follows from the extractability of SNARKs and the unforgeability of the underlying (plain) aggregate signature scheme.

We note that the above sketch serves as a proof of concept of the feasibility of locally verifiable aggregate signatures in the *multi-signer* setting. However, a direct construction is more interesting and desirable for several reasons. First, conceptually, SNARKs seem too big of a hammer to construct aggregate signatures. Secondly, in practice, SNARKs have a high concrete performance overhead, while direct constructions based on number theory are much more efficient (this is akin to why number-theoretic accumulators and plain aggregate signatures are used in practice as opposed to Merkle trees and SNARK-based plain aggregate signatures). Finally, SNARKs suffer from impossibility results in the plain model [GW11], and are often constructed in the random oracle model or from knowledge-type assumptions, while locally verifiable aggregate signatures can potentially be built from fully standard assumptions in the plain model. Our single-signer constructions demonstrate this in the static security model; we believe that adaptive security is achievable without random oracles, but leave it as a fascinating open problem. Yet another fascinating open problem is to construct a multi-signer locally verifiable aggregate signature scheme.

3 Preliminaries

Notation. We will let PPT denote probabilistic polynomial-time. We denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. Also, we use $[0, n]$ to denote the set of all non-negative integers up to n, i.e. $[0, n] := \{0\} \cup [n]$. Throughout this paper, unless specified, all polynomials we consider are positive polynomials. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$.

3.1 Pseudorandom Functions

A pseudorandom function (PRF) consists a pair of algorithms Setup and Eval with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^n) \to K$. The setup algorithm takes as input the security parameter λ and input length parameter n, and outputs a PRF key K.

$\mathsf{Eval}(K, x) \to y$. The evaluation algorithm, on input the PRF key K and string $x \in \{0,1\}^n$, outputs a bit string $y \in \{0,1\}^m$. Here $m = m(\lambda)$ denotes the output length of the PRF.

Definition 3.1 (Pseudorandomness). A PRF scheme $\mathsf{PRF} = (\mathsf{Setup}, \mathsf{Eval})$ is said to be secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{Eval}(K,\cdot)}(r_b) = b : \begin{array}{c} K \leftarrow \mathsf{Setup}(1^\lambda, 1^n),\ b \leftarrow \{0,1\} \\ x^* \leftarrow \mathcal{A}^{\mathsf{Eval}(K,\cdot)}(1^\lambda, 1^n) \\ r_0 \leftarrow \{0,1\}^m,\ r_1 = \mathsf{Eval}(K, x^*) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ must not query the challenge input x^* to the evaluation oracle $\mathsf{Eval}(K, \cdot)$.

3.2 Computational Assumptions

3.2.1 RSA Assumption and Shamir's Trick

Below we recall (one of the) standard versions of the RSA assumption [RSA78], RSA assumption with large prime exponents [MRV99], as well as the strong RSA assumption [BP97].

Assumption 3.2 (RSA assumption). Let λ be the security parameter. Let the positive integer N be the product of two $\lambda/2$-bit, distinct odd primes p, q. Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$. Given (N, e) and a random $y \in \mathbb{Z}_N^*$, it is hard to compute x such that $x^e \equiv y \bmod N$.

Assumption 3.3 (Large Exponent RSA assumption). Let λ be the security parameter. Let the positive integer N be the product of two $\lambda/2$-bit, distinct odd primes p, q. Let e be a randomly chosen $(\lambda + 1)$-bit prime. Given (N, e) and a random $y \in \mathbb{Z}_N^*$, it is hard to compute x such that $x^e \equiv y \bmod N$.

Assumption 3.4 (Strong RSA assumption)**.** Let λ be the security parameter. Let the positive integer N be the product of two $\lambda/2$-bit, distinct odd primes p, q. Given N and a random $y \in \mathbb{Z}_N^*$, it is hard to compute (x, e) such that $x^e \equiv y \bmod N$, where $e > 1$ is any positive integer.

We also make use of the following classical lemma due to Shamir whose proof is provided for completeness.

Lemma 3.5 (Shamir's trick [Sha83])**.** Given $x, y \in \mathbb{Z}_N$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b \pmod N$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_N$ such that $z^a = y \pmod N$.

Proof. Let $\alpha, \beta \in \mathbb{Z}$ be integers such that $\alpha a + \beta b = 1$. Then, $z = y^\alpha x^\beta$ is the desired number as $z^a = y^{\alpha a} x^{\beta a} = y^{\alpha a} y^{\beta b} = y^{\alpha a + \beta b} = y \pmod N$. □

3.2.2 (Bilinear) Diffie-Hellman Inversion Assumption

In this work, we will be using bilinear groups. Let Gen be a PPT algorithm that takes as input a security parameter λ (in unary), and outputs a λ-bit prime p, an efficient description of groups $\mathbb{G}, \mathbb{G}_T$ of order p, generator $g \in \mathbb{G}$ and an efficient non-degenerate bilinear mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ (that is, $e(g, g) \neq \mathbf{1}_{\mathbb{G}_T}$, and for all $a, b \in \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$). Here we rely on the well studied family of Diffie-Hellman Inversion problems [MSK02, BB04a, BB04b].

DHI Assumption. The q-DHI assumption states that no PPT adversary, given the sequence $(g, g^a, g^{a^2}, \ldots, g^{a^q})$ can compute $g^{\frac{1}{a}}$. In this experiment, a is chosen uniformly at random from $\mathbb{Z}_p^*$. It is called a q-type assumption because the experiment is parameterized by the length of the powers-in-exponent sequence given to the adversary. Here we are working in groups which have efficient bilinear pairing operation $e(\cdot, \cdot)$.

Assumption 3.6 (q-DHI Assumption)**.** The assumption is parameterized with an integer $q \in \mathbb{Z}$. We say that the q-DHI assumption holds if for all PPT $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, L) = g^{\frac{1}{a}} : \begin{array}{l} \Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda) \\ a \leftarrow \mathbb{Z}_p^*,\ L = (g^a, g^{a^2}, \ldots, g^{a^q}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

BDHI Assumption. In this work, we also consider the DHI assumption in the "target" group as well, which is commonly referred to as the Bilinear Diffie-Hellman Inversion assumption. The difference between the target and source version of the strong DH assumption is that in the target group version, the adversary is said to win the game even if it computes $\frac{1}{a}$ only in the target group, that is $e(g, g)^{\frac{1}{a}}$. Formally, we define both its search and decision versions.

Assumption 3.7 (q-BDHI Assumption)**.** The assumption is parameterized with an integer $q \in \mathbb{Z}$. We say that the q-BDHI assumption holds if for all PPT $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, L) = e(g, g)^{\frac{1}{a}} : \begin{array}{l} \Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda) \\ a \leftarrow \mathbb{Z}_p^*,\ L = (g^a, g^{a^2}, \ldots, g^{a^q}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Assumption 3.8 (Decision q-BDHI Assumption)**.** The assumption is parameterized with an integer $q \in \mathbb{Z}$. We say that the decision q-BDHI assumption holds if for all PPT $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, L, Z_b) = b : \begin{array}{c} \Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda) \\ a \leftarrow \mathbb{Z}_p^*,\ L = (g^a, g^{a^2}, \ldots, g^{a^q}) \\ b \leftarrow \{0,1\}, Z_0 = e(g,g)^{\frac{1}{a}}, Z_1 \leftarrow \mathbb{G}_T \end{array}\right] \leq \mathsf{negl}(\lambda)$$

4 Aggregate Cryptosystems with Local Properties

In this section, we recall the notion of single-signer aggregate signatures, and introduce the concept of local verification for aggregate signatures. Additionally, we also introduce the concept of aggregate identity-based encryption. Later in Appendix A, we also extend our concept of locally verifiable aggregate signatures to the multi-signer setting.

4.1 Aggregate Signatures

The notion of aggregate signatures as introduced by Boneh, Gentry, Lynn and Shacham [BGLS03] is simply a regular signature scheme that comes with two poly-time algorithms Aggregate and AggVerify, where Aggregate is used to aggregate an arbitrary polynomial number of message-signature pairs $\{(m_i, \sigma_i)\}_i$ generated using verification keys $\{\mathsf{vk}_i\}_i$, into a shorter aggregate signature $\widehat{\sigma}$, and AggVerify can be used to verify such aggregate signatures with respect to the sequence of messages $(m_1, \ldots, m_\ell)$ and the verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$.

An aggregate signature scheme is said to be a single-signer aggregate signature scheme if the aggregation algorithm requires all the verification keys $\{\mathsf{vk}_i\}_i$ to be the same. Below we define it formally.

Syntax. A single-signer aggregate signature scheme $\mathcal{S}$ for message space $\mathcal{M}$ consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. The setup algorithm, on input the security parameter λ, outputs a pair of signing and verification keys $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a signing key sk and a message $m \in \mathcal{M}$, and computes a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. The verification algorithm takes as input a verification key vk, a message $m \in \mathcal{M}$, and a signature σ. It outputs a bit to signal whether the signature is valid or not.

$\mathsf{Aggregate}\left(\mathsf{vk}, \{(m_i, \sigma_i)\}_i\right) \to \widehat{\sigma}/\perp$. The signature aggregation algorithm takes as input a verification key vk, a sequence of tuples, each containing a message m_i and signature σ_i, and it outputs either an aggregated signature $\widehat{\sigma}$ or a special abort symbol $\perp$.

$\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i\}_i, \widehat{\sigma}\right) \to 0/1$. The aggregate verify algorithm takes as input a verification key vk, a sequence of messages m_i, and it outputs a bit to signal whether the aggregated signature $\widehat{\sigma}$ is valid or not.

Correctness and Compactness. An aggregate signature scheme is said to be correct and compact if for all $\lambda, \ell \in \mathbb{N}$, every verification-signing key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, messages m_i for $i \in [\ell]$, and every signature $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$ for $i \in [\ell]$, the following holds:

Correctness of signing. For all $i \in [\ell]$, $\mathsf{Verify}(\mathsf{vk}, m_i, \sigma_i) = 1$.

Correctness of aggregation. If $\widehat{\sigma} = \mathsf{Aggregate}\left(\mathsf{vk}, \{(m_i, \sigma_i)\}_i\right)$, then

$$\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i\}_i, \widehat{\sigma}\right) = 1.$$

Compactness of aggregation. $|\widehat{\sigma}| \leq \mathsf{poly}(\lambda)$. That is, the size of an aggregated signature is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

Security. Next, we recall the security notion for regular signatures as well as for the setting of aggregate signatures.

Definition 4.1 (Unforgeability)**.** A signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ is said to be a secure signature scheme if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 : \begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

and $\mathcal{A}$ is admissible as long as it did not query m^* to the Sign oracle.

Definition 4.2 (Static Unforgeability)**.** We say the signature scheme is statically secure if the adversary in the above game is confined to make all of its message queries $\{m_i\}_{i\in[q]}$ and declare the challenge message m^* at the beginning of the game (defined in Definition 4.1) before it receives the verification key vk.

Definition 4.3 (Aggregated Unforgeability)**.** A single-signer aggregate signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ is said to be a secure aggregate signature scheme if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i^*\}_{i\in[\ell]}, \widehat{\sigma}^*\right) = 1 : \begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \left(\{m_i^*\}_{i\in[\ell]}, \widehat{\sigma}^*\right) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if there exists $i \in [\ell]$ such that m_i^* was not queried by $\mathcal{A}$ to the $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle.

Definition 4.4 (Static Aggregated Unforgeability)**.** We say the aggregate signature scheme is statically secure if the adversary in the above game is confined to make all of its message queries $\{m_i\}_{i\in[q]}$ and declare the challenge messages $\{m_i^*\}_{i\in[\ell]}$ at the beginning of the game (defined in Definition 4.3) before it receives the verification key vk.

Our definition of static security is identical to the weak-CMA security for plain signatures as defined by Boneh and Boyen [BB04b]. In addition to the above security properties, there are a number of other interesting properties such as unique signatures, multi-hop aggregation etc that have been considered in the literature. We describe them in detail later in Appendix A.1 when we define aggregate signatures in the multi-signer setting. Our aggregate signature schemes satisfy most of the properties as we discuss later.

4.1.1 Locally Verifiable Aggregate Signatures

In this work, we introduce the notion of local openings for aggregate signatures that enable faster local verification. As described above, in existing aggregate signatures the verification algorithm for an aggregate signature takes as input the entire sequence of messages $(m_1, \ldots, m_\ell)$ aggregated inside signature $\widehat{\sigma}$. Thus, the run-time of verification scales *polynomially* with the number of messages ℓ.

Aggregate signatures with local opening enable efficient verifiability, where the local verification algorithm takes as input only the message m that has to be verified against the claimed aggregated signature $\widehat{\sigma}$, instead of all ℓ messages. However, without any other modifications to the syntax of the aggregate signatures, the notion of local verifiability is impossible to achieve (as discussed in the introduction). In order to make the notion feasible, we introduce an auxiliary local opening generator that generates some auxiliary information specific to the message m being locally verified, and this algorithm does not require any of the input signatures $\{\sigma_i\}_i$ that were aggregated, but the final aggregated signature $\widehat{\sigma}$. Below we define the algorithms formally.

$\mathsf{LocalOpen}(\widehat{\sigma}, \mathsf{vk}, \{m_i\}_{i\in[\ell]}, j \in [\ell]) \to \mathsf{aux}_j$. The local opening algorithm takes as input an aggregated signature $\widehat{\sigma}$, a verification key vk, a sequence of messages m_i for $i \in [\ell]$, and an index $j \in [\ell]$. It outputs auxiliary information aux_j corresponding to the message m_i.

$\mathsf{LocalAggVerify}(\widehat{\sigma}, \mathsf{vk}, m, \mathsf{aux}) \to 0/1$. The local aggregate verification algorithm takes as input an aggregated signature $\widehat{\sigma}$, a verification key vk, a message m, and auxiliary information aux. It outputs a bit to signal whether the aggregate signature $\widehat{\sigma}$ contains a signature for message m under verification key vk, or not.

Correctness and Compactness of Local Opening. An aggregate signature scheme with local openings is said to be correct and compact if for all $\lambda, \ell \in \mathbb{N}$, every verification-signing key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, messages m_i for $i \in [\ell]$, and every signature $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$ for $i \in [\ell]$, the following holds:

Correctness of local opening. For all $k \in [\ell]$, we have

$$\mathsf{LocalAggVerify}\left(\widehat{\sigma}, \mathsf{vk}, m_k, \mathsf{LocalOpen}(\widehat{\sigma}, \mathsf{vk}, \{m_i\}_i, k)\right) = 1.$$

Compactness of opening. $|\mathsf{aux}| \leq \mathsf{poly}(\lambda)$. That is, the size of the auxiliary opening information is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

Security against adversarial openings. Now we define the security notion for aggregate signatures with local openings.

Definition 4.5 (Aggregated Unforgeability with Adversarial Opening)**.** A locally-verifiable aggregate signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify}, \mathsf{LocalOpen}, \mathsf{LocalAggVerify})$ is said to be a secure aggregate signature scheme against adversarial openings if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{LocalAggVerify}(\widehat{\sigma}^*, \mathsf{vk}, m^*, \mathsf{aux}^*) = 1 : \begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\widehat{\sigma}^*, \mathsf{aux}^*, m^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if m^* was not queried by $\mathcal{A}$ to the $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle.

Definition 4.6 (Static Aggregated Unforgeability with Adversarial Opening)**.** We say the locally-verifiable aggregate signature scheme is statically secure against adversarial openings if the adversary in the above game is confined to make all of its message queries $\{m_i\}_{i\in[q]}$ and declare the challenge message m^* at the beginning of the game (defined in Definition 4.5) before it receives the verification key vk.

Fully Public Openings for Aggregate Signatures. We additionally consider the setting where the local opening algorithm does not need an aggregate signature to provide an opening w.r.t., but only the sequence of messages.

Remark 4.7 (Fully Public Openings)**.** An aggregate signature scheme is said to have fully local public openings if the algorithm $\mathsf{LocalOpen}$ has the following syntax — $\mathsf{LocalOpen}(\mathsf{vk}, \{m_i\}_{i\in[\ell]}, j \in [\ell]) \to \mathsf{aux}_j$. That is, $\mathsf{LocalOpen}$ is oblivious to the aggregated signature.

Remark 4.8 (Optimal Compactness and Efficiency)**.** In our definitions, we consider the size of the aggregate signatures, auxiliary opening information, running time of the local verifier to be independent of the number of aggregations. However, one could also consider schemes where the compactness and efficiency of the scheme grows poly-logarithmically with the number of aggregations, as for most applications poly-logarithmically dependence can be asymptotically captured within the polynomial dependence on the security parameter.

4.2 Aggregate Identity-Based Encryption

In this section, we introduce the concept of aggregate identity-based encryption systems. We start by recalling the notion of identity-based encryption, and later describe the concept of secret key aggregation.

IBE Syntax. An Identity-Based Encryption (IBE) scheme IBE for set of identity spaces $\mathcal{I} = \{\{0,1\}^n\}_{n\in\mathbb{N}}$ and message spaces $\mathcal{M}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and identity length n. It outputs the public parameters mpk and the master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master secret key msk and an identity $\mathsf{id} \in \{0,1\}^n$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters mpk, a message $m \in \mathcal{M}$, and an identity $\mathsf{id} \in \{0,1\}^n$. It outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) \to m/\bot$. The decryption algorithm takes as input a secret key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct. It outputs either a message $m \in \mathcal{M}$ or a special symbol $\bot$.

Correctness. We say an IBE scheme is correct if for all $\lambda, n \in \mathbb{N}$, $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, $\mathsf{id} \in \{0,1\}^n$, $m \in \mathcal{M}$, $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$, we have that $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) = m$.

Security. Next, we recall the security notion for regular IBE systems.

Definition 4.9 (Secure IBE)**.** We say an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is secure if for any stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda, n \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \quad b \leftarrow \{0,1\} \\ (m_0, m_1, \mathsf{id}^*) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda, 1^n, \mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where all identities id queried by $\mathcal{A}$ satisfy $\mathsf{id} \neq \mathsf{id}^*$.

Definition 4.10 (Static Secure IBE)**.** We say the IBE scheme is statically secure if the adversary in the above game is confined to make all of its secret key queries $\{\mathsf{id}_i\}_{i \in [q]}$ and declare the challenge identity id^* at the beginning of the game (defined in Definition 4.9) before it receives the master public key mpk.

Aggregating Secret Keys. As mentioned previously, here we consider aggregation of secret keys associated with different identities into a compact aggregated key. Syntactically, this corresponds to introducing the following algorithms.

$\mathsf{KeyAgg}(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell) \to \widehat{\mathsf{sk}}$. The key generation algorithm takes as input a sequence of secret keys $\{\mathsf{sk}_i\}_i$ for $i \in [\ell]$ (for some $\ell > 1$). It outputs an aggregated key $\widehat{\mathsf{sk}}$.

$\mathsf{AggDec}(\widehat{\mathsf{sk}}, (\mathsf{id}_1, \ldots, \mathsf{id}_\ell), \mathsf{ct}, j \in [\ell]) \to m/\bot$. The aggregated decryption algorithm takes as input an aggregated key $\widehat{\mathsf{sk}}$, a list of identities $\mathsf{id}_1, \ldots, \mathsf{id}_\ell$, a ciphertext ct, and index $j \in [\ell]$ which denotes the identity used to compute ct. It outputs either a message $m \in \mathcal{M}$ or a special symbol $\bot$.

Correctness and Compactness. An aggregate IBE scheme is said to be correct and compact if for all $\lambda, \ell \in \mathbb{N}$, master key pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, identities id_i for $i \in [\ell]$, and every secret key $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}_i)$ for $i \in [\ell]$, the following holds:

Correctness of aggregated decryption. For all $i \in [\ell]$, every message $m \leftarrow \mathcal{M}$, ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}_i, m)$, we have that

$$\mathsf{AggDec}\left(\mathsf{KeyAgg}(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell), (\mathsf{id}_1, \ldots, \mathsf{id}_\ell), \mathsf{ct}, i\right) = m.$$

Compactness of aggregation. $|\widehat{\mathsf{sk}}| \leq \mathsf{poly}(\lambda)$. That is, the size of an aggregated key is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

5 RSA-based Locally Verifiable Aggregate Signatures

In this section, we provide a locally verifiable single-signer aggregate signature scheme based on the hardness of RSA. Our scheme satisfies a number of interesting properties, and relies on an efficient deterministic non-colliding prime sequence enumeration.

5.1 Deterministic Prime Sequence Enumeration

Here we are interested in an efficient injective mapping from the message space ($\mathcal{M}_\lambda = \{0,1\}^\lambda$) to the set of $(\lambda+1)$-bit prime numbers. Such injective mappings were constructed by Cachin, Micali, and Stadler [CMS99] by relying on $2\lambda^2$-wise independent hash functions, (randomized) primality testing [SS77, Rab80], and prime density theorems [DlVP97]. The idea is to enumerate over a fixed length ($\approx 2\lambda^2$) sequence of $(\lambda+1)$-bit numbers for each message in the message space, and select the lexicographically first prime number in that sequence (where the sequence is decided by the hash function). Since the hash function is pairwise independent, by relying on prime number density theorems, one gets that with all but negligible probability, such prime numbers for each message exist in the $2\lambda^2$ length sequence.

In this work, we rely on a similar prime sequence enumeration technique, but we slightly adapt it as it leads to different security proofs of our aggregate signature construction. Concretely, we rely on deterministic primality testing [AKS04] to avoid keeping random coins as part of the setup[2], and also replace the hash function with a PRF-based hash function in one instantiation (which results in static security of our signature scheme), and with a Random Oracle [BR93] in the second instantiation (which results in full security of our signature scheme). Additionally, we make the sampling process to be expected polynomial time instead as we consider exponential length sequences for the prime search. As we remark later in Remark B.2, the sampling time could be worst-case polynomial time by relying on well-known prime gap conjectures.

5.1.1 Prime Sequence Enumerator via Pseudorandom Functions

Let $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ be a secure PRF that outputs λ bits of output. Below, we describe our prime sequence enumerator based on PRFs. A (fully secure) prime sequence enumerator

[2] We point out that *we use deterministic primality testing only for the ease of exposition*, and this is not necessary as our scheme is secure even if we rely on efficient randomized primality testing. Such an approach was already outlined in [MRV99] where the idea is to generate a sequence of random coins as part of the setup, and use those random coins to run the randomized primality test deterministically on all those random coins. The proof relies on the fact that, with all but negligible probability over the choice of random coins sampled during setup, randomized primality test will fail on at least one random coins for a non-prime.

in the random oracle model (ROM) is described in Appendix B.

$\mathsf{PrimeSeq}^{\mathsf{PRF}}(1^\lambda) \to \mathsf{samp}$. It samples a PRF key $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{2\lambda})$, and sets $\mathsf{samp} = K$.

$\mathsf{PrimeSamp}^{\mathsf{PRF}}(\mathsf{samp} = K, m) \to e_m$. It proceeds as follows:

1. Set $\mathsf{count} := 0$, $\mathsf{flag} := \mathsf{false}$.
2. While $\mathsf{flag} = \mathsf{false}$:
   (a) Let $y := \mathsf{PRF.Eval}(K, m \,||\, \mathsf{count})$ where $m \,||\, \mathsf{count}$ is interpreted as a 2λ length bit string.
   (b) Run PrimalityTest to check if $2^\lambda + y$ is a prime. If it is a prime, set $\mathsf{flag} := \mathsf{true}$ and $e_m := 2^\lambda + y$. Otherwise, set $\mathsf{count} := \mathsf{count} + 1$.

Output e_m.

Theorem 5.1 (Efficient and Statically Secure Enumeration via PRFs)**.** *If PRF is a secure pseudorandom function as per Definition 3.1, then $(\mathsf{PrimeSeq}^{\mathsf{PRF}}, \mathsf{PrimeSamp}^{\mathsf{PRF}})$ satisfies the following properties:*

Efficient Sampling. For every $\lambda \in \mathbb{N}$, $m \in \{0,1\}^\lambda$, the prime sampling algorithm $\mathsf{PrimeSamp}^{\mathsf{PRF}}$ runs in expected polynomial time, where the probability is taken over the coins of setup algorithm $\mathsf{PrimeSeq}^{\mathsf{PRF}}$.

Statically Secure Non-Colliding Prime Enumeration. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\exists\, i \neq j \in [Q] \text{ s.t. } e_i = e_j \wedge m_i \neq m_j : \begin{array}{c} \{m_i\}_{i \in [Q]} \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{samp} \leftarrow \mathsf{PrimeSeq}^{\mathsf{PRF}}(1^\lambda) \\ \{e_i = \mathsf{PrimeSamp}^{\mathsf{PRF}}(\mathsf{samp}, m_i)\}_i \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Proof. The proof follows from [CMS99] which relied on $2\lambda^2$-wise independent hash function instead of a PRF as we do above. Also, as in [MRV99], we force the enumerator to output truly $(\lambda+1)$-bit primes by fixing the leading bit to be 1 (i.e., adding 2^λ to the randomly sampled number). Now by relying on the pseudorandomness property of the underlying PRFs, we get the desired properties for a sequence of polynomial but "a-priori unbounded" number of messages. Since PRFs are poly-wise independent functions by pseudorandomness for any arbitrary polynomial poly, thus the theorem follows. Note that here the PRF key is being released as part of the public sampling parameters, and despite that fact we are relying on PRF security for security of our samplers. Briefly, this is due to the fact that an attacker in the static non-colliding prime enumeration is required to commit all its messages at the beginning of the game, and the public sampling parameters (i.e., the PRF key) is sampled after the messages are committed by the adversary. Therefore, we do not need to supply the attacker the PRF key, and can simply check whether the non-colliding property failed by querying the PRF oracle. □

5.2 Construction

Below we provide our construction of single-signer aggregate signatures with λ-bit messages.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. The setup algorithm generates an RSA modulus $N = pq$, where p, q are random primes of $\lambda/2$ bits each. Next, it chooses a random element $g \leftarrow \mathbb{Z}_N^*$, and samples the public parameters for prime sequence enumeration as $\mathsf{samp} \leftarrow \mathsf{PrimeSeq}(1^\lambda)$. It sets the key pair as $\mathsf{vk} = (N, \mathsf{samp}, g)$ and $\mathsf{sk} = (p, q, \mathsf{samp}, g)$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. It parses sk as above, and computes the prime number $e_m = \mathsf{PrimeSamp}(\mathsf{samp}, m)$. It computes the signature as $g^{e_m^{-1}} \pmod N$ using p and q from the secret key and computing $e_m^{-1} \pmod{\phi(N)}$.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$. It parses vk as above, and computes the prime number $e_m = \mathsf{PrimeSamp}(\mathsf{samp}, m)$. It checks whether $\sigma^{e_m} \pmod N = g$. If the check succeeds, then it outputs 1 to signal that the signature is valid, otherwise it outputs 0.

$\mathsf{Aggregate}\left(\mathsf{vk}, \{(m_i, \sigma_i)\}_i\right) \to \widehat{\sigma}/\perp$. The signature aggregation algorithm first verifies all the input signatures σ_i, and outputs $\perp$ if any of these verifications fail. Otherwise, it computes the aggregated signature as

$$\widehat{\sigma} = \prod_i \sigma_i \pmod N.$$

$\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i\}_{i \in [\ell]}, \widehat{\sigma}\right)$. The signature verification algorithm parses the verification key as above, and computes the sequence of primes corresponding to the messages as $e_{m_i} = \mathsf{PrimeSamp}(\mathsf{samp}, m_i)$ for all $i \in [\ell]$ where ℓ is the number of aggregated messages. It then checks whether the following is true or not:

$$\widehat{\sigma}^{\prod_i e_{m_i}} = \prod_i g^{\prod_{j \neq i} e_{m_j}} \pmod N.$$

If the check succeeds, then it outputs 1 to signal that the aggregated signature is valid, otherwise it outputs 0.

$\mathsf{LocalOpen}(\widehat{\sigma}, \mathsf{vk}, \{m_i\}_{i \in [\ell]}, j \in [\ell]) \to \mathsf{aux}_j$. It parses vk as above, and computes the sequence of prime numbers corresponding to the messages as $e_{m_i} = \mathsf{PrimeSamp}(\mathsf{samp}, m_i)$ for all $i \in [\ell]$. It then computes the following terms:

$$e_{\mathbf{m} \setminus m_j} = \prod_{i \neq j} e_{m_i}, \quad f_j = \sum_{i \neq j} \prod_{k \neq \{i,j\}} e_{m_k}.$$

Note that since vk contains only N and not $\phi(N)$, thus the algorithm computes the above as large integers without performing any modular reductions. It then computes the following:

$$x = \widehat{\sigma}^{e_{\mathbf{m} \setminus m_j}} / g^{f_j} \pmod N.$$

And, it checks that $\gcd(e_{\mathbf{m} \setminus m_j}, e_{m_j}) = 1$. If the check fails, it outputs $\perp$, otherwise using Shamir's trick (Lemma 3.5), it computes aux_j as

$$\mathsf{aux}_j = \mathsf{Shamir}(x, y = g, a = e_{m_j}, b = e_{\mathbf{m} \setminus m_j}).$$

$\mathsf{LocalAggVerify}(\widehat{\sigma}, \mathsf{vk}, m, \mathsf{aux})$. The local verification algorithm simply runs the unaggregated verification and outputs $\mathsf{Verify}(\mathsf{vk}, m, \sigma = \mathsf{aux})$. That is, it interprets aux as the original signature on m, ignores $\widehat{\sigma}$, and verifies aux as a signature for m.

Basically, the aggregate signature scheme has the special property that the local opening algorithm is able to recover the signature for message under consideration from the aggregated signature, therefore the local opening for a message is simply its signature. Hence, the above local verification algorithm only needs to check that the opening information aux is a valid signature for m, and no extra checks are needed for the aggregated signature $\widehat{\sigma}$.[3]

In addition to the above algorithms, we want to point out that the scheme supports *unordered* sequential signing as well as multi-hop aggregation. Below we describe our sequential signing and verification algorithms:

$\mathsf{SeqAggSign}\left(\mathsf{sk}, m', \{m_i\}_i, \widehat{\sigma}\right) \rightarrow \widehat{\sigma}'$. The sequential signing algorithm first verifies the input aggregated signature $\widehat{\sigma}$, and outputs $\perp$ if the verification fails. Otherwise, it computes the prime $e_{m'}$ as $e_{m'} = \mathsf{PrimeSamp}(\mathsf{samp}, m')$, and computes the new aggregated signature as $\widehat{\sigma}^{e_{m'}^{-1}} \pmod N$ since it knows $\phi(N)$.

$\mathsf{SeqAggVerify}\left(\mathsf{vk}, \{m_i\}_{i\in[\ell]}, \widehat{\sigma}\right)$. The sequential aggregated verification algorithm parses the verification key as above, and computes the sequence of primes corresponding to the messages as $e_{m_i} = \mathsf{PrimeSamp}(\mathsf{samp}, m_i)$ for all $i \in [\ell]$ where ℓ is the number of aggregated messages. It then checks whether the following is true or not:

$$\widehat{\sigma}^{\prod_i e_{m_i}} = g \pmod N.$$

If the check succeeds, then it outputs 1 to signal that the aggregated signature is valid, otherwise it outputs 0.

5.3 Correctness, Compactness, and More Properties

Correctness of signing. This follows directly from the fact that $\mathsf{PrimeSamp}$ is a deterministic prime number sampler, and that $\left(g^{e_m^{-1}}\right)^{e_m} = g \pmod N$ for every m and $e_m = \mathsf{PrimeSamp}(\mathsf{samp}, m)$.

Correctness of Aggregation. Consider any sequence of messages $m_1, \ldots, m_\ell$, and corresponding signatures $\sigma_i = g^{e_{m_i}^{-1}}$ for $i \in [\ell]$ where $e_{m_i} = \mathsf{PrimeSamp}(\mathsf{samp}, m_i)$. We know that aggregating these signatures is done as $\widehat{\sigma} = \prod_i \sigma_i \pmod N$. And, the aggregated verification checks the following:

$$\widehat{\sigma}^{\prod_i e_{m_i}} = \prod_i g^{\prod_{j\neq i} e_{m_j}} \pmod N.$$

[3] We point out that this does not contradict our unforgeability property with adversarial openings. Since, irrespective of whether the adversary is maliciously aggregating signature or generating hints in a malicious way, the adversary is never allowed to make a sign query for the message associated with a forged signature. While it seems like since local verifier is independent of the aggregate signature $\widehat{\sigma}$, thus a verifier might supply any arbitrary string and still pass local verification. The point is in order for the local verification to accept, it must be provided with a valid signature (as a hint), thus an attacker can not forge by supplying only malformed aggregated signatures $\widehat{\sigma}$.

Now to verify that the above check succeeds for honestly computed and aggregated signatures, let us simplify the left side term $\widehat{\sigma}^{\prod_i e_{m_i}}$.

$$\widehat{\sigma}^{\prod_i e_{m_i}} = \left(\prod_j \sigma_j\right)^{\prod_i e_{m_i}} = \left(\prod_j g^{e_{m_j}^{-1}}\right)^{\prod_i e_{m_i}} .$$

Now since we have that $\left(g^{e_{m_j}^{-1}}\right)^{\prod_i e_{m_i}} = g^{\prod_{j \neq i} e_{m_j}} \pmod{N}$, the correctness of aggregated verification follows.

Compactness of Aggregation. The size of an aggregated signature is same as that of an *unaggregated* signature, which simply is a number between 0 and N.

Unique Signatues. Note that the above signature scheme is a unique signature scheme. This follows from the fact that the prime number enumeration samples $(\lambda+1)$-bit primes, and since all factors of $\phi(N)$ are primes less than $\lambda/2$-bits, thus $e_m^{-1} \pmod{\phi(N)}$ is uniquely and well defined. Thus, the inversion operation $g^{e_m^{-1}} \pmod{N}$ is an injective mapping.

Multi-Hop, Unordered and Interleavable Aggregation. We would like to point out that the above construction is a multi-hop aggregate signature scheme as well as the sequential signing and non-sequential aggregation can be arbitrarily interleaved. The multi-hop property follows directly from inspection since the aggregation algorithm is an unordered product of the corresponding signatures. And, since the product operation is independent of the sequence of multiplication, thus the aggregated verification does not depend on the order of aggregation, but only the needs the unordered sequence of aggregated messages. Lastly, we could also interleave the sequential and non-sequential signature aggregation algorithm, and the corresponding verification would need to be appropriately modified and altered.

5.4 Security

Static (Aggregated) Unforgeability. We show that if we instantiate the prime sequence enumeration based on PRFs in our above aggregate signature construction, then the resulting scheme satisfies static unforgeability. Formally, we prove the following.

Theorem 5.2 (Static Unforgeability). If the Strong RSA assumption (Assumption 3.4) holds, and $(\mathsf{PrimeSeq}, \mathsf{PrimeSamp})$ is instantiated based on secure PRFs (as described in Section 5.1), then the aggregate signature scheme described above satisfies static unforgeability, static aggregated unforgeability, and static aggregated unforgeability with adversarial openings (Definitions 4.2, 4.4 and 4.6).

Proof. We would like to point out that if the scheme satisfies aggregated unforgeability, then it also satisfies regular unforgeability. Below we first prove static aggregated unforgeability for our signature scheme.

Aggregated unforgeability. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks aggregated unforgeability with non-negligible probability ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the strong RSA assumption with $\epsilon - \mathsf{negl}$ probability for some negligible function negl. We describe reduction algorithm $\mathcal{B}$ below.

The strong RSA challenger samples an RSA modulus N and a random element $h \in \mathbb{Z}_N^*$, and sends (N, h) to $\mathcal{B}$. Since we are proving only static security, thus the adversary $\mathcal{A}$ also submits all its signature queries $\{m_i\}_{i\in[Q]}$ and the sequence of challenge messages $(m_1^*, \ldots, m_\ell^*)$ at the beginning. Also, since $\mathcal{A}$ is an admissible adversary, thus $\ell \geq 1$ and there exists an index j^* such that $m_{j^*}^* \notin \{m_i\}_{i\in[Q]}$. (Here $\ell = 1$ corresponds to regular unforgeability, whereas $\ell > 1$ corresponds to aggregated unforgeability.)

The reduction $\mathcal{B}$ then samples the public parameters for the PRF-based prime sequence enumeration as $\mathsf{samp} \leftarrow \mathsf{PrimeSeq}^{\mathsf{PRF}}(1^\lambda)$, and it computes the primes associated with all the above messages as $e_{m_i} = \mathsf{PrimeSamp}^{\mathsf{PRF}}(\mathsf{samp}, m_i)$ for all $i \in [Q]$ and $e_{m_i^*} = \mathsf{PrimeSamp}^{\mathsf{PRF}}(\mathsf{samp}, m_i^*)$ for all $i \in [\ell]$. It checks that there does not exist two distinct messages $m \neq m' \in \{m_i\}_i \cup \{m_i^*\}_i$ such that $e_m = e'_m$. In other words, it checks that the sequence of primes are non-colliding. If the check fails (i.e., there is a collision), then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ sets $g = h^{\prod_{i\in[Q]} e_{m_i}} \pmod N$, and sets the verification key vk as $\mathsf{vk} = (N, \mathsf{samp}, g)$. Here j^* is the special index such that $m_{j^*}^* \notin \{m_i\}_{i\in[Q]}$. It also computes the signatures corresponding to each queried message m_j as $\sigma_j = h^{\prod_{i\in[Q]\setminus\{j\}} e_{m_i}} \pmod N$ for $j \in [Q]$. $\mathcal{B}$ then sends vk and $\{\sigma_i\}_i$ to the adversary $\mathcal{A}$. $\mathcal{A}$ then sends it forged signature $\widehat{\sigma}$ (which corresponds to an aggregated signature for the sequence of challenge messages $m_1^*, \ldots, m_\ell^*$). $\mathcal{B}$ then checks if $\widehat{\sigma}$ is a valid accepting signature. It aborts if it is an invalid signature, otherwise it computes the following

$$z = \mathsf{Shamir}\left(x = \widehat{\sigma}^{\prod_{j\in[\ell]\setminus\{j^*\}} e_{m_j^*}}, y = h, a = e_{m_{j^*}^*}, b = \prod_{i\in[Q]} e_{m_i} \sum_{j\in[\ell]} \prod_{k\neq j} e_{m_k^*}\right). \tag{4}$$

Finally, $\mathcal{B}$ outputs $(z, e_{m_{j^*}^*})$ as its strong RSA solution to (N, h).

We claim that if $\mathcal{A}$ wins with probability ϵ, then $\mathcal{B}$ wins with probability $\epsilon - \mathsf{negl}$. First, we claim that if $\mathcal{A}$ is an admissible adversary, then $\mathcal{B}$ aborts with at most negligible probability. This follows from the static non-colliding property of the PRF-based prime sequence enumeration Theorem 5.1. Next, we argue that $z^{e_{m_{j^*}^*}} = h \pmod N$ whenever $\widehat{\sigma}$ is an accepting signature for $(m_1^*, \ldots, m_\ell^*)$ and $\mathcal{A}$ is admissible adversary. Note that if $\mathcal{A}$ is an admissible adversary, then j^* as defined above exists, and we have that

$$\widehat{\sigma}^{\prod_{j\in[\ell]} e_{m_j^*}} = \prod_{j\in[\ell]} g^{\prod_{k\neq j} e_{m_k^*}} \pmod N.$$

Substituting g in the right side, we get

$$\widehat{\sigma}^{\prod_{j\in[\ell]} e_{m_j^*}} = \prod_{j\in[\ell]} y^{\prod_{i\in[Q]} e_{m_i} \prod_{k\neq j} e_{m_k^*}} \pmod N.$$

Rewriting both sides in the above equation gives us

$$\left(\widehat{\sigma}^{\prod_{j\in[\ell]\setminus\{j^*\}} e_{m_j^*}}\right)^{e_{m_{j^*}^*}} = y^{\sum_{j\in[\ell]} \prod_{i\in[Q]} e_{m_i} \prod_{k\neq j} e_{m_k^*}} \pmod N.$$

That is, if $\widehat{\sigma}$ is an accepting signature, then the above equation gives that $x^a = y^b$. Now by Shamir's Lemma (Lemma 3.5), we get that $z^{e_{m^*_{j^*}}} = h \pmod N$ as long as $\gcd(a,b) = 1$. Here x, y, a, b, z are as defined in Eq. (4).

Thus, to complete the proof, we simply need to show that $\gcd(a,b) = 1$ where $a = e_{m^*_{j^*}}, b = \prod_{i\in[Q]} e_{m_i} \sum_{j\in[\ell]} \prod_{k\neq j} e_{m^*_k}$. First, note that a is a prime and $a < b$, thus either we will have $\gcd(a,b) = 1$ or $\gcd(a,b) = a$. Thus, to prove that $\gcd(a,b) = 1$, it is sufficient to show that $b \bmod a \neq 0$. Let us now show this precise statement.

$$\begin{aligned}
b \bmod a &= \prod_{i\in[Q]} e_{m_i} \sum_{j\in[\ell]} \prod_{k\neq j} e_{m^*_k} \bmod a \\
&= \prod_{i\in[Q]} e_{m_i} \left(\prod_{k\neq j^*} e_{m^*_k} + e_{m^*_{j^*}} \sum_{j\in[\ell]\setminus\{j^*\}} \prod_{k\neq j, j^*} e_{m^*_k} \right) \bmod a \\
&= \prod_{i\in[Q]} e_{m_i} \left(\prod_{k\neq j^*} e_{m^*_k} + a \sum_{j\in[\ell]\setminus\{j^*\}} \prod_{k\neq j, j^*} e_{m^*_k} \right) \bmod a \\
&= \prod_{i\in[Q]} e_{m_i} \left(\prod_{k\neq j^*} e_{m^*_k} \right) \bmod a \\
&\neq 0
\end{aligned}$$

The last inequality follows from the fact that e_{m_i} and $e_{m^*_k}$ are non-colliding sequence of primes for distinct messages. This concludes the proof of static aggregated unforgeability.

Aggregated unforgeability with adversarial openings. Next, we show that our signature scheme also satisfies aggregated unforgeability with adversarial openings. This mostly follows from the crucial observation that the local opening of an aggregated signature in our above construction is simply the original (unaggregated) signature for the corresponding message, and since the above signature scheme is a unique signature scheme, thus a pair of accepting aggregated signature $\widehat{\sigma}$ and auxiliary opening information aux for a message m simply contains the unique signature for m which will be aux. Thus, the proof of aggregated unforgeability with adversarial openings follows directly. □

Full (Aggregated) Unforgeability in ROM. Next, we show that if we instantiate the prime sequence enumeration in the ROM, then the above aggregate signature construction satisfies full unforgeability. Formally, we prove the following.

Theorem 5.3 (Full Unforgeability). *If the RSA assumption with large exponents (Assumption 3.3) holds, and* (PrimeSeq, PrimeSamp) *is instantiated in the ROM (as described in Section 5.1), then the aggregate signature scheme described above satisfies (full) unforgeability, aggregated unforgeability, and aggregated unforgeability with adversarial openings (Definitions 4.1, 4.3 and 4.5).*

Proof. We would like to point out that if the scheme satisfies aggregated unforgeability, then it also satisfies regular unforgeability. Below we first prove the full aggregated unforgeability for our signature scheme.

Aggregated unforgeability. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks full/adaptive aggregated unforgeability with non-negligible probability ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the RSA assumption with $\epsilon/Q^{\mathsf{RO}} - \mathsf{negl}$ probability for some negligible function negl, where Q^{RO} is the number of queries $\mathcal{A}$ makes to the random oracle.[4] We describe reduction algorithm $\mathcal{B}$ below.

The RSA challenger samples an RSA modulus N, a large $(\lambda+1)$-bit prime e, a random element $h \in \mathbb{Z}_N^*$, and sends (N, e, h) to $\mathcal{B}$. The reduction algorithm $\mathcal{B}$ is constructed analogous to that in the proof of Theorem 5.2, except now $\mathcal{B}$ uses the power of programmability. $\mathcal{B}$ guesses the index of the challenge message $m_{j^*}^*$ amongst all the RO queries, and programs the prime associated with it $e_{m_{j^*}^*}$ as the RSA challenge prime e. It also samples the remaining primes for all queried messages at the beginning of the game, and programs them for each queried message at the time the message is first queried. The entire reduction algorithm is defined as before, and the analysis remains identical except the reduction algorithm also aborts if guesses the index of the challenge message $m_{j^*}^*$ incorrectly. Since $\mathcal{B}$'s guess is correct with probability at least $1/Q^{\mathsf{RO}}$, thus $\mathcal{B}$'s final advantage will be $\epsilon/Q^{\mathsf{RO}} - \mathsf{negl}$. This concludes the proof.

Aggregated unforgeability with adversarial openings. As before, the proof of adaptive aggregated unforgeability with adversarial openings follows from the unique signature property and the adaptive (unaggregated) unforgeability of the signature scheme. □

6 Pairing-based Locally Verifiable Aggregate Signatures

In this section, we provide a locally verifiable single-signer aggregate signature scheme with *fully public* local openings based on the hardness of Diffie-Hellman Inversion problem. Our scheme satisfies a number of interesting properties that we discuss later, however it supports only bounded single-hop aggregation.

Injective Message Hashing. Similar to our RSA based construction, we are interested in an injective mapping from the message space ($\mathcal{M}_\lambda = \{0,1\}^\lambda$) to the prime field $\mathbb{Z}_p$ for $p > 2^\lambda$. We consider two simple such mappings $(\mathsf{HGen}, \mathsf{H})$ that lead to static and full adaptive security for our final construction respectively.

Identity Map. The hash setup $\mathsf{HGen}^{\mathcal{I}}$ is simply the empty algorithm that outputs $\mathsf{hk} = \epsilon$, and $\mathsf{H}^{\mathcal{I}}(\epsilon, m) = m$ where output m is interpreted as a field element of $\mathbb{Z}_p$.

RO Map. Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_\lambda$ be a family of hash functions where each $h \in \mathcal{H}_\lambda$ takes λ bits as input, and outputs λ-bits of output. The hash setup $\mathsf{HGen}^{\mathcal{H}}$ simply samples a hash function $h \in \mathcal{H}_\lambda$ and outputs $\mathsf{hk} = h$, and $\mathsf{H}^{\mathcal{H}}(\mathsf{hk} = h, m) = h(m)$ where output $h(m)$ is interpreted as a field element of $\mathbb{Z}_p$. Clearly, if h is modeled as a random oracle, then so is the resulting mapping.

[4] For simplicity, here we asssume that the adversary queries all its challenge messages to the random oracle. This can be assumed without loss of generality, as these could be added on top of an already admissible adversary.

Aggregating Inverse Exponents. Our aggregate signature scheme relies on the "key accumulation" algorithm of Delerablée, Paillier, and Pointcheval [DPP07, DP08]. We refer to the algorithm as the DPP algorithm which takes as input a sequence of group elements $\{g^{\frac{r}{\gamma+x_i}}, x_i\}_{i\in[\ell]}$, and outputs $g^{\frac{r}{\prod_{i\in[\ell]}(\gamma+x_i)}}$. However, as discussed in the introduction, we can rely on the alternate and more efficient Lagrange's inverse polynomial interpolation technique for a simpler aggregation algorithm. The idea behind our more efficient accumulation algorithm is as follows. By Lagrange's polynomial interpolation formula we know that for a degree-ℓ polynomial passing through points (x_i, y_i) for $i \in [\ell]$, the corresponding polynomial $p(x)$ can be written as follows

$$p(x) = \sum_{j\in[\ell]} y_j L_j(x), \qquad \text{where } L_j(x) = \frac{\prod_{i\neq j}(x - x_i)}{\prod_{i\neq j}(x_j - x_i)}.$$

Now if we set $y_i = 1$ for all $i \in [\ell]$. That is, $p(x_i) = 1$ for all i. Then, by inspection, we know that $p(x) = \prod_i (x - x_i) + 1$ is an identity. Thus, by using the above Lagrange's polynomial interpolation equation, we get that

$$\prod_i (x - x_i) + 1 = \sum_{j\in[\ell]} \frac{\prod_{i\neq j}(x - x_i)}{\prod_{i\neq j}(x_j - x_i)}.$$

Dividing both sides by $\prod_i (x - x_i)$ we get that

$$1 + \frac{1}{\prod_i (x - x_i)} = \sum_{i\in[\ell]} \frac{\Delta_i}{x - x_i}, \qquad \text{where } \Delta_i = \frac{1}{\prod_{j\neq i}(x_i - x_j)}.$$

Since Δ_i can be publicly computed given the list $x_1, \ldots, x_\ell$, thus the aggregation algorithm for group elements follows.

6.1 Construction

Below we provide our construction for single-signer aggregate signatures with λ-bit messages. Since we are in the single-signer setting, thus we no longer need to introduce the CRS algorithm as part of its description.

$\mathsf{Setup}(1^\lambda, 1^B) \to (\mathsf{vk}^{(\mathsf{local})}, \mathsf{vk}, \mathsf{sk})$. The setup algorithm takes as input the security parameter, λ, as well as the upper bound on number of aggregations, B. It samples the bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda)$, and samples a random exponent $\alpha \leftarrow \mathbb{Z}_p^*$. It also samples the public parameters for message hashing as $\mathsf{hk} \leftarrow \mathsf{HGen}(1^\lambda)$. It sets the key pair as $\mathsf{vk} = (\Pi, \mathsf{hk}, \{g^{\alpha^i}\}_{i\in[B]})$ and $\mathsf{sk} = (\Pi, \mathsf{hk}, \alpha)$. It also sets the local verification key $\mathsf{vk}^{(\mathsf{local})}$ as $\mathsf{vk}^{(\mathsf{local})} = (\Pi, \mathsf{hk}, g^\alpha)$.

NOTE. We would like to point out that the setup algorithm for aggregate signatures typically outputs only a verification-signing key pair. However, here also introduce a *local* verification key that is entirely contained inside the full verification key, but it serves as a shorter key for the local verification algorithm to use. Simply put, here we consider bounded aggregate signatures with local verification, and to make the notion of local verification interesting in the bounded aggregation setting, we introduce a local verification key whose size is independent of the aggregation bound B thereby enabling the local verification algorithm to be independent of the number of aggregations. One could have instead defined local verification algorithm to have RAM access over the full verification key, and require the worst case run-time of the local verification to not grow with the number of underlying aggregations whenever the local verification is modeled as a RAM.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. It parses sk as above, and hashes the message as $h_m = \mathsf{H}(\mathsf{hk}, m)$. It computes the signature as $g^{(\alpha+h_m)^{-1}}$ which can be computed efficiently since it knows α.[5]

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$. It parses vk as above, and computes the message hash as $h_m = \mathsf{H}(\mathsf{hk}, m)$. It checks whether $e(\sigma, g^{\alpha} g^{h_m}) = e(g, g)$ where g^{α} is taken from the verification key vk.[6] If the check succeeds, then it outputs 1 to signal that the signature is valid, otherwise it outputs 0.

$\mathsf{Aggregate}\,(\mathsf{vk}, \{(m_i, \sigma_i)\}_i) \to \widehat{\sigma}/\bot$. The signature aggregation algorithm first verifies all the input signatures σ_i, and outputs $\bot$ if any of these verifications fail. Otherwise, it computes the aggregated signature as

$$\widehat{\sigma} = \mathsf{DPP}(\{\sigma_i, x_i\}_i),$$

where $x_i = \mathsf{H}(\mathsf{hk}, m_i)$.

$\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i\}_{i\in[\ell]}, \widehat{\sigma}\right)$. The signature verification algorithm parses the verification key as above, and computes the sequence of hashed messages as $x_i = \mathsf{H}(\mathsf{hk}, m_i)$ for all $i \in [\ell]$ where ℓ is the number of aggregated messages. It then computes the following polynomial P symbolically to obtain the coefficients $\{\beta_i \in \mathbb{Z}_p\}_{i\in[\ell]}$:

$$P_{\{x_i\}_{i\in[\ell]}}(y) = \prod_{i\in[\ell]} (y + x_i) = \sum_{i=0}^{\ell} \beta_i y^i \pmod{p}. \tag{5}$$

It then checks that $\ell \leq B$ and whether the following is true or not:

$$e(\widehat{\sigma}, \prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i}) = e(g, g),$$

where g^{α^i} are taken from the verification key vk. If the check succeeds, then it outputs 1 to signal that the aggregated signature is valid, otherwise it outputs 0.

$\mathsf{LocalOpen}(\mathsf{vk}, \{m_i\}_{i\in[\ell]}, j \in [\ell]) \to \mathsf{aux}_j$. It parses vk as above, computes the sequence of hash messages as $x_i = \mathsf{H}(\mathsf{hk}, m_i)$ for all $i \in [\ell] \setminus \{j\}$, and computes the coefficients $\{\widetilde{\beta}_i \in \mathbb{Z}_p\}_{i\in[\ell-1]}$, similar to that in Eq. (5) except it removes $(y + x_j)$ from the list of monomials. Concretely, it computes

$$P_{\{x_i\}_{i\in[\ell]\setminus\{j\}}}(y) = \prod_{i\in[\ell]\setminus\{j\}} (y + x_i) = \sum_{i=0}^{\ell-1} \widetilde{\beta}_i y^i \pmod{p}. \tag{6}$$

[5] For simplicity, we ignore the possibility that $\alpha + h_m = 0$ as that could be easily handled as a special case by outputting the identity group element, but keeping it as part of the scheme description makes it cumbersome.

[6] Note that the verification algorithm does not the entire verification key, but the local portion of verification key would be sufficient.

It then outputs the auxiliary opening information $\mathsf{aux}_j = (\mathsf{aux}_{j,1}, \mathsf{aux}_{j,2})$ where $\mathsf{aux}_{j,1}, \mathsf{aux}_{j,2}$ are computed as

$$\mathsf{aux}_{j,1} = \prod_{i=0}^{\ell-1} (g^{\alpha^i})^{\widetilde{\beta}_i}, \qquad \mathsf{aux}_{j,2} = \prod_{i=0}^{\ell-1} (g^{\alpha^{i+1}})^{\widetilde{\beta}_i},$$

where g^{α^i} are taken from the verification key vk.

$\mathsf{LocalAggVerify}(\widehat{\sigma}, \mathsf{vk}^{(\mathsf{local})}, m, \mathsf{aux})$. The local verification algorithm parses the local verification key $\mathsf{vk}^{(\mathsf{local})}$ as above, and auxiliary opening $\mathsf{aux} = (\mathsf{aux}_1, \mathsf{aux}_2)$, and computes the message hash as $h_m = \mathsf{H}(\mathsf{hk}, m)$. It checks the following two conditions:

$$e(\widehat{\sigma}, {\mathsf{aux}_1}^{h_m} \mathsf{aux}_2) = e(g, g)$$
$$e(g^\alpha, \mathsf{aux}_1) = e(g, \mathsf{aux}_2)$$

where g^α is taken from the local verification key vk. If both the check succeed, then it outputs 1 to signal that the signature is valid, otherwise it outputs 0.

NOTE. As we pointed out before, instead of defining the local verification key, we could provide the local verifier RAM access to the full verification key, and since it only needs to extract g^α from the full verification key, thus the verification will be efficient even with that formalization.

In addition to the above algorithms, we want to point out that the scheme supports *unordered* sequential signing on top of single-hop aggregation. Below we describe our sequential signing and verification algorithms:

$\mathsf{SeqAggSign}\left(\mathsf{sk}, m', \{m_i\}_i, \widehat{\sigma}\right) \rightarrow \widehat{\sigma}'$. The sequential signing algorithm first verifies the input aggregated signature $\widehat{\sigma}$, and outputs $\perp$ if the verification fails. Otherwise, it hashes the message as $h_{m'} = \mathsf{H}(\mathsf{hk}, m')$, and computes the new aggregated signature as $\widehat{\sigma}^{(\alpha + h_{m'})^{-1}}$ since it knows α.

$\mathsf{SeqAggVerify}\left(\mathsf{vk}, \{m_i\}_{i \in [\ell]}, \widehat{\sigma}\right)$. The sequential verification algorithm simply runs the (non-sequential) aggregated verification and outputs $\mathsf{AggVerify}(\mathsf{vk}, \{m_i\}_i, \widehat{\sigma})$. That is, it interprets $\widehat{\sigma}$ as a non-sequential aggregated signature on $\{m_i\}_{i \in [\ell]}$, and verifies $\widehat{\sigma}$.

6.2 Correctness, Compactness, and More Properties

Correctness of signing. This follows directly from the fact that $e(g^{(\alpha + h_m)^{-1}}, g^\alpha g^{h_m}) = e(g, g)$ where $h_m = \mathsf{H}(\mathsf{hk}, m)$.

Correctness of Aggregation. Consider any sequence of messages $m_1, \ldots, m_\ell$, and corresponding signatures $\sigma_i = g^{(\alpha + h_{m_i})^{-1}}$ for $i \in [\ell]$ where $h_{m_i} = \mathsf{H}(\mathsf{hk}, m_i)$. We know that aggregating these signatures is done as $\widehat{\sigma} = \mathsf{DPP}(\{\sigma_i, h_{m_i}\}_i)$. Now by the correctness of the key accumulation algorithm of [DPP07, DP08], we have that $\widehat{\sigma} = g^{\prod_i (\alpha + h_{m_i})^{-1}}$. And, the aggregated verification checks the following:

$$e(\widehat{\sigma}, \prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i}) = e(g, g),$$

where β_i's are such that $\sum_{i=0}^{\ell} \beta_i y^i = \prod_{i \in [\ell]} (y + h_{m_i}) \pmod{p}$. Thus, we have that

$$\prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i} = g^{\sum_{i=0}^{\ell} \alpha^i \beta_i} = g^{\prod_{i \in [\ell]} (\alpha + h_{m_i})}.$$

Therefore, for honestly computed and aggregated signatures, the above check succeeds and correctness follows.

Compactness of Aggregation. The size of an aggregated signature is same as that of an *unaggregated* signature, which simply is a source group element (i.e., $\widehat{\sigma} \in \mathbb{G}$).

Unique Signatues. Note that the above signature scheme is a unique signature scheme. This follows from the fact that the mesage hashing is a determinstic function, and if $e(\sigma, g^{\alpha} g^{h_m}) = e(g, g)$, then it must be that $\sigma = g^{(\alpha + h_m)^{-1}}$ which can be uniquely computed since $\mathbb{G}$ is a prime order source group.

Single-Hop, Unordered Sequential Aggregation with Fully Public Local Openings. We would like to point out that the above construction is a single-hop aggregate signature scheme. And, since the product operation is independent of the sequence of multiplication, thus the aggregated verification does not depend on the order of aggregation, but only the needs the unordered sequence of aggregated messages. Here the sequential signing can be performed arbitrarily on top of an aggregated signature.

Lastly, an interesting feature of these signatures is that they provide fully public local openings, and the $\mathsf{LocalOpen}$ algorithm does not need an aggregated signature as an extra input.

6.3 Security

Static (Aggregated) Unforgeability. We show that if we instantiate the message hashing as the identity map in our above aggregate signature construction, then the resulting scheme satisfies static unforgeability. Formally, we prove the following.

Theorem 6.1 (Static Unforgeability)**.** If the DHI assumption (Assumption 3.6) holds, and $(\mathsf{HGen}, \mathsf{H})$ is an identity hash, then the aggregate signature scheme described above satisfies static unforgeability, and static aggregated unforgeability (Definitions 4.2 and 4.4).

Also, if the BDHI assumption (Assumption 3.7) holds, and $(\mathsf{HGen}, \mathsf{H})$ is an identity hash, then the aggregate signature scheme described above also satisfies static aggregated unforgeability with adversarial openings (Definition 4.6).

Proof. We would like to point out that the proof of aggregated unforgeability is more general than regular unforgeability, thus we first only prove static aggregated unforgeability for our signature scheme. Later, we prove unforgeability in presence of adversarial openings as well.

Aggregated unforgeability. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks aggregated unforgeability with non-negligible probability ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the q-DHI assumption with $\epsilon - \mathsf{negl}$ probability for some negligible function negl. We describe reduction algorithm $\mathcal{B}$ below.

Since we are proving only static security, thus the adversary $\mathcal{A}$ submits all its signature queries $\{m_i\}_{i \in [q_s]}$ and the sequence of challenge messages $(m_1^*, \ldots, m_\ell^*)$ at the beginning to the reduction algorithm $\mathcal{B}$. Since $\mathcal{A}$ is an admissible adversary, thus $\ell \geq 1$ and there exists an index j^* such that $m_{j^*}^* \notin \{m_i\}_{i \in [q_s]}$. (Here $\ell = 1$ corresponds to regular unforgeability, whereas $\ell > 1$ corresponds to aggregated unforgeability.)

Let $j^* \in [\ell]$ be the smallest index with aforementioned property, and B be the aggregation bound. The reduction algorithm then breaks the q-SDH assumption, where the hardness parameter q is such that $q \geq q_s + \ell + B - 1$. The DHI challenger samples the bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, h, e(\cdot, \cdot))$, and a sequence of group elements $\{h_i = h^{a^i}\}_{i=0}^{q}$ for a randomly chosen exponent $a \in \mathbb{Z}_p^*$, and sends $(\Pi, h_1, \ldots, h_q)$ to $\mathcal{B}$.

Let $\mathcal{S}$ denote the set of messages $\mathcal{S} := \{m_i\}_{i \in [q_s]} \cup \{m_j^*\}_{j \in [\ell] \setminus \{j^*\}}$, that is $\mathcal{S}$ contains only the distinct messages in the queried set and challenge set (after taking out $m_{j^*}^*$). We know that $|\mathcal{S}| \leq q_s + \ell - 1$. The reduction $\mathcal{B}$ then implicitly sets the signing key $\alpha = a - m_{j^*}^*$, and the base group element g to be $h_0^{\prod_{i=1}^{q_s}(\alpha + m_i)} = h_0^{\prod_{i=1}^{q_s}(a + m_i - m_{j^*}^*)}$. This also implies that the remaining group elements in the verification key $\{g_i = g^{\alpha^i}\}_{i \in [B]}$ are set as $g_i = h_0^{(a - m_{j^*}^*)^i \prod_{i=0}^{q_s}(a + m_i - m_{j^*}^*)}$. For ease of notation we refer to generator g as g_0 througout the proof.

Now in order to compute these elements in the verification key, $\mathcal{B}$ first computes the following polynomials P_j for $j \in [0, B]$ to obtain the coefficients $\{\beta_j^{(i)} \in \mathbb{Z}_p\}_{i=0}^{|\mathcal{S}|+j}$:

$$\forall j \in [0, B], \quad P_j(X) = (X - m_{j^*}^*)^j \prod_{m \in \mathcal{S}} (X + m - m_{j^*}^*) = \sum_{i=0}^{|\mathcal{S}|+j} \beta_j^{(i)} X^i \pmod{p}. \tag{7}$$

It samples a random exponent $\delta \in \mathbb{Z}_p^*$, and computes the group elements g_j to be included in the verification key as

$$\forall j \in [0, B], \quad g_j = h^{\delta P_j(a)} = h^{\delta \sum_{i=0}^{|\mathcal{S}|+j} \beta_j^{(i)} a^i} = \prod_{i=0}^{|\mathcal{S}|+j} h^{\delta \beta_j^{(i)} a^i} = \prod_{i=0}^{|\mathcal{S}|+j} h_i^{\delta \beta_j^{(i)}}. \tag{8}$$

It sets the verification keys vk and $\mathsf{vk}^{(\mathsf{local})}$ as $\mathsf{vk} = ((p, \mathbb{G}, \mathbb{G}_T, g_0, e(\cdot, \cdot)), \mathsf{hk} = \epsilon, \{g_i\}_{i=1}^{B})$, and $\mathsf{vk}^{(\mathsf{local})} = ((p, \mathbb{G}, \mathbb{G}_T, g_0, e(\cdot, \cdot)), \mathsf{hk} = \epsilon, g_1)$. That is, note that g_0 is not the same generator that the DHI challenger provided, but computed differently as described above.

Let Q_{-i} be the following polynomial with coefficients $\{\gamma_i^{(j)} \in \mathbb{Z}_p\}_{j=0}^{|\mathcal{S}|-1}$:

$$\forall i \in [q_s], \quad Q_{-i}(X) = \prod_{m \in \mathcal{S} \setminus \{m_i\}} (X + m - m_{j^*}^*) = \sum_{j=0}^{|\mathcal{S}|-1} \gamma_i^{(j)} X^j \pmod{p},$$

which is simply the polynomial P_0 (described in Eq. (7)) but with the monomial $(X + m_i - m_{j^*}^*)$

removed. $\mathcal{B}$ then computes the signatures for messages $m_1, \ldots, m_{q_s}$ as

$$\forall i \in [q_s], \quad \sigma_i = g^{(a+m_i-m^*_{j^*})^{-1}} = h_0^{\delta \frac{P_0(a)}{a+m_i-m^*_{j^*}}} = h_0^{\delta Q_{-i}(a)} = \prod_{j=0}^{|\mathcal{S}|-1} h_j^{\delta \gamma_i^{(j)}}. \tag{9}$$

$\mathcal{B}$ then sends $\mathsf{vk}, \mathsf{vk}^{(\mathsf{local})}$ and $\{\sigma_i\}_i$ to the adversary $\mathcal{A}$. $\mathcal{A}$ then sends its forged signature $\widehat{\sigma}$ (which corresponds to an aggregated signature for the sequence of challenge messages $m^*_1, \ldots, m^*_\ell$). $\mathcal{B}$ then checks if $\widehat{\sigma}$ is a valid accepting signature. It aborts if it is an invalid signature, otherwise it computes the following polynomial symbolically

$$P^*(X) = \frac{P_0(X)}{\prod_{j \in [\ell]}(X + m^*_j - m^*_{j^*})} = \frac{\prod_{m \in \mathcal{S} \setminus \{m^*_j\}_j}(X + m - m^*_{j^*})}{X} = \frac{\beta^*_{-1}}{X} + \sum_{i=0}^{|\mathcal{S}|-\ell} \beta^*_i X^i \pmod{p}, \tag{10}$$

and obtains coefficients $\{\beta^*_i\}_{i=-1}^{|\mathcal{S}|-\ell}$. Finally, it computes

$$Z = \left(\widehat{\sigma}^{1/\delta} \prod_{i=0}^{|\mathcal{S}|-\ell} h_i^{-\beta^*_i} \right)^{1/\beta^*_{-1}},$$

and outputs Z as its DHI solution.

We claim that if $\mathcal{A}$ wins with probability ϵ, then $\mathcal{B}$ wins with probability $\epsilon - \mathsf{negl}$. First, observe that if $\mathcal{A}$ is an admissible adversary, then $\mathcal{B}$ aborts with at most negligible probability. Next, we show that $Z^a = h_0$ whenever $\widehat{\sigma}$ is an accepting signature for $(m^*_1, \ldots, m^*_\ell)$ and $\mathcal{A}$ is an admissible adversary. This is because if $\mathcal{A}$ is an admissible adversary, then j^* as defined above exists, and we have that

$$e(\widehat{\sigma}, \prod_{j=0}^{\ell}(g_j)^{\widetilde{\beta}_j}) = e(g_0, g_0), \tag{11}$$

where $\{\widetilde{\beta}_j\}_{j=0}^{\ell}$ is such that $\prod_{j \in [\ell]}(X + m^*_j) = \sum_{j=0}^{\ell} \widetilde{\beta}_j X^j \pmod{p}$. Now, rewriting $\prod_{j=0}^{\ell}(g_j)^{\widetilde{\beta}_j}$ gives us that

$$\prod_{j=0}^{\ell}(g_j)^{\widetilde{\beta}_j} = g_0^{\sum_{j=0}^{\ell} \widetilde{\beta}_j \alpha^j} = g_0^{\prod_{j \in [\ell]}(\alpha + m^*_j)} = g_0^{\prod_{j \in [\ell]}(a + m^*_j - m^*_{j^*})}.$$

This combined with Eq. (11), we get that it must be the case that $\widehat{\sigma} = g_0^{\prod_{j \in [\ell]}(a+m^*_j-m^*_{j^*})^{-1}}$. And, since we have that $g_0 = h_0^{\delta \prod_{m \in \mathcal{S}}(a+m)}$, we get that the following must be true

$$\widehat{\sigma} = h_0^{\delta \prod_{m \in \mathcal{S}}(a+m-m^*_{j^*}) \prod_{j \in [\ell]}(a+m^*_j-m^*_{j^*})^{-1}} = h_0^{\delta P^*(a)}.$$

The last equality follows from the definition of P^* (see Eq. (10)) the fact that $m^*_{j^*} \notin \mathcal{S}$. And, we also have that $\beta^*_{-1} \neq 0$ as otherwise $m^*_{j^*} \in \mathcal{S}$, thus rewriting the above equation gives us that

$$\widehat{\sigma}^{1/\delta} = h_0^{P^*(a)} = h_0^{\frac{\beta^*_{-1}}{a} + \sum_{i=0}^{|\mathcal{S}|-\ell} \beta^*_i a^i} = h_0^{\frac{\beta^*_{-1}}{a}} \cdot \prod_{i=0}^{|\mathcal{S}|-\ell} h_i^{\beta^*_i}.$$

Rewriting this further gives us

$$\widehat{\sigma}^{1/\delta} \prod_{i=0}^{|\mathcal{S}|-\ell} h_i^{-\beta_i^*} = h^{\frac{\beta^*_{-1}}{a}}$$

$$\Rightarrow \left(\widehat{\sigma}^{1/\delta} \prod_{i=0}^{|\mathcal{S}|-\ell} h_i^{-\beta_i^*} \right)^{1/\beta^*_{-1}} = h^{\frac{1}{a}}.$$

Here the left side term is exactly how Z is computed, thus Z is a valid solution to the DHI problem. This concludes the proof of static aggregated unforgeability.

Aggregated unforgeability with adversarial openings. Next, we show that our signature scheme also satisfies aggregated unforgeability with adversarial openings. As the theorem says, this relies on the hardness of BDHI. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks aggregated unforgeability with adversarial openings with non-negligible probability ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the q-BDHI assumption in the target group with $\epsilon - \mathsf{negl}$ probability for some negligible function negl. We describe reduction algorithm $\mathcal{B}$ below.

Since we are proving only static security, thus the adversary $\mathcal{A}$ submits all its signature queries $\mathcal{S} := \{m_i\}_{i \in [q_s]}$ and the challenge message m^* at the beginning to the reduction algorithm $\mathcal{B}$. Since $\mathcal{A}$ is an admissible adversary, $m^* \notin \mathcal{S}$. Let B be the aggregation bound. The reduction algorithm then breaks the q-BDHI assumption in the target group, where the hardness parameter q is such that $q \geq q_s + B$. The BDHI challenger samples the bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, h, e(\cdot, \cdot))$, and a sequence of group elements $\{h_i = h^{a^i}\}_{i=0}^{q}$ for a randomly chosen exponent $a \in \mathbb{Z}_p^*$, and sends $(\Pi, h_1, \ldots, h_q)$ to $\mathcal{B}$. The reduction $\mathcal{B}$ then computes the polynomials P_j (for set $\mathcal{S}$) symbolically to obtain the coefficients $\{\beta_j^{(i)} \in \mathbb{Z}_p\}_{i=0}^{|\mathcal{S}|}$ corresponding to set $\mathcal{S}$ for $j \in [0, B]$ as in Eq. (7). It samples a random exponent $\delta \in \mathbb{Z}_p^*$, and computes the group element $g = g_0$ and $\{g_j\}_{j \in [B]}$ to be included in the verification key as in Eq. (8). It sets the verification keys vk and $\mathsf{vk}^{(\mathsf{local})}$, and computes the signatures for messages $m_i \in \mathcal{S}$ as in the reduction of aggregated unforgeability above (see Eq. (9)).

$\mathcal{B}$ then sends $\mathsf{vk}, \mathsf{vk}^{(\mathsf{local})}$ and $\{\sigma_i\}_i$ to the adversary $\mathcal{A}$. $\mathcal{A}$ then sends its forged signature $\widehat{\sigma}$ along with a local opening $\mathsf{aux} = (\mathsf{aux}_1, \mathsf{aux}_2)$ (which corresponds to an aggregated signature with local opening for the challenge message m^*). $\mathcal{B}$ then checks if $\widehat{\sigma}$ is a valid accepting signature by running the local verification algorithm. It aborts if it is an invalid signature, otherwise it compute the polynomial P^* to compute the coefficients $\{\beta_i^*\}_{i=-1}^{2\mathcal{S}-1}$ as

$$P^*(X) = \frac{P_0(X) \cdot P_0(X)}{X} = \frac{\beta^*_{-1}}{X} + \sum_{i=0}^{2|\mathcal{S}|-1} \beta_i^* X^i \pmod{p},$$

and it outputs Z as its BDHI solution where $Z \in \mathbb{G}_T$ is computed as

$$Z = \left(e(\widehat{\sigma}, \mathsf{aux}_1)^{1/\delta^2} \prod_{i=0}^{2|\mathcal{S}|-1} e(h_{\lceil i/2 \rceil}, h_{\lfloor i/2 \rfloor})^{-\beta_i^*} \right)^{1/\beta^*_{-1}},$$

We claim that if $\mathcal{A}$ wins with probability ϵ, then $\mathcal{B}$ wins with probability $\epsilon - \mathsf{negl}$. First, observe that if $\mathcal{A}$ is an admissible adversary, then $\mathcal{B}$ aborts with at most negligible probability. Next, we claim that $e(\widehat{\sigma}, \mathsf{aux}_1)^{\alpha+m^*} = e(\widehat{\sigma}, \mathsf{aux}_1)^a = e(g_0, g_0) = e(g, g)$ whenever $\widehat{\sigma}$ is an accepting signature for m^* with opening aux and $\mathcal{A}$ is admissible adversary. This is because if the signature is valid, then have that

$$e(\widehat{\sigma}, {\mathsf{aux}_1}^{m^*}\mathsf{aux}_2) = e(g_0, g_0), \text{ and } e(g_1, \mathsf{aux}_1) = e(g_0, \mathsf{aux}_2).$$

First, from the fact $e(g_1, \mathsf{aux}_1) = e(g_0, \mathsf{aux}_2)$ and $g_1 = g_0^{\alpha} = g_0^{a-m^*}$ we get that it must be the case that $\mathsf{aux}_1^{a-m^*} = \mathsf{aux}_2$. Now this gives that $e(\widehat{\sigma}, \mathsf{aux}_1)^{a-m^*} = e(\widehat{\sigma}, \mathsf{aux}_2)$, and combining this with $e(\widehat{\sigma}, {\mathsf{aux}_1}^{m^*}\mathsf{aux}_2) = e(g_0, g_0)$, we obtain $e(\widehat{\sigma}, \mathsf{aux}_1)^a = e(g_0, g_0)$. This gives us that

$$\begin{aligned}
e(\widehat{\sigma}, \mathsf{aux}_1) &= e(h_0^{\delta P_0(a)}, h_0^{\delta P_0(a)})^{1/a} \\
&= e(h_0, h_0)^{\frac{(\delta P_0(a))^2}{a}} \\
&= e(h_0, h_0)^{\delta^2 \left(\frac{\beta^*_{-1}}{a} + \sum_{i=0}^{2|\mathcal{S}|-1} \beta^*_i a^i \right)} \\
&= e(h_0, h_0)^{\delta^2 \left(\frac{\beta^*_{-1}}{a} \right)} \prod_{i=0}^{2|\mathcal{S}|-1} e(h_0^{a^{\lceil i/2 \rceil}}, h_0^{a^{\lfloor i/2 \rfloor}})^{\delta^2 \beta^*_i} \\
&= e(h_0, h_0)^{\delta^2 \left(\frac{\beta^*_{-1}}{a} \right)} \prod_{i=0}^{2|\mathcal{S}|-1} e(h_{\lceil i/2 \rceil}, h_{\lfloor i/2 \rfloor})^{\delta^2 \beta^*_i}
\end{aligned}$$

This can be further simplified as

$$e(\widehat{\sigma}, \mathsf{aux}_1)^{1/\delta^2} \prod_{i=0}^{2|\mathcal{S}|-1} e(h_{\lceil i/2 \rceil}, h_{\lfloor i/2 \rfloor})^{-\beta^*_i} = e(h, h)^{\frac{\beta^*_{-1}}{a}}.$$

Here the left side of the equation is precisely $Z^{\beta^*_{-1}}$, thus we get that $Z = e(h, h)^{1/a}$ whenever $\widehat{\sigma}$ is a valid signature for m^* with opening aux, and $\mathcal{A}$ is an admissible adversary. Thus, aggregated unforgeability with adversarial openings follows from hardness of BDHI. □

Full (Aggregated) Unforgeability in ROM. Next, we show that if we instantiate the message hashing in the ROM, then the above aggregate signature construction satisfies full unforgeability. Formally, we prove the following.

Theorem 6.2 (Full Unforgeability)**.** *If the DHI assumption (Assumption 3.6) holds, and $(\mathsf{HGen}, \mathsf{H})$ is instantiated in the ROM, then the aggregate signature scheme described above satisfies (full) unforgeability, and aggregated unforgeability (Definitions 4.1 and 4.3).*

Also, if the BDHI assumption (Assumption 3.7) holds, and $(\mathsf{HGen}, \mathsf{H})$ is instantiated in the ROM, then the aggregate signature scheme described above also satisfies (full) aggregated unforgeability with adversarial openings (Definition 4.5).

Proof. As in the case for RSA-based signatures, the proof of full security in the ROM is very similar to the proof of Theorem 5.3.

Aggregated unforgeability. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks full/adaptive aggregated unforgeability with non-negligible probability ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the strong DH assumption with $\epsilon/Q^{\mathsf{RO}} - \mathsf{negl}$ probability for some negligible function negl, where Q^{RO} is the number of queries $\mathcal{A}$ makes to the random oracle.[7] We describe reduction algorithm $\mathcal{B}$ below.

The reduction algorithm $\mathcal{B}$ is constructed analogous to that in the proof of Theorem 6.1, except now $\mathcal{B}$ uses the power of programmability. $\mathcal{B}$ guesses the index of the challenge message $m^*_{j^*}$ amongst all the RO queries, and programs the hashed message associated with $m^*_{j^*}$ as a random exponent $h_{m^*_{j^*}}$, and implicitly sets $\alpha = a - h_{m^*_{j^*}}$ similar to in the static security proof. It also samples the remaining hash messages for all queried messages at the beginning of the game, and programs them for each queried message at the time the message is first queried. The entire reduction algorithm is defined as before, and the analysis remains identical except the reduction algorithm also aborts if guesses the index of the challenge message $m^*_{j^*}$ incorrectly. Since $\mathcal{B}$'s guess is correct with probability at least $1/Q^{\mathsf{RO}}$, thus $\mathcal{B}$'s final advantage will be $\epsilon/Q^{\mathsf{RO}} - \mathsf{negl}$. This concludes the proof.

Aggregated unforgeability with adversarial openings. With similar modifications as in the above proof, we can reduce full aggregated unforgeability with adversarial openings to BDHI in the ROM. As above, the reduction suffers from a polynomial loss. □

7 Pairing-based Aggregate Identity-Based Encryption

In this section, we provide an aggregate IBE scheme based on the hardness of strong Diffie-Hellman problem. Our constructions supports bounded aggregation of secret keys.

7.1 Construction

$\mathsf{Setup}(1^\lambda, 1^B) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter, λ, as well as the upper bound on number of aggregations, B. It samples the bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda)$, and samples a random exponent $\alpha \leftarrow \mathbb{Z}_p^*$. It also samples the public parameters for identity hashing as $\mathsf{hk} \leftarrow \mathsf{HGen}(1^\lambda)$. It sets the key pair as $\mathsf{mpk} = (\Pi, \mathsf{hk}, \{g^{\alpha^i}\}_{i\in[B]})$ and $\mathsf{sk} = (\Pi, \mathsf{hk}, \alpha)$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. It parses msk as above, and hashes the identity as $h_{\mathsf{id}} = \mathsf{H}(\mathsf{hk}, \mathsf{id})$. It computes the secret key as $g^{(\alpha+h_{\mathsf{id}})^{-1}}$ which can be computed efficiently since it knows α.

$\mathsf{Aggregate}\left(\mathsf{mpk}, \{(\mathsf{id}_i, \mathsf{sk}_i)\}_i\right) \to \widehat{\mathsf{sk}}$. The key aggregation algorithm computes the aggregated key as

$$\widehat{\mathsf{sk}} = \mathsf{DPP}(\{\mathsf{sk}_i, x_i\}_i),$$

where $x_i = \mathsf{H}(\mathsf{hk}, \mathsf{id}_i)$.

[7] For simplicity, here we asssume that the adversary queries all its challenge messages to the random oracle. This can be assumed without loss of generality, as these could be added on top of an already admissible adversary.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m, 1^T) \to \mathsf{ct}$. The encryption algorithm takes as input key mpk (parsed as above), identity id, message m, and a bound T, where T declares *the maximum number of aggregations that any valid secret key could have and still be used for decryption.*[8]

It samples a random exponent $r \leftarrow \mathbb{Z}_p$, computes $h_{\mathsf{id}} = \mathsf{H}(\mathsf{hk}, \mathsf{id})$, outputs the ciphertext as

$$\mathsf{ct} = \left(\{ g^{r(\alpha + h_{\mathsf{id}})\alpha^i} \}_{i=0}^{T}, e(g,g)^r \cdot m \right).$$

Here $g^{r(\alpha+h_{\mathsf{id}})\alpha^i}$ is computed exponentiating the terms $g^{\alpha^{i+1}}, g^{\alpha^i}$ with $r, r \cdot h_{\mathsf{id}}$ (respectively) and then multiplying them.

$\mathsf{AggDec}(\widehat{\mathsf{sk}}, (\mathsf{id}_1, \ldots, \mathsf{id}_\ell), \mathsf{ct}, j \in [\ell])$. It parses ct as $(\{A_i\}_{i=0}^T, B)$, and computes the identity hash for all but j-th identity as $x_i = \mathsf{H}(\mathsf{hk}, \mathsf{id}_i)$ for $i \in [\ell] \setminus \{j\}$. It then computes the coefficients $\{\widetilde{\beta}_i \in \mathbb{Z}_p\}_{i \in [\ell-1]}$, as in Eq. (6). That is, it computes

$$P_{\{x_i\}_{i\in[\ell]\setminus\{j\}}}(y) = \prod_{i \in [\ell]\setminus\{j\}} (y + x_i) = \sum_{i=0}^{\ell-1} \widetilde{\beta}_i y^i \pmod{p}. \tag{12}$$

It then outputs the decrypted message as

$$\frac{B}{e(\prod_{i=0}^{\ell-1} A_i^{\widetilde{\beta}_i}, \mathsf{sk})}.$$

7.2 Correctness and Compactness

Correctness of aggregated decryption. Consider any sequence of identities $\mathsf{id}_1, \ldots, \mathsf{id}_\ell$, and corresponding keys $\mathsf{sk}_i = g^{(\alpha+h_{m_i})^{-1}}$ for $i \in [\ell]$ where $h_{\mathsf{id}_i} = \mathsf{H}(\mathsf{hk}, \mathsf{id}_i)$. We know that aggregating these keys is done as $\widehat{\mathsf{sk}} = \mathsf{DPP}(\{\mathsf{sk}_i, h_{m_i}\}_i)$. Now by the correctness of the key accumulation algorithm of [DPP07, DP08], we have that $\widehat{\mathsf{sk}} = g^{\prod_i (\alpha+h_{m_i})^{-1}}$. Now a ciphertext encrypted for identity id_{i^*} consists of the following

$$\mathsf{ct} = \left(\{ g^{r(\alpha + h_{\mathsf{id}_{i^*}})\alpha^i} \}_{i=0}^{T}, e(g,g)^r \cdot m \right).$$

And to decrypt, the aggregated decryption computes the following:

$$e(\widehat{\mathsf{sk}}, \prod_{i=0}^{\ell-1} (g^{r(\alpha+h_{\mathsf{id}_{i^*}})\alpha^i})^{\beta_i})$$

where β_i's are such that $\sum_{i=0}^{\ell-1} \beta_i y^i = \prod_{i\in[\ell]\setminus\{i^*\}} (y + h_{m_i}) \pmod{p}$. Thus, we have that

$$\prod_{i=0}^{\ell-1} (g^{r(\alpha+h_{\mathsf{id}_{i^*}})\alpha^i})^{\beta_i} = g^{\sum_{i=0}^{\ell-1} r(\alpha+h_{\mathsf{id}_{i^*}})\alpha^i \beta_i} = g^{\prod_{i\in[\ell]}(\alpha+h_{m_i})}.$$

[8] Basically, if $T = 0$, then the ciphertext can only be decrypted by regular (non-aggregated) keys, and otherwise it can be decrypted by any secret key which is an aggregation of secret key for id along with at most T other identity keys.

Therefore, for honestly computed ciphertexts, we get that

$$e(\widehat{\mathsf{sk}}, \prod_{i=0}^{\ell-1}(g^{r(\alpha+h_{\mathsf{id}_{i^*}})\alpha^i})^{\beta_i}) = e(g,g)^r$$

Thus, decryptor correctly recovers the message and correctness follows.

Compactness of Aggregation. The size of an aggregated secret key is same as that of an *unaggregated* secret key, which simply is a source group element (i.e., $\widehat{\mathsf{sk}} \in \mathbb{G}$).

7.3 Security

Static Semantic Security. We show that if we instantiate the message hashing as the identity map in our above aggregate IBE construction, then the resulting scheme satisfies static semantic security. Formally, we prove the following.

Theorem 7.1 (Static Security). If the Decisional BDHI assumption (Assumption 3.8) holds, and $(\mathsf{HGen}, \mathsf{H})$ is an identity hash, then the aggregate IBE scheme described above satisfies static semantic security (Definition 4.10).

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks static semantic security with non-negligible advantage ϵ. Then we construct a PPT adversary $\mathcal{B}$ that breaks the DBDHI assumption with ϵ advantage. We describe reduction algorithm $\mathcal{B}$ below.

In the static setting, the adversary $\mathcal{A}$ submits all its key queries $\mathcal{S} = \{\mathsf{id}_i\}_{i\in[q_k]}$ and the challenge identity id^* at the beginning to the reduction algorithm $\mathcal{B}$. Let B be the aggregation bound. The reduction algorithm then breaks the decisional q-BDHI assumption, where the hardness parameter q is such that $q \geq q_k + B - 1$. The decisional BDHI challenger samples the bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, h_0, e(\cdot,\cdot))$, and a sequence of group elements $\{h_i = h_0^{a^i}\}_{i=0}^{q}$ for a randomly chosen exponent $a \in \mathbb{Z}_p^*$, and sends $(\Pi, h_1, \ldots, h_q, Z)$ to $\mathcal{B}$ where Z is the BDHI challenge which is either sampled random target group element, or it is $e(h_0, h_0)^{1/a}$.

The reduction $\mathcal{B}$ then implicitly sets the master key $\alpha = a - \mathsf{id}^*$, and the base group element g to be $h_0^{\prod_{i=1}^{q_k}(\alpha+\mathsf{id}_i)} = h_0^{\prod_{i=1}^{q_k}(a+\mathsf{id}_i-\mathsf{id}^*)}$. In order to compute these elements in the master public key, $\mathcal{B}$ first computes the polynomials

$$\forall j \in [0, B], \quad P_{\mathcal{S},j,\mathsf{id}^*}(X) = (X - \mathsf{id}^*)^j \prod_{i\in[q_k]} (X + \mathsf{id}_i - \mathsf{id}^*) = \sum_{i=0}^{q_k+j} \beta^{(i)}_{\mathcal{S},j,\mathsf{id}^*} X^i \pmod p$$

to obtain the coefficients $\{\beta^{(i)}_{\mathcal{S},j,\mathsf{id}^*}\}_{j\in[0,B],i\in[0,q_k+j]}$. Using these coefficients, $\mathcal{B}$ computes the master public key

$$\mathsf{mpk} = \left((p, \mathbb{G}, \mathbb{G}_T, g_0, e(\cdot,\cdot)), \mathsf{hk} = \epsilon, \{g_j = g_0^{\alpha^j}\}_{j\in[B]}\right),$$

where the group elements $g_0, \ldots, g_B$ as:

$$\forall j \in [0, B], \quad g_j = \prod_{i=0}^{q_k+j} h_i^{\delta\beta^{(i)}_{\mathcal{S},j,\mathsf{id}^*}} = h_0^{\sum_{i=0}^{q_k+j} \delta\beta^{(i)}_{\mathcal{S},j,\mathsf{id}^*} a^i} = h_0^{\delta(a-\mathsf{id}^*)^j \prod_{i\in[q_k]}(a+\mathsf{id}_i-\mathsf{id}^*)},$$

where $\delta \in \mathbb{Z}_p^*$ is a randomly sampled exponent. It then computes the secret keys for the queried identities $\mathcal{S} = \{\mathsf{id}_j\}_{j\in[q_k]}$ by first computing the coefficients $\{\beta_{\mathcal{S},j}^{(i)}\}_{j\in[q_k],i\in[0,q_k-1]}$ associated with the following polynomials

$$\forall j \in [q_k], \quad P_{\mathcal{S},j}(X) = \prod_{i\in[q_k]\setminus\{j\}} (X + \mathsf{id}_i - \mathsf{id}^*) = \sum_{i=0}^{q_k-1} \beta_{\mathcal{S},j}^{(i)} X^i \pmod{p}$$

Using these, it computes the secret keys as

$$\forall j \in [q_k], \quad \mathsf{sk}_j = \prod_{i=0}^{q_k-1} h_i^{\delta\beta_{\mathcal{S},j}^{(i)}} = h_0^{\sum_{i=0}^{q_k-1} \delta\beta_{\mathcal{S},j}^{(i)} a^i} = h_0^{\delta \prod_{i\in[q_k]\setminus\{j\}}(a+\mathsf{id}_i-\mathsf{id}^*)}.$$

$\mathcal{B}$ then sends the master public key mpk, and secret keys $\{\mathsf{sk}_j\}_{j\in[q_k]}$ as computed above to $\mathcal{A}$. The adversary submits two challenge messages m_0, m_1, and $\mathcal{B}$ after receiving the challenge messages, it computes the challenge ciphertext by implicitly setting the random exponent $r = \gamma/(\alpha+\mathsf{id}^*) = \gamma/a$ for a randomly sampled exponent $\gamma \leftarrow \mathbb{Z}_p^*$. By implicitly setting $r = \gamma/a$, the challenge ciphertext look as follows:

$$\mathsf{ct}^* = \left(\{g_0^{\gamma(a-\mathsf{id}^*)^i}\}_{i=0}^T, e(g_0, g_0)^{\gamma/a} \cdot m_b\right),$$

where b is a randomly sampled challenge bit. Note that $g_j = h_0^{\delta(a-\mathsf{id}^*)^j \prod_{i\in[q_k]}(a+\mathsf{id}_i-\mathsf{id}^*)}$, thus we get that $g_0^{\gamma(a-\mathsf{id}^*)^i} = g_i^\gamma$. Also, note that $e(g_0, g_0)^{\gamma/a}$ can be computed as follows:

$$\begin{aligned}
e(g_0, g_0)^{\gamma/a} &= e(\prod_{i=0}^{q_k} h_i^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}}, \prod_{i=0}^{q_k} h_i^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}})^{\gamma/a} \\
&= \prod_{0\le i,j\le q_k} e(h_i^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}}, h_j^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(j)}})^{\gamma/a} \\
&= e(h_0, h_0)^{(\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(0)})^2\gamma/a} \prod_{\substack{0\le i,j\le q_k \\ (i,j)\neq(0,0)}} e(h_i^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}}, h_j^{\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(j)}})^{\gamma/a} \\
&= e(h_0, h_0)^{(\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(0)})^2\gamma/a} \prod_{\substack{0\le i,j\le q_k \\ (i,j)\neq(0,0)}} e(h_i, h_j)^{\delta^2\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}\beta_{\mathcal{S},0,\mathsf{id}^*}^{(j)}\gamma/a} \\
&= (e(h_0, h_0)^{1/a})^{(\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(0)})^2\gamma} \prod_{\substack{0\le i,j\le q_k \\ (i,j)\neq(0,0)}} (e(h_i, h_j)^{1/a})^{\delta^2\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}\beta_{\mathcal{S},0,\mathsf{id}^*}^{(j)}\gamma}
\end{aligned}$$

where $e(h_i, h_j)^{1/a}$ can be computed as $e(h_{i-1}, h_j)$ whenever $i \neq 0$ and $e(h_i, h_{j-1})$ whenever $j \neq 0$, and the component $e(h_0, h_0)^{1/a}$ is replaced with the BDHI challenge Z. Thus, the challenge ciphertext is set as

$$\mathsf{ct}^* = \left(\{g_i^\gamma\}_{i=0}^T, \quad Z^{(\delta\beta_{\mathcal{S},0,\mathsf{id}^*}^{(0)})^2\gamma} \prod_{\substack{0\le i,j\le q_k \\ (i,j)\neq(0,0)}} (e(h_i, h_j)^{1/a})^{\delta^2\beta_{\mathcal{S},0,\mathsf{id}^*}^{(i)}\beta_{\mathcal{S},0,\mathsf{id}^*}^{(j)}\gamma} \cdot m_b\right).$$

$\mathcal{B}$ sends ct^* as the challenge ciphertext to $\mathcal{A}$, and $\mathcal{A}$ responds with its guess b'. If $b' \neq b$ (i.e. $\mathcal{A}$'s guess is incorrect), then $\mathcal{B}$ outputs 1 signalling Z is a random target group element, otherwise it outputs 0 to signal that $Z = e(h_0, h_0)^{1/a}$ to the DBDHI challenger.

We claim that if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ wins with advantage ϵ as well. First, observe that $\mathcal{B}$ simulates all the public parameters and secret keys perfectly as in the original IBE security game, since it can compute all these terms as appropriate polynomials of the DBDHI challenge tuple. Next, note that if Z is a random target group element, then ct^* completely hides the challenge bit b from the adversary, whereas if $Z = e(h_0, h_0)^{1/a}$, then ct^* perfectly encrypts the challenge message m_b under id^*. Thus, $\mathcal{A}$'s advantage is 0 when $Z \leftarrow \mathbb{G}_T$ whereas it's advantage is ϵ when $Z = e(h_0, h_0)^{1/a}$. This can be used to argue that $\mathcal{B}$'s advantage is also ϵ, thereby concluding the proof of static security.

□

Full Semantic Security. Next, we show that if we instantiate the identity hashing in the ROM, then the above aggregate IBE construction satisfies full security. Formally, we prove the following.

Theorem 7.2 (Full Security)**.** If the Decisional BDHI assumption (Assumption 3.8) holds, and $(\mathsf{HGen}, \mathsf{H})$ is instantiated in the ROM, then the aggregate IBE scheme described above satisfies (full) semantic security (Definition 4.9).

Proof. As in the case for bilinear-based aggregate signatures, the proof of full security in the ROM is identical to the proof of static security as described in Theorem 6.1, except now we use the fact that the identity hash is modeled as a RO to program each queried identity with a randomly sampled exponent, where all such exponents are sampled at the beginning of the game. And, the reduction $\mathcal{B}$ also guesses the query index of the challenge identity id^* amongst all the RO queries[9], and programs the hashed identity associated with id^* as a random exponent h_{id^*}, and implicitly sets $\alpha = a - h_{\mathsf{id}^*}$ as in the static security proof. It programs the sampled non-challenge identity hashes for each queried identity at the time it is first queried. The entire reduction algorithm is defined as before (except this programming), and the analysis remains identical except the reduction algorithm also aborts if it guesses the query index of the challenge identity id^* incorrectly, or the challenge exponent h_{id^*} matches the DBDHI exponent a (in which case it can distinguish DBDHI challenge itself). Since $\mathcal{B}$'s guess is correct with probability at least $1/Q^{\mathsf{RO}}$, thus $\mathcal{B}$'s final advantage will be $\epsilon/Q^{\mathsf{RO}} - \mathsf{negl}$. This concludes the proof. □

Acknowledgements. We thank the anonymous reviewers for their helpful comments. And, we thank Brent Waters for fruitful discussions.

References

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 473–484, 2010.

[9] For simplicity, here we asssume that the adversary queries even the challenge identity to the random oracle. This can be assumed without loss of generality, as this could be added on top of an already admissible adversary.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[BB04a] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 56–73. Springer, 2004.

[BGLS] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. I. a survey of two signature aggregation techniques. *CryptoBytes*, page 1.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Proceedings of Eurocrypt '03*, volume 2656 of LNCS, pages 416–432, 2003.

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 276–285, 2007.

[BGR14] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations. *Information and computation*, 239:356–376, 2014.

[BJ10] Ali Bagherzandi and Stanisław Jarecki. Identity-based aggregate and multi-signature schemes based on rsa. In *International Workshop on Public Key Cryptography*, pages 480–498. Springer, 2010.

[BLK13] Adam Langley Ben Laurie and Emilia Kasper. Certificate transparency. https://datatracker.ietf.org/doc/html/rfc6962, 2013.

[BMP16] Rachid El Bansarkhani, Mohamed Saied Emam Mohamed, and Albrecht Petzoldt. Mqsas-a multivariate sequential aggregate signature scheme. In *International Conference on Information Security*, pages 426–439. Springer, 2016.

[BN07] Mihir Bellare and Gregory Neven. Identity-based multi-signatures from rsa. In *CryptographersâĂŹ Track at the RSA Conference*, pages 145–162. Springer, 2007.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT '97*, volume 1233 of LNCS, pages 480–494, 1997.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[CD96] Ronald Cramer and Ivan Damgård. New generation of secure and practical rsa-based signatures. In *Annual International Cryptology Conference*, pages 173–185. Springer, 1996.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 402âĂŞ414, Berlin, Heidelberg, 1999. Springer-Verlag.

[Cra36] Harald CramÃľr. On the order of magnitude of the difference between consecutive prime numbers. *Acta Arithmetica*, 2(1):23–46, 1936.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong rsa assumption. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):161–185, 2000.

[CTg] Certificate transparency project. https://certificate.transparency.dev/.

[DKS16] David Derler, Stephan Krenn, and Daniel Slamanig. Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In *International Conference on Cryptology and Network Security*, pages 211–227. Springer, 2016.

[DlVP97] Charles-Jean De la Vallée Poussin. *Recherches analytiques sur la théorie des nombres premiers*. Hayez, Imprimeur de l'Académie royale de Belgique, 1897.

[DN94] Cynthia Dwork and Moni Naor. An efficient existentially unforgeable signature scheme and its applications. In *Annual International Cryptology Conference*, pages 234–246. Springer, 1994.

[DP08] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2008.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *Pairing-Based Cryptography - Pairing 2007, First International Conference, Tokyo,* *Japan, July 2-4, 2007, Proceedings*, volume 4575 of *Lecture Notes in Computer Science*, pages 39–59. Springer, 2007.

[DPSS15] David Derler, Henrich C Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In *ICISC 2015*, pages 3–19. Springer, 2015.

[FHPS13] Eduarda SV Freire, Dennis Hofheinz, Kenneth G Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *Advances in Cryptology–CRYPTO 2013*, pages 513–530. Springer, 2013.

[Fia89] Amos Fiat. Batch rsa. In *Conference on the Theory and Application of Cryptology*, pages 175–185. Springer, 1989.

[Fis03] Marc Fischlin. The cramer-shoup strong-rsa signature scheme revisited. In *International Workshop on Public Key Cryptography*, pages 116–129. Springer, 2003.

[FLS12] Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. In *International Conference on Security and Cryptography for Networks*, pages 113–130. Springer, 2012.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 1999.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[Gol17] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

[Gor18] Sergey Gorbunov. How not to use aggregate signatures in your blockchain, 2018.

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *PKC*, 2006.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 99–108, 2011.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 3–34. Springer, 2015.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *CRYPTO*, 2013.

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the rsa assumption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 197–229. Springer, 2018.

[IN83] Kazuharu Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC J. Res. Dev.*, 71, 1983.

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CryptographersâĂŹ track at the RSA conference*, pages 244–262. Springer, 2002.

[LLY13a] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures made shorter. In *International Conference on Applied Cryptography and Network Security*, pages 202–217. Springer, 2013.

[LLY13b] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures with short public keys: Design, analysis and implementation studies. In *International Workshop on Public Key Cryptography*, pages 423–442. Springer, 2013.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2004.

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, 2006.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 245–254, 2001.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *In Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS*, pages 120–130. IEEE, 1999.

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 85(2):481–484, 2002.

[MT07] Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 86–91. IEEE, 2007.

[Nev08] Gregory Neven. Efficient sequential aggregate signed data. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 52–69. Springer, 2008.

[Oka88] Tatsuaki Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Transactions on Computer Systems (TOCS)*, 6(4):432–441, 1988.

[OO99] Kazuo Ohta and Tatsuaki Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 82(1):21–31, 1999.

[Rab80] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[RS09] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *International Conference on Information Security and Assurance*, pages 750–759. Springer, 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *International Conference on Information Security and Cryptology*, pages 285–304. Springer, 2001.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO '84*, volume 196 of LNCS, pages 47–53, 1984.

[SS77] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM journal on Computing*, 6(1):84–85, 1977.

[Sud09] Madhu Sudan. Probabilistically checkable proofs. *Commun. ACM*, 52(3):76–84, 2009.

[Yek12] Sergey Yekhanin. Locally decodable codes. *Found. Trends Theor. Comput. Sci.*, 6(3):139–255, 2012.

A Locally Verifiable Multi-Signer Aggregate Signature

In this section, we extend the concept of locally-verifiable aggregate signatures defined in Section 4 to the multi-signer setting. Naturally, in the multi-signer setting, all the local and global aggregation and verification algorithms would take as input a sequence of verification keys instead of a single verification key now. Below we define it formally.

Syntax. A locally-verifiable multi-signer aggregate signature scheme $\mathcal{S}$ for message space $\mathcal{M}$ consists of the following polynomial time algorithms:

$\mathsf{CRS}(1^\lambda) \to \mathsf{crs}$. The CRS generation algorithm samples global parameters crs. All the remaining algorithms take crs as input, and for ease of notation we do not write it explicitly.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. The setup algorithm, on input the security parameter λ, outputs a pair of signing and verification keys $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a signing key sk and a message $m \in \mathcal{M}$, and computes a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. The verification algorithm takes as input a verification key vk, a message $m \in \mathcal{M}$, and a signature σ. It outputs a bit to signal whether the signature is valid or not.

$\mathsf{Aggregate}\left(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i\right) \to \widehat{\sigma}/\bot$. The signature aggregation algorithm takes as input a sequence of tuples, each containing a verification key vk_i, a message m_i, a signature σ_i, and it outputs either an aggregated signature $\widehat{\sigma}$ or a special abort symbol $\bot$.

$\mathsf{AggVerify}\left(\{(\mathsf{vk}_i, m_i)\}_i, \widehat{\sigma}\right) \to 0/1$. The aggregate verify algorithm takes as input a sequence of tuples, each containing a verification key vk_i, a message m_i, and it outputs a bit to signal whether the aggregated signature $\widehat{\sigma}$ is valid or not.

$\mathsf{LocalOpen}(\widehat{\sigma}, \{(\mathsf{vk}_i, m_i)\}_{i \in [\ell]}, j \in [\ell]) \to \mathsf{aux}_j$. The local opening algorithm takes as input an aggregated signature $\widehat{\sigma}$, a sequence of tuples (each containing a verification key vk_i and a message m_i for $i \in [\ell]$), and an index $j \in [\ell]$ where ℓ is the length of the sequence. It outputs auxiliary information aux_j corresponding to the key-message pair (vk_i, m_i).

$\mathsf{LocalAggVerify}(\widehat{\sigma}, \mathsf{vk}, m, \mathsf{aux}) \to 0/1$. The local aggregate verification algorithm takes as input an aggregated signature $\widehat{\sigma}$, a verification key vk, a message m, and auxiliary information aux. It outputs a bit to signal whether the aggregate signature $\widehat{\sigma}$ contains a signature for message m under verification key vk, or not.

Correctness and Compactness. A locally-verifiable multi-signer aggregate signature scheme is said to be correct and compact if for all $\lambda, \ell, N \in \mathbb{N}$, parameters $\mathsf{crs} \leftarrow \mathsf{CRS}(1^\lambda)$, verification-signing key pairs $(\mathsf{vk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Setup}(1^\lambda)$ for $j \in [N]$, messages m_i for $i \in [\ell]$, every key mapping function $\pi : [\ell] \to [N]$, and every signature $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_{\pi(i)}, m_i)$ for $i \in [\ell]$, the following holds:

Correctness of signing. For all $i \in [\ell]$, $\mathsf{Verify}(\mathsf{vk}_{\pi(i)}, m_i, \sigma_i) = 1$.

Correctness of aggregation. If $\widehat{\sigma} = \mathsf{Aggregate}\left(\{\left(\mathsf{vk}_{\pi(i)}, m_i, \sigma_i\right)\}_i\right)$, then

$$\mathsf{AggVerify}\left(\{\left(\mathsf{vk}_{\pi(i)}, m_i\right)\}_i, \widehat{\sigma}\right) = 1.$$

Correctness of local opening. For all $k \in [\ell]$, we have

$$\Pr\left[\mathsf{LocalAggVerify}\left(\widehat{\sigma}, \mathsf{vk}_{\pi(k)}, m_k, \mathsf{LocalOpen}(\widehat{\sigma}, \{(\mathsf{vk}_{\pi(i)}, m_i)\}_i, k)\right)\right] = 1.$$

Compactness of aggregation. $|\widehat{\sigma}| \leq \mathsf{poly}(\lambda)$. That is, the size of an aggregated signature is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

Compactness of opening. $|\mathsf{aux}| \leq \mathsf{poly}(\lambda)$. That is, the size of the auxiliary opening information is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

Security. Now for security, as in the single-signer setting, we have regular unforgeability, aggregated unforgeability, and aggregated unforgeability with adversarial openings. The regular unforgeability and aggregated unforgeability with adversarial openings is defined identically in the multi-signer setting (which the only difference that the challenger also samples the global parameters crs at the beginning of the experiments).

Below we simply provide the aggregated unforgeability property for the multi-signer setting for completeness.

Definition A.1 (Aggregated Unforgeability)**.** A multi-signer aggregate signature scheme $(\mathsf{CRS}, \mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ is said to be a secure aggregate signature scheme if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{AggVerify}\left(\{(\mathsf{vk}_i^*, m_i^*)\}_{i\in[\ell]}, \widehat{\sigma}^*\right) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRS}(1^\lambda), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \left(\{(\mathsf{vk}_i^*, m_i^*)\}_{i\in[\ell]}, \widehat{\sigma}^*\right) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if there exists $i \in [\ell]$ such that $\mathsf{vk}_i^* = \mathsf{vk}$ and m_i^* was not queried by $\mathcal{A}$ to the $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle.

Definition A.2 (Static Aggregated Unforgeability)**.** We say the aggregate signature scheme is statically secure if the adversary in the above game is confined to make all of its message queries $\{m_i\}_{i\in[q]}$ and declare the challenge key-message tuples $\{(\mathsf{vk}_i^*, m_i^*)\}_{i\in[\ell]}$ at the beginning of the game (defined in Definition 4.3) before it receives the verification key vk.

Fully Public Openings. As in the single-signer setting, we additionally consider the setting where the local opening algorithm does not need an aggregate signature to provide an opening w.r.t., but only the sequence of key-message pairs. Thus, syntactically we have — $\mathsf{LocalOpen}(\{(\mathsf{vk}_i, m_i)\}_{i\in[\ell]}, j \in [\ell]) \to \mathsf{aux}_j$.

A.1 Properties of (Aggregate) Signatures

An (aggregate) signature scheme can additionally satisfy following properties and/or suffer from the corresponding restriction.

Unique Signatures. A signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ is said to be a unique signature scheme if for all tuples $(\mathsf{vk}, m, \sigma_1, \sigma_2)$ where $\sigma_1 \neq \sigma_2$, $\mathsf{Verify}(\mathsf{vk}, m, \sigma_i) = 1$ for at most one i.

Single-Signer Aggregation. An aggregate signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ is said to be a single-signer aggregate signature if the aggregation algorithm and aggregate-verify algorithms require all the verification keys to be the same. That is, the aggregation algorithm only takes as input a single verification key vk, thus aggregation is only defined for a single signer. Syntactically, the algorithms are modified as follows — $\mathsf{Aggregate}\left(\mathsf{vk}, \{(m_i, \sigma_i)\}_i\right)$ and $\mathsf{AggVerify}\left(\mathsf{vk}, \{m_i\}_i, \widehat{\sigma}\right)$.

Multi-Hop vs. Single-Hop Aggregation. An aggregate signature scheme is said to be multi-hop aggregatable if the Aggregate algorithm also takes as input an aggregated signature as well. Syntactically, the algorithm is modified as follows — $\mathsf{Aggregate}\left(\{(\mathbf{vk}_i, \mathbf{m}_i, \widehat{\sigma}_i)\}_i\right)$, where each $\mathbf{vk}_i, \mathbf{m}_i$ are non-empty sequence of verification keys and messages, and $\widehat{\sigma}_i$ is a possibly aggregated signature.

On the other hand, a single-hop aggregate signature requires the input signatures to the Aggregate algorithm to be regular (non-aggregated) signatures.

Bounded Aggregation. For any $q > 1$, an aggregate signature scheme is said to be q-bounded aggregatable if the Aggregate algorithm takes at most q message-signature pairs. That is, $\mathsf{Aggregate}\left(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[\ell]}\right)$ is well defined as long as $\ell \leq q$. To make the notion of bounded aggregation non-trivial, it is essential to require that the size of aggregated signatures to be independent of the bound q.

Sequential Aggregation. An aggregate signature scheme is said to be sequentially aggregatable if the aggregation algorithm takes as input the signing key sk and a single (possibly aggregated) signature $\widehat{\sigma}$ instead of a sequence of signatures. Syntactically, the aggregated signing algorithm works as follows.

$\mathsf{Aggregate}\left(\mathsf{sk}, m', \{(\mathsf{vk}_i, m_i)\}_i, \widehat{\sigma}\right) \to \widehat{\sigma}'/\perp$. The sequential aggregation algorithm takes as input a signing key sk, message m', a sequence of tuples (each containing a verification key vk_i and a message m_i), and an input aggregated signature $\widehat{\sigma}$. It outputs either an aggregated signature $\widehat{\sigma}'$ or a special abort symbol $\perp$.

Typically, the aggregate verification algorithm requires the sequence of messages $(m_1, \ldots, m_\ell)$ (being verified with respect to the aggregated signature $\widehat{\sigma}$) to be provided in the same order as the one in which sequential aggregation was performed. That is, typically the verification is sensitive to the sequence of aggregation. Now sequential aggregate signature schemes where the verification is oblivious to the ordering are said to be *unordered* sequential aggregation schemes.

B Sampling via Random Oracles

Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_\lambda$ be a family of hash functions where each $h \in \mathcal{H}_\lambda$ takes 2λ bits as input, and outputs λ-bits of output.

$\mathsf{PrimeSeq}^{\mathcal{H}}(1^\lambda) \to \mathsf{samp}$. It samples a hash function $h \leftarrow \mathcal{H}_\lambda$, and sets $\mathsf{samp} = h$.

$\mathsf{PrimeSamp}^{\mathcal{H}}(\mathsf{samp} = h, m) \to e_m$. It proceeds as follows:

```
1. Set $\mathsf{count} := 0$, $\mathsf{flag} := \mathsf{false}$.
2. While $\mathsf{flag} = \mathsf{false}$:
    (a) Let $y := h(m \,||\, \mathsf{count})$ where $m \,||\, \mathsf{count}$ is interpreted as a $2\lambda$ length bit string.
    (b) Run PrimalityTest to check if $2^\lambda + y$ is a prime. If it is a prime, set $\mathsf{flag} := \mathsf{true}$ and $e_m := 2^\lambda + y$. Otherwise, set $\mathsf{count} := \mathsf{count} + 1$.
```

Output e_m.

Theorem B.1 (Efficient and Adaptive Programmable Enumeration in ROM)**.** If $\mathcal{H}_\lambda$ is modeled as a random oracle, then $(\mathsf{PrimeSeq}^{\mathcal{H}}, \mathsf{PrimeSamp}^{\mathcal{H}})$ satisfies the following properties:

Efficient Sampling. For every $\lambda \in \mathbb{N}$, $m \in \{0,1\}^\lambda$, the prime sampling algorithm $\mathsf{PrimeSamp}^{\mathcal{H}}$ runs in expected polynomial time, where the probability is taken over the coins of setup algorithm $\mathsf{PrimeSeq}^{\mathcal{H}}$.

Adaptive Non-Colliding Prime Enumeration. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\mathsf{PrimeSamp}^{\mathcal{H}}(\mathsf{samp}, m_1) = \mathsf{PrimeSamp}^{\mathcal{H}}(\mathsf{samp}, m_2) \wedge m_1 \neq m_2 : \begin{array}{r} \mathsf{samp} \leftarrow \mathsf{PrimeSeq}^{\mathcal{H}}(1^\lambda) \\ (m_1, m_2) \leftarrow \mathcal{A}(1^\lambda, \mathsf{samp}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Programmable Prime Enumeration. $\mathsf{PrimeSamp}^{\mathcal{H}}$ behaves as a programmable random prime oracle.

Proof. The proof of this theorem is similar to that of Theorem 5.1 with the modification that now since $\mathcal{H}_\lambda$ is modeled as a random oracle, thus the resulting prime sequence enumerator can be modeled as a “programmable" random prime oracle. Note that the adaptive non-colliding prime enumeration property simply follows from collision resistance of the hash function, and we do not need to model the hash function as a random oracle for that. □

Remark B.2 (Efficiency of Sampling)**.** We would like to point out that for our above samplers we only prove efficiency in the average case, thus the running time of sampler could be exponential in the worst case. However, we could prove worst case running time to be polynomial as well if we assume the well-known prime gap conjectures such as Cramér’s conjecture [Cra36], or even weaker variants of those conjectures which state that the prime gaps are bounded poly-logarithmically in the prime value. Thus, by relying on such prime gap conjectures, we get that worst case running time of sampling will also be polynomial. In order to complete the argument, we would need to make minor edit to the sampling procedure where instead of now evaluating the function on $m \,||\, 0, m \,||\, 1, m \,||\, 2, \ldots$ and so on, we would simply evaluate the function on m and then find the closest prime greater than or equal to the function evaluation on m. Thus, by prime gap conjectures, this can always be computed in time $\mathsf{poly}(\lambda)$.