Pairing-Based Aggregate Signatures without Random Oracles

Susan Hohenberger
Johns Hopkins University
susan@cs.jhu.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

An aggregate signature scheme allows a user to take N signatures from N users and aggregate them into a single short signature. One approach to aggregate signatures uses general-purpose tools like indistinguishability obfuscation or batch arguments for NP. These techniques are general, but lead to schemes with very high concrete overhead. On the practical end, the seminal work of Boneh, Gentry, Lynn, and Shacham (EUROCRYPT 2003) gives a simple and practical scheme, but in the random oracle model. In the plain model, current practical constructions either rely on interactive aggregation or impose restrictions on how signatures can be aggregated (e.g., same-message aggregation, same-signer aggregation or only support sequential or synchronized aggregation).

In this work, we focus on simple aggregate signatures in the plain model. We construct a pairing-based aggregate signature scheme that supports aggregating an a priori bounded number of signatures N. The size of the aggregate signature is just two group elements. Security relies on the (bilateral) computational Diffie-Hellman (CDH) problem in a pairing group. To our knowledge, this is the first group-based aggregate signature in the plain model where (1) there is no restriction on what type of signatures can be aggregated; (2) the aggregated signature contains a constant number of group elements; and (3) security is based on static falsifiable assumptions in the plain model. The limitation of our scheme is that our scheme relies on a set of public parameters (whose size scales with N) and individual signatures (before aggregation) also have size that scale with N. Essentially, individual signatures contain some additional hints to enable aggregation.

Our starting point is a new notion of *slotted* aggregate signatures. Here, each signature is associated with a "slot" and we only support aggregating signatures associated with distinct slots. We then show how to generically lift a slotted aggregate signature scheme into a standard aggregate signature scheme at the cost of increasing the size of the original signatures.

1 Introduction

An aggregate signature scheme allows a user to take N signatures from N users and aggregate them into a single short signature. More precisely, given a collection of N triples $(\mathsf{vk}_1, m_1, \sigma_1), \ldots, (\mathsf{vk}_N, m_N, \sigma_N)$, where σ_i is a signature on m_i with respect to verification key vk_i, it should be possible to publicly obtain an aggregate signature σ_{agg} on the list of messages $(m_1, \ldots, m_N)$ that verifies with respect to the list of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$. Moreover, the size of the aggregate signature σ_{agg} should be sublinear in the number of signatures N. In this work, we focus exclusively on schemes with *non-interactive* aggregation. Namely, there is no interaction between the aggregator and the individual signers.

Aggregate signatures are useful whenever we have an application that requires communicating multiple signatures from different users. For instance, when a client connects to a server over TLS, the server will send a certificate chain that authenticates its public key. Each certificate in the chain contains a signature from one certificate authority on a cryptographic key. An aggregate signature would allow the server to send a single signature rather than N signatures (when considering a certificate chain of length N). More recently, aggregate signatures have found applications to blockchains. Here, signatures from many different users (e.g., authorizing different transactions) are compressed into a single short signature that is recorded on the blockchain. Aggregate signatures are also a useful consensus mechanism where signatures from multiple independent validators are compressed into a single short certificate of validity.

One of the simplest aggregate signature schemes is the pairing-based scheme of Boneh, Gentry, Lynn, and Shacham [BGLS03], which essentially augments the Boneh-Lynn-Shacham (BLS) signature scheme [BLS01] to support aggregation. While this scheme is simple and lightweight, its security relies on the random oracle heuristic. A natural

question is to design efficient aggregate signatures in the *plain* model without random oracles. This is the focus of this work.

Aggregate signatures in the plain model. Previously, the works of [BCCT13, HKW15] built *universal* signature aggregators from succinct non-interactive arguments of knowledge (SNARKs) for NP and from indistinguishability obfuscation, respectively. These schemes allow aggregating *arbitrary* signatures. More recently, the flurry of works on constructing batch arguments for NP [CJJ21a, CJJ21b, WW22, DGKV22, PP22, KLVW23, CGJ+23, CEW25] also directly imply aggregate signatures in the plain model (c.f., [WW22, DGKV22]). However, the reliance on indistinguishability obfuscation or general-purpose batch arguments incurs substantial concrete overheads. For instance, these approaches all make non-black-box use of an existing digital signature scheme. Our goal in this work is to develop a more direct approach for constructing aggregate signatures in the plain model. Specifically, we seek constructions that only need to make black-box use of a (pairing) group, and moreover, the size of the aggregate signature consists of a *constant* number of group elements.[1] There has also been a line of work studying relaxations of aggregate signatures to enable efficient constructions in the plain model. Typically, these schemes impose restrictions on what types of signatures can be aggregated; we refer to Section 1.2 for a discussion of these approaches.

This work. In this work, we construct an aggregate signature from the (bilateral) computational Diffie-Hellman (CDH) assumption in pairing groups. Our scheme supports aggregating an a priori bounded number of signatures N. An aggregate signature on up to N messages consists of just two group elements. To our knowledge, this is the first group-based aggregate signature in the plain model where (1) there is no restriction on what type of signatures can be aggregated (see Section 1.2 for more discussion); (2) the aggregated signature contains a constant number of group elements; and (3) security is based on static falsifiable assumptions in the plain model. The limitation is that our scheme requires a set of long public parameters ($N^{1+\alpha}$ for any constant $\alpha > 0$) and moreover, the size of individual signatures *before* aggregation scales with N (specifically, individual signatures have size $N \cdot \mathsf{poly}(\lambda, \log N)$, where λ is the security parameter).

The individual signatures in our scheme contain additional information that enables aggregation, but are essentially unnecessary for signature verification. For this reason, the aggregation process still yields short signatures consisting of a constant number of group elements. While it may seem undesirable for individual signatures to be long, in many applications of aggregate signatures, only the aggregated signature needs to be transmitted or stored long term. For instance, when communicating certificate chains in a TLS connection, a server only needs to transmit an aggregate signature to the clients. In blockchain applications, only the aggregated signature for a set of transactions needs to be stored on the blockchain. In both types of applications, the long individual signatures do not need to be communicated or stored. For these applications, our construction offers a compelling solution to aggregate signatures without random oracles.

1.1 Technical Overview

To design an aggregate signature scheme in the plain model, it helps to start with an ordinary signature scheme. In this work, we consider the signature scheme that is derived from the Boneh-Boyen [BB04] selectively-secure identity-based encryption (IBE) scheme.[2] The Boneh-Boyen signature scheme has appealing properties:

- **Based on simple assumptions.** The Boneh-Boyen signature scheme is a simple scheme whose security reduces to the standard computational Diffie-Hellman (CDH) problem over bilinear groups.
- **Easily extensible to adaptive security.** While the Boneh-Boyen signature scheme is selectively-secure, one can swap in the Waters [Wat05] hash and adapt the analysis to prove adaptive security.

[1] If we combine a group-based batch argument for NP (e.g., [WW22, CGJ+23, CEW25]) with an existing digital signature scheme, the size of the resulting signature would contain $\mathsf{poly}(\lambda)$ group elements, where λ is a security parameter. Specifically, in these constructions, the size of the aggregate signature grows with the size of the Boolean circuit that computes the signature verification algorithm. This circuit has super-constant size since it must read the signature. The resulting aggregate signature then contains a super-constant number of group elements.

[2] Specifically, the signature for a message m is an IBE decryption key for identity m. However, instead of using this key to decrypt a ciphertext (as in IBE), the user verifies the signature by using the bilinear map to check that the decryption key is well-formed. We remark that BLS signatures [BLS01] are derived in an analogous manner from the Boneh-Franklin [BF01] IBE scheme (in the random oracle model).

- **Aggregatable multi-signature.** The work of Lu, Ostrovsky, Sahai, Shacham, and Waters [LOS+06] previously showed how to aggregate signatures from many signers under the restriction that every signature is on the *same* message.[3] In other words, the [LOS+06] scheme is an aggregatable "multi-signature." However, as we demonstrate below, it is unclear how to support general aggregation (of signatures with arbitrary messages under arbitrary verification keys).

The Boneh-Boyen signature scheme. To illustrate our techniques, we start by recalling the Boneh-Boyen [BB04] signature scheme. To simplify the exposition, we describe everything using a symmetric pairing group.[4] Let $(\mathbb{G}, \mathbb{G}_T)$ be a symmetric bilinear pairing group with prime order p. Let g be a generator for $\mathbb{G}$, and $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently-computable bilinear map. The Boneh-Boyen signature scheme then works as follows:

- **Key-generation:** The key-generation algorithm samples a random exponent $\alpha \xleftarrow{R} \mathbb{Z}_p$ and random group elements $u, h \xleftarrow{R} \mathbb{G}$. The secret key is the exponent $\mathsf{sk} = \alpha$ while the public key is the triple $\mathsf{vk} = (e(g,g)^\alpha, u, h)$.
- **Signature:** To sign a message $m \in \mathbb{Z}_p$, the signer samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs $\sigma = (\sigma_1, \sigma_2)$ where

$$\sigma_1 = g^\alpha (u^m h)^r \quad \text{and} \quad \sigma_2 = g^r.$$

  We can view the element $u^m h$ as the "Boneh-Boyen hash" of the message m. We could alternatively replace this with the bit-by-bit hash function from [Wat05] to obtain an analogous signature scheme. Using [Wat05] allows us to prove adaptive security in the plain model, but at the expense of longer public keys.
- **Verification:** To check a signature $\sigma = (\sigma_1, \sigma_2)$ with respect to a verification key $\mathsf{vk} = (e(g,g)^\alpha, u, h)$, the verifier checks that

$$e(g,g)^\alpha \stackrel{?}{=} \frac{e(\sigma_1, g)}{e(\sigma_2, u^m h)}.$$

Supporting same-message aggregation. Next, we describe the [LOS+06] approach of extending the Boneh-Boyen scheme to an aggregatable multi-signature (i.e., a scheme that supports aggregating signatures on the *same* message). First, the [LOS+06] scheme assumes that the users' verification keys share a common u, h. In other words, the group elements u, h are now part of the public parameters for the aggregate multi-signature. Each user's individual verification key is $e(g,g)^\alpha$ and the exponent α is their signing key.

Suppose we have two users with public keys $\mathsf{vk}_1 = e(g,g)^{\alpha_1}$ and $\mathsf{vk}_2 = e(g,g)^{\alpha_2}$. Moreover, suppose σ_1 is a signature on m under vk_1 (with signing randomness r_1) and σ_2 is a signature on m under vk_2 (with signing randomness r_2). Then, we can write

$$\sigma_1 = (\sigma_{1,1},\ \sigma_{1,2}) = \left(g^{\alpha_1}(u^m h)^{r_1},\ g^{r_1}\right)$$
$$\sigma_2 = (\sigma_{2,1},\ \sigma_{2,2}) = \left(g^{\alpha_2}(u^m h)^{r_2},\ g^{r_2}\right)$$

The aggregate signature σ_{agg} on m is then

$$\sigma_{\mathsf{agg}} = \left(\sigma_{1,1}\sigma_{2,1},\ \sigma_{1,2}\sigma_{2,2}\right) = \left(g^{\alpha_1+\alpha_2}(u^m h)^{r_1+r_2},\ g^{r_1+r_2}\right).$$

By construction, this is a signature on m with respect to the "aggregated verification key"

$$\mathsf{vk}_{\mathsf{agg}} = \mathsf{vk}_1 \cdot \mathsf{vk}_2 = e(g,g)^{\alpha_1+\alpha_2}$$

and signing randomness $r_1 + r_2$. This approach naturally extends to N users, and the size of the aggregate multi-signature remains at exactly two group elements. The above aggregation procedure critically relies on the the fact that σ_1 and σ_2 are signatures on the *same* message, and thus, share a *common* base $u^m h$.

[3] Technically they showed this for the Waters [Wat05] signature, but since the Waters' scheme shares the same basic structure as the Boneh-Boyen construction, their approach applies equally well to the Boneh-Boyen scheme.

[4] Everything here readily extends to asymmetric groups and in the technical sections (e.g., Section 3.1), we describe our constructions using asymmetric pairing groups.

Aggregating signatures with common randomness. Consider now the more general setting where the signatures are on *different* messages m_1 and m_2:

$$\sigma_1 = (\sigma_{1,1},\ \sigma_{1,2}) = \left(g^{\alpha_1}(u^{m_1}h)^{r_1},\ g^{r_1}\right)$$
$$\sigma_2 = (\sigma_{2,1},\ \sigma_{2,2}) = \left(g^{\alpha_2}(u^{m_2}h)^{r_2},\ g^{r_2}\right)$$

As written, there does not appear to be a simple way to aggregate these signatures since the signatures have different bases: ($u^{m_1}h$ in σ_1 and $u^{m_2}h$ in σ_2) *and* different randomness r_1, r_2. However, suppose for a moment that the users had used the *same* randomness to construct σ_1 and σ_2: namely, let $r_1 = r_2 = r$. In this case, a simple aggregation strategy is to compute

$$\sigma_{\text{agg}} = (\gamma_1, \gamma_2) = (\sigma_{1,1}\sigma_{2,1},\ \sigma_{1,2}) = \left(g^{\alpha_1+\alpha_2}(u^{m_1}h)^{r}(u^{m_2}h)^{r}, g^{r}\right). \tag{1.1}$$

Observe first that the aggregate signature only consists of two group elements, which is exactly the same size as a plain Boneh-Boyen signature. Moreover, given the public parameters $\text{pp} = (u, h)$, the aggregate signature $\sigma_{\text{agg}} = (\gamma_1, \gamma_2)$, the users' verification keys $\text{vk}_1 = e(g,g)^{\alpha_1}, \text{vk}_2 = e(g,g)^{\alpha_2}$, and the messages m_1, m_2, one can verify the aggregate signature by checking

$$\text{vk}_1 \cdot \text{vk}_2 = e(g,g)^{\alpha_1} \cdot e(g,g)^{\alpha_2} \stackrel{?}{=} \frac{e(\gamma_1, g)}{e(\gamma_2, u^{m_1}h \cdot u^{m_2}h)}. \tag{1.2}$$

We can view $\text{vk}_1 \cdot \text{vk}_2$ as the aggregated verification key associated with $(\text{vk}_1, \text{vk}_2)$. This approach naturally extends to the setting where there are N signatures, provided that all of the signatures share the same signing randomness.

Introducing helper terms to facilitate aggregation. The problem is we cannot expect independent signers to choose the same randomness when signing. In fact, security of the Boneh-Boyen scheme critically relies on the signing randomness r being random and unknown to the adversary. To tackle this, we start by considering a relaxed version of the problem where there are two types of public keys and signatures (which we denote by "Type-1" and "Type-2"). Moreover, the scheme only supports aggregating a Type-1 signature with a Type-2 signature. We now describe our modified approach:

- **Public parameters:** The public parameters for Type-1 and Type-2 signatures are (independently-generated) public parameters for the Boneh-Boyen multi-signature. Namely, the public parameters for Type-1 signatures is the pair (u_1, h_1); similarly, the public parameters for Type-2 signatures is the pair (u_2, h_2).
- **Signing and verification keys:** A user's signing key is still $\text{sk} = \alpha \in \mathbb{Z}_p$ and their verification key is $\text{vk} = e(g,g)^{\alpha}$, exactly as before.
- **Signature:** A Type-1 signature σ_1 on a message m_1 (with signing key α_1 and signing randomness r_1) contains a standard Boneh-Boyen signature with respect to the Type-1 public parameters together with additional helper components with respect to the Type-2 public parameters. Namely,

  $$\sigma_1 = \left(g^{\alpha_1}(u_1^{m_1}h_1)^{r_1},\ g^{r_1},\ u_2^{r_1},\ h_2^{r_1}\right),$$

  Similarly, a Type-2 signature σ_2 on a message m_2 (with signing key α_2 and signing randomness r_2) has the following structure:

  $$\sigma_2 = \left(g^{\alpha_2}(u_2^{m_2}h_2)^{r_2},\ g^{r_2},\ u_1^{r_2},\ h_1^{r_2}\right).$$

  The helper components in σ_1, σ_2 essentially couple the signing randomness used to construct the signature with the public parameters of the other type. These extra components will enable aggregation.

To aggregate the Type-1 signature σ_1 with the Type-2 signature σ_2, we use the additional helper components to transform each into a signature with *common* randomness $r_{\text{agg}} = r_1 + r_2$ (and with respect to public parameters (u_1u_2, h_1h_2)). Specifically, given σ_1, σ_2 and m_1, m_2, the aggregation algorithm first computes

$$\sigma_1' = \left(g^{\alpha_1}(u_1^{m_1}h_1)^{r_1} \cdot (u_1^{r_2})^{m_1} \cdot h_1^{r_2},\ g^{r_1} \cdot g^{r_2}\right) = \left(g^{\alpha_1}(u_1^{m_1}h_1)^{r_1+r_2},\ g^{r_1+r_2}\right)$$
$$\sigma_2' = \left(g^{\alpha_2}(u_2^{m_2}h_2)^{r_2} \cdot (u_2^{r_1})^{m_2} \cdot h_2^{r_1},\ g^{r_1} \cdot g^{r_2}\right) = \left(g^{\alpha_2}(u_2^{m_2}h_2)^{r_1+r_2},\ g^{r_1+r_2}\right).$$

At this point, σ'_1 and σ'_2 are Boneh-Boyen signatures with common randomness $r_1 + r_2$, so we can aggregate them using the procedure from Eq. (1.1):

$$\sigma_{\mathsf{agg}} = (\gamma_1, \gamma_2) = (\sigma'_{1,1}\sigma'_{2,1},\ \sigma'_{1,2}) = \left(g^{\alpha_1+\alpha_2}(u_1^{m_1}h_1)^{r_1+r_2}(u_2^{m_2}h_2)^{r_1+r_2}, g^{r_1+r_2}\right).$$

Given the Type-1 public parameters $\mathsf{pp}_1 = (u_1, h_1)$, the Type-2 public parameters $\mathsf{pp}_2 = (u_2, h_2)$, the aggregate signature $\sigma_{\mathsf{agg}} = (\gamma_1, \gamma_2)$, the users' verification keys $\mathsf{vk}_1 = e(g,g)^{\alpha_1}$, $\mathsf{vk}_2 = e(g,g)^{\alpha_2}$, and the messages m_1, m_2, the corresponding aggregate verification algorithm is then

$$\mathsf{vk}_1 \cdot \mathsf{vk}_2 = e(g,g)^{\alpha_1} \cdot e(g,g)^{\alpha_2} \stackrel{?}{=} \frac{e(\gamma_1, g)}{e(\gamma_2, u_1^{m_1}h_1 \cdot u_2^{m_2}h_2)}.$$

It is not difficult to see that the ideas above can generalize to support aggregation between N different types of signatures where N is fixed at system setup. The public parameters now scale linearly with N (corresponding to N sets of public parameters, one for each signature type). For all $s \in [N]$, a Type-s signature contains a total of $2N$ group elements. This includes (1) a Boneh-Boyen signature with respect to the Type-s parameters; and (2) cross terms that relate the signing randomness with the Type-t public parameters for all $t \neq s$. The additional cross terms enable our aggregation procedure described above. The size of the aggregate signature always consists of exactly two group elements.

Slotted aggregate signatures. In Section 3, we refer to the above scheme as a "slotted aggregate signature" scheme. Namely, in this setting, there are N slots (corresponding to the N different signature types). A Type-s signature is associated with the slot $s \in [N]$. The aggregation algorithm takes as input a list of tuples $\{(s, \mathsf{vk}_s, m_s, \sigma_s)\}_{s\in S}$ where each σ_s is a Type-s signature (associated with the slot $s \in S$) and aggregates the list of signatures into a short signature σ_{agg}. The restriction in a slotted signature scheme is we can only aggregate a list of signatures where each signature in the list is associated with a *distinct* slot (e.g., we cannot aggregate two signatures that are both associated with the same slot s). Subsequently, we will describe a generic way to lift a slotted aggregate signature scheme into a standard aggregate signature scheme that supports bounded aggregation without any slot restrictions.

Security for a slotted aggregate signature scheme. The security requirement on a slotted aggregate signature scheme is that the adversary cannot forge an aggregate signature on any list $\{(s, \mathsf{vk}_s, m_s)\}_{s\in S}$ where there exists some slot $s \in S \subseteq [N]$ where vk_s is uncorrupted (i.e., honestly-generated), and moreover, the adversary did not request a Type-s signature on m_s under vk_s. For ease of exposition, we consider a selective version of the security game where the adversary commits to both the slot index $s^* \in [N]$ as well the challenge message m_{s^*} associated with slot s^* at the beginning of the security game. Note that we can generically lift the scheme to an adaptively-secure one by having the reduction guess the slot index s^* (incurring a $1/N$ loss) as well as the challenge message m^* (incurring a $1/2^\lambda$ loss[5]) and then relying on sub-exponential hardness. With complexity leveraging, the scheme parameters would scale with $\mathsf{poly}(\lambda, \log N)$, which maintain our succinctness property. Alternatively, we could also replace the Boneh-Boyen hash function [BB04] implicitly used in our construction with the Waters hash function [Wat05] and adapt his techniques to directly argue adaptive security. We elected to focus on the simpler case of selective security to highlight the techniques novel to our construction.

Slot-specific user keys. The basic scheme described above supports aggregation, but does not achieve the security property we described above. The problem is that the verification equation (Eq. (1.2)) for aggregate signatures is agnostic to the association between verification keys and messages. For instance, consider the following two scenarios where we have two users with signing keys α_1 and α_2:

- User 1 creates a Type-1 signature σ_1 on m_1 with randomness r_1 and User 2 creates a Type-2 signature σ_2 on m_2 with randomness r_2.
- User 1 creates a Type-2 signature σ'_1 on m_2 with randomness r_1 and User 2 creates a Type-1 signature σ'_2 on m_1 with randomness r_2.

[5]Without loss of generality, it suffices to support signatures on λ-bit messages as we can always compose with a collision-resistant hash function to support signing longer messages.

This yields the following signatures:

$$\sigma_1 = \left(g^{\alpha_1}(u_1^{m_1}h_1)^{r_1},\ g^{r_1},\ u_2^{r_1},\ h_2^{r_1}\right) \qquad \sigma_1' = \left(g^{\alpha_1}(u_2^{m_2}h_2)^{r_1},\ g^{r_1},\ u_1^{r_1},\ h_1^{r_1}\right)$$
$$\sigma_2 = \left(g^{\alpha_2}(u_2^{m_2}h_2)^{r_2},\ g^{r_2},\ u_1^{r_2},\ h_1^{r_2}\right) \qquad \sigma_2' = \left(g^{\alpha_2}(u_1^{m_1}h_1)^{r_2},\ g^{r_2},\ u_2^{r_2},\ h_2^{r_2}\right).$$

Aggregating both (σ_1, σ_2) and (σ_1', σ_2') in this case yields the same aggregate signature σ_{agg}:

$$\sigma_{\mathsf{agg}} = \left(g^{\alpha_1+\alpha_2}(u_1^{m_1}h_1u_2^{m_2}h_2)^{r_1+r_2},\ g^{r_1+r_2}\right).$$

Essentially, in the above scheme, σ_{agg} binds to a pair of verification keys $\mathsf{vk}_1 = e(g,g)^{\alpha_1}$ and $\mathsf{vk}_2 = e(g,g)^{\alpha_2}$ as well as a pair of messages m_1, m_2, but it does not say whether which verification key vk_1 or vk_2 a particular message is associated. A normal aggregate signature scheme should bind each message to a specific verification key.

The basic scheme described above does bind each message to a specific slot. Namely, if a message m is associated with slot s, then the "hash" of the message used during verification is $(u_s^m h_s)$, where (u_s, h_s) are the public parameters associated with slot s. The problem is that the verification keys are independent of s. A simple way to fix this is to have each user generate N different verification keys, one associated with each slot. For instance, a user's verification key could be $e(g,g)^{\alpha_1}, \ldots, e(g,g)^{\alpha_N}$. When verifying a signature on the set $\{(s, \mathsf{vk}_s, m_s)\}$, where $\mathsf{vk}_s = (\mathsf{vk}_{s,1}, \ldots, \mathsf{vk}_{s,N})$, the aggregated verification key associated with this set would now be $\prod_{s\in S} \mathsf{vk}_{s,s}$. The aggregated verification key establishes an association between verification keys and slots. Since there is already a binding between slots and messages, this combination establishes a binding between verification keys and messages. This in turn suffices for security.

While having each user sample N independent verification keys suffices for correctness and security, it results in long verification keys. We observe that we can use the pairing to compress them. Namely, instead of having the user verification keys be $e(g,g)^{\alpha_1}, \ldots, e(g,g)^{\alpha_N}$, where $\alpha_1, \ldots, \alpha_N$ are uniform, we instead set them in a correlated manner. Namely, we take $\alpha_s = \alpha a_s$ where each $a_s \xleftarrow{R} \mathbb{Z}_p$ is a fixed value determined by the public parameters. To facilitate this, the user would publish g^α as their verification key and the public parameters would include $A_s = g^{a_s}$ for all $s \in [N]$. Given $\mathsf{vk} = g^\alpha$, the verification algorithm computes $e(\mathsf{vk}, A_s) = e(g,g)^{\alpha a_s}$, which is the user's verification key associated with slot s. With this optimization, we reduce the size of each user's verification key back to a single group element while still maintaining a binding between verification keys and messages. We give the full scheme in Construction 3.3.

Proving security. Security of our construction relies on the computational Diffie-Hellman (CDH) assumption in $\mathbb{G}$. When we instantiate the scheme over asymmetric groups, we rely on the bilateral version of CDH where the CDH challenge is given out in *both* base groups. As noted above, we consider selective security where the adversary commits to both a slot $s^* \in [N]$ and a challenge message m^* at the beginning of the security game. Its goal is to forge an aggregate signature where slot s^* is associated with message m^* and the challenge verification key. In the security proof, we program the CDH challenge into the public parameters for the chosen slot s^*. We rely on a similar cancellation trick used to simulate identity keys in the Boneh-Boyen IBE scheme to answer the signing queries. We refer to Section 3.1, and specifically, the proof of Theorem 3.5 for the full details.

Lifting a slotted scheme into an unslotted scheme. The limitation of a slotted aggregate signature scheme is it only supports a list of signatures associated with distinct slots. However, if there is no coordination among signers, it is not clear how to enforce this distinct-slot requirement. Our second contribution in this work is a generic compiler that transforms a slotted aggregate signature scheme into a standard aggregate signature scheme that supports aggregating an a priori bounded number of signatures where there is no slot restriction. Our approach follows the methodology from [GLWW23], who described a similar compiler to transform a distributed broadcast encryption scheme into a flexible broadcast encryption scheme. They describe a solution based on bipartite matching. A similar approach applies in our setting. A caveat of the approach in [GLWW23] is they rely on random oracles for the analysis. Since we focus on the plain model in this work, we describe an alternative realization of the [GLWW23] idea using expander graphs [GUV07]. This allows us to generically transform the slotted aggregate signature scheme into an unslotted aggregate signature scheme with only modest overhead (and without needing random oracles). Our setup is as follows:

- Suppose we want to support aggregation for up to N signatures. We will use a slotted aggregate signature scheme with $M \geq N$ slots.

- To sign a message m, the signer will choose a set of slots $S \subseteq [M]$ and generates a signature on m with respect to each slot $s \in S$. The resulting signature is the collection $\sigma = \{(s, \sigma_s)\}_{s \in S}$.

- Suppose we have N signatures $\sigma_1, \ldots, \sigma_N$ where $\sigma_i = \{(s, \sigma_{i,s})\}_{s \in S_i}$. To aggregate these signatures, the goal is to "assign" each signature σ_i to a slot $s_i \in S_i$ such that $s_1, \ldots, s_N$ are all distinct. Similar to [GLWW23], we can model this as a bipartite matching problem. Specifically, we consider a bipartite graph with N nodes on the left (corresponding to the N signatures) and M nodes on the right (corresponding to the slots for the slotted signature scheme). There is an edge between node i on the left and node j on the right if $j \in S_i$ (i.e., the i^{th} signature σ_i contains a slotted signature in slot j). If there is a way to associate a unique slot with each signature, then we can invoke the aggregation algorithm for the underlying slotted scheme to obtain the final aggregate signature.

The question now is choosing the sets $S_i \subseteq [M]$ associated with individual signatures so as to guarantee that for *any* choice of N (valid) signatures, there always exists a complete matching in the associated bipartite graph.[6] We describe two approaches:

- The simplest approach is to take $M = N$ (i.e., use a slotted scheme with N slots) and require that each signature includes a signature for *every* slot (i.e., $S = [M]$). In this case, the graph associated with any collection of signatures is complete, and as such, a complete matching always exists. Thus, correctness is immediate in this case. The drawback, of course, is that this approach blows up the size of individual signatures by a factor of N. When applied to our pairing-based construction, this yields a construction where the public parameters contain $O(N)$ group elements, normal signatures contain $O(N^2)$ group elements, and the aggregate signature consists of two group elements.

- We then show a more efficient approach where instead of including a signature for *every* slot, we choose the slots based on the edges of an expander graph. In particular, the induced bipartite graph associated with any set of N signatures is an expander, and by setting the parameters properly, we can ensure that a complete matching always exists. If we instantiate the expander graph using the explicit construction of [GUV07], a complete matching exists so long as the number of slots satisfies $M = N^{1+\alpha} \cdot \mathsf{poly}(\lambda, \log N)$ and each signature consists of $D = \mathsf{poly}(\lambda, \log N)$ signatures for the underlying slotted scheme. Here, $\alpha > 0$ can be any constant (and where the $\mathsf{poly}(\cdot)$ factors depend on α; see Corollary 4.16). When applied to our pairing-based construction, this yields a construction where the public parameters contain $N^{1+\alpha} \cdot \mathsf{poly}(\lambda, \log N)$ group elements, normal signatures contain $N \cdot \mathsf{poly}(\lambda, \log N)$ group elements, and the aggregate signature consists of two group elements.

We describe our approach and these two instantiations in Section 4. We refer to Corollaries 4.11 and 4.16 for the specific instantiations of our approach. Taken together, we obtain an aggregate signature scheme that supports aggregating an a priori bounded number of signatures. Individual signatures in our scheme grow with the bound, but the aggregated signature is short (consisting of just two group elements).

1.2 Additional Related Work

Aggregate signatures in the plain model. A natural question is whether we can directly prove security of the classic pairing-based (aggregate) signature schemes from [BLS01, BGLS03] *without* the random oracle heuristic. This would immediately give an aggregatable signature in the plain model. However, this seems to be a challenging question. Namely, the [BLS01] signature scheme (the basis for the aggregate signature scheme in [BGLS03]) can be viewed as a "full domain hash" signature. There have been some attempts to argue security of such schemes in the plain model, but so far, existing approaches have either required multilinear maps [FHPS13, HSW13] or indistinguishability obfuscation [HSW14]. This is the case regardless of whether one is aiming for adaptive security or a relaxed notion such as selective or static security. Proving selective security of [BLS01, BGLS03] in the plain model from simple pairing-based assumptions like CDH would be a major breakthrough in this area.

[6] Note that because we want aggregation to succeed for *any* set of valid signatures, this rules out the strategy of having the signer pick $D < N$ slots at random. If we allow the signer to choose slots uniformly at random, then there will always exist a collection of N (valid) signatures that occupy the same set of slots. If we work in the random oracle, then we can derive the slots deterministically via the random oracle, which does yield a viable strategy. The work of [GLWW23] take such an approach in the context of flexible broadcast encryption. Since we focus exclusively on plain model constructions in this work, we opt for a different approach.

Interactive aggregation. A number of works have studied signature schemes that support *interactive* aggregation [BN06, MPSW19, DEF+19, BK20]. In this setting, signers are allowed to interact with each other to construct an aggregate signature. Our focus in this work is on non-interactive aggregation.

Other notions of aggregation. Many relaxations of aggregate signatures have been proposed in order to enable simple or more efficient constructions in the plain model. In a synchronized aggregate signature [GR06, AGH10, LLY13, HW18, FSZ22, KS23], signers are assumed to share a synchronized clock and aggregation is possible for signatures generated in the same epoch. Moreover, there is an additional assumption that each users signs at most one message per epoch. Sequential aggregate signatures [LMRS04, LOS+06, BGOY07] consider a different relaxation where the *signers* perform the aggregation. Namely, signer i would take an aggregate signature σ'_{agg} on the first $i-1$ messages and output a new aggregate signature σ_{agg} on the first i messages. In the standard setting of aggregate signatures, we do not require interaction or coordination between signers.

Multisignatures [Ita83, OO99, MOR01, Bol03, LOS+06, BN06, BN07, RY07, BDN18, MPSW19, DGNW20, FSZ22, WTW+24, BPW25] are a special case of aggregate signatures where we only support aggregating signatures from multiple parties on the *same* message. Finally, Goyal and Vaikuntanathan [GV22] consider an aggregate signature scheme with the opposite limitation where one can aggregate signatures on different messages but only from a single party. They give a direct bilinear map construction in this setting where the number of signatures that can be aggregated is a priori bounded. Security in this case can be proven in the plain model if the attacker declares all of its signing queries ahead of time (i.e., the scheme satisfies static security).

In this work, we focus on the standard notion of aggregate signatures where there are no restrictions on how signatures are aggregated (other than the bound on the number of signatures that can be aggregated).

Registration-based cryptography. Our techniques are also conceptually similar to techniques used in recent works on registration-based cryptography [GHMR18]. At a high level, the goal in registration-based cryptography is to aggregate public keys from multiple independent parties into a single short key. Many recent pairing-based and lattice-based schemes for registration-based cryptography [HLWW23, ZZGQ23, FFM+23, KMW23, GLWW24, AT24, CW24, BLM+24, CHW25, WW25] rely on a similar cross-term technique to facilitate key aggregation and decryption. In each of these schemes, users are associated with a slot, and each user's public key includes cross terms that are a function of the user's key-generation secret key and the slot-specific components in the scheme parameters. In our setting, we also start with a slotted scheme and rely on having the user include cross terms in their signatures in order to facilitate aggregation.

In the setting of registration-based cryptography, a number of works have also studied way to lift from a slotted scheme (where users register keys associated with a slot) to a general scheme where there are no such restrictions. These include approaches based on the powers-of-two trick [GHMR18, GHM+19, HLWW23], bipartite matching [GLWW23], or cuckoo hashing [FKdP23]. In this work, we leverage the bipartite matching approach from [GLWW23], who previously considered it in the setting of distributed broadcast encryption.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in the parameter λ and $\mathsf{negl}(\lambda)$ to denote a negligible function in λ (i.e., a function $f(\lambda)$ where $f = o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$).

Prime-order pairing groups. We recall the notion of a prime-order pairing group and the bilateral computational Diffie-Hellman (CDH) assumption we use in this work. The bilateral CDH assumption is essentially the standard CDH assumption over asymmetric groups where the challenge is given out in both groups.

Definition 2.1 (Prime-Order Bilinear Group)**.** An (asymmetric) prime-order group generator GroupGen is an efficient algorithm that takes as input the security parameter 1^λ and outputs a description $\mathcal{G} = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, p, g, g', e)$ consisting of groups $\mathbb{G}$, $\mathbb{G}'$, and $\mathbb{G}_T$, each of prime order $p = 2^{\Theta}(\lambda)$, and where g is a generator of $\mathbb{G}$, g' is a generator of $\mathbb{G}'$, and $e\colon \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$ is a non-degenerate bilinear map. We additionally require that the group operation in $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$

and the pairing operation e are efficiently-computable. We assume that GroupGen outputs a *fixed* prime $p = p(\lambda)$ for each security parameter $\lambda \in \mathbb{N}$.

Assumption 2.2 (Bilateral Computational Diffie-Hellman)**.** Let GroupGen be a prime-order group generator. We say that the bilateral computational Diffie-Hellman (bilateral CDH) assumption holds with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathcal{A}(\mathcal{G}, g^x, g^y, (g')^x, (g')^y) = g^{xy}] = \mathsf{negl}(\lambda),$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, p, g, g', e) \leftarrow \mathsf{GroupGen}(1^\lambda)$, and $x, y \xleftarrow{R} \mathbb{Z}_p$.

3 Slotted Aggregate Signatures

In this section, we give the formal definition of a slotted aggregate signature scheme. As described in Section 1.1, in a slotted aggregate signature scheme, the signer associates a signature with a specific slot, and the aggregation algorithm only works for signatures assigned to *distinct* slots. We give the formal definition below:

Definition 3.1 (Slotted Aggregate Signatures)**.** A slotted aggregate signature scheme on message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{SAS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{pp}$: On input the security parameter λ and the number of slots N, the setup algorithm outputs the public parameters pp. We assume that pp includes a description of 1^λ and 1^N.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: On input the public parameters pp, the key-generation algorithm outputs a verification key vk and a signing key sk.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m, s) \to \sigma$: On input the public parameters pp, the signing key sk, a message $m \in \mathcal{M}$, and a slot $s \in [N]$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, s, \sigma) \to b$: On input the public parameters pp, a verification key vk, a message $m \in \mathcal{M}$, a slot $s \in [N]$, and a signature σ, the verification algorithm outputs $b = 1$ if the signature is valid and $b = 0$ otherwise.
- $\mathsf{Aggregate}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s, \sigma_s)\}_{s \in S}) \to \sigma_{\mathsf{agg}}$: On input the public parameters pp, a list of verification keys vk_s, messages $m_s \in \mathcal{M}_\lambda$, and signatures σ_s for $s \in S$ where $S \subseteq [N]$, the aggregation algorithm outputs an aggregate signature σ_{agg} (or a special symbol $\perp$ to indicate a failure).
- $\mathsf{AggVerify}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\mathsf{agg}}) \to b$: On input the public parameters pp, a list of verification keys vk_s and messages $m_s \in \mathcal{M}_\lambda$ for $s \in S$ where $S \subseteq [N]$, and a signature σ_{agg}, the aggregate-verification algorithm outputs $b = 1$ if σ_{agg} is a valid signature and $b = 0$ otherwise.

We require Π_{SAS} to satisfy the following correctness, succinctness, and unforgeability properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all polynomials $N = \mathsf{poly}(\lambda)$, all slot indices $s \in [N]$, and all messages $m \in \mathcal{M}_\lambda$,

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, s, \sigma) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N); \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m, s) \end{array}\right] = 1.$$

  In addition, for all pp in the support of $\mathsf{Setup}(1^\lambda, 1^N)$ and all collections $\{(s, \mathsf{vk}_s, m_s, \sigma_s)\}_{s \in S}$ where $S \subseteq [N]$ and

$$\forall s \in S : \mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_s, m_s, s, \sigma_s) = 1,$$

  we have that

$$\Pr\left[\mathsf{AggVerify}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\mathsf{agg}}) = 1\right] = 1,$$

  where $\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s, \sigma_s)\})$.

- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot)$ such that in the completeness experiment above, the size of the aggregate signature σ_{agg} satisfies $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda, \log N)$.
- **Unforgeability:** We define unforgeability against a fully malicious adversary that is allowed to choose arbitrary verification keys other than the target one. Formally, we begin by defining the unforgeability game, which is parameterized by a security parameter λ and an adversary $\mathcal{A}$:
  - **Setup:** The challenger gives 1^λ to $\mathcal{A}$ and receives from $\mathcal{A}$ the number of slots 1^N. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. It sends $(\mathsf{pp}, \mathsf{vk}^*)$ to the adversary $\mathcal{A}$.
  - **Signing queries:** The adversary can now make adaptive signing queries. On each query, the adversary specifies a slot $s \in [N]$ and a message $m \in \mathcal{M}_\lambda$. The challenger responds with $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, m, s)$.
  - **Output:** At the end of the game, the adversary outputs a pair $(\{(s, \mathsf{vk}_s, m_s)\}_{s\in S}, \sigma_{\mathsf{agg}})$ where $S \subseteq [N]$. The challenger outputs 1 if the following conditions hold:
    - $\mathsf{AggVerify}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s)\}_{s\in S}, \sigma_{\mathsf{agg}}) = 1$.
    - There exists some $s \in S$ where $\mathsf{vk}_s = \mathsf{vk}^*$ and moreover, algorithm $\mathcal{A}$ did not make a signing query on slot s with message m_s.

    Otherwise, the challenger outputs 0.

  We say the signature scheme is unforgeable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the unforgeability game.

Definition 3.2 (Selective Security)**.** For a slotted aggregate signature scheme Π_{SAS}, we define the selective unforgeability game exactly as in Definition 3.1, except we require the adversary to output a slot index $s^* \in [N]$ and the message $m^* \in \mathcal{M}_\lambda$ at the beginning of the setup phase (*before* seeing the public parameters). The adversary in the selective unforgeability game wins if the conditions in Definition 3.1 hold, and moreover, $s^* \in S$, $\mathsf{vk}_{s^*} = \mathsf{vk}^*$, and $m_{s^*} = m^*$. We say Π_{SAS} is *selectively-secure* if the advantage of any efficient adversary is bounded by $\mathsf{negl}(\lambda)$ in the selective unforgeability game.

3.1 Slotted Aggregate Signatures in the Plain Model

In this section, we describe our slotted aggregate signature scheme from the bilateral CDH assumption in a prime-order pairing group.

Construction 3.3 (Slotted Aggregate Signature)**.** Let GroupGen be an asymmetric prime-order group generator (Definition 2.1). Let $p = p(\lambda)$ be the order of the groups output by GroupGen. We construct a slotted aggregate signature scheme $\Pi_{\mathsf{SAS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ with message space $\mathcal{M} = \{\mathbb{Z}_{p(\lambda)}\}_{\lambda\in\mathbb{N}}$ as follows:[7]

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of slots N, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, p, g, g', e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. For each $i \in [N]$, the setup algorithms samples $a_i, v_i, t_i \xleftarrow{R} \mathbb{Z}_p$ and sets
$$u_i = g^{v_i},\ h_i = g^{t_i},\ u_i' = (g')^{v_i},\ h_i' = (g')^{t_i},\ A_i = g^{a_i}.$$
It outputs the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u_i', h_i', A_i)\}_{i\in[N]})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u_i', h_i', A_i)\}_{i\in[N]})$ the key-generation algorithm samples $\alpha \xleftarrow{R} \mathbb{Z}_p$. It outputs the verification key $\mathsf{vk} = (g^\alpha, (g')^\alpha)$ and the signing key $\mathsf{sk} = \alpha$.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m, s)$: On input the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u_i', h_i', A_i)\}_{i\in[N]})$, the signing key $\mathsf{sk} = \alpha$, a message $m \in \mathbb{Z}_p$, and a slot index $s \in [N]$, the signing algorithm samples $r \xleftarrow{R} \mathbb{Z}_p$ and computes randomization components $U_j = u_j^r$ and $H_j = h_j^r$ for all $j \neq s$. Then it outputs the signature
$$\sigma = \left(A_s^\alpha \cdot (u_s^m h_s)^r,\ g^r,\ \{(j, U_j, H_j)\}_{j\neq s}\right).$$

[7] Note that we can support signing arbitrary messages by first hashing into $\mathbb{Z}_p$ using any collision-resistant hash function.

- Verify(pp, vk, m, s, σ): On input the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u'_i, h'_i, A_i)\}_{i\in[N]})$, the verification key $\mathsf{vk} = (V, V')$, a message $m \in \mathbb{Z}_p$, a slot index $s \in [N]$, and a signature $\sigma = (\sigma_1, \sigma_2, \{(j, U_j, H_j)\}_{j\neq s})$, the verification algorithm checks the following conditions:
  - $e(g, V') = e(V, g')$;
  - $e(\sigma_1, g') = e(A_s, V') \cdot e(\sigma_2, (u'_s)^m h'_s)$; and
  - for all $j \in [N]$ where $j \neq s$, $e(U_j, g') = e(\sigma_2, u'_j)$ and $e(H_j, g') = e(\sigma_2, h'_j)$.

  If all checks pass, the verification algorithm outputs 1; otherwise, it outputs 0.
- Aggregate(pp, $\{(s, \mathsf{vk}_s, m_s, \sigma_s)\}_{s\in S}$): On input the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u'_i, h'_i, A_i)\}_{i\in[N]})$, a list of verification keys vk_s, messages $m_s \in \mathbb{Z}_p$, and signatures $\sigma_s = (\sigma_{s,1}, \sigma_{s,2}, \{(j, U_{s,j}, H_{s,j})\}_{j\neq s})$, the aggregation algorithm proceeds as follows:
  - Compute $\gamma_2 = \prod_{s\in S} \sigma_{s,2} \in \mathbb{G}$.
  - Then for $s \in S$, compute an intermediate value $\delta_s \in \mathbb{G}$ where
$$\delta_s = \sigma_{s,1} \cdot \prod_{j\neq s} U_{j,s}^{m_s} H_{j,s}$$
    Finally, it computes $\gamma_1 = \prod_{s\in S} \delta_s \in \mathbb{G}$.

  Finally it outputs the aggregate signature $\sigma_{\mathrm{agg}} = (\gamma_1, \gamma_2)$.
- AggVerify(pp, $\{(s, \mathsf{vk}_s, m_s)\}_{s\in S}, \sigma_{\mathrm{agg}}$): On input the public parameters $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u'_i, h'_i, A_i)\}_{i\in[N]})$, a list of verification keys $\mathsf{vk}_s = (V_s, V'_s)$ and messages $m_i \in \mathbb{Z}_p$, and an aggregate signature $\sigma_{\mathrm{agg}} = (\gamma_1, \gamma_2)$, the verification algorithm outputs 1 if for all $s \in S$, $e(g, V'_s) = e(V_s, g')$, and moreover,
$$e(\gamma_1, g') = \prod_{s\in S} \left(e(A_s, V'_s) \cdot e(\gamma_2, (u'_s)^{m_s} h'_s)\right). \tag{3.1}$$
  Otherwise, it outputs 0.

Theorem 3.4. *Construction 3.3 is correct.*

Proof. Take any $\lambda \in \mathbb{N}$ and $N = \mathrm{poly}(\lambda)$. Take $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u'_i, h'_i, A_i)\}_{i\in[N]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Take any slot $s \in [N]$ and any message $m \in \mathbb{Z}_p$. Let $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m, s)$. By construction, $u_i = g^{v_i}$, $h_i = g^{t_i}$, $u'_i = (g')^{v_i}$, and $h'_i = (g')^{t_i}$. In addition, $\mathsf{sk} = \alpha$, $\mathsf{vk} = (V, V') = (g^\alpha, (g')^\alpha)$ and

$$\sigma = (\sigma_1, \sigma_2, \{(j, U_j, H_j)\}_{j\neq s}) = \left(A_s^\alpha \cdot (u_s^m h_s)^r,\ g^r,\ \{(j, U_j, H_j)\}_{j\neq s}\right),$$

where $U_j = u_j^r$ and $H_j = h_j^r$, and $r \in \mathbb{Z}_p$ is the signing randomness. By definition, $e(V, g') = e(g, g')^\alpha = e(g, V')$. Next, it follows that

$$\begin{aligned} e(\sigma_1, g') &= e(A_s^\alpha \cdot (u_s^m h_s)^r, g') \\ &= e(A_s^\alpha, g') \cdot e((u_s^m h_s)^r, g') \\ &= e(A_s, (g')^\alpha) \cdot e(g^r, (u'_s)^m h'_s) \\ &= e(A_s, V') \cdot e(\sigma_2, (u'_s)^m h'_s) \end{aligned}$$

which matches the verification equation. Next, for all $j \neq s$, we also have

$$\begin{aligned} e(U_j, g') &= e(u_j^r, g') = e(g^r, u'_j) = e(\sigma_2, u'_j) \\ e(H_j, g') &= e(h_j^r, g') = e(g^r, h'_j) = e(\sigma_2, h'_j). \end{aligned}$$

Thus, $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, s, \sigma) = 1$. We now move on to verifying the correctness of aggregate signatures. Take any collection $\{(s, \mathsf{vk}_s, m_s, \sigma_s)\}_{s \in S}$ where

$$\forall s \in S : \mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_s, m_s, s, \sigma_s) = 1.$$

Let $\sigma_{\mathsf{agg}} = (\gamma_1, \gamma_2) \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s, \sigma_s)\})$ and consider $\mathsf{AggVerify}(\mathsf{pp}, \{(s, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\mathsf{agg}})$. Write $\mathsf{vk}_s = (V_s, V_s')$. Since $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_s, m_s, \sigma_s) = 1$, this means $e(V_s, g') = e(g, V_s')$ for all $s \in S$, so the first verification requirement is satisfied. Consider now the second relation (Eq. (3.1)). Write $\sigma_s = (\sigma_{s,1}, \sigma_{s,2}, \{(j, U_{s,j}, H_{s,j})\}_{j \neq s})$ and $\sigma_{s,2} = g^{r_s}$ for some $r_s \in \mathbb{Z}_p$. Again, since $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_s, m_s, \sigma_s) = 1$, the following properties must hold:

- Since $e(\sigma_{s,1}, g') = e(A_s, V_s') \cdot e(\sigma_{s,2}, (u_s')^{m_s} h_s')$, this means

  $$e(\sigma_{s,1}, g') = e(g^{a_s}, (g')^{\alpha_s}) e(g^{r_s}, (g')^{m_s v_s + t_s}) = e(g^{a_s \alpha_s + r_s (m_s v_s + t_s)}, g').$$

  This means

  $$\sigma_{s,1} = g^{a_s \alpha_s + r_s (m_s v_s + t_s)} = A_s^{\alpha_s} (u_s^{m_s} h_s)^{r_s}.$$

- Since $e(U_{s,j}, g') = e(\sigma_{s,2}, u_j')$, this means

  $$e(U_{s,j}, g') = e(g^{r_s}, (g')^{v_j}) = e(g^{r_s v_j}, g').$$

  Thus, $U_{s,j} = g^{r_s v_j} = u_j^{r_s}$. Similarly, since $e(H_{s,j}, g') = e(\sigma_{s,2}, h_j')$, this means $H_{s,j} = h_j^{r_s}$.

Let $r^* = \sum_{s \in S} r_s$. Then, the Aggregate algorithm computes

$$\gamma_2 = \prod_{s \in S} \sigma_{s,2} = \prod_{s \in S} g^{r_s} = g^{r^*}.$$

Next, for each $s \in S$, it also computes

$$\begin{aligned}
\delta_s &= \sigma_{s,1} \cdot \prod_{j \neq s} U_{j,s}^{m_s} H_{j,s} \\
&= A_s^{\alpha_s} (u_s^{m_s} h_s)^{r_s} \cdot \prod_{j \neq s} u_s^{m_s r_j} h_s^{r_j} \\
&= A_s^{\alpha_s} (u_s^{m_s} h_s)^{r_s} \cdot (u_s^{m_s})^{\sum_{j \neq s} r_j} \cdot h_s^{\sum_{j \neq s} r_j} \\
&= A_s^{\alpha_s} (u_s^{m_s} h_s)^{r_s} \cdot (u_s^{m_s})^{r^* - r_s} \cdot h_s^{r^* - r_s} \\
&= A_s^{\alpha_s} (u_s^{m_s} h_s)^{r_s} \cdot (u_s^{m_s} h_s)^{r^* - r_s} \\
&= A_s^{\alpha_s} (u_s^{m_s} h_s)^{r^*}
\end{aligned}$$

We can now think of the pair (δ_s, γ_2) as a signature on the message m_s under the *aggregated randomness* r^*. Finally, we have that

$$\gamma_1 = \prod_{s \in S} \delta_s = \prod_{s \in S} A_s^{\alpha_s} (u_s^{m_s} h_s)^{r^*}.$$

Finally, consider the aggregate verification relation (Eq. (3.1)):

$$\begin{aligned}
e(\gamma_1, g') &= e\left(\prod_{s \in S} A_s^{\alpha_s} (u_s^{m_s} h_s)^{r^*}, g' \right) \\
&= \prod_{s \in S} \left(e(A_s^{\alpha_s}, g') \cdot e((u_s^m h_s)^{r^*}, g') \right) \\
&= \prod_{s \in S} \left(e(A_s, (g')^{\alpha_s}) \cdot e(g^{r^*}, (u_s')^{m_s} h_s') \right) \\
&= \prod_{s \in S} \left(e(A_s, V_s') \cdot e(\gamma_2, (u_s')^{m_s} h_s') \right).
\end{aligned}$$

Corresponding, Eq. (3.1) holds and AggVerify outputs 1, as required. □

Theorem 3.5 (Selective Unforgeability)**.** *Suppose the bilateral CDH assumption (Assumption 2.2) holds with respect to* GroupGen*. Then, Construction 3.3 satisfies selective unforgeability.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that wins the selective unforgeability game (Definition 3.2) with non-negligible probability ε. We construct a reduction algorithm $\mathcal{B}$ that solves the bilateral CDH problem with respect to GroupGen. Algorithm $\mathcal{B}$ works as follows:

- **Initialization:** Algorithm $\mathcal{B}$ receives the security parameter 1^λ and the bilateral CDH challenge $(\mathcal{G}, X, X', Y, Y)$, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, p, g, g', e)$, $X = g^x$, $X' = (g')^x$, $Y = g^y$, and $Y' = (g')^y$, and $x, y \xleftarrow{R} \mathbb{Z}_p$. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ responds by outputting the number of slots 1^N, a challenge index $s^* \in [N]$, and a challenge message $m^* \in \mathbb{Z}_p$.
- **Setup:** For each $i \in [N]$, algorithm $\mathcal{B}$ samples random integers $v_i, t_i, a_i \xleftarrow{R} \mathbb{Z}_p$. Then for all $i \in [N] \setminus \{s^*\}$ it sets
$$u_i = g^{v_i},\ u_i' = (g')^{v_i},\ h_i = g^{t_i},\ h_i' = (g')^{t_i},\ A_i = g^{a_i}$$
as in the normal setup algorithm. For index s^*, it sets
$$u_{s^*} = Xg^{v_{s^*}},\ u_{s^*}' = X'(g')^{v_{s^*}},\ h_{s^*} = X^{-m^*} g^{t_{s^*}},\ h_{s^*}' = (X')^{-m^*}(g')^{t_{s^*}},\ A_{s^*} = X.$$
In particular, the challenge message m^* is "programmed" into the parameters for the challenge slot s^*. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (\mathcal{G}, \{(i, u_i, h_i, u_i', h_i', A_i)\}_{i \in [N]})$ and $\mathsf{vk}^* = (Y, Y')$ to $\mathcal{A}$.
- **Signing queries:** Whenever $\mathcal{A}$ makes a signing query on a slot $s \in [N]$ and a message $m \in \mathbb{Z}_p$ (where $(s, m) \neq (s^*, m^*)$), algorithm $\mathcal{B}$ proceeds as follows:
  - **Case 1:** For a query (s, m) where $s \neq s^*$, algorithm $\mathcal{B}$ computes the signature by sampling $r \xleftarrow{R} \mathbb{Z}_p$. Then, for each $j \neq s$, it computes $U_j = u_j'^r$ and $H_j = h_j'^r$. Finally, it responds to $\mathcal{A}$ with the signature
$$\sigma = \left(Y^{a_s} \cdot (u_s^m h_s)^r,\ g^r,\ \{(j, U_j, H_j)\}_{j \neq s}\right).$$
This is nearly identical to the actual signing algorithm except algorithm replaces A_s^α with Y^{a_s}. This change does not effect the distribution of signatures since in vk^* we implicitly set $\alpha = y$. Thus $A_s^\alpha = A_s^y = g^{a_s y} = Y^{a_s}$. In the actual signing algorithm, the signer knows the discrete log α of the verification key. The reduction algorithm (which cannot know the signing key $\alpha = y$) compensates by using knowledge of the exponent a_s associated with A_s instead.
  - **Case 2:** For a query (s, m) where $s = s^*$ and $m \neq m^*$, algorithm $\mathcal{B}$ cannot compute $A_{s^*}^y = X^y = g^{xy}$ itself, since this is the solution to the bilateral CDH problem. Thus, to construct a signature on a message $m \neq m^*$, it will sample the signing randomness in a careful way that cancels out this component in a spirit similar to the Boneh-Boyen identity-based encryption (IBE) scheme [BB04]. The reduction algorithm does this in two steps:
    - First, algorithm $\mathcal{B}$ will deterministically construct a signature on the message m (using knowledge of the exponents v_i, t_i algorithm $\mathcal{B}$ chose at the beginning). However, the structure of this signature reveals a correlation with m^* being planted in the public parameters (i.e., it does not have the correct distribution).
    - In the second step, the reduction algorithm re-randomizes the signature from the first step to ensure that the resulting signature has the correct distribution. Unlike the first step, this second re-randomization step only requires knowledge of the public parameters and not the exponents v_i, t_i.

    Concretely, algorithm $\mathcal{B}$ works as follows:
    - The reduction algorithm computes
$$\begin{aligned} \tilde{\sigma}_2 &= Y^{-1/(m-m^*)} \\ \tilde{\sigma}_1 &= \tilde{\sigma}_2^{m v_{s^*} + t_{s^*}} = Y^{-(1/(m-m^*))(m v_{s^*} + t_{s^*})} \\ \forall j \neq s^* : \tilde{U}_j &= \tilde{\sigma}_2^{v_j} = Y^{-(1/(m-m^*))v_j} \\ \forall j \neq s^* : \tilde{H}_j &= \tilde{\sigma}_2^{t_j} = Y^{-(1/(m-m^*))t_j}. \end{aligned}$$

These terms are efficiently computable by $\mathcal{B}$ since they are simply raising Y to an exponent known to $\mathcal{B}$. We remark that $(1/(m-m^*))$ is well defined in $\mathbb{Z}_p$ since $m \neq m^*$. Observe that

$$\begin{aligned} A_{s^*}^{y} \cdot (u_{s^*}^{m} h_{s^*})^{-y/(m-m^*)} &= g^{xy} (X^{m-m^*} g^{m v_{s^*} + t_{s^*}})^{-y/(m-m^*)} \\ &= g^{(m v_{s^*} + t_{s^*})(-y/(m-m^*))} \\ &= \tilde{\sigma}_2^{m v_{s^*} + t_{s^*}} = \tilde{\sigma}_1. \end{aligned}$$

In particular, this means that

$$\tilde{\sigma} = \left(\tilde{\sigma}_1,\ \tilde{\sigma}_2,\ \{(j, \tilde{U}_j, \tilde{H}_j)\}_{j \neq s}\right)$$

is a well-defined signature with randomness $r = -y/(m-m^*)$.

* Now $\mathcal{B}$ computes the final signature by re-randomizing $\tilde{\sigma}$. Specifically, it samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. It then computes

$$\sigma_1 = \tilde{\sigma_1} \cdot (u^m h)^{\tilde{r}},\ \sigma_2 = \tilde{\sigma_2} \cdot g^{\tilde{r}}$$

and for all $j \neq s^*$, it computes $U_j = \tilde{U}_j u_j^{\tilde{r}}$ and $H_j = \tilde{H}_j h_j^{\tilde{r}}$.

Algorithm $\mathcal{B}$ replies with the signature $\sigma = (\sigma_1, \sigma_2, \{U_j, H_j\}_{j \neq s^*})$. By construction, σ is a signature with randomness $r = -y/(m-m^*) + \tilde{r}$.

- **Output:** At the end of the game, the attacker outputs a list $\{(s, \mathsf{vk}_s, m_s)\}_{s \in S}$ where $S \subseteq [N]$ together with a signature $\sigma_{\mathsf{agg}} = (\gamma_1, \gamma_2)$. The reduction $\mathcal{B}$ first checks that
  - $\mathsf{AggVerify}(\mathsf{pp}, \{(x, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\mathsf{agg}}) = 1$.
  - $s^* \in S$ and $\mathsf{vk}_{s^*} = \mathsf{vk}^*$ and $m_{s^*} = m^*$.

  Algorithm $\mathcal{B}$ rejects if either property does not hold. If both properties hold, then algorithm $\mathcal{B}$ parses each $\mathsf{vk}_s = (V_s, V_s')$ and computes the values

$$\tilde{\gamma}_1 = \gamma_1 \prod_{j \neq s^*} V_j^{-a_j} \quad \text{and} \quad \tilde{\gamma}_2 = \gamma_2,$$

  and outputs

$$\tilde{\gamma_1} \cdot \tilde{\gamma_2}^{-\sum_{s \in S}(v_s m_s + t_s)}.$$

By construction, algorithm $\mathcal{B}$ is efficient if $\mathcal{A}$ is efficient. Next, we argue that algorithm $\mathcal{B}$ currently simulates an execution of the unforgeability game. Consider first the distribution of the public parameters:

- First, the bilateral CDH challenger samples $\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$, exactly as in the real scheme. In addition, we can write $X = g^x$, $X' = (g')^x$, $Y = g^y$, and $Y' = (g')^y$, where the bilateral CDH challenger samples $x, y \xleftarrow{R} \mathbb{Z}_p$.
- Next, for all $i \neq s^*$ the reduction constructs $u_i, u_i', h_i, h_i', A_i$ exactly as in the real scheme.
- What remains is to analyze the *joint* distribution of $\mathsf{vk}^*, u_{s^*}, u_{s^*}', h_{s^*}, h_{s^*}', A_{s^*}$. By construction, the reduction algorithm sets

$$\begin{aligned} \mathsf{vk}^* &= \left(g^y, (g')^y\right) & A_{s^*} &= g^x \\ u_{s^*} &= X g^{v_{s^*}} = g^{x + v_{s^*}} & u_{s^*}' &= X' (g')^{v_{s^*}} = (g')^{x + v_{s^*}} \\ h_{s^*} &= X^{-m^*} g^{t_{s^*}} = g^{-m^* x + t_{s^*}} & h_{s^*}' &= (X')^{-m^*} (g')^{t_{s^*}} = (g')^{-m^* x + t_{s^*}}. \end{aligned}$$

  Let $\tilde{v}_{s^*} = x + v_{s^*}$, $\tilde{t}_{s^*} = -m^* x + t_{s^*}$, $\tilde{\alpha} = y$, and $\tilde{a}_{s^*} = x$. Since x, y, v_{s^*}, t_{s^*} are all uniform over $\mathbb{Z}_p$ (and sampled independently of all previous elements), the values $\tilde{\alpha}, \tilde{v}_{s^*}, \tilde{t}_{s^*}, \tilde{a}_{s^*}$ are uniform and independent over $\mathbb{Z}_p$. This means we can alternatively write

$$\mathsf{vk}^* = \left(g^{\tilde{\alpha}}, (g')^{\tilde{\alpha}}\right),\ u_{s^*} = g^{\tilde{v}_{s^*}},\ u_{s^*}' = (g')^{\tilde{v}_{s^*}},\ h_{s^*} = g^{\tilde{t}_{s^*}},\ h_{s^*}' = (g')^{\tilde{t}_{s^*}},\ A_{s^*} = g^{\tilde{a}_{s^*}}.$$

  This coincides with the distribution in the real scheme.

Consider now the signing queries. We again consider the two cases:

- **Case 1:** As described in the reduction, $\mathcal{B}$ constructs signatures in the same manner as the actual scheme except it replaces the term A_s^{α} with Y^{a_s}. Since $A_s^{\alpha} = A_s^{y} = g^{a_s y} = Y^{a_s}$, this produces an identical result.
- **Case 2:** As shown in the reduction algorithm itself, the signature σ algorithm $\mathcal{B}$ constructs can be viewed as a signature output by the signing algorithm with randomness $r = -y/(m - m^*) + \tilde{r}$. Since the reduction samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r is also uniform over $\mathbb{Z}_p$. This coincides with the distribution of the signatures in the real scheme.

We conclude that algorithm $\mathcal{B}$ perfectly simulates an execution of the selective unforgeability game for $\mathcal{A}$. This means with probability ε, algorithm $\mathcal{A}$ outputs $\{(s, \mathsf{vk}_s, m_s)\}_{s \in S}$ where $S \subseteq [N]$ together with a signature $\sigma_{\text{agg}} = (\gamma_1, \gamma_2)$ where

- $\mathsf{AggVerify}(\mathsf{pp}, \{(x, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\text{agg}}) = 1$.
- $s^* \in S$ and $\mathsf{vk}_{s^*} = \mathsf{vk}^*$ and $m_{s^*} = m^*$.

In this case, consider the output of algorithm $\mathcal{B}$. Write $\mathsf{vk}_s = (V_s, V_s')$. Since $\mathsf{AggVerify}(\mathsf{pp}, \{(x, \mathsf{vk}_s, m_s)\}_{s \in S}, \sigma_{\text{agg}}) = 1$, this means

$$\forall s \in S : e(g, V_s') = e(g', V_s), \tag{3.2}$$

and moreover,

$$e(\gamma_1, g') = \prod_{s \in S} \left(e(A_s, V_s') \cdot e(\gamma_2, (u_s')^{m_s} h_s') \right). \tag{3.3}$$

Since $\tilde{\gamma}_1 = \gamma_1 \cdot \prod_{j \neq s^*} V_j^{-a_j}$ and $\tilde{\gamma}_2 = \gamma_2$, we now appeal to Eqs. (3.2) and (3.3) to conclude that

$$\begin{aligned}
e(\tilde{\gamma}_1, g') \cdot \prod_{s \in S} e\big(\tilde{\gamma}_2, (u_s')^{m_s} h_s'\big)^{-1} &= e(\gamma_1, g') \cdot \prod_{j \neq s^*} e\big(V_j^{-a_j}, g'\big) \cdot \prod_{s \in S} e\big(\gamma_2, (u_s')^{m_s} h_s'\big)^{-1} \\
&= \prod_{s \in S} e(A_s, V_s') \cdot \prod_{j \neq s^*} e\big(V_j^{-a_j}, g'\big) \\
&= e(A_{s^*}, V_{s^*}') \cdot \prod_{j \neq s^*} \left(e(g, V_j')^{a_j} \cdot e(V_j, g')^{-a_j} \right) \\
&= e(A_{s^*}, V_{s^*}') \cdot \prod_{j \neq s^*} \left(e(V_j, g')^{a_j} \cdot e(V_j, g')^{-a_j} \right) \\
&= e(A_{s^*}, V_{s^*}') = e(X, Y') = e(g, g')^{xy}.
\end{aligned} \tag{3.4}$$

Now, for all $j \neq \{s^*\}$, we have that $(u_j')^{m_j} h_j' = (g')^{v_j m_j + t_j}$. Similarly,

$$(u_{s^*}')^{m^*} h_{s^*}' = (X')^{m^*} (g')^{m^* v_{s^*}} (X')^{-m^*} (g')^{t_{s^*}} = (g')^{v_{s^*} m^* + t_{s^*}} = (g')^{v_{s^*} m_{s^*} + t_{s^*}}$$

since $m_{s^*} = m^*$. It then follows that

$$\prod_{s \in S} e\big(\tilde{\gamma}_2, (u_s')^{m_s} h_s'\big)^{-1} = \prod_{s \in S} e\big(\tilde{\gamma}_2^{-(v_s m_s + t_s)}, g'\big) = e\big(\tilde{\gamma}_2^{-\sum_{s \in S}(v_s m_s + t_s)}, g'\big). \tag{3.5}$$

Combining Eqs. (3.4) and (3.5), we see that

$$\begin{aligned}
e(g^{xy}, g') &= e(\tilde{\gamma}_1, g') \cdot \prod_{s \in S} e\big(\tilde{\gamma}_2, (u_s')^{m_s} h_s'\big)^{-1} && \text{by Eq. (3.4)} \\
&= e(\tilde{\gamma}_1, g') \cdot e\big(\tilde{\gamma}_2^{-\sum_{s \in S}(v_s m_s + t_s)}, g'\big) && \text{by Eq. (3.5)} \\
&= e\big(\tilde{\gamma}_1 \tilde{\gamma}_2^{-\sum_{s \in S}(v_s m_s + t_s)}, g'\big) && \text{by bilinearity.}
\end{aligned}$$

It follows that $\tilde{\gamma}_1 \tilde{\gamma}_2^{-\sum_{s \in S} v_s m_s + t_s}$ is a solution to the bilateral CDH problem and so, algorithm $\mathcal{B}$ succeeds with the same advantage ε. □

Theorem 3.6 (Succinctness). *Construction 3.3 is succinct.*

Proof. By construction, the size of the aggregate signature in Construction 3.3 consists of two group elements, so it has size $2 \cdot |\mathbb{G}| = \mathsf{poly}(\lambda)$, as required. □

Corollary 3.7 (Slotted Aggregate Signatures). *Let λ be a security parameter and N be the number of slots. Then, under the bilateral CDH assumption in a pairing group, there exists a slotted aggregate signature scheme with the following properties:*

- *The public parameters contain $3N + 1$ elements in $\mathbb{G}$ and $2N + 1$ elements in $\mathbb{G}'$.*
- *Each verification key consists of one element in each of $\mathbb{G}$ and $\mathbb{G}'$. The secret key is a field element in $\mathbb{Z}_p$.*
- *Each signature contains $2N$ elements in $\mathbb{G}$. Signing requires $O(N)$ exponentiations in $\mathbb{G}$ and verification also requires $O(N)$ pairings.*
- *An aggregate signature consists of just two elements in $\mathbb{G}$. Aggregation requires $O(N)$ exponentiations. Verifying an aggregate signature requires $O(N)$ pairings.*

Remark 3.8 (Reducing the Verification Cost). To verify an aggregate signature on $|S| \leq N$ messages, the aggregate verification algorithm in Construction 3.3 needs to compute $3|S| + 2$ pairings: $|S|$ pairings are needed to compute the per-slot public key $e(A_s, V'_s)$ and $2|S|$ pairings are needed to check well-formedness of each individual (i.e., that $e(g, V'_s) = e(V_s, g')$). We first remark that all of these pairings can be precomputed (e.g., in an offline phase) if the signers and slots information are known before the signature is verified. Also, in settings where one expects to verify multiple signatures from the same set of signers, the per-slot verification keys can be cached and reused (and similarly, there is no need to re-check the well-formedness of their verification keys).

Second, when checking well-formedness of the $|S|$ verification keys, a simple strategy to reduce the number of pairings (and replace them with exponentiations) is to have the verifier sample random $\beta_s \xleftarrow{R} \mathbb{Z}_p$ for all $s \in S$ and checking $e(g, \prod_{s \in S}(V'_s)^{\beta_s}) = e(\prod_{s \in S} V_s{}^{\beta_s}, g')$. This will detect if any key is invalid with $1 - 1/p$ probability. One could also use a "small exponents" technique to choose β_s values from a smaller range to reduce the exponentiation time in exchange for a higher probability of accepting a signature with respect to a malformed set of verification keys [FGHP09].

4 From a Slotted Scheme to an Unslotted Scheme

In this section, we describe how to upgrade a slotted aggregate signature scheme into a standalone aggregate signature scheme (without slots). We work in the setting of bounded aggregation where there is an a priori bound on the number of signatures that will be aggregated. We start with the formal definition:

Definition 4.1 (Aggregate Signatures with Bounded Aggregation). An aggregate signature scheme that supports bounded aggregation on message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{AS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{pp}$: On input the security parameter λ and a bound on the number of signatures that will be aggregated, the setup algorithm outputs the public parameters pp. We assume that pp includes a description of 1^λ and 1^N.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: On input the public parameters pp, the key-generation algorithm outputs a verification key vk and a signing key sk.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m) \to \sigma$: On input the public parameters pp, the signing key sk, and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, \sigma) \to b$: On input the public parameters pp, a verification key vk, a message $m \in \mathcal{M}$, and a signature σ, the verification algorithm outputs $b = 1$ if the signature is valid and $b = 0$ otherwise.

- $\mathsf{Aggregate}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[K]}) \to \sigma_{\mathsf{agg}}$: On input the public parameters pp, verification keys vk_i, messages $m_i \in \mathcal{M}_\lambda$, and signatures σ_i for all $i \in [K]$ where $K \le N$, the aggregation algorithm outputs an aggregate signature σ_{agg} (or a special symbol $\perp$ to indicate a failure). Without loss of generality, we assume that each (vk_i, m_i) pair is distinct in the input.
- $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}) \to b$: On input the public parameters pp, verification keys vk_i and messages $m_i \in \mathcal{M}_\lambda$ for all $i \in [K]$ where $K \le N$, and a signature σ_{agg}, the aggregate-verification algorithm outputs $b = 1$ if σ_{agg} is a valid signature and $b = 0$ otherwise.

We require Π_{AS} to satisfy the following correctness, succinctness, and unforgeability properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all polynomials $N = \mathsf{poly}(\lambda)$, and all messages $m \in \mathcal{M}_\lambda$,

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, \sigma) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N); \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m) \end{array}\right] = 1.$$

  In addition, for all pp in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all $K \le N$, and all collections $\{(\mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[K]}$ where

$$\forall i \in [K] : \mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_i, m_i, \sigma_i) = 1,$$

  we have that

$$\Pr\left[\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}) = 1\right] = 1,$$

  where $\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[K]})$.
- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot, \cdot)$ such that in the completeness experiment above, the size of the aggregate signature σ_{agg} satisfies $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda, \log N)$.
- **Unforgeability:** For a security parameter λ and an adversary $\mathcal{A}$, we define the unforgeability game as follows:
  - **Setup:** The challenger gives 1^λ to $\mathcal{A}$ and receives the bound 1^N. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. It sends $(\mathsf{pp}, \mathsf{vk}^*)$ to the adversary $\mathcal{A}$.
  - **Signing queries:** The adversary can now make adaptive signing queries. On each query, the adversary specifies a message $m \in \mathcal{M}_\lambda$. The challenger responds with $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, m)$.
  - **Output:** At the end of the game, the adversary outputs $\{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}$ for some $K \le N$ together with a signature σ_{agg}. The challenger outputs 1 if the following conditions hold:
    - $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}) = 1$.
    - There exists some $i \in [K]$ where $\mathsf{vk}_i = \mathsf{vk}^*$ and moreover, algorithm $\mathcal{A}$ did not make a signing query on message m_i.

    Otherwise, the challenger outputs 0.

  We say the signature scheme is unforgeable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the unforgeability game.

Definition 4.2 (Selective Unforgeability)**.** Similar to Definition 3.2, we can define a selective variant of the unforgeability game in Definition 4.1 where the adversary has to declare a challenge message $m^* \in \mathcal{M}_\lambda$ at the beginning of the game (*before* it sees the public parameters). At the end of the game the adversary is successful if it outputs $\{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}$ and σ_{agg} where the following conditions hold:

- $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}) = 1$ as in Definition 4.1.
- There exists $i \in [K]$ where $\mathsf{vk}_i = \mathsf{vk}^*$ and $m_i = m^*$ and algorithm $\mathcal{A}$ does not make a signing query on message m_i.

We say Π_{AS} is *selectively-secure* if the advantage of any efficient adversary is bounded by $\mathsf{negl}(\lambda)$ in the selective unforgeability game.

4.1 Slotted-to-Unslotted Transformation

We now show how to construct an aggregate signature scheme with bounded aggregation from any slotted aggregate signature scheme. As described in Section 1.1, to support aggregation of up to N signatures, the idea is to instantiate a slotted aggregate signature scheme with $M \geq N$ slots. Each signature for the main scheme will consist of $D \leq M$ signatures on the same message but to different slots. Suppose we now have a collection of $K \leq N$ signatures. We can use the underlying slotted aggregate signature scheme to aggregate the signatures as long as we can associate a distinct slot (of the slotted scheme) with each of the K signatures. Similar to [GLWW23], we can formulate this assignment process as a bipartite graph matching problem. Namely, we consider a bipartite graph with K nodes on the left (representing the K signatures we are aggregating) and M nodes on the right (representing the slots of the slotted aggregate signature scheme). An edge exists between a signature σ_i and a slot $s \in [M]$ if σ_i includes a slotted signature for slot s. Since each signature contains D slotted signatures, each node on the left has degree D. Whenever there is a complete matching in this graph, we can aggregate the signatures using the underlying slotted scheme. Before giving the formal construction, we recall the basic graph-theoretic tools we will use. The definitions of bipartite graphs, Hall's theorem, and the statement of the Hopcroft-Karp algorithm is taken (nearly) verbatim from [GLWW23, §3.1].

Bipartite graphs and matchings. A bipartite graph $G = (U, V, E)$ consists of two sets of vertices U and V and a set of edges E. Each edge $e \in E$ is a pair of nodes $(u, v) \in U \times V$. We say G has left-degree D if for every node $u \in U$ has degree exactly D. A matching $M = (S, \rho)$ on G from U to V is a set of nodes $S \subseteq U$ and an *injective* labeling function $\rho \colon S \to V$ where

$$\forall u \in S : (u, \rho(u)) \in E.$$

We say $M = (S, \rho)$ is a complete matching if $S = U$ and that it is maximal if for every matching $M' = (S', \rho')$ on G from U to V, it holds that $|S| \geq |S'|$. For a set $S \subseteq U$, we write $\Gamma(S) \subseteq V$ to denote the neighborhood of S: $\Gamma(S) = \{v \in V \mid \exists u \in S : (u, v) \in E\}$.

Theorem 4.3 (Hall's Marriage Theorem [Hal35])**.** *Let $G = (U, V, E)$ be a bipartite graph. Then, G has a complete matching from U to V if and only if for every subset $S \subseteq U$, $|\Gamma(S)| \geq |S|$.*

Theorem 4.4 (Hopcroft-Karp [HK71])**.** *There exists a* deterministic *algorithm* FindMatch *that takes as input a bipartite graph $G = (U, V, E)$ and outputs a maximal matching from U to V in time $O(|E| \cdot |V|^{1/2})$.*

Slotted-to-unslotted transformation. We now give our slotted-to-unslotted transformation.

Construction 4.5. Our construction relies on the following primitives:

- Let $\Pi_{\mathsf{SAS}} = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}', \mathsf{Aggregate}', \mathsf{AggVerify}')$ be a slotted aggregate signature scheme with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$. Let $\ell = \ell(\lambda)$ be the length of a pair (vk, m), where vk is a verification key and $m \in \mathcal{M}_\lambda$ is a message for Π_{SAS}.
- Let FindMatch be the bipartite matching algorithm from Theorem 4.4.
- Let $\mathcal{H} = \{H_{\ell,M,D}\}_{\ell,M,D \in \mathbb{N}}$ be a family of hash functions $H_\ell \colon \{0,1\}^\ell \to [M]^D$. We assume the description of $H_{\ell,M,D}$ includes a description of the parameters ℓ, M, and D. Let $M = M(\lambda, N)$ and $D = D(\lambda, N)$ be polynomials which will be set in the security analysis.

We construct an aggregate signature scheme $\Pi_{\mathsf{AS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{AggVerify})$ that supports bounded aggregation with the same message space $\mathcal{M}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and a bound on the number of signatures, compute $\ell = \ell(\lambda)$, $M = M(\lambda, N)$, $D = D(\lambda, N)$, and $\mathsf{pp}' \leftarrow \mathsf{Setup}'(1^\lambda, 1^M)$. Output $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$, output $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{pp})$. Output the signing key $\mathsf{sk} = (\mathsf{vk}', \mathsf{sk}')$ and the verification key $\mathsf{vk} = \mathsf{vk}'$.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$, a signing key $\mathsf{sk} = (\mathsf{vk}', \mathsf{sk}')$, and a message $m \in \mathcal{M}_\lambda$, the signing algorithm computes $(s_1, \ldots, s_D) = H_{\ell,M,D}(\mathsf{vk}', m)$. Then, for each $i \in [D]$, it computes $\sigma'_i \leftarrow \mathsf{Sign}'(\mathsf{pp}', \mathsf{sk}', m, s_i)$. Output $\sigma = (\sigma'_1, \ldots, \sigma'_D)$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, \sigma)$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$, a verification key $\mathsf{vk} = \mathsf{vk}'$, a message $m \in \mathcal{M}_\lambda$, and a signature $\sigma = (\sigma'_1, \ldots, \sigma'_D)$, the verification algorithm computes $(s_1, \ldots, s_D) = H_{\ell,M,D}(\mathsf{vk}', m)$ and output 1 if $\mathsf{Verify}'(\mathsf{pp}', \mathsf{vk}', m, s_i, \sigma'_i) = 1$ for all $i \in [D]$. Otherwise, it outputs 0.
- $\mathsf{Aggregate}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[K]})$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$, verification keys $\mathsf{vk}_i = \mathsf{vk}'_i$, messages $m_i \in \mathcal{M}_\lambda$, and signatures $\sigma_i = (\sigma_{i,1}, \ldots, \sigma_{i,D})$ for all $i \in [K]$ where $K \le N$, the aggregation algorithm proceeds as follows:
  - First, define a bipartite graph $G = (U, V, E)$ where $U = [K]$ and $V = [M]$. For each $i \in [K]$, compute $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(\mathsf{vk}'_i, m_i)$. Define the edges to be $E = \{(i, s_{i,j}) \mid i \in [K], j \in [D]\}$.
  - Compute a matching $(U', \rho) = \mathsf{FindMatch}(G)$. If $U' \neq U$, then output $\perp$.
  - For each $i \in [K]$, let $j_i \in [D]$ be the index such that $s_{i,j_i} = \rho(i)$. Compute and output $\sigma'_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i, \sigma'_{i,j_i})\}_{i\in[K]})$
- $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}})$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$, the verification keys $\mathsf{vk}_i = \mathsf{vk}'_i$ and messages $m_i \in \mathcal{M}_\lambda$ for each $i \in [K]$ where $K \le N$, and the aggregated signature $\sigma_{\mathsf{agg}} = \sigma'_{\mathsf{agg}}$, the aggregation algorithm proceeds as follows:
  - First, define a bipartite graph $G = (U, V, E)$ where $U = [K]$ and $V = [M]$. For each $i \in [K]$, compute $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(\mathsf{vk}'_i, m_i)$. Define the edges to be $E = \{(i, s_{i,j}) \mid i \in [K], j \in [D]\}$.
  - Compute a matching $(U', \rho) = \mathsf{FindMatch}(G)$. If $U' \neq U$, then output $\perp$.
  - Compute and output $\mathsf{AggVerify}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i)\}_{i\in[K]}, \sigma'_{\mathsf{agg}})$.

Correctness and security analysis. We now give the correctness and security analysis of our construction. Observe that Construction 4.5 uses the hash function $H_{\ell,M,D}$ to define a bipartite graph that is used to assign signatures to slots during aggregation. For aggregation to be successful, this mapping must define a bipartite graph with a complete matching. Thus, we say the hash function $H_{\ell,M,D}$ is "N-match-inducing" if for all distinct inputs $x_1, \ldots, x_N \in \{0,1\}^\ell$, the bipartite graph induced by $H_{\ell,M,D}(x_1), \ldots, H_{\ell,M,D}(x_N)$ contains a matching (where the graph is the one from Construction 4.5). We formally define this property below and then give the correctness and security analysis. Afterwards, we describe several ways to instantiate the match-inducing family of hash functions that provide various trade-offs (in terms of signature size vs. public parameter size).

Definition 4.6 (Match-Inducing Hash Function)**.** Let $H_{\ell,M,D} \colon \{0,1\}^\ell \to [M]^D$ be a hash function. For a collection of distinct inputs $x_1, \ldots, x_K \in \{0,1\}^\ell$, we define the bipartite graph $G = (U, V, E)$ induced by $H_{\ell,M,D}$ on inputs $(x_1, \ldots, x_K)$ as follows:

- Let $U = [K]$ and $V = [M]$.
- For each $i \in [K]$, let $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(x_i)$. Let $E = \{(i, s_{i,j}) \mid i \in [K], j \in [D]\}$.

We say $H_{\ell,M,D}$ is N-match-inducing if for every collection of distinct inputs $x_1, \ldots, x_K \in \{0,1\}^\ell$, where $K \le N$, the bipartite graph $G = (U, V, E)$ induced by $H_{\ell,M,D}$ on inputs $(x_1, \ldots, x_K)$ has a complete matching from U to V.

Theorem 4.7 (Correctness)**.** *Suppose Π_{SAS} is correct. In addition, suppose that for all $\lambda, N \in \mathbb{N}$, the hash function $H_{\ell(\lambda),M(\lambda,N),D(\lambda,N)}$ is N-match-inducing. Then, Construction 4.5 is correct.*

Proof. Take any $\lambda \in \mathbb{N}$, any polynomial $N = N(\lambda)$, and any message $m \in \mathcal{M}_\lambda$. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$. Then we have the following:

- First $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$ where $\mathsf{pp}' \leftarrow \mathsf{Setup}(1^\lambda, 1^M)$. Since $M = \mathsf{poly}(\lambda, N)$ and $N = \mathsf{poly}(\lambda)$, M is also polynomially-bounded in λ.

- Next, $\mathsf{sk} = (\mathsf{vk}', \mathsf{sk}')$ and $\mathsf{vk} = \mathsf{vk}'$ where $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{pp})$.
- Finally, $\sigma = (\sigma'_1, \ldots, \sigma'_D)$ where $\sigma'_i \leftarrow \mathsf{Sign}'(\mathsf{pp}', \mathsf{sk}', m, s_i)$ and $(s_1, \ldots, s_D) = H_{\ell,M,D}(\mathsf{vk}', m)$.
- By correctness of Π_{SAS}, $\mathsf{Verify}'(\mathsf{pp}', \mathsf{vk}', m, s_i, \sigma'_i) = 1$ for all $i \in [D]$. This means $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, m, \sigma) = 1$, as required.

For the second property, take any collection of tuples $(\mathsf{vk}_i, m_i, \sigma_i)$ where $i \in [K]$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}_i, m_i, \sigma_i) = 1$. This means for all $i \in [N]$ and all $j \in [D]$, we can write $\mathsf{vk}_i = \mathsf{vk}'_i$ and $\sigma_i = (\sigma'_{i,1}, \ldots, \sigma'_{i,D})$ where

$$\mathsf{Verify}'(\mathsf{pp}', \mathsf{vk}'_i, m_i, \sigma'_{i,j}, s_{i,j}) = 1, \tag{4.1}$$

and $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(\mathsf{vk}'_i, m_i)$. Then, the following holds:

- Let $G = (U, V, E)$ be the bipartite graph induced by $H_{\ell,M,D}$ on the input $((\mathsf{vk}'_1, m_1), \ldots, (\mathsf{vk}'_K, m_K))$ computed by Aggregate. By construction, $U = [K]$ and $V = [M]$.
- Since $H_{\ell,M,D}$ is N-match-inducing and $K \le N$, the graph G contains a complete matching from U to V. By Theorem 4.4, this means $(U', \rho) = \mathsf{FindMatch}(G)$ outputs a matching where $U' = U$. This means the values of $\rho(1), \ldots, \rho(K) \in [M]$ are all distinct. By construction of G, for every $i \in [K]$, there exists an index $j_i \in [D]$ such that $s_{i,j_i} = \rho(i)$. By Eq. (4.1), this means

$$\forall i \in [K] : \mathsf{Verify}'(\mathsf{pp}', \mathsf{vk}'_i, m_i, \sigma'_{i,j_i}, \rho(i)) = 1. \tag{4.2}$$

- Let $\sigma'_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i, \sigma_{i,j_i})\}_{i\in[K]})$. By Eq. (4.2) and correctness of Π_{SAS}, this means

$$\mathsf{AggVerify}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i)\}_{i\in[K]}) = 1.$$

  By construction of AggVerify, this means $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma'_{\mathsf{agg}}) = 1$. Correctness holds since AggVerify constructs the graph G using the identical procedure as Aggregate and the FindMatch algorithm is deterministic. As such, AggVerify and Aggregate compute the same matching (U', ρ). □

Theorem 4.8 (Succinctness). *If Π_{SAS} is succinct, then Construction 4.5 is also succinct.*

Proof. Take any $\lambda, N \in \mathbb{N}$. The size of an aggregate signature in Construction 4.5 is $\mathsf{poly}(\lambda, \log M)$. Since $M = \mathsf{poly}(\lambda, N)$, we conclude that $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda, N)$, as required. □

Theorem 4.9 (Selective Unforgeability). *If Π_{SAS} satisfies selective unforgeability, then Construction 4.5 is also selectively unforgeable.*

Proof. Let $\mathcal{A}$ be an adversary for the selective unforgeability game. We define a sequence of hybrid experiments:

- Hyb_0: This is the real selective unforgeability experiment:
  - **Setup:** In this experiment, the challenger gives 1^λ to $\mathcal{A}$ and receives the bound 1^N along with a challenger message m^*. Then, it computes $\ell = \ell(\lambda)$, $M = M(\lambda, N)$, and $D = D(\lambda, N)$, and samples $\mathsf{pp}' \leftarrow \mathsf{Setup}(1^\lambda, 1^M)$. The challenger also samples $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{pp})$. The challenger gives $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$ and $\mathsf{vk}^* = \mathsf{vk}'$ to $\mathcal{A}$.
  - **Signing queries:** Whenever algorithm $\mathcal{A}$ makes a signing query on a message $m \ne m^* \in \mathcal{M}_\lambda$, the challenger computes $(s_1, \ldots, s_D) = H_{\ell,M,D}(\mathsf{vk}^*, m)$. Then it computes $\sigma'_i \leftarrow \mathsf{Sign}'(\mathsf{pp}', \mathsf{sk}', m, s_i)$ and gives $\sigma = (\sigma'_1, \ldots, \sigma'_D)$ to $\mathcal{A}$.
  - **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs $\{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}$ and a signature σ_{agg}. The output of the experiment is 1 if the following conditions hold:

* $\mathsf{AggVerify}(\mathsf{pp}, \{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}) = 1$. Specifically, the challenger first defines the bipartite graph $G = (U, V, E)$ where $U = [K]$ and $V = [M]$. For each $i \in [K]$, compute $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(\mathsf{vk}_i, m_i)$. Define the edges to be $E = \{(i, s_{i,j}) \mid i \in [K], j \in [D]\}$. Then the challenger computes a matching $(U', \rho) = \mathsf{FindMatch}(G)$. It checks that

$$U' = U \quad \text{and} \quad \mathsf{AggVerify}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}).$$

* There exists an index $i \in [K]$ where $\mathsf{vk}_i = \mathsf{vk}^* = \mathsf{vk}'$ and $m_i = m^*$.

- Hyb_1: Same as Hyb_0, except in the setup phase, the challenger samples an index $j^* \xleftarrow{R} [M]$. At the end of the game, the challenger additionally checks that $\rho(i) = j^*$ where $i \in [K]$ is the index where $\mathsf{vk}_i = \mathsf{vk}'$ and $m_i = m^*$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ to denote the output distribution of an execution of Hyb_0 and Hyb_1 with adversary $\mathcal{A}$, respectively. First, Hyb_0 and Hyb_1 are identical experiments from the view of the adversary. If the challenger outputs 1 in Hyb_0, then it must be the case that $\rho(i) \in [M]$ where $i \in [K]$ is the special index that satisfies $\mathsf{vk}_i = \mathsf{vk}'$ and $m_i = m^*$. Since the challenger samples $j^* \xleftarrow{R} [M]$, with probability $1/M$, $j^* = \rho(i)$. Thus, we conclude that

$$\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \frac{1}{M} \cdot \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1].$$

Suppose now that algorithm $\mathcal{A}$ breaks selective unforgeability with non-negligible probability ε. This means $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \varepsilon$ and $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \frac{\varepsilon}{M}$, which is also non-negligible since $M = \mathsf{poly}(\lambda, N)$ and $N = \mathsf{poly}(\lambda)$. We now use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks selective unforgeability of Π_{SAS}:

- **Setup phase:** On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{B}$ on the same input 1^λ. Algorithm $\mathcal{B}$ outputs the bound 1^N and a target message $m^* \in \mathcal{M}_\lambda$. Algorithm $\mathcal{B}$ then computes $\ell = \ell(\lambda)$, $M = M(\lambda, N)$, and $D = D(\lambda, N)$. It then samples an index $j^* \xleftarrow{R} [M]$, and gives the slot count 1^M, the slot index $j^* \in [M]$, and the message m^* to its challenger. The challenger responds with the public parameters pp' and a verification key vk'. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (\mathsf{pp}', H_{\ell,M,D})$ and $\mathsf{vk}^* = \mathsf{vk}'$ to $\mathcal{A}$.

- **Signing queries:** Whenever $\mathcal{B}$ issues a signing query on a message $m \neq m^*$, algorithm $\mathcal{B}$ first computes $(s_1, \ldots, s_D) = H_{\ell,M,D}(\mathsf{vk}^*, m)$. For each $i \in [D]$, algorithm $\mathcal{B}$ makes a signing query on message m and slot s_i to obtain a signature σ'_i. Algorithm $\mathcal{B}$ responds with $\sigma = (\sigma'_1, \ldots, \sigma'_D)$ to $\mathcal{A}$.

- **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs $\{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}$ and a signature σ_{agg}. Algorithm $\mathcal{B}$ first defines the bipartite graph $G = (U, V, E)$ where $U = [K]$ and $V = [M]$. For each $i \in [K]$, it computes $(s_{i,1}, \ldots, s_{i,D}) = H_{\ell,M,D}(\mathsf{vk}'_i, m_i)$. and then defines the edges to be $E = \{(i, s_{i,j}) \mid i \in [K], j \in [D]\}$. Then algorithm $\mathcal{B}$ computes a matching $(U', \rho) = \mathsf{FindMatch}(G)$. Algorithm $\mathcal{B}$ outputs $\{(\rho(i), \mathsf{vk}'_i, m_i)\}_{i\in[K]}$ together with the signature σ_{agg}.

By construction, the challenger for the selective unforgeability game for Π_{SAS} samples $\mathsf{pp}' \leftarrow \mathsf{Setup}'(1^\lambda, 1^M)$ and $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{pp})$, which coincides with the distribution in Hyb_1. Next, for the signing queries on a message m and slot s_i, the challenger responds with $\sigma'_i \leftarrow \mathsf{Sign}'(\mathsf{pp}', \mathsf{sk}', m, s_i)$, which again coincides with the behavior in Hyb_1. We conclude that algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{A}$. Thus, with probability ε/M, algorithm $\mathcal{B}$ outputs $\{(i, \mathsf{vk}_i, m_i)\}_{i\in[K]}$ and a signature σ_{agg} where the following conditions hold:

- Let $G = (U, V, E)$ be the bipartite graph induced by $H_{\ell,M,D}$ on input $((\mathsf{vk}_1, m_1), \ldots, (\mathsf{vk}_K, m_K))$ and $(U', \rho) = \mathsf{FindMatch}(G)$. Then

$$U' = U \quad \text{and} \quad \mathsf{AggVerify}'(\mathsf{pp}', \{(\rho(i), \mathsf{vk}'_i, m_i)\}_{i\in[K]}, \sigma_{\mathsf{agg}}).$$

- There exists an index $i \in [K]$ where $\rho(i) = j^*$, $\mathsf{vk}_i = \mathsf{vk}^*$, and $m_i = m^*$.

In this case, algorithm $\mathcal{B}$ wins the selective unforgeability game for Π_{SAS}, and the claim holds. □

4.2 Constructing Match-Inducing Hash Functions

In this section, we describe two possible ways to instantiate the match-inducing hash function in Construction 4.5. The two constructions offer different trade-offs. The first is a "brute force" approach that increases the size of the signatures by a factor of N, but keeps the public parameter size unchanged relative to the slotted aggregate signature scheme. The second approach is based on expander graphs and increases the size of the signature by a $\mathsf{poly}(\lambda, \log N)$ factor, but slightly increases the size of the public parameters relative to the slotted aggregate signature scheme.

Construction 4.10 (Brute-Force Match-Inducing Hash Function)**.** Take any input length $\ell = \ell(\lambda)$ and set $M(\lambda, N) = D(\lambda, N) = N$. Define $H_{\ell,M,D}(x) := (1, 2, \ldots, M)$. Namely, the hash function $H_{\ell,M,D}$ is a constant function that outputs $(1, 2, \ldots, M)$ on every input.

By construction, for all $\lambda, N \in \mathbb{N}$, all inputs $x_1, \ldots, x_K$ where $K \leq N$, the graph $G = (U, V, E)$ induced by the hash function $H_{\ell(\lambda),M(\lambda,N),D(\lambda,N)} \equiv H_{\ell,N,N}$ from Construction 4.10 is a *complete* bipartite graph. Since $|U| = K \leq N = |V|$, this is always a matching from U to V, so $H_{\ell,N,N}$ is N-match-inducing. In the context of the aggregate signature scheme, this corresponds to the setting where we use a slotted aggregate signature scheme with N slots and a signature on a message m consists of N signatures for the slotted scheme, one for each slot. By construction, this allows us to aggregate any collection of N signatures. Combined with Construction 3.3, we obtain the following corollary:

Corollary 4.11 (Aggregate Signature with Bounded Aggregation)**.** *Under the bilateral CDH assumption in a prime-order pairing group, there exists an aggregate signature scheme that supports bounded aggregation with the following efficiency properties:*

- *For a security parameter λ and a bound N on the number of signatures that can be aggregated, the public parameters consists of $O(N)$ group elements.*
- *The verification key consists of a single group element and the secret key consists of a single field element.*
- *A signature consists of $O(N^2)$ group elements.*
- *An aggregate signature on any number of $K \leq N$ message/verification key pairs consists of 2 group elements.*

A construction based on expander graphs. A limitation of using Construction 4.10 in Construction 4.5 is that it blows up the signature size by a factor of N, where N is the maximum number of signatures that can be aggregated. The blowup is due to the use of a hash function that always induces a *complete* bipartite graph to achieve the match-inducing property. A natural approach to reduce the signature size overhead is to use a *sparse* bipartite graph instead (where a matching is still guaranteed to exist). Here, we describe another instantiation based on expander graphs. We first recall some basic definitions. All of our definitions are adapted from [GUV07]:

Definition 4.12 (Bipartite Expander)**.** A bipartite graph $G = (U, V, E)$ with left-degree D is a (K, A)-expander if for every set $S \subseteq U$ of size at most K, we have $|\Gamma(S)| \geq A \cdot |S|$. We say that G has an *explicit* description if there is an algorithm that takes as input a node $u \in U$ and outputs $\Gamma(u)$ in $\mathsf{poly}(\log |U| + \log D)$ time.

Theorem 4.13 (Explicit Bipartite Expander [GUV07, Theorem 1.3])**.** *For every constant $\alpha > 0$, every $T \in \mathbb{N}$, every $K \leq T$, and every $\varepsilon > 0$, there is an explicit $(K, (1-\varepsilon)D)$-expander $G = (U, V, E)$ with left-degree $D = O((\log T)(\log K)/\varepsilon)^{1+1/\alpha}$, $U = [T]$, and $V = [M]$ where $M \leq D^2 K^{1+\alpha}$.*

Construction 4.14 (Expander-Based Match-Inducing Hash Function)**.** Take any constant $\alpha > 0$, and any function $\ell = \ell(\lambda)$. Let $G = (U', V', E')$ be the explicit expander graph from Theorem 4.13 with parameters α, $T = 2^\ell$, $K = N$, and $\varepsilon = 1/2$. Let $M(\lambda, N) = |V|$ and $D(\lambda, N)$ be the left-degree of G. We associate the elements of $U' = [2^\ell]$ with the bit strings $\{0,1\}^\ell$ in the canonical way (e.g., lexicographically). Then, on input $x \in \{0,1\}^\ell$ define $H_{\ell,N,D}(x)$ to be the D neighbors of x in the graph G. Note that $\Gamma(x)$ can be computed in $\mathsf{poly}(\ell, \log D)$-time since G has an explicit description.

Theorem 4.15 (Expander-Based Match-Inducing Hash Function)**.** *Take any constant $\alpha > 0$. Then, for all sufficiently large $\ell = \ell(\lambda)$, the hash function $H_{\ell,M,D}$ from Construction 4.14 is an N-match-inducing hash function where $D = \mathsf{poly}(\ell, \log N)^{1+1/\alpha}$ and $M = D^2 N^{1+\alpha}$.*

Proof. Let $G = (U', V', E')$ be the expander graph from Construction 4.14. By Theorem 4.13, G has left-degree $D = O(\ell \log N)^{1+1/\alpha}$ and moreover, G is a $(N, D/2)$-expander. For sufficiently-large ℓ, this means $D/2 > 1$. In this case, the expansion property ensures that every collection of $K \leq N$ distinct nodes $x_1, \ldots x_K \in [2^\ell]$ has a neighborhood of size $|\Gamma(\{x_1, \ldots, x_K\})| \geq (D/2) \cdot K \geq K$. By Theorem 4.3, this means there is a complete matching from $\{x_1, \ldots, x_K\} \subseteq U$ to $V = [M]$ in G. By construction, the graph induced by $H_{\ell,M,D}$ with respect to any set of inputs $x_1, \ldots, x_K$ is the subgraph of G obtained by taking $\{x_1, \ldots, x_K\}$ to be the left nodes and the nodes $[M]$ on the right. As argued previously, there exists a matching in this graph, so $H_{\ell,M,D}$ is N-match-inducing, as required. □

Combining Construction 3.3 with Construction 4.5 and the match-inducing hash function from Theorem 4.13, we obtain the following corollary:

Corollary 4.16 (Aggregate Signature with Bounded Aggregation)**.** *Take any constant $\alpha > 0$. Under the bilateral CDH assumption in a prime-order pairing group, there exists an aggregate signature scheme that supports bounded aggregation with the following efficiency properties:*

- *For a security parameter λ and a bound N on the number of signatures that can be aggregated, the public parameters consists of $\mathsf{poly}(\lambda, \log N)^{2+2/\alpha} N^{1+\alpha}$ group elements.*
- *The verification key consists of a single group element and the secret key consists of a single field element.*
- *A signature consists of $N \cdot \mathsf{poly}(\lambda, \log N)^{1+1/\alpha}$ group elements.*
- *An aggregate signature on any number of $K \leq N$ message/verification key pairs consists of 2 group elements.*

Suppose we set $\alpha = 1$. Then Corollary 4.16 gives an aggregate signature scheme where the public parameters have size $N^2 \cdot \mathsf{poly}(\lambda, \log N)^4$ and where the signature size is $N \cdot \mathsf{poly}(\lambda, \log N)^2$. Asymptotically, by setting α to be arbitrarily small, we obtain a scheme where the public parameters have size $N^{1+\alpha} \cdot \mathsf{poly}(\lambda, \log N)$ and individual signatures have size $N \cdot \mathsf{poly}(\lambda, \log N)$. In this case, the slotted-to-unslotted transformation only incurs $\mathsf{poly}(\lambda, \log N)$-overhead in the signature size over the underlying slotted construction.

Acknowledgments

Susan Hohenberger is supported by NSF CNS-2318702 and a JPMC Faculty Research Award. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, and an Amazon Research Award.

References

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM CCS*, 2010.

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In *CRYPTO*, 2024.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, 2013.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, 2018.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, 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 *ACM CCS*, 2007.

[BK20] Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices, 2020.

[BLM+24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. In *ASIACRYPT*, 2024.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, 2001.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS*, 2006.

[BN07] Mihir Bellare and Gregory Neven. Identity-based multi-signatures from RSA. In *CT-RSA*, 2007.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *PKC*, 2003.

[BPW25] Dan Boneh, Aditi Partap, and Brent Waters. Accountable multi-signatures with constant size public keys. In *PKC*, 2025.

[CEW25] Binyi Chen, Noel Elias, and David J. Wu. Pairing-based batch arguments for NP with a linear-size CRS. In *ASIACRYPT*, 2025.

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In *CRYPTO*, 2023.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, 2021.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *IEEE S&P*, 2019.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, 2022.

[DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In *USENIX Security Symposium*, 2020.

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[FGHP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *CT-RSA*, 2009.

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO*, 2013.

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT*, 2023.

[FSZ22] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. In *ACM CCS*, 2022.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *ACM CCS*, 2023.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

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

[GUV07] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *CCC*, 2007.

[GV22] Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In *CRYPTO*, 2022.

[Hal35] Peter Hall. On representatives of subsets. *Journal of The London Mathematical Society*, 1935.

[HK71] John E. Hopcroft and Richard M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *SWAT*, 1971.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT*, 2015.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

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

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, 2014.

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In *EUROCRYPT*, 2018.

[Ita83] Kazuharu Itakura. A public-key cryptosystem suitable for digital multisignature. *NEC research and development*, 71, 1983.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, 2023.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KS23] R. Kabaleeshwaran and Panuganti Venkata Shanmukh Sai. Synchronized aggregate signature under standard assumption in the random oracle model. In *INDOCRYPT*, 2023.

[LLY13] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Aggregating CL-signatures revisited: Extended functionality and better efficiency. In *Financial Cryptography and Data Security*, 2013.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT*, 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: extended abstract. In *ACM CCS*, 2001.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptogr.*, 87(9), 2019.

[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), 1999.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *FOCS*, 2022.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, 2005.

[WTW+24] Jianghong Wei, Guohua Tian, Ding Wang, Fuchun Guo, Willy Susilo, and Xiaofeng Chen. Pixel+ and Pixel++: Compact and efficient forward-secure multi-signatures for PoS blockchain consensus. In *USENIX Security Symposium*, 2024.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.