Locally Verifiable Aggregate Signatures and Cryptographic Accumulators: different names, same thing?

Anaïs Barthoulot[1,2], Olivier Blazy[3], Sébastien Canard[4]

[1] Orange, Caen, France
[2] XLim, Université de Limoges, France
[3] École Polytechnique, Palaiseau, France
[4] Télécom Paris, Palaiseau, France
anais.barthoulot@orange.com, olivier.blazy@polytechnique.edu, sebastien.canard@telecom-paris.fr

Cryptographic accumulator schemes, a primitive established almost thirty years ago, are widely used in cryptography as a building block of several applications while locally verifiable aggregate signature schemes are an extremely recent primitive with potential many future applications. When looking at (informal) definitions and (RSA and pairing based) instantiations of both primitives, a similarity appears. The interest of establishing the relation between those two primitives is obviously a better (and simplified, if they are actually the same) understanding of them, and an improvement of the state of the art: e.g. the pairing-based instantiation of locally verifiable aggregate signature scheme will bring new features to accumulators' literature if it can be turned into an accumulator scheme. In this paper we try to prove that (single-signer) locally verifiable aggregate signature schemes are actually accumulator schemes. We were able to prove that accumulators can be built from locally verifiable aggregate signature schemes, and that the construction is working the other round under some conditions on the used accumulator. However, in both cases we were not able to prove the security of our constructions and leave it as an open problem.

1 (Informal) Definitions

Signatures. A signature scheme is composed of three algorithms $\mathsf{Setup}, \mathsf{Sign}$ and Verify such that

- Setup takes as input the security parameter of the scheme and returns a pair of signing and verification keys $(\mathsf{sk}, \mathsf{vk})$;
- Sign takes as input a signing key and a message, and outputs a signature σ;
- and Verify takes as input a verification key, a message and a signature and returns 1 if the signature is valid, 0 otherwise.

Signature schemes must satisfy *correctness of signing*, meaning that for all security parameter, all signing and verification keys generated honestly, all messages and all signatures generated honestly, the Verify algorithm outputs 1. Regarding security, a signature scheme must satisfy *unforgeability*: an adversary that only knows the verification key (and has access to a signature oracle) cannot produce a message-signature pair that will pass the verification algorithm.

In the sequel we consider the *single-signer* setting, meaning that aggregation of signatures is possible only when signatures were generated using the same key verification. We restrict our attention to this setting as there are similarities with cryptographic accumulators only when considering single-signer (locally verifiable) aggregate signature schemes.

Aggregate Signatures. An aggregate signature scheme [BGLS03] is a signature scheme that also provides two poly-time algorithms $\mathsf{Aggregate}$ and $\mathsf{AggVerify}$, where

- $\mathsf{Aggregate}$ takes as input a verification key, along with a set of message-signature pairs and returns a shorter aggregate signature $\hat{\sigma}$;
- and $\mathsf{AggVerify}$ takes as input a verification key, a set of messages and an aggregate signature, and outputs 1 if the aggregate signature is valid with respect to the set of messages, 0 otherwise.

Aggregate signatures must satisfy *correctness of signing* (as any signature scheme) and must also satisfy *correctness of aggregation*, meaning that for all security parameters, all signing and verification keys generated honestly, all messages, all signatures and all aggregate signatures generated honestly, the $\mathsf{AggVerify}$

algorithm outputs 1. Regarding efficiency, aggregate signature schemes have a *compactness of aggregation* requirement: the size of an aggregate signature is a fixed polynomial in the security parameter, independent of the number of aggregations. As for security, an aggregate signature scheme must satisfy *aggregated unforgeability*, which is the same definition as *unforgeability* of signature schemes except that now the adversary must produce a set of messages along with a forged aggregate signature.

Locally Verifiable Aggregate Signatures. A locally verifiable aggregate signatures scheme [GV22] (LVAS), is an aggregate signature scheme with two additional algorithms $\mathsf{LocalOpen}$ and $\mathsf{LocalAggVerify}$ such that

- $\mathsf{LocalOpen}$ takes as input a verification key, an aggregate signature, a set of $l \in \mathbb{N}$ messages and an index ind in $[l]$ and returns auxiliary information aux corresponding to the message of index ind;
- and $\mathsf{LocalAggVerify}$ takes as input a verification key, an aggregate signature, *one* message and auxiliary information associated, and returns 1 if the aggregate signature contains the signature of the message, 0 otherwise.

This local opening brings efficient verification: the verification algorithm takes as input *one* specific message instead of all messages, to prove that the signature of this message is indeed in the aggregate signature. Such an aggregate signature scheme must satisfy *correctness of local opening* meaning that for all security parameter, all signing and verification keys generated honestly, all messages, all honestly computed signatures and aggregate signatures, and all honestly generated auxiliary information, the $\mathsf{LocalAggVerify}$ algorithm returns 1. It must also satisfy *compactness of aggregation* as for aggregate signature schemes and *compactness of opening* that requires that the size of the auxiliary information is fixed polynomial in the security parameter, and is independent of the number of aggregations.
Regarding security, LVAS scheme must satisfy *aggregated unforgeability with adversarial opening*: this definition is similar to *unforgeability* of signature schemes except that this time the adversary must produce a tuple of aggregate signature-auxiliary information-message that passes the $\mathsf{LocalAggVerify}$ algorithm.

Asymmetric Cryptographic Accumulators. Cryptographic accumulators [Bd94] were introduced in 1993 by Benaloh and De Mare as a compact way to represent a set of elements. There exist in the literature two kinds of accumulators, as identified by [KLL14]: *asymmetric* and *symmetric* accumulators. An asymmetric accumulator scheme is composed of four algorithms:

- Gen, the generation algorithm that takes as input the security parameter of the scheme, and outputs the accumulator keys, which are a pair of public key-master secret key $(\mathsf{pk}_{\mathsf{acc}}, \mathsf{sk}_{\mathsf{acc}})$;
- Eval, the evaluation algorithm that takes as input the accumulators keys, and a set of elements and outputs the compact representation acc of this set (which is called the “accumulator”);
- $\mathsf{WitCreate}$, the witness creation algorithm that creates a witness wit_x that an element x, given in input, is in the set given as input along with its accumulator;
- Verify, the verification algorithm that outputs 1 if the witness is correct, meaning that the element is indeed in the accumulated set, 0 otherwise.

One aim of cryptographic accumulators is to produce accumulators and witnesses that have constant size, meaning that the sizes do not dependent on the number of elements in the accumulated set (but are polynomial in the security parameter). An accumulator is said to be *correct* if for all honestly generated keys, all honestly computed accumulators and witnesses, the Verify algorithm returns 1.
Regarding security, there are several properties but, in this work, we only focus use on *collision resistance* [BP97] meaning that given the accumulator public key it is hard for an adversary to find a set $\mathcal{X}$ and a value $y \notin \mathcal{X}$ and build a witness wit_y such that $\mathsf{Verify}(\mathsf{pk}_{\mathsf{acc}}, \mathsf{acc}_{\mathcal{X}}, \mathsf{wit}_y) = 1$, where $\mathsf{acc}_{\mathcal{X}} = \mathsf{Eval}(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}, \mathcal{X})$.
The Eval (resp. $\mathsf{WitCreate}$) algorithm can take as input either both the public and secret keys of the scheme, or only the public key. In the first case, we say that the scheme has private evaluation (resp. witness generation), while in the second case we say that the scheme has public evaluation (resp. witness generation). It is also possible to have private (resp. public) evaluation while having public (resp. private) witness generation.

Symmetric Cryptographic Accumulators. A symmetric accumulator is the same than an asymmetric accumulator, except that it does not need witness for verification and therefore does not have a $\mathsf{WitCreate}$ algorithm. Regarding security, it must be hard for an adversary that is given an accumulator along with its associated set, to find another set such that it also corresponds to the given accumulator.

2 Formalizing the Link Between Both Primitives

When reading the above (informal) definitions, many similarities caught our attention. Therefore, we tried to establish a link between all those primitives. We start with the relation between symmetric accumulator schemes and signatures schemes as cryptographic accumulator were originally defined in the symmetric way and as an alternative to signature schemes.

Signature and Symmetric Accumulator. We can easily build a signature scheme from an accumulator scheme with private evaluation and public generation. Indeed, set $\mathsf{sk} = \mathsf{sk}_{\mathsf{acc}}$, $\mathsf{vk} = \mathsf{pk}_{\mathsf{acc}}$, let the signature of a message m be the accumulator of the set $\{\mathsf{m}\}$ and let the signature verification be the accumulator verification algorithm. Correctness of the signature scheme comes straight from the correctness of the accumulator scheme. The construction can be done the other way round: the accumulator scheme can be build from the signature scheme, however it results in a *bounded* accumulator scheme [CL02], meaning that only a given number of values can be accumulated, with bound equals to 1 in our case. As for security, the question is more tricky: an adversary of the accumulator security will try to find another message that has the same signature as one given as challenge, while an adversary of the signature security will try to produce a new pair of message-signature. We were not able to make the reduction from one to the other as both adversaries are requiring different inputs, and have different outputs that cannot be used to solve the other's security game.

Aggregate Signatures and Symmetric Accumulator. Using an aggregate signature scheme to build a symmetric accumulator will solve the above problem of the bound equals to 1. Indeed, when the set to accumulate contains more than one element, the Eval algorithm runs the Sign algorithm on all elements of the set, then queries the $\mathsf{Aggregate}$ algorithm on all pairs of element-signature and returns the aggregate signature as the accumulator of the set. However, there is some issue when defining the verification algorithm Verify from the aggregate verification algorithm $\mathsf{AggVerify}$: the latter is expecting as input a set of messages while the former is only expecting one element (i.e. one message). This difference of syntax traduces the fact that $\mathsf{AggVerify}$ is doing verification for all messages at the same time, while Verify is doing verification for one element (i.e. message) only. Thus the construction in this way is not working.

Now let us do the construction in the other way round. It is more complicated as we need to find a way to define $\mathsf{Aggregate}$. Using an *additive* accumulator scheme [CL02] might be the answer, as such accumulator schemes provide an algorithm Add that takes as input the scheme (secret and) public key(s), a set of elements along with its accumulator and an element y to add, and returns an accumulator corresponding to the union of the original set and $\{y\}$. Then $\mathsf{Aggregate}$ iq defined as follow: it runs Eval on one message of the set given as input, then runs the addition algorithm Add to obtain an accumulator of all messages, that will be output as an aggregate signature. However notice that the $\mathsf{Aggregate}$ algorithm takes as input only the verification key, therefore we need to use an accumulator scheme with *public* addition, meaning that Add only takes as input the accumulator public key and not the accumulator secret key. As for the aggregate verification algorithm $\mathsf{AggVerify}$, we can define it such that it runs the accumulator verification algorithm Verify for all messages taken as input and outputs 0 if Verify returns 0 at least once.

Regarding security, we try to prove security of the above construction through a security reduction but again due to different inputs and outputs in the accumulator and aggregate signature security game, we were not able to make the proof.
Also we notice something important regarding the above construction: while this is not proven formally in the literature, it is admitted that symmetric accumulator schemes have accumulators sizes linear in the number of accumulated elements [KLL14]. Thus, if we build an aggregate signature scheme from a symmetric accumulator, it will not satisfy *compactness of aggregation* as an aggregate signature is an accumulator in our construction.

To solve the above issue on the verification algorithms different inputs, one might want to use locally verifiable aggregate signature scheme instead of aggregate signature scheme as the former provides a verification algorithm for *one* message only. However this verification algorithm requires also as input some auxiliary information, that symmetric accumulator schemes cannot provide but asymmetric accumulator schemes can through the use of witnesses. That is why we finally compare asymmetric accumulators to locally verifiable aggregate signature schemes.

Locally Verifiable Aggregate Signatures and Asymmetric Accumulators. We now come to our initial objective: finding a relation between LVAS and asymmetric accumulators. We can easily build an asymmetric accumulator scheme (with private evaluation, *i.e.* with $\mathsf{sk}_{\mathsf{acc}}$ taken as input, and public key generation, *i.e.* with only $\mathsf{pk}_{\mathsf{acc}}$ as input) from a LVAS scheme:

- the Gen algorithm runs the Setup algorithm;
- the Eval algorithm runs the Sign algorithm for each element of the set, then runs the Aggregate algorithm on all obtained signatures;
- the WitCreate algorithm runs the LocalOpen algorithm;
- and the Verify algorithm runs the LocalAggVerify algorithm.

Here all algorithm are consistent in the inputs/outputs, the correctness of the accumulator scheme directly comes from the LVAS *correctness of local opening* and the scheme has constant size accumulators and witnesses thanks to the LVAS *compactness of aggregation* and *compactness of local opening*.

The other way round, the construction is a little bit more complicated but still possible:

- Setup runs the accumulator scheme algorithm Gen;
- Sign runs the Eval algorithm on a singleton;
- Verify runs the algorithms WitCreate and Verify of the accumulator scheme;
- Aggregate runs the Eval algorithm on a set composed of one message then run the Add algorithm to include all the other messages' signatures;
- AggVerify runs the accumulator scheme algorithms WitCreate and Verify for each messages;
- LocalOpen runs the algorithm WitCreate;
- and LocalAggVerify runs the accumulator Verify algorithm.

To make the construction work, the accumulator scheme must have private evaluation and public witness creation, and it also must be additive with a publicly computable Add algorithm (as already seen for the symmetric accumulator). Notice that if the accumulator scheme allows *subset queries* [DKNS04] meaning that witnesses can be done for a subset of elements instead of for one element only, then AggVerify becomes more efficient. It is straightforward that correctness of the LVAS comes from the accumulator scheme correctness. Regarding compactness of aggregation and local opening, they are satisfied if the accumulator scheme has constant size accumulators and witnesses.

As for the security, the *unforgeability of local opening* of the LVAS scheme seems kind of similar to the *collision resistance* security property of accumulator schemes. But there are tricky parts in these reductions: first what about the unforgeability of the aggregate signature? To prove that an adversary cannot forge an aggregate signature that is actually an accumulator, we have to prove that the accumulator itself is somewhat "unforgeable". To our knowledge, there is not such security property existing in the cryptographic accumulator literature. Then, what about oracles queries? In the collision resistance security game, the adversary is allowed to query the oracle on any sets without restrictions. In the aggregated unforgeability with adversarial opening security game, the adversary can query any messages to the oracle except the challenge message. When making security reductions from one adversary to the other, we encountered some restrictions that changed the adversary advantage in winning the collision resistance security game, in a way that we were not able to properly estimate.

Conclusion. While at first glance, single-signer locally verifiable aggregate signature schemes and asymmetric cryptographic accumulator schemes seems to work in the same manner, we were not able to prove that actually they were the same. Developing a more precise study of both primitives security might answer our question. Table 1 summarizes the different constructions we made in this study.

In red we highlight the most inefficient construction, and in green the best construction we made.

References

[Bd94] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT'93*, *LNCS* 765, pages 274–285. Springer, Heidelberg, May 1994.

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, *LNCS* 2656, pages 416–432. Springer, Heidelberg, May 2003.

Table 1. Summary of the different constructions. $\surd$ means "working", $\times$ means "not working", $\approx$ means "working under some conditions".

From	To	Problems	Working	Security
Symmetric accumulator	Signature	Existence	$\approx$	$\times$
Signature	Symmetric accumulator	Bounded by 1	$\surd$	$\times$
(Additive) symmetric accumulator	Aggregate signature	public Add, sizes	$\approx$	$\times$
Aggregate signature	Symmetric accumulator	Verify/AggVerify	$\times$	$\times$
LVAS	Asymmetric accumulator	No	$\surd$	$\times$
(Additive) Asymmetric accumulator	LVAS	Public Add	$\approx$	$\times$

[BP97] N. Bari and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT'97*, *LNCS* 1233, pages 480–494. Springer, Heidelberg, May 1997.

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002*, *LNCS* 2442, pages 61–76. Springer, Heidelberg, August 2002.

[DKNS04] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In *EUROCRYPT 2004*, *LNCS* 3027, pages 609–626. Springer, Heidelberg, May 2004.

[GV22] R. Goyal and V. Vaikuntanathan. Locally verifiable signature and key aggregation. In *CRYPTO 2022, Part II*, LNCS, pages 761–791. Springer, Heidelberg, August 2022.

[KLL14] A. Kumar, P. Lafourcade, and C. Lauradoux. Performances of cryptographic accumulators. *Local Computer Networks*, 2014.