Locally Verifiable Signature and Key Aggregation

Crypto’22

Authors: **Rishab Goyal (MIT)** **Vinod Vaikuntanathan(MIT)**

Speaker: **Xiaotong Zhou**

Problem statement

Aggregate Signatures

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

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

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

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

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

Problem statement

- Aggregate signatures [BGLS03] enable compressing a set of N signatures on N different messages into a short aggregate signature. **This reduces the space complexity of storing the signatures from linear in N to a fixed constant** (that depends only on the security parameter).

- However, verifying the aggregate signature requires access to all N messages, **resulting in the complexity of verification being at least Ω(N).**

- In many practical scenarios, the verifier **is merely interested in checking** if $\hat{\sigma}$ is an aggregated signature of **some set** that contains a particular message m.

Can we construct locally verifiable aggregate signatures?

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

Contribution

- Introduced the **notion of locally verifiable aggregate signatures** that enable efficient verification.

  - Given a short aggregate signature σ, the verifier can **check whether a particular message *m* is in the set**, in time independent of N. Verification does not require knowledge of the entire set M.

  - Natural applications:
    - ✓ Certificate transparency logs;
    - ✓ Blockchains;
    - ✓ Redacting signatures
    - ✓ All the original signatures are produced by a single user.

- Two constructions: **RSA-based** and **Bilinear Diffie-Hellman** based locally verifiable aggregate signature

- An additional contribution: introduce the notion of **compressing cryptographic keys in identity-based encryption (IBE) schemes**

  - The secret keys for **N identities** can be compressed into **a single aggregate key**, which can then be used to decrypt ciphertexts sent to any of the N identities.

Locally Verifiable Aggregate Signatures: Definition and Applications

Definition：(single-signer setting)

- In addition to the key generation, signing, and verification algorithms, a locally verifiable aggregate signature scheme consists of three additional algorithms.

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

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

✓ ***LocalAggVerify*** is the **local verification algorithm** that verifies the aggregate signature $\hat{\sigma}$ and the short hint *h* for a message *m*. **(the run-time of *LocalAggVerify* is independent of N)**

Applications :

- **Certificate Transparency (CT) Logs**

  - Certificate transparency (CT) is an internet security standard that creates public logs which record all certificates issued by certificate authorities (CAs);

  - Aggregate signatures can ease the burden of storage on the CT log;

  - However, the only way to verify whether a particular entry exists in the log is to download all entries;

  - **Locally verifiable aggregate signatures** allow the CT log to compress the certificates into a short aggregate signature **while allowing the user to verify the existence of an entry by downloading just a few additional kilobytes** (in the form of a short hint) and performing a fast computation.

Applications :

- **Blockchains**
  - A user or an organization wants to aggregate the signatures on the set of all transactions originating from a single payer, and later wishes to quickly and with little communication convince a third party **(e.g. an auditor)** of the existence of a particular transaction;
  - The user/organization can prove knowledge of a single transaction **without revealing the remaining transactions**.

Applications :

- **Redactable Signatures**

  - Redactable signature schemes allow a signature holder to publicly censor parts of a signed document such that the corresponding signature σ can be efficiently updated without the secret signing key, and the updated signature can still be verified given only the redacted document;

  - Locally verifiable aggregate signatures provide a fresh approach to redactability and sanitization.

  - **First, split** the document into small message blocks; **second, sign** the message blocks individually, together with their index; and **third, aggregate** these individual signatures and output the aggregated signature as the final signature for the full document. **For redaction**, the redacting party can generate short hints for each of the unredacted portions of the document, and include these as part of the redacted signature.

Constructions

- **RSA-based Locally Verifiable Aggregate Signature**
- **Pairing-based Locally Verifiable Aggregation**
- **Pairing-based Aggregate Identity-Based Encryption**

RSA-based Locally Verifiable Aggregate Signature

Technical Overview

- Classical RSA-based single-signer aggregate signature

$$PK = (N, e)$$

$$\sigma = H(m)^d \pmod N$$
$$ed = 1 \pmod{\varphi(N)}$$

$$\{(m_i, \sigma_i)\}_{i=1}^{L}$$

$$\hat{\sigma} = \prod_{i=1}^{L} \sigma_i \pmod N$$

$$\hat{\sigma}^e = \prod_{i=1}^{L} {\sigma_i}^e \pmod N = \prod_{i=1}^{L} H(m_i)^{ed} \pmod N$$

Problem:

It is completely unclear how to "locally" verify a single message m_i given $\hat{\sigma}$ and some hint h_i related to the message vector **m**.

How to define the message dependent hint clearly?

- ✓ Define the hint $h_i = \prod H(m_j)$ for $j \neq i$
- ✓ The local verifier check that $\hat{\sigma}^e = h_i \cdot H(m_i)$

$$h = \hat{\sigma} \cdot H(m_i)^{-1} \bmod N$$

Problem: However, **a malicious hint generator can easily fool the verifier:** the hint is adversarially generated and **the verifier has no mechanism to check that the hint is well-formed** without recomputing it which, in turn, seems to require the verifier to know all the underlying messages, in direct conflict with the requirement of local verification.

- To avoid this issue, they looked at other RSA-based signature schemes

  ✓ Unlike the classical RSA-based signature scheme, **most of these schemes do not support aggregation**.

- **A notable exception is the Gennaro, Halevi, and Rabin** [GHR99] **scheme** which works as follows.

$PK = (N, g)$

$\sigma = g^{1/H(m)} \ (mod\ N)$
$g \in \mathbb{Z}_N^*, e_{m_i} = H(m_i)$
$\sigma_i = g^{1/e_{m_i}} \ (mod\ N)$

$\{(m_i, \sigma_i)\}_{i=1}^{L}$

$\hat{\sigma} = \prod_{i=1}^{L} \sigma_i \ \ (mod\ N)$

$$\hat{\sigma}^{\prod_{i=1}^{L} e_{m_i}} = \left(\prod_{i=1}^{L} \sigma_i\right)^{\prod_{i=1}^{L} e_{m_i}} \ (\boldsymbol{mod\ N}) = \prod_{i=1}^{L} g^{\prod_{i \neq j} e_{m_j}} \ (\boldsymbol{mod\ N})$$

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. Advances in Cryptology. EUROCRYPT 1999.

$$\hat{\sigma}^{\prod_{i=1}^{L} e_{m_i}} = \left(\prod_{i=1}^{L} \sigma_i\right)^{\prod_{i=1}^{L} e_{m_i}} \pmod{N} = \prod_{i=1}^{L} g^{\prod_{i \neq j} e_{m_j}} \pmod{N}$$

✓ the aggregate signatures $\hat{\sigma}$ **can also be locally verified** w.r.t a message $m_j \in \mathbf{m}$ (the latter being the set of all messages whose signatures have been aggregated into $\hat{\sigma}$) without knowing **m** but **given only a short verification hint that depends on m and $\hat{\sigma}$.**

- To generate the following two whole numbers as the hint.

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

- The key observation is the following equation

$$(\hat{\sigma})^{e_{\mathbf{m}\backslash m_j}} = g^{f_j} \cdot g^{e_{\mathbf{m}\backslash m_j}/e_{m_j}} \pmod{N}$$

$$\left(g^{\sum_1^L \frac{1}{e_{m_i}}}\right)^{\prod_{i \neq j} e_{m_i}} = \left(g^{\sum_{(i \neq j)} \frac{1}{e_{m_i}}} \cdot g^{\frac{1}{e_{m_j}}}\right)^{\prod_{i \neq j} e_{m_i}}$$

- This can be translated into the following verification equation:

$$\left((\hat{\sigma})^{e_{\mathbf{m}\backslash m_j}} / g^{f_j}\right)^{e_{m_j}} \stackrel{?}{=} g^{e_{\mathbf{m}\backslash m_j}} \pmod{N}$$

- Since $\boldsymbol{e}_{\boldsymbol{m_j}}$ can be computed from just the target message m_j, releasing $\boldsymbol{e}_{\mathbf{m}\setminus\boldsymbol{m_j}}$ and f_j as the hint enables local verification of the aggregate signature $\hat{\sigma}$ via the above equation.

- It can also be proven secure in the presence of malicious hint generators as long as **the local verification algorithm also checks that the numbers $\boldsymbol{e}_{\mathbf{m}\setminus\boldsymbol{m_j}}$ and $\boldsymbol{e}_{\boldsymbol{m_j}}$ are co-prime** (that is, gcd($\boldsymbol{e}_{\mathbf{m}\setminus\boldsymbol{m_j}}$, $\boldsymbol{e}_{\boldsymbol{m_j}}$) = 1).)

Problem: The hints $\boldsymbol{e}_{\mathbf{m}\setminus\boldsymbol{m_j}}$ and f_j have to be computed modulo φ(N), but the hint generator does not (and must not) know φ(N). The only way out seems to be to compute them over the integers which again does not work as they could be large O(L)-bit numbers, which is decidedly not short.These together seem like an unfortunate limitation to obtaining local verifiability.

- Luckily, this conundrum can be resolved in a rather simple, yet elegant, way using the surprising power of **Shamir's trick [Sha84].**

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

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

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. CRYPTO 84.

✓ From the following equation:

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

$a = e_{m_j}, b = e_{\mathbf{m}\setminus m_j}$
$x = g^{e_{\mathbf{m}\setminus m_j}/e_{m_j}}, y = g$

$x^a = y^b$

$z^a = y$
$z^{e_{m_j}}$

✓ In more detail, the hint generator first computes:

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

✓ Note that $e_{\mathbf{m}\setminus m_j}$ and e_{m_j} are co-prime, thus there exist efficently computable integers α and β such that $\alpha \cdot e_{\mathbf{m}\setminus m_j} + \beta \cdot e_{m_j} = 1$.

✓ The hint generator next re-computes the signature and outputs it as hint.

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

- In fact, the local verification algorithm is independent of the aggregated signature $\hat{\sigma}$, and **only needs the hint for verification**.

- RSA-based scheme satisfies several additional properties such as support for **multi-hop aggregation** as well as **unordered sequential aggregation**.

- The aforementioned exact re-computation property of our aggregate signatures is very useful for **obtaining a redactable signature scheme which has constant-size redacted** as well as unredacted signatures. In a nutshell, the redaction algorithm can first compute the individual signatures of all message blocks whose signature it wants to release, and can aggregate them again to create a shorter signature.

The concrete Construction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

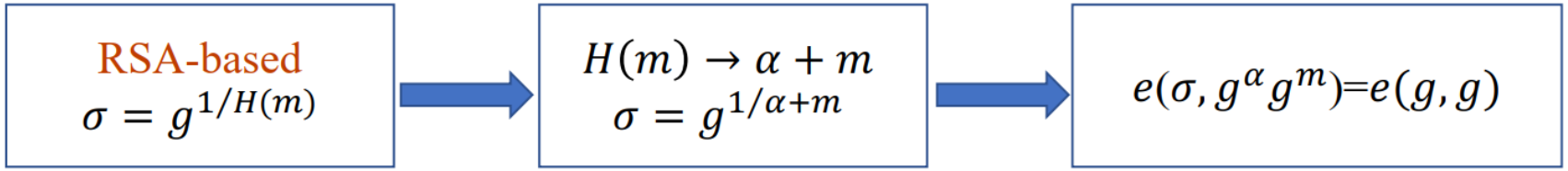
$\widehat{\boldsymbol{\sigma}}^{\prod_{i\neq j} e_{m_i}} = \boldsymbol{g}^{1/e_{m_j}} = \boldsymbol{\sigma_j}$

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

Pairing-based Locally Verifiable Aggregation

Technical Overview

- Their starting point is to translate the above process of RSA-based signature generation to bilinear maps.



- Coincidentally, this is exactly the weakly secure short signature scheme of Boneh and Boyen (BB) [BB04b].

Problem: Unfortunately, **the BB scheme is also not known to be aggregatable**, and while there exist pairing- based (multi-signer) aggregate signature schemes [BGLS03], they are algebraically similar to the classical RSA-based schemes, thus **do not appear to support local verifiability**.

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

[BGLS03] Dan Boneh, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. Eurocrypt 2003.

- The BB scheme is a single-signer aggregatable scheme.
- They observe that, by **Lagrange's inverse polynomial interpolation technique**, they can aggregate a sequence of signatures $\boldsymbol{\sigma_i = g^{\frac{1}{\alpha+m_i}}}$ **into** $\boldsymbol{\hat{\sigma} = g^{\prod_i \frac{1}{\alpha+m_i}}}$.
- Lagrange's inverse polynomial interpolation allows the following computation **without the knowledge of the secret exponent α.** where the coefficients γ_i can be publicly computed given only the sequence of messages $\{m_i\}_{i=1}^{L}$

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

- Thus, the aggregate signature $\hat{\sigma}$ can be computed as

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

- The aggregate verifier symbolically evaluates the polynomial $\boldsymbol{\Pi_{i=1}^{L}(\alpha + m_i)}$ **to simplify it as** $\boldsymbol{\Sigma_{i=0}^{L}\delta_i\alpha_i}$, and using bilinear maps it can verify the aggregate signature **as** $\boldsymbol{e\left(\hat{\sigma}, \Pi_i\left(g^{\alpha^i}\right)^{\delta_i}\right) = e(g, g)}$ but this **needs the monomials g^{α} as part of the public key.**

- This scheme is locally verifiable. The non-local verification algorithm works in two phases.

  ✓ First, it pre-processes the public key, given only the set of messages, to compute $\prod_i (g^{\alpha^i})^{\delta_i} = g^{\prod_i (\alpha + m_i)}$

  ✓ Second, it uses the bilinear map to pair this with the aggregate signature $\hat{\sigma}$ and compare with e(g, g).

  ✓ **The first step in the verification is inefficient.**

- ✓ In order to solve the above problems, **a hint generator can speed it up for any target message m_j** by generating the following two group elements as part of the short hint. Note that both h_1 and h_2 can also be publicly computed given only the public key, and set of messages contained in the aggregated signature.

Sign: $\hat{\sigma} = \prod_i \sigma_i^{\gamma_i} = g^{\prod_i \frac{1}{\alpha + m_i}}$ **Verify:** $e\left(\hat{\sigma}, \Pi_{i=1}^{L}(\alpha + m_i)\right) = e\left(\hat{\sigma}, \Pi_i \left(g^{\alpha^i}\right)^{\delta_i}\right) = e(g, g)$

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

- ✓ Given h_1 and h_2, a verifier can locally verify the aggregate signature as $e(\hat{\sigma}, h_1^{m_j} h_2) = e(g, g)$

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

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

Summary

- The pairing-based locally verification single-signer aggregate signature scheme **has the long public key**.

- They prove this to be **statically secure in the standard model**, and adaptively secure in the random oracle model by replacing $\alpha + m$ terms with $\alpha + \mathrm{H}(m)$.

- The **hint generation algorithm is fully public**, and does not even depend on the aggregate signature.

- Such fully public hint generation will be useful in applications **where the hint generator is unaware of the underlying aggregate signature**, or the user wants to generate the hint even before the aggregate signature has been generated or made available.

- **The aggregatable IBE scheme builds on the same idea**.

Pairing-based Aggregate Identity-Based Encryption

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

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

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

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

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

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

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

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

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

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

Pairing-based Aggregate Identity-Based Encryption

They provided an aggregate IBE scheme based on the hardness of strong Diffie-Hellman problem. Their constructions supports bounded aggregation of secret keys.

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

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

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

$$\widehat{\mathsf{sk}} = \mathsf{DPP}(\{\mathsf{sk}_i, x_i\}_i), = g^{\Pi_i \frac{1}{\alpha+h_{id}}}$$

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

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

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

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

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

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

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

It then outputs the decrypted message as

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

Open Problems

- Reducing the CRS size and **proving adaptively secure without the random oracle**

- How to construct a multi-signer locally verifiable aggregate signature scheme.