Practical Non-interactive Multi-signatures, and a Multi- to Aggregate Signatures Compiler

Matthieu Rambaud
Télécom Paris

Christophe Levrat
Inria Saclay

ABSTRACT

In a fully non-interactive multi-signature, resp. aggregate-signature scheme (fNIM, resp. fNIA), signatures issued by many signers on the same message, resp. on different messages, can be succinctly "combined", resp. "aggregated". fNIMs are used in the Ethereum consensus protocol, to produce the certificates of validity of blocks which are to be verified by billions of clients. fNIAs are used in some PBFT-like consensus protocols, such as the production version of Diem by Aptos, to replace the forwarding of many signatures by a new leader. In this work we address three complexity bottlenecks. (i) fNIAs are costlier than fNIMs, e.g., we observe that the fNIA of BGLS (Eurocrypt'03) over 3000 signatures, takes 100x longer verification time than a batch verification of 3000 Schnorr signatures. (ii) fNIMs impose that each verifier processes the setup published by the group of potential signers. This processing consists either in verifying proofs of possession (PoPs), such as in Pixel (Usenix'20) and in the IETF'22 draft inherited from Ristenpart-Yilek (Eurocrypt'07), which costs a product of pairings over all published keys. Or, it consists in re-randomizing the keys, such as in SMSKR (FC'24). (iii) Existing proven security bounds on efficient fNIMs do not give any guarantee in practical curves with 256bits-large groups, such as BLS12-381 (used in Ethereum) or BLS12-377 (used in Zexe). Thus, computing in much larger curves is required to have provable guarantees.

Our first contribution is a new fNIM called dms, it addresses both (ii) and (iii). It is as simple as adding Schnorr PoPs to the schoolbook pairing-based fNIM of Boldyreva (PKC'03). (ii) For a group of 1000 signers, verification of these PoPs is: 5+ times faster than for the previous pairing-based PoPs; and 3+ times faster than the Verifier's processing of the setup in SMSKR (and contrary to the latter, needs not be re-started when a new member joins the group). (iii) We prove a tight reduction to the discrete logarithm (DL), in the algebraic group model (AGM). Given the current estimation of roughly 128 bits of security for the DL in both the curves BLS12-381 and BLS12-377, we deduce a probability of forgery of dms no higher than about 2^{-93} for a time 2^{80} adversary. This reduction is our main technical contribution. The only related proof before was for an interactive Schnorr-based multi-signature scheme, using Schnorr PoPs. Our strategy easily fills a gap in their proof, since we take into account that the adversary has access to a signing oracle even before publishing its PoPs. But in our context of pairing-based multi-signatures, extraction of the keys of the adversary is significantly more complicated, since the signing oracle produces a correlated random string. We finally provide another application of dms, which is that it can be plugged in recent threshold signatures without setup (presented by Das et al at CCS'23, and Garg et al at SP'24), since these schemes implicitly build on any arbitrary BLS-based fNIM.

Our second contribution addresses (i), it is a very simple compiler: $\mathscr{M}to\mathscr{A}$ (multi-to-aggregate). It turns any fNIM into an fNIA, suitable for aggregation of signatures on messages with a prefix in common, with the restriction that a signer must not sign twice using the same prefix. The resulting fNIA is post-quantum secure as soon as the fNIM is, such as Chipmunk (CCS'23). We demonstrate the relevance for Diem by applying $\mathscr{M}to\mathscr{A}$ to dms: the resulting fNIA enables to verify 39x faster an aggregate of 129 signatures, over messages with 7 bits-long variable parts, than BGLS.

1 Introduction 1
2 Preliminaries 4
3 New definitions 5
4 $\mathcal{M}$to$\mathcal{A}$: Multi to Aggregate Compiler 7
5 dms 7
6 Proof of Theorem 4 9
7 Evaluation and Comparison 11
8 Acknowledgements 13
A Further formalization and Optimization of $\mathcal{M}$to$\mathcal{A}$ 15
B Details for Applications of dms 16
C Details for application of $\mathcal{M}$to$\mathcal{A}$ to consensus 16
D Further details on related works 19

1 INTRODUCTION

In an aggregate signature scheme [23, 15, 78, 37, 74, 60, 25, 64, 30, 45, 69, 81, 90], a single short string takes the place of n individual signatures by n signers on n messages. Our focus is what we call fully non-interactive aggregate signature schemes (fNIAs) [23, 15, 37, 74, 64, 45, 69, 81, 90]. They offer an aggregation algorithm Ag which, roughly, takes as input any multiset of triples of public key - message - valid individual signature: $(\mathsf{pk}_i, m_i, \Sigma_i)_{i \in [n]}$, and outputs a single aggregate signature Σ. Finally, there is a public verification algorithm Vf which takes as input a purported aggregate signature Σ with respect to a multiset of pairs of public key - message: $(\mathsf{pk}_i, m_i)_{i \in [n]}$, and outputs a bit denoting acceptance or rejection. Such schemes have the unforgeability property, that acception implies that for any pair $(\dot{\mathsf{pk}}, \dot{m})$ in the multiset such that the key $\dot{X}$ was generated by some honest process, then it must have signed the corresponding message $\dot{m}$. *Fully non-interactive multisignature schemes* (fNIMs) enable aggregation only over *identical* messages. We then denote N the number of messages, instead of n, and dub aggregation the combination algorithm: Cb : $m, (\mathsf{pk}_i, \Sigma_i)_{i \in [N]} \longrightarrow \Sigma$. Pairing-based fNIMs are used in the consensus protocol of Ethereum [48, 50], they enable clients to verify that a block was voted by enough validators [3].

Goal (i): reducing the efficiency gap between the verification times of fNIAs and of fNIMs. In all pairing-based fNIAs [15, 37, 74, 69, 81], following BGLS [23], the complexity of the Verifier is at least a product of $n + 1$ pairings. By contrast, in nearly all pairing-based fNIMs [20, 78, 86, 21][22, §6][47, 56, 9], the online verification complexity of the Verifier is mostly two pairings. Our terminology "online" is to differentiate from the other task of the Verifier, which we will call "processing of the group setup" and discuss separately. Concretely, in Table 7 we observe that the online verification time of the verifier in any existing pairing-based fNIA, for 3000 signatures over different message contents, is at least 300× higher than the online verification of any of the previous pairing-based fNIMs over 3000 signatures on identical message contents. Turning to lattice-based fNIMs, the state of the art called Chipmunk [52, 51] enjoys a verification 6.5× faster for $N = 8192$ than the naive verification of individual Falcon signatures. Whereas, there exists no public evaluation, to our knowledge, of the Verifier runtimes of the state of the art lattice-based fNIAs [90, 1]. They consist of SNARKs of signatures, using the system called Labrador [19], which has linear Verifier complexity. Apart from them, the fNIA [45] was recently broken [29].

This efficiency gap is best illustrated by blockchain consensus algorithms, say, among n_{C} processes. The fastest ones are known as "leader-based": [62, 72, 58, 44] (their liveness requires partial synchrony). The most recent implementation used in production is the one of Facebook's Diem21 ([44]) by Aptos. It proceeds by iterations called "rounds". Under good conditions, the leader of a new round in Diem21 only has to combine $(2/3)n_{\mathrm{C}}$ identical votes with a fNIM, into a multi-signature dubbed a "quorum certificate" (QC), which it multicasts. But if the leader of the previous round was corrupt or the network not synchronous, then the current leader must aggregate $(2/3)n_{\mathrm{C}}$ signatures over *different* so-called "timeout messages". It multicasts the aggregate signature, called a "timeout certificate". Aggregation is done by Aptos ([5]) with BGLS. Hence, already for $(2/3)n_{\mathrm{C}} = 129$, we observe in Table 7 that verification of a timeout certificate is 49× slower than verification of a QC. For convenience we recall Diem21 in Figure 9, for simplicity with aggregation instead instantiated as a naive concatenation of signatures.

Goal (ii): reducing the complexity of processing of the group setup. All pairing-based fNIMs [86, 24, 47, 21, 56, 9] require costly additional tasks from the Verifier. Namely, the group of potential signers must initially publish the outcome of their group setup, which we call generically the "keys of the group" and denote $\mathcal{KG}$. The Verifier must then process $\mathcal{KG}$, we call this task *processing of the group setup*.

In a first category of fNIMs ($\mathcal{MSP}$*-pop* [86, 22, 24], Pixel and $\mathcal{ASMP}$*-pop* [86, 47, 22, 24]), each signer incorporates a so-called *proof of possession* (PoP): π into its public key: $\mathsf{pk} = (X, \pi)$. The purpose of the PoP is to enforce (possibly with a loose reduction) that its issuer "knows" a secret key corresponding to X. PoPs thus somehow emulate the model called "knowledge-of-secret-key" (kosk). The kosk assumes that the adversary gives to the reduction a secret key for every public key appearing in its forgery (excepted the target one). Interest of the kosk model is that the security of the two fastest known fNIMs: [20, 78] is proven only in this model. The fNIM of Boldyreva [20] has verification complexity of only two pairings, but without the kosk it is vulnerable to so-called "rogue key attacks" [21]. There, the adversary creates a forgery involving public keys, other than the target one, for which it does not know corresponding secret keys. In all previous works, the PoP π is equal to a pairing-based signature (BLS [26]) on the public key itself. The Verifier then has to verify the PoPs of all the keys: $\mathcal{KG}$ of the group of signers. In Table 6 we estimate that their (batched & optimized) verification for an 2702-sized group takes 1947ms on a laptop. As clear from Table 7, this time is orders of magnitude longer than the online verification of a multi-signature.

In a second category of fNIMs ($\mathcal{ASMP}$/ $\mathcal{ASMP}$*-pop* [22], SMSKR [9] and SIG_1 [27]), the processing of the group setup consists in (re-)computing a so-called verification key for the group, out of the list of their published keys: $\mathcal{KG}$ In $\mathcal{ASMP}$ and SIG_1 this verification key is of constant size (at the cost of an interactive setup). While in SMSKR, each signer in the group re-randomizes its secret signing key based on all other published keys: $\mathcal{KG}$. The Verifier then has to compute the re-randomized public keys accordingly: these will be the ones used for verifying signatures (both individual and combined). Again, as evidenced in Tables 6 and 7, this task is the bottleneck of the Verifier since it takes three orders of magnitude longer than the online verification of signatures. Worse: unlike verification of PoPs, the group key must be re-computed each time there is a new group member, since re-randomization depends on the list of published keys: $\mathcal{KG}$.

A last category of fNIMs (the blog version $\mathcal{MSP}$*-blog* [21, 56]) does not require processing of the group setup tasks, beyond verification of membership of the keys in the subgroup $\mathbb{G}_2$. But its online verification requires to compute a combined verification key, equal to the sum of the re-randomizations of the public keys of the signers. We note that this computation is comparatively faster than computing re-randomized keys separately, since it can be done in one single N-sized multi-exponentiation (plus N times $N\kappa$-sized hashes).

Goal (iii): achieving provable security for use with the curves used in practice. In Table 8 we observe that no existing fNIM is proven safe to use with the curves used in practice: BLS12-381, adopted by Ethereum, and BLS12-377, proposed by Zexe [31]. Despite these curves having a discrete-log-in-subgroup (DL) problem of estimated security close to 128 bits [71, 6], when instantiating with them the fNIM $\mathcal{MSP}$*-pop* [86, 22, 24] (IETF draft standard 2022), we find that the proven bound on the probability of a forgery after 2^{80} clock cycles is higher than 2^{-13}. This estimate is optimistic, since under the assumption that co-CDH would be as hard as DL. Other pairing-based fNIMs [47, 21, 56, 9] are not either proven secure with such curves (the proven formulas would give an upper-bound higher than 1). Finally, the lattice-based fNIMs [52, 51] have public keys of logarithmic size in the number of signatures allowed, since they are equal to commitments to vectors of one-time public keys. A first exception is the fNIA called $\mathcal{AS}$-4 (long version of Bellare-Namprempre-Neven [15], following a trick of Katz-Wang [73]). But its verification complexity is prohibitive, since it is a product of $N + 1$ pairings, as in all related fNIA schemes. The second and last exception is the recent scheme SIG_1 of [27], but which has a quadratic Verifier's processing of keys (see Sec. 7 and below). So this raises the natural question of finding a fNIM with tight reduction to the discrete logarithm (DL), and thus usable with practical curves. This has been achieved so far with multisignatures [42, 13] requiring two rounds of interaction, and thus which are not usable in consensus protocols such as Diem21 [44].

1.1 An Efficient Multi-to-Aggregate Compiler

We address Goal (i) by introducing $\mathcal{M}\text{to}\mathcal{A}$: it is a compiler which transforms any fNIM into a special-purpose fNIA. The resulting fNIA applies to messages divided into two parts: a common prefix τ, called the tag, and the remaining message v_i, of bitlength denoted $|v|$. The resulting fNIA is particularly efficient when the length $|v|$ of the variable contents v_i is only of a few bits. $\mathcal{M}\text{to}\mathcal{A}$ operates as follows. Each signer prepares and publishes $2|v|$ public verification keys. To sign a message $m_i = (\tau, v_i)$, it signs τ $|v|$ times: each time using the public key indicated by the j-th bit of v_i, for $j \in [|v|]$. As all the signatures of all signers are on the same τ, the verification cost is equal to verifying a combined multisignature. We see that a signer should not sign two different messages: v_i and v_i' that share the same tag τ (otherwise the adversary could cherry-pick signed bits from both v_i and v_i' to forge another signature). For this reason, the resulting fNIA is called "one-time-tagged".

1.1.1 *Performance and Applications of $\mathcal{M}\text{to}\mathcal{A}$; and related works.*

We resume the example of the production implementation [5] of Diem21 [44] by Aptos. A timeout message of a process $\mathcal{P}_j$ is formatted as the signed message: $\{r, r_j\}_j$. The tag r is the current round number, while $r_j < r - 1$ is roughly the highest round number in which $\mathcal{P}_j$ saw a QC: $qc_{high,j}$. Hence, r_j can be advantageously encoded as the difference $v_j := r - r_j - 1$ (this improves by -1 a nice idea of [61]). Thus, after the network becomes synchronous and assuming that each leader is honest with probability 2/3, then the expected value of v_j is only 0.5. Hence, the timeout messages fall in the regime where $\mathcal{M}\text{to}\mathcal{A}$ is efficient. In Table 7 we report on the Verifier's runtime for an $\mathcal{M}\text{to}\mathcal{A}$ aggregate over 129 signers, calibrated with a variable parts of bitlength $|v| = 7$ (which is overkill for Diem21, by the above considerations). Thus, verification consists of verifying a $N = 7n$-wise multisignature. We used as input any of the three BLS-based fNIMs: $\mathcal{MSP}$*-pop*, SMSKR and our dms (below), which have the same online verification complexity for a given curve. We achieve an online verification runtime of 3.4ms. It is close to the batch verification of a naive concatenation of 129 Schnorr signatures (of 1.9ms, note that $\mathcal{M}\text{to}\mathcal{A}$ is 3× faster for 3000 signers). This is 39× faster than the verification time: 116.6ms of an 129-wise BGLS aggregate signature, as used so far in the implementation of Diem21 [5].

One-time-tagged fNIA schemes were considered in [4, 70, 61], but all based on non-post-quantum assumptions (the latter inspired $\mathcal{M}\text{to}\mathcal{A}$). By contrast, $\mathcal{M}\text{to}\mathcal{A}$ has post-quantum security whenever applied to any post-quantum fNIM, e.g. to [51]. Another use-case where $\mathcal{M}\text{to}\mathcal{A}$ is advantageous is suggested by [4]: they consider connected devices signing short measurements (such as the temperature, weight, speed), one-time-tagged with the time of the measure.

1.2 A Faster and Tightly Secure fNIM

We introduce a fully non-interactive multisignature scheme which achieves both goals (ii) and (iii), called Dynamic Multi-signature with Schnorr proofs of possession (dms). It is as simple as augmenting the pairing-based (BLS) multisignature scheme of Boldyreva [20] with proofs of possession (PoP) consisting of Schnorr signatures of processes on their public keys. Since batch verifying N Schnorr signatures [14] is much faster than computing $N + 1$ pairings, it is not surprising that the verification of PoPs in dms achieves a $> 5\times$ speedup over the most optimized verification of the ones of $\mathcal{MSP}$*-pop* [86, 22, 24]: see Table 6. As any PoP-based fNIM, dms comes with the bonus of being dynamic, i.e., the signing algorithm needs not taking as input a group of potential signers $\mathcal{KG}$. In turn, individual signatures can be combined without the restriction that the signers agreed together on some common group $\mathcal{KG}$. Finally, as in any PoP-based fNIM, when new potential signers, say, 14, publish their keys, the marginal cost of the Verifier is only to verify 14 PoPs. In Table 6 we evaluated this task to be $> 500\times$ faster than the Verifier's processing in SMSKR when new members join the group, since it must re-randomize all published keys.

Proving tight reduction of dms *to* DL *(in the AGM).* The proof does not follow from previous works, it is one of our main contributions. The literature of the last 25 years suggests that mere Schnorr ZK PoKs of secret keys might not simply provably thwart rogue key attacks ([80] Micali-Ohta-Reyzin "That is, for the simulator to be polynomial time, there can be at most logarithmically many signers" and Bellare-Neven [16] "one would require ZK PoKs extractable under such concurrent conditions. This eliminates many standard protocols, including standard POKs of discrete logarithms."). Worse, the attempt [8] had its proof invalidated by [46], and finally a proof attempt, on a related issue (CCA security from Schnorr PoKs of randomness) [18], required a sophisticated revisiting [54], despite being in the algebraic adversary model. We consider the *algebraic group model* (AGM), in line with [84, 2, 18, 54, 53, 11, 12, 13, 9, 7, 41]. Namely, our security bounds hold against so-called *algebraic adversaries*. Heuristics partly supporting the AGM are that, according to [71, §4][6], no better attacks are known against DL in BLS12-377/381, than the generic square-root algebraic one (see Sec. 7.3).

The main difficulty of our proof consists, upon being given a forgery w.r.t. a set of public keys $(X_i, \pi_i)_{i \in [N]}$, in extracting from the forger all the secret keys, i.e., ξ s.t. $X_i = \xi.G_2$ where G_2 is a public generator of the subgroup $\mathbb{G}_2$ (except the one of the challenge public key $\dot{X}$). The last step from there is that the reduction builds a valid individual signature for $\dot{X}$ on the challenge message, thus breaking unforgeability of BLS. This last step is as in [20], with the minor twist that the set $(X_i)_{i \in [N]}$ could well contain multiple copies of $\dot{X}$, without the forger giving $\dot{x}$ to the reduction. The *main difficulty* is singled-out in Sec. 6 as an abstract extractability game, called ssc (Schnorr straight-line extraction despite correlations). It considers an adversary having access to a signing oracle for the BLS signature of a given key X (playing the role of $\dot{X}$). Then it outputs one or several keys appended with valid proofs of possession: (X^*, R^*, z^*),... (other than the one of the challenge). It wins the game if the challenger fails to extract in straight-line one of the corresponding secret keys x^*. The game is not comparable to [53, §6], where the adversary had instead to forge a BLS signature. In

Theorem 4 we show that the advantage of the adversary is bounded by the advantage in the DL game. The proof is strictly more difficult than in [42, §A], since their adversary does not have access to a signing oracle for the target key $\dot{X}$. Absence of this oracle in [42, §A] invalidates the security proof of their multisignature (page 10, step "key registration"), because in the unforgeability definition ([42, §5.1] and our Sec. 3) the adversary *has* the power to query such an oracle *potentially before* it chooses the set of keys of its forgery. Recall that the adversary has indeed the power to engage in signing sessions with concurrent groups containing the same target key $\dot{X}$, before it registers the set of keys $(X_i)_{i \in [N]}$ of its forgery. On the one hand, we observe that [42, p10] can easily be fixed (we notified this on 01-23 2024 to the authors). Indeed, the proof of [42, §A] would go unchanged after adding the necessary Schnorr-signing oracle, since it would return uniformly random group elements. Whereas our signing oracle produces a *correlated* sequence of group elements: $\left\{\left(\mathrm{H}(m_j), \dot{x}.\mathrm{H}(m_j)\right)_j, \dot{X} = \dot{x}.G_2, G_2\right\}$, j running over the signing queries. These correlations make necessary for our reduction to DL to follow two alternative behaviors (the second, called D, is designed to cope with the event which we call "very bad", defined in Eq. (16)), instead of one single behavior in [42, §A].

Applications of dms . It enables to divide by N the online storage size of certificates of validity for blocks [48, 50, 3], compared to a batch of Schnorr signatures. Although fNIMs are already used in Ethereum for this purpose, Table 8 shows that previous fNIMs were proven secure only in curves much larger than BLS12-381 or -377. This would imply non-standard curves, larger storage size, and longer verification time. Moreover the 5× processing of the group setup speedup of dms (for equal curves) directly impacts billions of verifiers. Contrary to a common belief, individual verification of signatures of a fNIM can be made faster than the one of Schnorr signatures. *First*, as observed in [38] and confirmed by Table 7, *batch verification* of BLS individual signatures (as used in $\mathcal{MSP}$*-pop*, SM-SKR, dms) is 3× faster than batch verification of Schnorr signatures, for $n = 3073$ signatures. The Verifier simply combines the signatures (the cost of these n additions is negligible) then verifies the obtained multisignature. Batch verification virtually always succeeds in use-cases such as signatures published in blocks (otherwise, the validators of the blocks would be severely punished). *Second*, for use-cases where invalid signatures often occur, then it was observed by [56, 34] that signers can add a Chaum-Pedersen proof of equality of discrete logs to their BLS signatures, enabling an individual verification time comparable to the one of a Schnorr signature. We give further details in Appendix B. In Appendix B we also explain how dms can be advantageously plugged in the threshold signature schemes [43, 57], in place of $\mathcal{MSP}$*-pop*.

2 PRELIMINARIES

We use the formalism of games [17], with the simplification that we merge the finalization in the main body, as in [53], and that we use the more mainstream meaning of the *advantage of an adversary $\mathcal{A}$ in a game* $\mathbf{g}$ [12], denoted explicitely $\mathbb{P}(\mathbf{g}^{\mathcal{A}} = 1)$, to designate the probability that $\mathcal{A}$ wins the game, i.e., that $\mathbf{g}$ sets the flag win ← 1.

Bilinear groups. A bilinear group description ([55]) is a tuple $\mathbb{G} = (\mathbb{G}_1, \mathbb{G}_2, G_T, e, \phi, \psi, p)$ such that $\mathbb{G}_i$ is a cyclic group of prime order p for $i \in \{1, 2, T\}$, in additive notation; e is a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, i.e., for all $a, b \in \mathbb{Z}_p$ and all generators G_1 of $\mathbb{G}_1$ and G_2 of $\mathbb{G}_2$ we have that $G_T := e(G_1, G_2)$ generates $\mathbb{G}_T$ and $e(a.G_1, b.G_2) = ab.e(G_1, G_2) = ab.G_T$; $\phi : \mathbb{G}_1 \to \mathbb{G}_2$ is an isomorphism, and $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is an isomorphism. All group operations and the bilinear map e must be efficiently computable. $\mathbb{G}$ is of Type 1 if the maps ϕ and ψ are efficiently computable, in which case we will consider without loss of generality that $\mathbb{G}_1 = \mathbb{G}_2$; $\mathbb{G}$ is of Type 2 if there is no efficiently computable map ϕ; and $\mathbb{G}$ is of Type 3 if there are no efficiently computable maps ϕ and ψ.

In line with [37], all our assumptions and statements are with respect to a choice of fixed public generators G_1, G_2. Note that since we are in the random oracle model, we have a fortiori a public uniform random string (URS). So G_1, G_2 could be fixed by seeding any public uniform sampling algorithm with the URS.

The algebraic group model (AGM). In line with [84, 2, 18, 54, 53, 11, 12, 13, 9, 7, 41], we consider provable security against adversaries known as *algebraic algorithms.* We recall the most recent model in the setting of bilinear groups, from [11, Def 2]. An algorithm $\mathcal{A}$ executed in a security game ([17]) is called algebraic if for all group elements $Z \in \mathbb{G}_i$, $i \in \{1, 2, T\}$ that $\mathcal{A}$ outputs to any oracle of the game, it additionally provides a representation in terms of received group elements in $\mathbb{G}_i$ and those from groups from which there is an efficient mapping to $\mathbb{G}_i$. In particular for $Z \in \in \{\mathbb{G}_1, \mathbb{G}_2\}$: if $U_0, \ldots, U_\ell \in \mathbb{G}_1$ and $V_0, \ldots, V_m \in \mathbb{G}_2$ are the group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ received so far then $\mathcal{A}$ provides a list of coefficients in $\mathbb{Z}_p$ $(\mu_i)_{i \in [\ell]}$, and possibly $(\zeta_j)_{j \in [m]}$, such that, depending on the case:

- $Z \in \mathbb{G}_1$ (Type 1 and 2): $Z = \sum_i \mu_i U_i + \sum \zeta_j \psi(V_j)$
  (Type 3): $Z = \sum_i \mu_i U i$
- $Z \in \mathbb{G}_2$ (Type 1): $Z = \sum_i \mu_i U_i + \sum \zeta_j V_j$
  (Type 2 and 3): $Z = \sum_j \zeta_j V_j$

Signatures, example of BLS*.* We recall the standard notion of a digital signature scheme with existential unforgeability under chosen message attacks (EUF-CMA, [63, 20]). It is the data of algorithms for key generation Kg, signature Sign and verification Vf, the latter returning a bit denoting acceptance or rejection. They furthermore have the following properties. Correctness requires that $\forall(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Kg}(), \forall m, \mathsf{Vf}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$. Unforgeability requires that a polynomial adversary $\mathcal{B}$, dubbed forger, is unable to forge valid a signature on a message m for which it did not query a signature. More formally it is defined by a game, which for concreteness we examplify in Figure 1 on the example of the well-known BLS signature scheme [26, 37]. BLS is parametrized by a bilinear group with public generators: $G_1 \in \mathbb{G}_1$, $G_2 \in \mathbb{G}_2$, and by a hash-to-curve map $m \to \mathrm{H}(m) \in \mathbb{G}_2$ which we model as a random oracle.

- BLS.Kg(): sample $x \xleftarrow{\$} \mathbb{Z}_p$, output $(\mathsf{sk}, \mathsf{pk}) = (x, x.\mathbb{G}_2)$;
- BLS.Sign : $m \to \mathsf{sk}.\mathrm{H}(m)$;
- BLS.Vf : $\mathsf{pk}, m, \Sigma \to e(\Sigma, G_2) == e(\mathrm{H}(m), X)$.

In the random oracle (RO) model, the security of BLS has a loose reduction to the computational Diffie Hellman (CDH) problem. More precisely, over bilinear groups of type I and II it has a loose

```
BLS-uf
$x \xleftarrow{\$} \mathbb{Z}_p,\ X = x.G_2$
$(m^*, \Sigma^*) \leftarrow \mathcal{B}^{\textsc{Sign},\mathrm{H}}(X)$
return $\left(m \notin Q_{\mathrm{sig}}\ \wedge\ e(\Sigma^*, G_2) == e(\mathrm{H}(m^*), X)\right)$

oracle $\mathrm{H}(m)$
return $M \leftarrow \mathrm{H}(m)$

oracle $\textsc{Sign}(m)$
$Q_{\mathrm{sig}} \leftarrow Q_{\mathrm{sig}} \cup \{m\}$
return $x.\mathrm{H}(m)$
```

Figure 1: **EUF-CMA game for BLS**

reduction [26] to the problem called co-cdh: given $(Q \xleftarrow{\$} \mathbb{G}_1,\ a.G_2)$ for $a \xleftarrow{\$} \mathbb{Z}_p$, compute $a.Q$. Whereas over bilinear groups of type III, it has a loose reduction [37] to the problem called co-cdh*: given $(Q \xleftarrow{\$} \mathbb{G}_1,\ a.G_1, a.G_2)$ for $a \xleftarrow{\$} \mathbb{Z}_p$, compute $a.Q$. Note that in BDN18 [22], co-cdh and co-cdh* are respectively renamed ψ-co-cdh and co-cdh. In the RO+AGM model, the security of BLS has a tight reduction [53, §6] to the discrete logarithm (DL) in $\mathbb{G}_1$. We observe that over type II groups, a minor adaptation of the proof shows a tight reduction to the DL in $\mathbb{G}_2$ (use ψ to port the DL challenge in $\mathbb{G}_1$). We observe that over type III groups, the same proof shows a reduction to the type III variant of DL known as (1, 1)-DL [11]: given $(a.G_1, a.G_2)$, compute a.

3 NEW DEFINITIONS

We first define fully non-interactive multisignature schemes (fNIM), then fully non-interactive one-time-tagged aggregate signature schemes (fNIA), which are output by our compiler $\mathcal{M}$to$\mathcal{A}$. Our definitional choices aim at capturing most existing non-interactive schemes, capturing the remaining ones would require only straightforward adaptations. Our syntax is meant to be safely used as such by the Combiner/Aggregator or the Verifier, no hidden additional check is required from them. In the last subsection we discuss our choices w.r.t. related specifications. All algorithms take as additional input a multiset of public keys denoted $\mathcal{KG}$, called the keys of the group (of potential signers). A scheme is called dynamic when this input plays no role. In turn, all $\mathcal{KG}$-dependent restrictions in the specifications and security definitions are then removed. For simplicity we formalize our definitions and results with schemes in which the setup is non-interactive (provided implicit coordination within the group $\mathcal{KG}$ for non-dynamic schemes). Our compiler $\mathcal{M}$to$\mathcal{A}$ obviously extends to schemes with interactive setup, of which the three existing ones are [78], $\mathcal{ASMP}$ [22, §4] and $\mathcal{ASMP}$*-pop* [22, §6]. Both n and N denote arbitrary positive integers. The definitions involve only n-(or N-)sized multisets of tuples, such as $(\mathrm{pk}_i, \Sigma_\mathrm{i})_{i \in [N]}$ or $(\mathrm{pk}_i, v_i, \Sigma_\mathrm{i})_{i \in [n]}$, i.e., without ordering between each other. So our indices 1, i etc. are just here to name elements. The numbers N and n of messages to be combined/aggregated are variable inputs to the algorithms.

Fully Non-Interactive Multi-signatures

Definition 1 (fNIM). A fully non-interactive multisignature scheme consists of the following local algorithms.

- Kg() →(sk, pk) (key generation)
- $\mathrm{Sign}(\mathcal{KG}, \mathrm{sk}_i, m) \rightarrow \Sigma_\mathrm{i}$ (signing)
- $\mathrm{iVf}(\mathcal{KG}, \mathrm{pk}_i, m, \Sigma_\mathrm{i}) \rightarrow 0/1$ (individual signature verification)
- $\mathrm{Cb}\big(\mathcal{KG}, m, (\mathrm{pk}_i, \Sigma_\mathrm{i})_{i \in [N]}\big) \rightarrow \Sigma$ (combination)
- $\mathrm{Vf}\big(\mathcal{KG}, (\mathrm{pk}_i)_{i \in [N]}, m, \Sigma\big) \rightarrow 0/1$ (verification)

Moreover they should satisfy the following three properties.

Individual completeness states that for any message m, then a correctly generated signature on m by any correctly generated key with respect to any group $\mathcal{KG}$ to which it belongs, must pass individual verification. Formally:

(1)
$$\mathbb{P}\left(\ 0 \leftarrow \mathrm{iVf}\big(\mathcal{KG}, \mathrm{pk}, m, \dot{\Sigma}\big)\ \middle|\ \begin{array}{l} \dot{\mathrm{sk}}, \dot{\mathrm{pk}} \leftarrow \mathrm{Kg}() \\ \wedge\, \dot{\mathrm{pk}} \in \mathcal{KG} \\ \wedge\, \dot{\Sigma} \leftarrow \mathrm{Sign}\big(\mathcal{KG}, \dot{\mathrm{sk}}, m\big) \end{array}\right) = \mathrm{negl}$$

Combination-robustness states that combination must return a valid multisignature whenever applied on any number n of individual signatures on any message m, with respect to a common group $\mathcal{KG}$ containing all signers, as soon as they all pass individual verification. Formally, for any polynomial adversary $\mathcal{A}$:

$$(2)\quad \mathbb{P}\Bigg(0 \leftarrow \mathrm{Vf}\big(\mathcal{KG}, (\mathrm{pk}_i)_{i \in [n]}, m, \Sigma\big)\ \Bigg|\ \begin{array}{l} \mathcal{KG}, (\mathrm{pk}_i, \Sigma_\mathrm{i})_{i \in [n]} \leftarrow \mathcal{A} \\ \wedge\, (\mathrm{pk}_i)_{\,i \in [N]} \subset \mathcal{KG} \\ \wedge\, \mathrm{iVf}\big(\mathcal{KG}, \mathrm{pk}_i, m, \Sigma_\mathrm{i}\big) = 1\ \forall i \in [N] \\ \wedge\, \Sigma \leftarrow \mathrm{Cb}\big(\mathcal{KG}, m, (\mathrm{pk}_i, \Sigma_\mathrm{i})_{i \in [N]}\big) \end{array}\Bigg) = \mathrm{negl}$$

Unforgeability states that any polynomial forger $\mathcal{F}$ has negligible advantage in the following game denoted m-uf. A challenger exhibits to $\mathcal{F}$ an honestly generated key $\dot{\mathrm{pk}}$, then $\mathcal{F}$ commits to a group of public keys $\mathcal{KG}$, then it is granted access to a $\dot{\mathrm{pk}}$-signing oracle with respect to $\mathcal{KG}$. The forger wins if it can create a valid multi-signature on behalf of some subgroup of signers containing $\dot{\mathrm{pk}}$, on a message never queried to the oracle.

```
m-uf
$(\dot{\mathrm{sk}}, \dot{\mathrm{pk}}) \leftarrow \mathrm{Kg}()$
$\mathcal{KG} \leftarrow \mathcal{F}$
$\left((\mathrm{pk}_i)_{i \in [N]}, m^*, \Sigma^*\right) \leftarrow \mathcal{F}^{\textsc{Sign}}(\dot{\mathrm{pk}})$
return $\Big(m \notin Q_{\mathrm{sig}}\ \wedge\ \dot{\mathrm{pk}} \in (\mathrm{pk}_i)_{i \in [N]} \subset \mathcal{KG}$
  $\wedge\ \mathrm{Vf}(\mathcal{KG}, (\mathrm{pk}_i)_{i \in [N]}, m, \Sigma^*) = 1\Big)$

oracle $\textsc{Sign}(m)$
if $\dot{\mathrm{pk}} \in \mathcal{KG}$
  $Q_{\mathrm{sig}} \leftarrow Q_{\mathrm{sig}} \cup \{m\}$
  return $\dot{\Sigma} \leftarrow \mathrm{Sign}(\mathcal{KG}, \dot{\mathrm{sk}}, m)$
```

Figure 2: **Multi-signatures unforgeability game. Instructions removed in dynamic schemes are shaded-out**

Note that the definition in the non-dynamic case is weak, because the forger can make signing queries only *after* it has committed to its challenge group $\mathcal{KG}$ of keys. So this non-dynamic restriction rules-out a forger which would concurrently interact with multiple groups. But this restriction appears only in [9] to our knowledge, it is absent from most security definitions, even those of non-dynamic fNIMs ($\mathcal{ASMP}$ [22, §4.1],[13]). As observed in the Introduction, the definition of [42, §5.1] does not either make this restriction, although their security proof implicitly makes it.

Fully Non-Interactive (One-time-tagged) Aggregate-signatures

The following definition applies to messages to be signed which come as $m_i = (\tau_i | v_i)$ where the τ_i are called their tags and v_i their variable parts. Again, the number n of signatures aggregated is a variable input. Compared to classical aggregate signatures, aggregation is enabled only on messages with the same tag τ, and unforgeability is guaranteed only if honest signers do not sign two different messages with the same tag.

Definition 2 (fNIA). A fully non-interactive one-time-tagged aggregate signature scheme consists of the following local algorithms.

- $\mathsf{Kg}() \rightarrow (\mathsf{sk}, \mathsf{pk})$ (key generation)
- $\mathsf{Sign}(\mathcal{KG}, \mathsf{sk_i}, \tau_i, v_i) \rightarrow \Sigma_\mathrm{i}$ (signing)
- $\mathsf{iVf}(\mathcal{KG}, \mathsf{pk}_i, \tau_i, v_i, \Sigma_\mathrm{i}) \rightarrow 0/1$ (individual signature verification)
- $\mathsf{Ag}\big(\mathcal{KG}, \tau, (\mathsf{pk}_i, \Sigma_\mathrm{i}, v_i)_{i \in [n]}\big) \rightarrow \Sigma$ (aggregation)
- $\mathsf{Vf}\big(\mathcal{KG}, \tau, (\mathsf{pk}_i, v_i)_{i \in [n]}, \Sigma\big) \rightarrow 0/1$ (verification)

Moreover they should satisfy the following three properties.

Individual completeness requires that for any tagged message $(\tau|v)$, then a correctly generated signature on $(\tau|v)$ by any correctly generated key with respect to any group $\mathcal{KG}$ to which it belongs, must pass individual verification. Formally:

(3)

$$\mathbb{P}\left(0 \leftarrow \mathsf{iVf}\big(\mathcal{KG}, \dot{\mathsf{pk}}, \tau, v, \dot{\Sigma}\big) \;\middle|\; \begin{array}{c} \dot{\mathsf{sk}}, \dot{\mathsf{pk}} \leftarrow \mathsf{Kg}() \\ \wedge\, \dot{\mathsf{pk}} \in \mathcal{KG} \\ \dot{\Sigma} \leftarrow \mathsf{Sign}\big(\mathcal{KG}, \dot{\mathsf{sk}}, \tau, v\big) \end{array} \right) = \mathsf{negl}$$

Aggregation-robustness requires that aggregation must return a valid aggregate signature whenever applied on any number n of individual signatures on any identically tagged messages $m_i = (\tau, v_i)$, with respect to a common group $\mathcal{KG}$ containing all signers, as soon as they all pass individual verification. Formally, for any polynomial adversary $\mathcal{A}$:

$$(4) \quad \mathbb{P}\Bigg(0 \leftarrow \mathsf{Vf}\Big(\mathcal{KG}, \tau, (\mathsf{pk}_i, v_i)_{i \in [n]}, m, \Sigma\Big) \;\Bigg|\; \begin{array}{l} \mathcal{KG}, \tau, (\mathsf{pk}_i, v_i, \Sigma_\mathrm{i})_{i \in [n]} \leftarrow \mathcal{A} \\ \wedge\, (\mathsf{pk}_i)_{\, i \in [n]} \subset \mathcal{KG} \\ \wedge\, \mathsf{iVf}\big(\mathcal{KG}, \mathsf{pk}_i, \tau, v_i, \Sigma_\mathrm{i}\big) = 1 \;\forall i \in [n] \\ \wedge\, \Sigma \leftarrow \mathsf{Ag}\big(\mathcal{KG}, \tau, (\mathsf{pk}_i, v_i, \Sigma_\mathrm{i})_{i \in [n]}\big) \end{array} \Bigg) = \mathsf{negl}$$

Unforgeability requires that any polynomial forger $\mathcal{F}$ has negligible advantage in the following game denoted a-uf. A challenger exhibits to $\mathcal{F}$ an honestly generated key $\dot{\mathsf{pk}}$, then $\mathcal{F}$ commits on a group of public keys $\mathcal{KG}$, then it is granted access to a $\dot{\mathsf{pk}}$-signing oracle with respect to $\mathcal{KG}$, which however refuses to sign twice with the same tag. The forger wins if it can create a valid aggregate signature on behalf of some subgroup of signers on a list of key - messages pairs containing some $(\dot{\mathsf{pk}}, v^*)$, for some tag τ^*, such that the oracle never delivered a signature on the tagged message $(\tau^*|v^*)$. Formally:

```
a-uf
$(\dot{\mathsf{sk}}, \dot{\mathsf{pk}}) \leftarrow \mathsf{Kg}()$
$\mathcal{KG} \leftarrow \mathcal{F}$
$\big(\mathcal{KG}, \tau^*, (\mathsf{pk}_i, v_i)_{i \in [n]}, \Sigma^*\big) \leftarrow \mathcal{F}^{\textsc{Sign}}(\dot{\mathsf{pk}})$
return $\Big(\exists v^* \text{ s.t. } (\dot{\mathsf{pk}}, v^*) \in (\mathsf{pk}_i, v_i)_{i \in [n]} \wedge (\tau^*, v^*) \notin Q_{\mathrm{sig}}$
$\wedge\, (\mathsf{pk}_i)_{i \in [n]} \subset \mathcal{KG} \;\wedge\, \mathsf{Vf}(\mathcal{KG}, \tau^*, (\mathsf{pk}_i, v_i)_{i \in [n]}, \Sigma^*) = 1\Big)$

oracle $\textsc{Sign}(\tau, v)$
if $\dot{\mathsf{pk}} \in \mathcal{KG} \wedge \nexists (\tau, v' \neq v) \in Q_{\mathrm{sig}}$:
    $Q_{\mathrm{sig}} \leftarrow Q_{\mathrm{sig}} \cup \{(\tau, v)\}$
return $\dot{\Sigma} \leftarrow \mathsf{Sign}(\mathcal{KG}, \dot{\mathsf{sk}}, \tau, v)$
```

Figure 3: **Aggregate signatures unforgeability game. Instructions removed in dynamic schemes are shaded-out**

Comments on Definitions and Related Works

Blob of signatures and messages. Virtually all existing fNIM schemes allow the combiner to take as separate inputs a multiset of keys, and a multiset of messages: $\mathsf{Cb}\big(\mathcal{KG}, m, (\mathsf{pk}_i)_{i \in [N]}, (\Sigma_\mathrm{i})_{i \in [N]}\big)$. The reason why our syntax requires to regroup them in pairs: $(\mathsf{pk}_i, \Sigma_\mathrm{i})_{i \in [n]}$ is because it is necessary in the fNIA called $\mathcal{AS}$-4 [16], which we observe is a fortiori a fNIM.

Key-aggregation. In all existing fNIM schemes, the verification can be factored in a first step called key-aggregation, which shrinks the multiset of the keys of the subgroup of signers into a short aggregate key: $\mathsf{KAg} : \big(\mathcal{KG}, (\mathsf{pk}_i)_{i \in [n]}\big) \rightarrow \mathsf{apk}$. Then, verification proceeds using only apk: $\mathsf{Vf\text{-}lazy}(\mathsf{apk}, m, \Sigma) \rightarrow 0/1$. We purposedly did not include this decomposition in the specifications, in line with [56, 51], because we believe it to be bug-prone. Indeed, some existing specifications only specify the last part Vf-lazy, but it would be insecure to run such a Verifier algorithm without any additional safe means of checking, in a way or another, that apk was correctly output by KAg from the keys of the purported signers. A more subtle example is the scheme RSMS-pop in [56, §4.3]: its Verifier algorithm, VerifyMul, does not check validity of the PoPs on the public keys, although it does aggregate them with KAg. The problem is that its KAg does not either check validity of the PoPs. In conclusion, it seems to us that rogue-key attacks [21] invalidate the unforgeability of RSMS-pop, in the sense of [56, def. 3.3]. Of course solving this apparent issue would just require to specify that the RSMS-pop.VerifyMul verifies PoPs, in a way or another.

Interactive setups. The issue of enabling verification of apk is even clearer in the cases, not captured by our definitions, where KAg would be an interactive protocol between the signers. Fortunately the only such fNIM schemes to our knowledge are MOR01 [80] and $\mathcal{ASMP}$ & $\mathcal{ASMP}$*-pop* [22, §4 & §6], and in all of them the protocol is *publicly verifiable* as long as it is executed over a broadcast channel.

Concurrent groups. Although our unforgeability definitions in Figures 2 and 3 follow the ones of [9], in which the adversary must commit on a single group $\mathcal{KG}$ and has then access to a signing oracle w.r.t. this group only, this limitation is relaxed in the specifications of [22, §4]. More precisely, provided interactive group setups, they show that the security of their schemes $\mathcal{ASMP}/\mathcal{ASMP}$-*pop* decreases linearly only in the total number of signers over all groups in which the target key $\dot{\mathsf{pk}}$ is involved. Of course all these issues disappear in dynamic schemes.

Not fully non-interactive schemes. In the scheme $\mathcal{MSP}$ [22], the Sign algorithm takes as input the exact list of the keys which are meant to sign the message: $(\mathsf{pk}_i)_{i\,\in\,[N]}$. Upon receiving N signatures on some message m generated by the keys $(\mathsf{pk}_i)_{i\,\in\,[N]}$, the combination Cb can combine them only if they were all generated with input $(\mathsf{pk}_i)_{i\,\in\,[N]}$. Said in the other way: Cb cannot combine any signature on m generated with input $(\mathsf{pk}_i)_{i\,\in\,[N]}$, unless receiving such signatures from *all* the intended signers $(\mathsf{pk}_i)_{i\,\in\,[N]}$. Said otherwise: aggregation fails as soon as one of the intended signers aborts (this issue is lifted in the blog version $\mathcal{MSP}$-*blog* [21]). This limitation also shows-up in the two-round schemes ([46, 42, 13]): although they require interaction only in a first round which is message-independent, the signatures produced in the second round can be combined only if they are produced by all participants of the first round.

Aggregation robustness. This is a property which we credit to [56, 51]. Prior specifications guaranteed a successful aggregation only over correctly generated signatures and keys.

4 $\mathcal{M}$to$\mathcal{A}$: MULTI TO AGGREGATE COMPILER

We convey all ideas of $\mathcal{M}$to$\mathcal{A}$, further formalism and the (obvious) proof can be found in Appendix A. We consider any fNIM: M, and describe the resulting one-time-tagged fNIA, called A. Each potential signer i generates then outputs a list of $2|v|$ M-public keys: $\{\mathsf{pk}_i^{j,b}, j \in |v|, b \in \{0,1\}\}$. To sign some $(\tau|v_i)$, the signer i: parses $v_i = (v_i^j)_{j\,\in\,[|v|]}$ the bit decomposition of the variable part, then outputs a signature *on* τ under each public key pk_i^{j,v_i^j} for $j \in [|v|]$. That is, the data of the variable part v_i of the message intended to be signed is *not* encoded by the actual content signed, which is just equal to the fixed part τ, but instead by the list of the *keys* which signed τ. The Aggregator: upon receiving signatures on n messages $(v_i)_n$ from n signers, where we recall each signature consists of a $|v|$-sized list of M-individual signatures, applies the M-combination algorithm: M.Cb on all $N = |v|n$ signatures received. Finally, the verifier checks the multisignature Σ received against the multiset of keys $\mathcal{PK}$ which it reads from the messages $(v_i)_{i\,\in\,[n]}$. That is, for each message $v_i = (v_i^j)_{j\,\in\,[|v|]}$ it appends $\mathcal{PK} \leftarrow \mathcal{PK} \cup \{\mathsf{pk}_i^{j,v_i^j}\}_{j\,\in\,[|v|]}$, then outputs $\mathsf{M.Vf}(\mathcal{PK}, m, \Sigma)$.

5 dms

dms is specified over any bilinear group $(\mathbb{G}_1, G_1, \mathbb{G}_2, G_2, \mathbb{G}_T, e)$ and operates with any hash-to-curve random oracle $\mathrm{H} : \{0,1\}^* \rightarrow \mathbb{G}_1$ and random oracle $\mathrm{H}_{\mathrm{pop}} : \{0,1\}^* \rightarrow \mathbb{Z}_p$. We will show that its security tightly reduces to the one of BLS signatures (Figure 1). Themselves tightly reduce to the hardness of the DL, in the AGM, by [53, §6]. The syntax of dms does not contain any group $\mathcal{KG}$, hence, it is a *dynamic* fNIM. Note that in the description below of dms, if one removes the PoP: π from the Kg, and its verification: kVf from both the Cb and Vf algorithms, then we are brought back to the plain BLS multisignature scheme [20] (also recalled in Appendix 4.2).

- **Kg()**: sample $x \xleftarrow{\$} \mathbb{Z}_p$, set $X \leftarrow x.G_2$; sample $r \xleftarrow{\$} \mathbb{Z}_p$, $R \leftarrow r.G_2$, set $c \leftarrow \mathrm{H}_{\mathrm{pop}}(X, X, R)$ and $z := r + c.x$, let $\pi \leftarrow (R, z)$ the proof of possession (PoP), output $\mathsf{sk} := x$ and $\mathsf{pk} := (X, \pi)$.

Since PoPs are to be verified by both the individual verification algorithm (iVf) and the combination algorithm (Cb), we factor-out their verification with the following helper function:

- $\mathsf{kVf}(X)$: parse $(X, \pi) \leftarrow \mathsf{pk}$ and $(R, z) \leftarrow \pi$; $c \leftarrow \mathrm{H}_{\mathrm{pop}}(X, X, R)$, output $(X \in \mathbb{G}_2 \ \wedge \ z.G_2 == R + c.X)$.

Noticeably, and unlike pairing-based PoPs, no subgroup membership in $\mathbb{G}_2$ is to be performed on the PoP π, since the verified relation $R = z.G_2 - c.X$ automatically implies membership of R in $\mathbb{G}_2$.

- $\mathbf{Sign(sk}, \boldsymbol{m})$ $= \mathsf{sk}.\mathrm{H}(m)$;
- $\mathbf{iVf(pk}_{\boldsymbol{i}}, \boldsymbol{m}, \boldsymbol{\Sigma}_{\mathbf{i}})$: parse $\mathsf{pk}_i \leftarrow (X_i, \pi_i)$, return $\big(\Sigma_{\mathrm{i}} \in \mathbb{G}_1 \ \wedge \ \mathsf{kVf}(\mathsf{pk}_i) == 1 \ \wedge \ e(\Sigma_{\mathrm{i}}, G_2) == e(\mathrm{H}(m), X_i)\big)$.
- $\mathbf{Cb}\big(\boldsymbol{m}, (\mathbf{pk}_{\boldsymbol{i}}, \boldsymbol{\Sigma}_{\mathbf{i}})_{\boldsymbol{i}\,\in\,[\boldsymbol{N}]}\big)$: if $\Big\{\mathsf{iVf}(\mathsf{pk}_i, m, \Sigma_{\mathrm{i}}) == 1 \ \forall i \in [N]\Big\}$ then return $\sum_{i\,\in\,[N]} \Sigma_{\mathrm{i}}$.
- $\mathbf{Vf}\big((\mathbf{pk}_{\boldsymbol{i}})_{\boldsymbol{i}\,\in\,[\boldsymbol{N}]}, \boldsymbol{m}, \boldsymbol{\Sigma}\big)$: parse $(X_i, \pi_i) \leftarrow \mathsf{pk}_i \ \forall i \in [N]$; return $\Sigma \in \mathbb{G}_1 \ \wedge \big\{\, \mathsf{kVf}(\mathsf{pk}_i) == 1 \, \forall i \in [N] \,\big\}$ $\wedge \ e(\Sigma, G_2) == e\Big(\mathrm{H}(m), \sum_{i\,\in\,[N]} X_i\Big)$.

Theorem 3. dms *is a fNIM in the AGM. For all three types of bilinear groups, its unforgeability tightly reduces to hardness of the discrete logarithm (DL) problem.*

The proofs of Individual completeness and aggregation-robustness are identical to the ones of standalone BLS multisignatures [20] (plus correctness of Schnorr signatures used as PoPs), so we skip them. In Sec. 5.1 we state the main ingredient of unforgeability of dms. It states that whenever an algebraic adversary $\mathcal{A}$ outputs a public key X^*along with a Schnorr signature on itself: (R^*, z^*), then the discrete logarithm of X^*, i.e., a secret key, can be efficiently computed from the decomposition of X^* given by the adversary. Then in Sec. 5.2 we conclude the proof of unforgeability from this ingredient.

5.1 Extracting Schnorr despite correlations

We formalize the game called **ssc** in Figure 4, which stands for "Schnorr Straight-line extraction in presence of BLS Correlations". The game samples a public key $X = x.G_2$, which we dub the "honest key", generates a Schnorr signature on it: $\pi = (R, z)$, and shows (X, π) to the adversary $\mathcal{A}$. Then the adversary is given access to hash-into-$\mathbb{G}_1$ and hash-into-$\mathbb{Z}_p$ random oracles: $\mathrm{H} : m_i \rightarrow M_i$ and $\mathrm{H}_{\mathrm{pop}}$; and to a BLS signing oracle for the honest secret key $\textsc{Sign} : m_i \rightarrow \Sigma_{\mathrm{i}} := x.\mathrm{H}(m_i)$. All in all, up to delivering to $\mathcal{A}$ both replies from $\textsc{Sign}$ and H for every queried m, we have that $\mathcal{A}$ is delivered a random string with a hidden structure: $(M_i,\ x.M_i)_{i\,\in\,[q_{\mathrm{H}}]}$, in addition to the Schnorr proof of knowledge π for the same exponent: x of X. In the game **ssc** in Figure 4, the challenger tries to extract in straight-line a discrete logarithm x^* upon being submitted some X^* and some Schnorr signature (R^*, z^*) on it, valid for X^*. The goal of the adversary is to defeat this extraction: it wins as soon as one extraction attempt fails over all its submissions. The extractor is defined as follows. Upon submitting some (X^*, R^*, z^*), the adversary gives the decompositions of X^*, R^*in terms of all group elements received so far, of which all hashs M_i and signatures Σ_{i}

returned by the oracles so far:

$$X^* = \alpha.G + \beta.X + \sum_{i \in [q_H]} \gamma_i \Sigma_i + \sum_{i \in [q_H]} \delta_i M_i \tag{5}$$

$$R^* = \alpha'.G + \beta'.X + \sum_{i \in [q_H]} \gamma'_i \Sigma_i + \sum_{i \in [q_H]} \delta'_i M_i \tag{6}$$

Note that, without loss of generality, we assumed that R does not appear in the decompositions, since $R = z.G - c.X$. As will be precised in the proof, these decompositions are to be understood as those which $\mathcal{A}$ gave when outputting $X^*$and $R^*$for the first time. The extractor is then simply defined as the function which returns α. So this extractor outputs a correct discrete logarithm if $X^* == \alpha.G_2$, else, this means that the adversary wins. The following theorem states that escaping this extractor is as hard as solving DL. The proof is done in Sec. 6 and is one of our main technical contributions.

Theorem 4. *From any* SSC*-adversary $\mathcal{A}$ with advantage ϵ and making q_H RO queries, one can build an adversary $\mathcal{E}$ against the discrete logarithm* (**DL**) *in $\mathbb{G}_2$ with advantage $\epsilon' \geqslant (1 - \frac{q_H+1}{p})(\epsilon - q_H/p)$ and with at most twice the running time.*

```
ssc
$x \xleftarrow{\$} \mathbb{Z}_p,\ X = \delta.G_2$
$r \xleftarrow{\$} \mathbb{Z}_p,\ R \leftarrow r\, G_2,\ c \leftarrow \mathrm{H}_{\mathrm{pop}}(X, X, R),\ z \leftarrow r + cx,\ \pi \leftarrow (R, z)$
foreach $(X^*, R^*, z^*) \leftarrow \mathcal{A}^{\mathrm{Sign},\mathrm{H},\mathrm{H}_{\mathrm{pop}}}(X, \pi)$    ▷ up to $q_{\mathrm{H}}$ attempts
    $c^* \leftarrow \mathrm{H}(X^*, X^*, R^*)$
    if $z^*.G_2 = R^* + c^*.X^* \wedge (X^*, R^*, z^*) \neq (X, R, z)$
        receive the decomposition (5): $X^* = \alpha.G_2 + \ldots$ (see above)
        if $X^* \neq \alpha.G_2$ then win $\leftarrow 1$
return win

oracle Sign$(m)$
    return $\Sigma \leftarrow x.\mathrm{H}(m)$
```

Figure 4: **Schnorr Straight-line extraction in presence of Correlations oracles.** H **and** $\mathrm{H}_{\mathrm{pop}}$ **are random oracles: into** $\mathbb{G}_1$ **and into** $\mathbb{Z}_p$**; Sign is a BLS signing oracle.**

5.2 Tight reduction of dms to DL.

The proof follows from the following chain of tight reductions:

$$\text{m-uf(dms)} \xrightarrow{\text{Lem. 5}} \mathbf{ssc} \wedge \mathbf{bls}\text{-uf} \xrightarrow{\text{Thm. 4}} \mathbf{dl} \wedge \mathbf{bls}\text{-uf} \xrightarrow{[53]} \mathbf{dl} \tag{7}$$

We now outline each of the reductions. The game on the left is the unforgeability of dms. The first reduction is stated more precisely as follows, and will be proven below:

Lemma 5. *For any algebraic forger $\mathcal{F}$ in the unforgeability game* m-uf *of* dms*, with advantage ϵ, running time t and making at most q_H RO queries; then there exists a forger $\mathcal{B}$ in the unforgeability game* **BLS**-uf *of* BLS *signatures, which has advantage $\epsilon' \geqslant \epsilon - q_H/p - \mathrm{UB}^{SSC}(t)$ and at most twice the running time t, where $\mathrm{UB}^{SSC}(t)$ denotes the upper-bound on the advantage of a time-t adversary in game* **SSC**.

The third reduction, labelled by Thm. 4 is because Theorem 4 shows a tight reduction from **SSC** to **DL**.

The last reduction, from BLS unforgeability to the hardness of **DL**, is proven in [53, §6]. Note the their proof holds for type I bilinear groups, and could be easily adapted to type II. On the other hand, for their proof to carry over type III groups, the DL problem to be considered is when the adversary is given challenges in both groups: $(\ell.G_1, \ell.G_2)$. This type III DL problem is formalized in [11] as $(1, 1)$-DL. Note that in the plain model (non-AGM), then BLS-uf in type III groups reduces (non-tightly) to an analogous variant of computational Diffie-Hellman, which is called co-cdh∗ in [37], and simply "co-DH" in [22, Def. 2].

Now, all what remains to be proven is Lemma 5.

Proof of Lemma 5. The following proof holds for all three types of bilinear groups. $\mathcal{B}$ receives the challenge honest key X and simulates the required Schnorr PoP π on X using the standard strategy. Namely: it samples $(r, z) \leftarrow \mathbb{Z}_p^2$, programs the random oracle as $c \leftarrow \mathrm{H}_{\mathrm{pop}}(X, X, R)$, up to the q_{H}/p-probability event where $\mathcal{A}$ would already have queried $\mathrm{H}_{\mathrm{pop}}(X, X, R)$, then outputs $\pi \leftarrow (R := r.G_2, z)$. Then it gives $\mathsf{pk} \leftarrow (X, \pi)$ as the challenge honest key to $\mathcal{F}$. The rest of the proof closely follows [20]. The only difference, which is all the point of our work, is that instead of being given for free all valid secret keys from $\mathcal{F}$, $\mathcal{B}$ extracts them (except those identical to X) thanks to an obvious reduction to **SSC**. Whenever $\mathcal{F}$ makes an m-uf-Sign signature query on some message m, $\mathcal{B}$ simply makes the query on the same m to its own **BLS**-uf-Sign oracle, then forwards the result to $\mathcal{F}$. At some point, $\mathcal{F}$ outputs a dms forgery: $\left((\mathsf{pk}_i)_{i \in [N]}, m, \Sigma\right)$, in particular such that m was not queried before to Sign. For simplicity, let us re-index the keys such that all those *different* from X come first: $(\mathsf{pk}_i)_{i \in [N_1]}$, then the $N - N_1$ ones identical to X come last. Note that by definition of a forgery, $N - N_1 \geqslant 1$.

Claim: Parse $(X_i, \pi_i) \leftarrow \mathsf{pk}_i\ \forall i \in [N]$. Then, $\mathcal{F}$ is able to extract the discrete logarithms $(x_i)_{i \in [N_1]}$ of the $(X_i)_{i \in [N_1]}$ from the queries of $\mathcal{F}$ to H and $\mathrm{H}_{\mathrm{pop}}$, except with probability $\mathrm{UB}^{\mathrm{SSC}}(t)$, in at most twice the running time t of $\mathcal{F}$.

proof of the Claim. Consider formally the adversary $\mathcal{F}^{\perp}$, which is equal to $\mathcal{F}$ except that the last two outputs of its forgery are removed: $\left((\mathsf{pk}_i)_{i \in [N]}, \perp, \perp\right)$. Then $\mathcal{F}^{\perp}$ is a game-**SSC** adversary, which concludes the Claim.

End of the proof. Informally, the reduction $\mathcal{B}$ removes, from the forgery Σ, the contributions of the individual signatures from the non-X keys. Then, what remains is valid signature on m for the key $(N - N_1).X$, so it scales it down to a valid signature for X. Formally, $\mathcal{B}$ computes the individual signatures $\Sigma_i \leftarrow x_i.\mathrm{H}(m)$, then outputs the forgery:

$$\Sigma_X \leftarrow \frac{1}{N - N_1}\Big(\Sigma - \sum_{i \in [N_1]} \Sigma_i\Big)\ . \tag{8}$$

Let us formally verify that Σ_X is indeed a valid BLS signature on m for the challenge key X, which will conclude the proof. For readability we multiply everywhere by $(N - N_1)$. By construction:

$$(N - N_1)e(\Sigma_X, G_2) = e\Big(\Sigma - \sum_{i \in [N_1]} x_i.\mathrm{H}(m),\ G_2\Big)\ . \tag{9}$$

On the other hand, Σ being a valid dms multi-signature, we have

$$(10) \qquad e(\Sigma, G_2) = e\Big(\mathrm{H}(m)\,,\ \sum_{i\,\in\,[N_1]} x_i.G_2 + (N - N_1)X.G_2\Big)\ .$$

Developping by linearity the RHS of Eq. (9), then replacing $e(\Sigma, G_2)$ by Eq. (10), we obtain that this RHS is equal to

$$(11) \quad e\Big(\mathrm{H}(m)\,,\ \sum_{i\,\in\,[N_1]} x_i.G_2 + (N - N_1)X.G_2\Big) - e\Big(\sum_{i\,\in\,[N_1]} x_i.\mathrm{H}(m),\ G_2\Big)$$

Finally, cancelling-out all equalities $e(\mathrm{H}(m), x_i.G_2) = e(x_i.\mathrm{H}(m), G_2)$, we obtain:

$$(12) \qquad (N - N_1)e(\Sigma_X, G_2) = e\Big(\mathrm{H}(m), (N - N_1)X\Big)\ ,$$

which, after dividing by $(N - N_1)$, proves the validity of Σ_X.

6 PROOF OF THEOREM 4

As in most works on BLS [53, 9, 7], we describe only the proof in the case of type I bilinear groups, so in what follows we identify $\mathbb{G} := \mathbb{G}_1 = \mathbb{G}_2$ and $G := G_1 = G_2$, excepted in the formal descriptions of Figures 4 and 5. We will explain in the end why the proof is much simpler in the cases of type II and III groups.

Simplifications w.l.o.g. Note that in the definition of the game **ssc**, we could assume without loss of generality that $\mathcal{A}$ submits at most one Schnorr signature to the game **ssc**, instead of q_{H} submissions. Indeed it can make submissions to itself and run the extractor on itself to check if it won or not. For this reason, without loss of generality (w.l.o.g.), we now consider that $\mathcal{A}$ makes at most one submission to the game. Also, w.l.o.g., consider that when the adversary makes a query m to H then it immediately makes the same query m to Sign, and conversely.

The reduction $\mathscr{E}$ against **dl** receives a challenge L and its goal is to output the exponent ℓ, i.e., s.t. $L = \ell.G$. To this end, it runs the ssc adversary $\mathcal{A}$ and simulates **ssc** to it, as follows. We call $\mathscr{E}$ the *master reduction* because it tosses a coin and, depending on its value 1 or 0, behaves towards $\mathcal{A}$ as reduction C or D, both described in Figure 5. We now convey their intuition. As will be clear from their description, both are (almost) perfect simulations of game **ssc**, and furthermore C and D are information-theoretically indistinguishable from each other. The difference between them, hidden to $\mathcal{A}$, is how the challenge L is embedded. Both reductions use procedures, denoted $\widetilde{\mathrm{H}}$, $\widetilde{\textsc{Sign}}$ and $\widetilde{\mathrm{H}}_{\mathrm{pop}}$, to simulate to $\mathcal{A}$ the responses to its queries to oracles H, Sign and $\mathrm{H}_{\mathrm{pop}}$ respectively.

Reduction C embeds the **dl** challenge L as the honest public key of **ssc**, i.e., sets $X := L$. It simulates the required Schnorr PoP π on X following the standard technique, namely: samples $(r, z) \leftarrow \mathbb{Z}_p^2$, programs the random oracle as $c \leftarrow \widetilde{\mathrm{H}}_{\mathrm{pop}}(X, X, R)$, then outputs $\pi \leftarrow (R := r.G_2, z)$. So its simulation of ssc is perfect, up to the q_{H}/p-probability event where $\mathcal{F}$ would have queried $\mathrm{H}_{\mathrm{pop}}(X, X, R)$ before it was programmed on c.

Reduction D honestly generates the public key X as $x \xleftarrow{\$} \mathbb{Z}_p$, $X \leftarrow x.G_2$. It embeds the dl challenge in the simulated hash-to-curve oracle $\widetilde{\mathrm{H}}$, using the following trick of [53, §6]. Upon queried a new message m_i, $\widetilde{\mathrm{H}}$ samples $(\widehat{h_i}, b_i) \xleftarrow{\$} \mathbb{Z}_p^2$ then returns $M_i \leftarrow b_i.L + \widehat{h_i}.G$. In particular, the output M_i varies uniformly in $\mathbb{G}$, so $\widetilde{\mathrm{H}}$ perfectly simulates a random oracle. In conclusion, it can perfectly simulate the signing oracle, as: $m \leftarrow x.\widetilde{\mathrm{H}}(m)$. Note that since D knows the secret key x, it could also honestly generate a Schnorr PoP as $r \xleftarrow{\$} \mathbb{Z}_p$, $R \leftarrow r.G_2$, $c \leftarrow \mathrm{H}_{\mathrm{pop}}(X, X, R)$ then $z \leftarrow r + c.x$. So it would not have to program $\mathrm{H}_{\mathrm{pop}}$, so its simulation of **ssc** would be perfect. But we make the choice to specify that D does instead generate a simulated Schnorr proof and programs $\widetilde{\mathrm{H}}$, like C does. The reason for this choice is that this makes the view of the adversary identically distributed against C and D, so this will simplify the proof.

Strategy of C to win against **dl**. In what follows we consider the event $\mathrm{C}^{\mathcal{A}} = 1$ where $\mathcal{A}$ outputs a winning triple (X^*, R^*, z^*) to C. Namely, it passes verification of Schnorr proofs, i.e., s.t. for $c^* \leftarrow \widetilde{\mathrm{H}}_{\mathrm{pop}}(X^*, X^*, R^*)$ then $z^*.G = R^* + c^*.X^*$. Being algebraic, $\mathcal{A}$ also submits a linear decomposition of X^* and R^* on all group elements which it was delivered so far. We are now more precise, and specify that the decompositions given in equations (5) (6) are those that $\mathcal{A}$ submitted when it output X^* and R^* *for the first time*, i.e., either to oracle $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ or directly to the main C procedure. Since $\mathcal{A}$ won, then α cannot be the only nonzero coefficient in X^*. With these notations, recall that the goal of C is to find the exponent $x := \ell$ of $X = L$, i.e., s.t. $x.G = X$. To explain how C tries to find it efficiently, start from the relation $z^*.G = R^* + c^*X^*$, substitute R^* and X^* by their decompositions in Eq. (5), then we obtain:

$$(13) \quad z^*G = c^*\Big(\alpha.G + \beta.X + \sum_i \gamma_i.\Sigma_{\mathrm{i}} + \sum_i \delta_i.M_i\Big) \\ + \alpha'.G + \beta'.X + \sum_i \gamma_i'.\Sigma_{\mathrm{i}} + \sum_i \delta_i'.M_i \ \ .$$

Replacing the oracle responses by their values: $\Sigma_{\mathrm{i}} = h_i.X$ and $M_i = h_i.G$, and substituting $X = x.G$, we obtain:

$$(14) \quad z^*G = x\Big(c^*\beta + c^*\sum_i \gamma_i h_i + \beta' + \sum_i \gamma_i' h_i\Big) + c^*\alpha + c^*\sum_i \delta_i h_i + \alpha' + \sum_i \delta_i' h_i$$

Thus C can efficiently recover $\ell = x$ by division by the scalar:

$$(15) \qquad f^* := c^*(\beta + \sum_i \gamma_i h_i) + \beta' + \sum_i \gamma_i' h_i$$

... unless this scalar is zero.

To analyze this bad ($f^* = 0$) event, the important observation is that in all games considered (both **ssc** and its reductions C and D), the decompositions (5)(6) of X^* and R^*, were handed-out by the adversary $\mathcal{A}$ *strictly before* c^* was sampled uniformly at random. Let us prove it on the example of C. Either $\mathcal{A}$ queried (X^*, X^*, R^*) to $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ *before* outputting (X^*, R^*, z^*) to the main C procedure, then c^* was sampled by $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ just after. Or, $\mathcal{A}$ gave (X^*, R^*, z^*) to the main C procedure without having queried $\widetilde{\mathrm{H}}_{\mathrm{pop}}(X^*, X^*, R^*)$ before, then C makes the query to its internal procedure $\widetilde{\mathrm{H}}_{\mathrm{pop}}(X^*, X^*, R^*)$ just after, which then samples c^*.

Lemma 6. *Consider, as before, the event (up to probability q_H/p) where no query $\widetilde{\mathrm{H}}_{\mathrm{pop}}(X, X, R)$ is made before $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ is programmed as*
$\widetilde{\mathrm{H}}_{\mathrm{pop}}(X, X, R) \rightarrow c$. Consider any query $\widetilde{\mathrm{H}}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$ made for the first time, the decompositions of $\dot{X}$ and $\dot{R}$ denoted as in (5) (6) *(so we omit adding a dot above the coefficients $\alpha, \beta, \gamma, \dots$), and the response $\dot{c}$. Consider the indices $i = 1, \dots, i_0$ of all queries m_i to $\widetilde{\mathrm{H}}$ and*

$\widetilde{\text{SIGN}}$ (responding M_i and Σ_i, respectively) which were made before the query $\widetilde{\mathrm{H}}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$, i.e., before $\dot{c}$ was sampled. Then:

- *$\dot{c}$ is sampled independently from $\big(\alpha, \beta, (\delta_i, \delta'_i, \gamma_i, \gamma'_i, h_i)_{i \leqslant \mathrm{i}_0}\big)$;*
- *for all $\mathrm{i} > \mathrm{i}_0$, $\delta_i = \delta'_i = \gamma_i = \gamma'_i = 0$.*

The rest of the proof strategy is as follows. Let us consider the event where the adversary wins against the master reduction: $(\mathscr{E}^{\mathcal{A}} = 1) := (\mathrm{C}^{\mathcal{A}} = 1 \wedge \mathscr{E} = \mathrm{C}) \vee (\mathrm{D}^{\mathcal{A}} = 1 \wedge \mathscr{E} = \mathrm{D})$, where $\mathscr{E} = \mathrm{C}$ and $\mathscr{E} = \mathrm{D}$ denote the events where the coin was 1 or 0, i.e., where $\mathscr{E}$ behaves as C or D. We are going to consider the sub-event, denoted $\neg V \subset (\mathscr{E}^{\mathcal{A}} = 1)$, defined as the non-vanishing of at least one of the coefficient in f^* (Eq. (15)), i.e., $\neg V := \big(\beta + \sum_i \gamma_i h_i \neq 0\big) \vee \big(\beta' + \sum_i \gamma'_i h_i \neq 0\big)$. An immediate consequence of Lemma 6 is that, under C, in the sub-event $\neg V \wedge (\mathrm{C}^{\mathcal{A}} = 1)$, then the bad event $(f^* = 0)$ (almost) never happens, and thus C is (almost) always able to find the DL challenge $x = \ell$. The "almost" will be quantified later. So what remains to conclude the proof is to show that, in the complementary event $V \subset (\mathscr{E}^{\mathcal{A}} = 1)$ which we call *"very bad"*, then the reduction D will (almost always) be able to find ℓ efficiently. Our first task is to formalize a predicate equivalent to V and which is well-defined under D. To do so, we multiply the relations defining $\neg V$ by X then take the negation, which yields:

$$V := \Big\{\beta.X + \sum_i \gamma_i M_i = 0 \ \wedge\ \beta'.X + \sum_i \gamma'_i M_i = 0\Big\}\ . \tag{16}$$

This equivalent predicate being purely in terms of the view of the adversary, it is also meaningful under D.

In order to conclude, we recall the fact that C and D are perfectly indistinguishable from the adversary $\mathcal{A}$. A consequence is that the coin tossed by the master-reduction $\mathscr{E}$, i.e., its choice of behavior C or D, is independent of which event V or $\neg V$ happens (otherwise the adversary could distinguish between C and D). In conclusion, each time the adversary wins, i.e., $(\mathscr{E}^{\mathcal{A}} = 1)$, whatever V or $\neg V$ is the most likely to happen, the master-reduction $\mathscr{E}$ will have almost probability 1/2 to extract the DL challenge ℓ. We now formalize the above claims as Lemma 7, then formalize the above conclusion of Theorem 4 from it, then prove Lemma 7.

LEMMA 7. *There exists reductions* C *and* D *from* **SSC** *to the* **DL** *game, such that for any* SSC*-adversary $\mathcal{A}$:*

- *both the views of $\mathcal{A}$ against* C *and* D*, are identically distributed as in* **SSC***, except with q_H/p probability;*
- *the views of $\mathcal{A}$ against* C *and* D *are identically distributed;*
- C *and* D *enjoy the following probabilities of success, i.e., of* $(\mathrm{C}^{\mathcal{A}} = 1)$ *and* $(\mathrm{D}^{\mathcal{A}} = 1)$*:*

$$\mathbb{P}\big(\mathbf{DL}^{\mathrm{C}} = 1\big) = (1 - \frac{q_H}{p}).\mathbb{P}\big(\mathrm{C}^{\mathcal{A}} = 1 \wedge \neg V\big) \tag{17}$$

$$\mathbb{P}\big(\mathbf{DL}^{\mathrm{D}} = 1\big) = (1 - \frac{q_H + 1}{p}).\mathbb{P}\big(\mathrm{D}^{\mathcal{A}} = 1 \wedge V\big) \tag{18}$$

Assuming Lemma 6, let us conclude Theorem 4. Since the master reduction $\mathscr{E}$ behaves as C or D with probability 1/2 each, we have $\mathbb{P}\big(\mathbf{DL}^{\mathscr{E}} = 1\big) = 1/2\,\mathbb{P}\big(\mathbf{DL}^{\mathrm{C}} = 1\big) + 1/2\,\mathbb{P}\big(\mathbf{DL}^{\mathrm{D}} = 1\big)$. By Equations (17) and (18), it is in turn $\geqslant \big(1 - \frac{q_\mathrm{H}+1}{p}\big)\Big[\mathbb{P}\big(\mathrm{C}^{\mathcal{A}} = 1 \wedge \neg V\big) + \mathbb{P}\big(\mathrm{D}^{\mathcal{A}} = 1 \wedge V\big)\Big]$. Now, note that for any fixed $\mathcal{A}$, we have $\mathbb{P}(\mathrm{C}^{\mathcal{A}} = 1 \wedge \neg V) = \mathbb{P}(\mathrm{D}^{\mathcal{A}} = 1 \wedge \neg V)$. Indeed if not, then an unlimited adversary $\mathcal{A}$ could distinguish between C and D, a contraction. Substituting, we obtain: $\mathbb{P}\big((\mathbf{DL})^{\mathscr{E}} = 1\big) \geqslant 1/2\big(1 - \frac{q_\mathrm{H}+1}{p}\big)[\mathbb{P}\big(\mathrm{C}^{\mathcal{A}} = 1\big)]$. By the first claim of Lemma 7, since the view of $\mathcal{A}$ against C is q_H/p-close to its view in **SSC**, we have that $\mathbb{P}\big(\mathrm{C}^{\mathcal{A}} = 1\big) \geqslant \epsilon - q_\mathrm{H}/p$. Replacing in the previous formula of $\mathbb{P}\big((\mathbf{DL})^{\mathscr{E}} = 1\big)$ yields the theorem.

Proof of Lemma 7. The first claim wsa already argued along with the definitions of C and D, namely, the only difference between the view against **SSC** is in the event where the adversary had already queried $\widetilde{\mathrm{H}}_{\mathrm{pop}}(X, X, R)$ before it was programmed.

Proof of Eq. (17). Since the bound is to be proven under reduction C only, we consider only the reduction C, i.e., we condition on the event $\mathscr{E} = \mathrm{C}$. By Lemma 6, for each query $\widetilde{\mathrm{H}}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$ in the execution, and denoting accordingly $\dot{f}$ following formula (15), we have $\mathbb{P}(\dot{f} = 0 | \neg V) = 1/p$ where the probability is taken over the sampling of the answer c^*. Taking the union bound over all q_H queries in the execution, we thus have probability q_H/p that none of their $\dot{f}$ is equal to 0. In particular, for the specific f^* of the winning triple, we thus have

$$\mathbb{P}(f^* = 0 | \neg V) \leqslant q_\mathrm{H}/p\ . \tag{19}$$

Since C is able to extract $x = \ell$ when $f^* \neq 0$, this concludes the proof.

Proof of Eq. (18) Since the bound is to be proven under reduction D only, we consider only the reduction D, i.e., we condition on the event $\mathscr{E} = \mathrm{D}$. Let us start from Eq. (13) and, since we assumed V, simplify by Eq. (16). Replacing $M_i = b_i.L + \widehat{h_i}.G$ and $L = \ell.G$ we obtain

$$z^*G = c^*\Big(\alpha.G + \sum_i \delta_i(b_i\,\ell.G + \widehat{h_i}.G)\Big) + \alpha'.G + \sum_i \delta'_i(b_i\,\ell.G + \widehat{h_i}.G) \tag{20}$$

Thus D can efficiently recover ℓ by division by the scalar:

$$\lambda := c^* \sum_i \delta_i b_i + \sum_i \delta'_i b_i \tag{21}$$

... unless this scalar λ is zero.

Let us assume that it is the case, then we cannot be in the event $W := \Big\{(\delta_i = 0\ \wedge\ \delta'_i = 0)\,\forall i \wedge V\Big\}$. Indeed, substituting in Eq. (5) those vanishings and those of Eq. (16), would yield $X^* = \alpha.G$, contradicting that $\mathcal{A}$ wins. Hence, we must be in the event $\neg W \subset V$ where at least one of the coefficients δ_i or δ'_i is nonzero. To conclude, we apply the same kind of reasoning as in the proof of Eq. (17). Let us consider one query $\widetilde{\mathrm{H}}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$ in the execution. By Lemma 6, $\mathbb{P}(\dot{c} = 0 | \neg W) = 1/p$ where the probability is taken over the sampling of c^* and on the coins of the adversary. Taking the union bound over all q_H queries in the execution, we thus have probability q_H/p that none of their $\dot{c}$ is equal to 0. In particular, for the specific c^* of the winning triple, we thus have

$$\mathbb{P}(c^* = 0 | \neg W) \leqslant q_\mathrm{H}/p\ . \tag{22}$$

In the event $c^* \neq 0$, we Claim that $\mathbb{P}(\lambda = 0 | c^* \neq 0 \wedge \neg W) \leqslant 1/p$, which concludes the proof. The Claim follows from the fact that, by construction, all $(b_i)_i$ are information-theoretically hidden from the adversary, hence they are independent of $\big((\delta_i, \delta'_i)_i$, of which at least one is nonzero by definition of $\neg W$.

```
reductions C $\boxed{\text{and D}}$ against DL from SSC adversary $\mathcal{A}$
receive the DL challenge $L = \ell.G_2 \in \mathbb{G}_2$ with unknown exponent $\ell$
$X \leftarrow L$ (and thus $X = x.G_2$ s.t. $x = \ell$)
$(c, z) \xleftarrow{\$} \mathbb{Z}_p^2$; $R \leftarrow z\,G_2 - c.X$; $\pi \leftarrow (R, z)$
Program $\widetilde{\mathrm{H}}_{\mathrm{pop}}(X, X, R) \rightarrow c$
$(X^*, R^*, z^*) \leftarrow \mathcal{A}(X, \pi)$
$c^* \leftarrow \widetilde{\mathrm{H}}(X^*, X^*, R^*)$
if $z^*.G_2 = R^* + c^*.X^* \wedge (X^*, R^*, z^*) \neq (X, R, z)$
  receive the decomposition of $X^*$(eq. (5))
  if $\alpha.G_2 \neq X^*$   ▷ i.e., other coefficients than $\alpha$ are nonzero in eq. (5)
    set flag win ← 1
    C: if $\beta.X + \sum_i \gamma_i \Sigma_i \neq 0$ $\boxed{\text{D: if} = 0}$   ▷ see eq. (16)
      compute then return $\ell$ (see above)   ▷ up to $1/p$ failure proba
```

```
simulated $\widetilde{\mathrm{H}}(m_i)$
if $H[m_i] = \perp$
  C: $h_i \xleftarrow{\$} \mathbb{Z}_p$; $M_i \leftarrow h_i.G_1$;
  $\boxed{\text{D: } (\widehat{h}_i, b_i) \xleftarrow{\$} \mathbb{Z}_p^2;\ M_i \leftarrow b_i.L + \widehat{h}_i.G_1}$
  $H[m_i] \leftarrow M_i$
return $H[m_i]$
```

```
simulated $\widetilde{\mathrm{SIGN}}(m_i)$
C: return $\Sigma_i \leftarrow h_i.X$
$\boxed{\text{D: return } \Sigma_i \leftarrow x.\widetilde{\mathrm{H}}(m_i)}$
```

```
simulated $\widetilde{\mathrm{H}}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$
if $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ was programmed on $(X, X, R)$ and $(\dot{X}, \dot{X}, \dot{R}) == (X, X, R)$
  return $c$
else return $\mathrm{H}_{\mathrm{pop}}(\dot{X}, \dot{X}, \dot{R})$
```

Figure 5

Comments on **ssc** *and on the proof.* Compared to [54], which consider a forger against Schnorr signatures, the goal of our adversary in game **ssc** is easier, and thus our proof apparently harder. Indeed, their forger has to forge a Schnorr signature for a *given target key*. Whereas, our forger succeeds as long as it outputs *any* Schnorr proof, such that the discrete logarithm cannot be extracted.

We credit to [54] the crucial observation that c^* is sampled after the adversary first returns the decomposition of the Schnorr proof that it submits. Notice that it would be fallacious to conclude that, for a given winning triple output by the adversary: (X^*, R^*, z^*), then the $c^* \leftarrow \widetilde{\mathrm{H}}(X^*, X^*, R^*)$ would be independent from the decompositions of X^* and R^*. Indeed, the adversary could well make a unique output to the game: (X^*, R^*, z^*), chosen among possibly many winning triples, as the one which maximizes the number of digits of c^* in common with, e.g., the coefficient δ_1 in the decomposition of X^*. Such correlations are captured by the overhead q_{H} in the probabilities of our bad events, i.e., they have probability q_{H}/p instead of just $1/p$. This overhead captures all possible queries (X^*, X^*, R^*) that could have been made to $\widetilde{\mathrm{H}}_{\mathrm{pop}}$ in order to find winning triples.

Although the proof immediately reduces to the case where $\mathcal{A}$ submits at most one Schnorr signature to the game **ssc**, we defined **ssc** with multi-submissions to make it easier to use in the analysis of dms.

The proof can be much simplified in the case of Type II or III bilinear groups. The DL challenge L is in $\mathbb{G}_2$, but in type II and III groups the algebraic adversary is further restricted to decompose $\mathbb{G}_2$ elements in $\mathbb{G}_2$ only. So all the complicated terms in $\mathbb{G}_1$ in the decompositions Equations (5) and (6) disappear (so the reduction needs not anymore program the hash-to-curve).

7 EVALUATION AND COMPARISON

All our implementations were run on a laptop with Core i5-8265U (8 cores at 1.6GHz), 16GB of RAM, with the library gnark-crypto on Go [28]. The curve used was BLS12-377, offering a pairing of type III, for which the uncompressed size of a point in $\mathbb{G}_1$ is 768 bits and of a point in $\mathbb{G}_2$ is 1536 bits. Compressed points, i.e., their x-coordinate plus one bit, are twice smaller. Each number is the mean over 10 executions. The code is available at [1].

7.1 Comparing processing of the group setup runtimes. We dub “Verifier” the verification function $\mathsf{Vf}\left(\mathcal{KG}, (\mathrm{pk}_i)_{i\in[N]}, m, \Sigma\right) \rightarrow 0/1$ of a fNIM. We call processing of the group setup the tasks of the Verifier which can be done straight upon learning the group of potential signers: $\mathcal{KG}$; and online verification the remaining tasks performed upon learning the actual subset of signers $(\mathrm{pk}_i)_{i\in[N]} \subset \mathcal{KG}$ and the signed message (m, Σ). Of course, in dynamic fNIMs such as $\mathcal{MSP}$*-pop* [86][22, §6][24] and dms, the Verifier does not take any group of public keys $\mathcal{KG}$ as input. What we call processing of the group setup in dynamic fNIMs is the task of verifying the proofs of possession (PoP) of the published keys $\mathcal{KG}$. Recall that in dms we formalized this task as the *key verification* function $\mathsf{kVf}(\mathrm{pk}_i), \forall i \in [N]$. In Table 6 we consider the three fNIMs which have the fastest online verification: SMSKR [9], $\mathcal{MSP}$*-pop* and dms. The online verification is identical in all of them, i.e., returns $\Sigma \in \mathbb{G}_1 \wedge e(\Sigma, G_2) == e\left(\mathrm{H}(m), \sum_{i\in[N]} X_i\right)$. On the other hand, most of the runtime of the Verifiers in both $\mathcal{MSP}$*-pop* and SMSKR (recalled in Appendices 4.3 and 4.4) is explained by their processing of the group setups. As evidenced in Table 6, dms removes this bottleneck. We now detail the figures.

On the first line we measure the time of the processing of the group setup of a group of $|\mathcal{KG}| = 2702$ keys all-at-once. In what follows we denote $N = |\mathcal{KG}| = 2702$ for simplicity. This number was chosen as $2702 = 2|v|.193$, for $|v| = 7$. These numbers illustrate the use-case of the compiler $\mathcal{M}$to$\mathcal{A}$ applied to a group of $n_{\mathrm{C}} = 193$ potential signers and to messages of $|v| = 7$-bits-long variable parts. We batched the verifications of the N pairing-based PoPs as follows. First, we used the trick of [35] for reducing batch verification of N BLS signatures into a product of N pairings (recalled in Appendix 4.4). Second, we computed this product using the optimized algorithm of gnark-crypto, inherited from [65]. Analogously, we batched the verification of the N PoPs in dms, by using the method of [14] for batch verification of Schnorr signatures (recalled in Appendix 4.5).

[1] https://anonymous.4open.science/r/MtoA-830A/

On the second line we consider the incremental processing of the group setup in the scenario where: there is a group $\mathcal{KG}$ of $N = 2702$ keys, over which processing of the group setup was already performed, and then there are 14 new keys $(\mathsf{pk}'_i)_{i \in [14]}$ which join the group, in place of the old keys $(\mathsf{pk}_i)_{i \in [14]}$. Note that this corresponds to the same use-case of $\mathcal{M}$to$\mathcal{A}$, for messages of variable parts $|v| = 7$ bits, when one of the $n = 193$ real group members leaves and is replaced by a new member. This results in the new group $\mathcal{KG}' = \left(\mathcal{KG} \backslash (\mathsf{pk}_i)_{i \in [14]}\right) \cup (\mathsf{pk}'_i)_{i \in [14]}$. Since the resulting group $\mathcal{KG}'$ is different from $\mathcal{KG}$, in SMSKR the Verifier needs to compute again all the 2702 rerandomized keys relatively to the new group $\mathcal{KG}'$, in addition to checking $\mathbb{G}_1$ membership of the 14 new keys. Hence, we see that the incremental processing of the group setup of SMSKR is nearly as costly as the processing of the group setup of a whole new group of keys, as evidenced by the first column of Table 6. Whereas in both $\mathcal{MSP}$*-pop* and dms, the incremental processing of the group setup only consists in verifying the PoPs of the 14 new keys (and the $\mathbb{G}_1/\mathbb{G}_2$-memberships).

in ms	SMSKR [9]	$\mathcal{MSP}$*-pop* [86][22, §6][24]	**dms**
Batch $\lvert\mathcal{KG}\rvert$ keys	1134.9	1947.4	366.6
14 new keys	828.7	12.5	3.3

Table 6: **processing of the group setup runtimes (in ms) for each new group of $|\mathcal{KG}| = 2702$ public keys, in three fNIMs. First line: for a group $\mathcal{KG}$ of completely new keys. Second line: incremental processing of the group setup when 14 new keys join the group $\mathcal{KG}$.**

7.2 Comparing verification times of $\mathcal{M}$to$\mathcal{A}$ + multi-BLS; vs BGLS. In the first three lines of Table 7 we compare the online verification times of three aggregate signature schemes, for an aggregate signature over n messages. The messages are of the form $m_i = (\tau, v_i)$ with have any arbitrary common prefix τ and variable suffixes v_i, all of size $|v| = 7$bits. On the last column with display the size of signatures, including the data of the public keys of the signers. Given a known group of public keys $\mathcal{KG}$, the public keys of the subgroup of n signers can be encoded as a $|\mathcal{KG}|$-sized array of bits. We approximated $|\mathcal{KG}| \cong n$, since $|\mathcal{KG}| = n_C = (3/2)n$ in Diem21-like consensus algorithms. On the last two lines, in grey: we display the times for BLS multisignatures and threshold signatures, which is of course not an apples-to-apples comparison. *In more detail:*

First line: BGLS [23, 15, 37, 74, 22], of which the verification takes $n + 1$ pairings (recalled in Appendix 4.1). More precisely, we evaluated verification of the fastest variant of BGLS: $e(\Sigma, \mathbb{G}_2) == \sum_{i \in [n]} e(\mathsf{H}(m_i), X_i)$. This variant, dubbed $\mathcal{AS}$-1 in [15], is restricted to pairwise different messages m_i. Hence, for a fair comparison with $\mathcal{M}$to$\mathcal{A}$ + dms, which is unrestricted and has tight security, we should have instead evaluated the costlier variant of BGLS called $\mathcal{AS}$-4 in [15]. The verification would then have taken even more time. The signature is a $\mathbb{G}_1$ element, which has uncompressed size equal to 92bytes = 768bits.

Second line: $\mathcal{M}$to$\mathcal{A}$ instantiated with any BLS-based fNIM with optimal online verification, i.e., either $\mathcal{MSP}$*-pop*, SMSKR or our dms. Namely, a $\mathcal{M}$to$\mathcal{A}$ signature comes as a multisignature over $N = |v|n = 7n$ keys, and its verification is as in dms without verification of PoPs, i.e., as in [20], recalled in Appendix 4.2. As in consensus protocols, we considered that the Verifier knows the tag τ in advance, and thus could pre-compute $\mathsf{H}(\tau)$.

Third line: naive concatenation of n Schnorr signatures. Since no pairing is necessary, we used the faster curve secp256k1. Each signature is of the form (R, z), where the $\mathbb{G}_1$ element R is now only of size 64bytes, and the $\mathbb{Z}_p$-element z is of size 32bytes.

Fourth line: multisignature over n signers with any BLS-based fNIM with optimal online verification, i.e., either $\mathcal{MSP}$*-pop*, SMSKR or our dms.

Fifth line: BLS threshold signature [20, 12]. The signature output is a (standard, single-key) BLS signature.

times in ms	$n = 129$	$n = 3073$	size (bits)
BLS aggregate sig.	116.6	2661.6	$768 + n$
$\mathcal{M}$to$\mathcal{A}$ with BLS multisig.	**3.4**	**35.9**	**$768 + n$**
Batch Schnorr	1.9	22.4	$768n + n$
BLS multisig.	2.4	7.1	$768 + n$
BLS threshold sig.	1.9	1.9	768

Table 7: **Online verification times over n messages (see above)**

7.3 Comparison with previous provable securities. In Table 8 we state proven upper-bounds: $\mathsf{UB}^{\text{m-uf}}$ on the probability to forge a multisignature, i.e., the advantage in the m-uf game.

- *The first column states the formulas of* $\mathsf{UB}^{\text{m-uf}}$, not directly in terms of the running time t of the adversary, but instead in terms of: q_{H} the number of its random oracle requests (hash-to-curve); q_{s} the number of its signing requests; and of the upper-bounds: UB^{DL}, $\mathsf{UB}^{\text{co-cdh}}$, $\mathsf{UB}^{\text{co-bdh}}$ and $\mathsf{UB}^{\text{BLS-uf}}$ on the advantage in the games of DL, co-cdh, co-bdh ([27]) and of forgery against standalone BLS signatures. The upper-bounds are for any adversary with roughly the same running time t as the forger, neglecting additive time overheads. In the proofs, such additive overheads typically amount to roughly $+q_{\mathsf{H}}.\tau_{\exp}$, where $\tau_{\exp}$ is a scalar multiplication, also known as exponentiation. In line with [22, Thm 5], we corrected the bound for $\mathcal{MSP}$*-pop* displayed in [86, Thm 4.1], in which q_{H} was replaced by the much smaller number q_{s} of signature queries. We also corrected a bug in the bounds of [9, Thms 1–4] (as confirmed by the authors on 26/1/2024).

- *The second column* states the models in which the formulas are provable: RO stands for random oracle, and "RMSS" is, for simplicity, the assumption that the "random modular subset sum" [9, Def. 3] is at least as hard as DL ([9, §C.3]).

- *The third column* are numerical applications when assuming furthermore the AGM. Concretely, under the AGM then $\mathsf{UB}^{\text{co-bdh}} = \mathsf{UB}^{\text{co-cdh}} = \mathsf{UB}^{\text{dl}}$ [11]. We took $q_{\mathsf{H}} = 2^{80}$; the number of clock cycles $t = 2^{80}\tau_{\exp}$; $q_{\mathsf{s}} = 2^{30}$ (in line with [13]); groups of size $p \sim 2^{253}$, and the hardness of DL estimated as $\mathsf{UB}^{\text{dl}} \leqslant t^2/p$. The latter formula is shown for generic groups in [87] with t the number of group operations. Whereas, we apply it more conservatively to t the number of clock cycles. This estimate seems recently validated [71, 6]

	Proven security: $\mathrm{UB}^{\text{m-uf}}$			Verifier's efficiency		Dynamic	PoP
	theoretical	Model	$\log_2(p) \approx 253$	Group setup	Online		needed
[20, 78] if kosk model	$\mathrm{UB}^{\text{BLS-uf}}$	RO	2^{-93} (if AGM)	●	●	●	
$\mathcal{AS}$-4 [73, 16]	$\mathrm{UB}^{\text{co-cdh}}$	RO	2^{-93} (if AGM)	●	○	●	
$\mathcal{MSP}$-*pop* [86, Th 4.1][22, §6][24] & $\mathcal{ASMP}$-*pop* [22, §6] & Pixel [47]	$q_{\mathrm{H}}.\mathrm{UB}^{\text{co-cdh}}$	RO	2^{-13} (if AGM)	◒	○	●	yes
$\mathcal{MSP}$-*blog* [21, 56]	${q_{\mathrm{H}}}^{3/2}.\sqrt{\mathrm{UB}^{\text{co-cdh}}}$	RO	1	●	◒	●	
$\mathcal{ASM}$[22, §4.2]	${q_{\mathrm{H}}}^{3/2}.\sqrt{\mathrm{UB}^{\text{co-cdh}}}$	RO	1	○	●	○	
SMSKR [9]	${q_{\mathrm{H}}}^{3/2}.\sqrt{\mathrm{UB}^{\text{dl}}}$	RO+AGM+RMSS	1	○	●	○	
SIG$_1$ [27] (Appendix 4.6)	$q_{\mathrm{s}}.\mathrm{UB}^{\text{co-bdh}}$	RO	2^{-63} (if AGM)	○	◒	○	
Squirrel & Chipmunk [52, 51]	(Lattice-based)					●	
dms (this work)	UB^{DL}	RO+AGM	2^{-93}	●	●	●	yes

Table 8: **fNIMs**

for both the popular curves BLS12-377 ($p \sim 2^{253}$), used in Zexe [31], and BLS12-381 ($p \sim 2^{255}$), used in Ethereum, since they are both estimated to have close to 126 bits of security. It is however estimated by Duquesne-Barbulescu and NCC group [10, 66] that both those BLS12 curves, in order to match this security, should be instantiated with a base prime q of size at least 460bits.

8 ACKNOWLEDGEMENTS

We thank Zhuolun Xiang for informing of the use of aggregate signatures in the production implementation of Diem21 by Aptos [5].

REFERENCES

[1] M. A. Aardal, D. F. Aranha, K. Boudgoust, S. Kolby, and A. Takahashi. *Aggregating Falcon Signatures with LaBRADOR.* ePrint 2024/311. 2024.

[2] M. Abdalla, F. Benhamouda, and P. MacKenzie. "Security of the J-PAKE Password-Authenticated Key Exchange Protocol". In: *IEEE SP.* 2015.

[3] S. Agrawal, J. Neu, E. N. Tas, and D. Zindros. "Proofs of Proof-Of-Stake with Sublinear Complexity". In: *AFT.* 2023.

[4] J. H. Ahn, M. Green, and S. Hohenberger. "Synchronized aggregate signatures: new definitions, constructions and applications". In: *CCS.* 2010.

[5] Aptos. *Implementation of Aptos consensus, following Diem.* https://github.com/aptos-labs/aptos-core/blob/main/consensus/consensus-types/src/timeout_2chain.rs Retrieved on June 23, 2024. 2024.

[6] D. F. Aranha, Y. E. Housni, and A. Guillevic. *A survey of elliptic curves for proof systems.* Des. Codes, Cryptogr. 2022.

[7] R. Bacho and J. Loss. "On the Adaptive Security of the Threshold BLS Signature Scheme". In: *CCS.* 2022.

[8] A. Bagherzandi, J.-H. Cheon, and S. Jarecki. "Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma". In: *CCS.* 2008.

[9] F. Baldimtsi, K. K. Chalkias, F. Garillot, J. Lindstrom, B. Riva, A. Roy, A. Sonnino, P. Waiwitlikhit, and J. Wang. "Subset-optimized BLS Multi-signature with Key Aggregation". In: *FC.* 2024.

[10] R. Barbulescu and S. Duquesne. *Updating key size estimations for pairings.* JOC. 2018.

[11] B. Bauer, G. Fuchsbauer, and J. Loss. "A Classification of Computational Assumptions in the Algebraic Group Model". In: *CRYPTO.* 2020.

[12] M. Bellare, E. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. "Better than Advertised Security for Non-interactive Threshold Signatures". In: *CRYPTO.* 2022.

[13] M. Bellare and W. Dai. "Chain Reductions for Multi-Signatures and the HBMS Scheme". In: *ASIACRYPT.* 2021.

[14] M. Bellare, J. A. Garay, and T. Rabin. "Fast batch verification for modular exponentiation and digital signatures". In: *EUROCRYPT.* 1998.

[15] M. Bellare, C. Namprempre, and G. Neven. "Unrestricted Aggregate Signatures". In: *ICALP.* long version. 2007.

[16] M. Bellare and G. Neven. "Multi-signatures in the plain public-Key model and a general forking lemma". In: *CCS.* 2006.

[17] M. Bellare and P. Rogaway. "Code-Based Game-Playing Proofs and the Security of Triple Encryption". In: *EUROCRYPT.* 2006.

[18] D. Bernhard, M. Fischlin, and B. Warinschi. "On the Hardness of Proving CCA-security of Signed ElGamal". In: *PKC* (2016).

[19] W. Beullens and G. Seiler. "LaBRADOR: Compact Proofs for R1CS from Module-SIS". In: *CRYPTO.* 2023.

[20] A. Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC.* Latest long version at https://faculty.cc.gatech.edu/~aboldyre/papers/b.pdf. 2003.

[21] D. Boneh, M. Drijvers, and G. Neven. *Bls multi-signatures with public-key aggregation.* https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html. 2018.

[22] D. Boneh, M. Drijvers, and G. Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT.* 2018.

[23] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps". In: *EUROCRYPT.* 2003.

[24] D. Boneh, S. Gorbunov, R. S. Wahby, H. Wee, C. A. Wood, and Z. Zhang. *BLS Signatures draft-irtf-cfrg-bls-signature-05*. https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature/. 2022.

[25] D. Boneh and S. Kim. *One-Time and Interactive Aggregate Signatures from Lattices*. https://crypto.stanford.edu/~skim13/agg_ots.pdf. 2020.

[26] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptol.* (2004).

[27] D. Boneh, A. Partap, and B. Waters. *Accountable Multi-Signatures with Constant Size Public Keys*. ePrint 2023/1793. 2023.

[28] G. Botrel, T. Piellard, Y. E. Housni, A. Tabaie, and I. Kubjas. *ConsenSys/gnark-crypto: v0.6.1*. 2022.

[29] K. Boudgoust, E. Gachon, and A. Pellet-Mary. "Some Easy Instances of Ideal-SVP and Implications on the Partial Vandermonde Knapsack Problem". In: *CRYPTO*. 2022.

[30] K. Boudgoust and A. Takahashi. *Sequential Half-Aggregation of Lattice-Based Signatures*. ePrint 2023/159. 2023.

[31] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. "ZEXE: Enabling Decentralized Private Computation". In: *IEEE SP*. 2020.

[32] M. Bravo, G. V. Chockler, and A. Gotsman. "Liveness and Latency of Byzantine State-Machine Replication". In: *DISC*. 2022.

[33] M. Bravo, G. V. Chockler, and A. Gotsman. "Making Byzantine consensus live". In: *Distributed Comput.* (2022).

[34] J. Burdges, O. Ciobotaru, S. Lavasani, and A. Stewart. *Efficient Aggregatable BLS Signatures with Chaum-Pedersen Proofs*. ePrint 2022/1611. 2022.

[35] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen. "Batch Verification of Short Signatures". In: *EUROCRYPT*. 2007.

[36] B. Y. Chan and R. Pass. "Simplex Consensus: A Simple and Fast Consensus Protocol". In: *TCC*. 2023.

[37] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. "Comparing Two Pairing-Based Aggregate Signature Schemes". In: *Des. Codes, Cryptogr.* (2010).

[38] H. Cheng, Y. Lu, Z. Lu, Q. Tang, Y. Zhang, and Z. Zhang. *JUMBO: Fully Asynchronous BFT Consensus Made Truly Scalable*. 2024.

[39] G. Chockler. *Modular Construction of Live Byzantine Consensus Protocols*. an abstract appearing on https://lp.jetbrains.com/sptdc-2023/. 2023.

[40] S. Cohen, R. Gelashvili, E. Kokoris-Kogias, Z. Li, D. Malkhi, A. Sonnino, and A. Spiegelman. "Be Aware of Your Leaders". In: *FC*. 2022.

[41] E. Crites, C. Komlo, and M. Maller. "Fully Adaptive Schnorr Threshold Signatures". In: *CRYPTO*. 2023.

[42] E. Crites, C. Komlo, and M. Maller. *How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures*. ePrint 2021/1375. Merged into Crypto'2022 "Better than advertised". 2021.

[43] S. Das, P. Camacho, Z. Xiang, J. Nieto, B. Bunz, and L. Ren. "Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold". In: *CCS*. 2023.

[44] Diem. *DiemBFT v4: State Machine Replication in the Diem Blockchain*. https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf. 2021.

[45] Y. Doröz, J. Hoffstein, J. H. Silverman, and B. Sunar. *MMSAT: A Scheme for Multimessage Multiuser Signature Aggregation*. ePrint 2020/520. 2020.

[46] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. "On the Security of Two-Round Multi-Signatures". In: *IEEE Security and Privacy*. 2019.

[47] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee. "Pixel: Multi-signatures for Consensus". In: *USENIX security*. 2020.

[48] B. Edgington. *Upgrading Ethereum*. https://eth2book.info/latest/book.pdf. 2023.

[49] M. F. Esgin, O. Ersoy, V. Kuchta, J. Loss, A. Sakzad, R. Steinfeld, X. Yang, and R. K. Zhao. "A New Look at Blockchain Leader Election: Simple, Efficient, Sustainable and Post-Quantum". In: *AsiaCCS*. 2023.

[50] Ethereum. *Ethereum Altair upgrade*. https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/bls.md. 2023.

[51] N. Fleischhacker, G. Herold, M. Simkin, and Z. Zhang. "Chipmunk: Better Synchronized Multi-Signatures from Lattices". In: *CCS*. 2023.

[52] N. Fleischhacker, M. Simkin, and Z. Zhang. "Squirrel: Efficient Synchronized Multi-Signatures from Lattices". In: *CCS*. 2022.

[53] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: *CRYPTO*. 2018.

[54] G. Fuchsbauer, A. Plouviez, and Y. Seurin. "Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model". In: *EUROCRYPT*. 2020.

[55] S. D. Galbraith, K. G. Paterson, and N. P. Smart. "Pairings for cryptographers". In: *Discret. Appl. Math.* (2008).

[56] D. Galindo and J. Liu. "Robust Subgroup Multi-signatures for Consensus". In: *CT-RSA*. 2022.

[57] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. "hinTS: Threshold Signatures with Silent Setup". In: *IEEE SP*. 2024.

[58] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang. "Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback". In: *FC*. we refer to the 18 June 2021 version on arxiv. 2022.

[59] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang. "Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback". In: version 2024-04-30. 2024.

[60] C. Gentry, A. O'Neill, and L. Reyzin. *A Unified Framework for Trapdoor-Permutation-Based Sequential Aggregate Signatures*. ePrint 2018/070. 2018.

[61] N. Giridharan, H. Howard, I. Abraham, N. Crooks, and A. Tomescu. *No-commit proofs: Defeating livelock in bft*. eprint 2021/1308. 2021.

[62] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. "SBFT: A Scalable and Decentralized Trust Infrastructure". In: *DSN*. 2019.

[63] S. Goldwasser, S. Micali, and R. L. Rivest. “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks”. In: *SIAM J. Comput.* (1988).

[64] B. Goodell and A. Feickert. *Fusion One-Time Non-Interactively-Aggregatable Digital Signatures From Lattices.* ePrint 2023/303. 2023.

[65] R. Granger and N. P. Smart. “On Computing Products of Pairings”. In: *eprint 2006/172* (2006).

[66] N. Group. *Zcash Overwinter Consensus and SaplingCryptography Review.* https://research.nccgroup.com/wp-content/uploads/2020/07/NCC_Group_Zcash2018_Public_Report_2019-01-30_v1.3.pdf. 2019.

[67] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. “Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice”. In: *NDSS.* 2022.

[68] K. Guo, K. Hu, and Z. Zhang. “Liveness Attacks on HotStuff: The Vulnerability of Timer Doubling Mechanism”. In: *The Computer Journal* (2024). URL: %5Curl%7Bhttps://doi.org/10.1093/comjnl/bxae027%7D.

[69] C. Hébant and D. Pointcheval. “Traceable Constant-Size Multi-Authority Credentials”. In: *SCN.* 2022.

[70] S. Hohenberger and B. Waters. “Synchronized Aggregate Signatures from the RSA Assumption”. In: *EUROCRYPT.* 2018.

[71] Y. E. Housni and A. Guillevic. *Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition.* CANS. 2020.

[72] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai. “Fast-HotStuff: A Fast and Resilient HotStuff Protocol”. In: *IEEE Transactions on Dependable and Secure Computing* (2023).

[73] J. Katz and N. Wang. “Efficiency improvements for signature schemes with tight security reductions”. In: *CCS.* 2003.

[74] M. Lacharité. “Security of BLS and BGLS signatures in a multi-user setting”. In: *Cryptogr. Commun.* (2018).

[75] A. Lewis-Pye and I. Abraham. “Fever: OptiFmal Responsive View Synchronisation”. In: *Opodis.* 2023.

[76] A. Lewis-Pye, D. Malkhi, O. Naor, and K. Nayak. “Lumiere: Making Optimal BFT for Partial Synchrony Practical”. In: *Podc.* 2024.

[77] Z. Li, A. Sonnino, and P. Jovanovic. “Performance of EdDSA and BLS Signatures in Committee-Based Consensus”. In: *ApPLIED at PODC.* 2023.

[78] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. “Sequential Aggregate Signatures and Multisignatures Without Random Oracles”. In: *EUROCRYPT.* 2006.

[79] D. Malkhi, C. Stathakopoulou, and M. Yin. “BBCA-CHAIN: One-Message, Low Latency BFT Consensus on a DAG”. In: *FC.* 2024.

[80] S. Micali, K. Ohta, and L. Reyzin. “Accountable-Subgroup Multisignatures: Extended Abstract”. In: *CCS.* 2001.

[81] O. Mir, B. Bauer, S. Griffy, A. Lysyanskaya, and D. Slamanig. “Aggregate Signatures with Versatile Randomization and Issuer-Hiding Multi-Authority Anonymous Credentials”. In: *CCS.* 2023.

[82] O. Naor, M. Baudet, D. Malkhi, and A. Spiegelman. “Cogsworth: Byzantine View Synchronization”. In: *arxiv 1909.05204* (2019).

[83] O. Naor and I. Keidar. “Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR”. In: *DISC.* 2020.

[84] P. Paillier and D. Vergnaud. “Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log”. In: *ASIACRYPT.* 2005.

[85] M. Rambaud. “(Section 6 of version 2020-11-29) Malicious Security Comes for Free in Consensus with Leaders”. In: *eprint 2020/1480* (2020).

[86] T. Ristenpart and S. Yilek. “The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks”. In: *EUROCRYPT.* 2007.

[87] V. Shoup. “Lower Bounds for Discrete Logarithms and Related Problems”. In: *EUROCRYPT.* 1997.

[88] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. “Bullshark: DAG BFT Protocols Made Practical”. In: *CCS.* 2022.

[89] X. Sui, S. Duan, and H. Zhang. “Marlin: Two-Phase BFT with Linearity”. In: *DSN.* 2022.

[90] T. Tomita and J. Shikata. *Compact Aggregate Signature from Module-Lattices.* ePrint 2023/471. 2023.

[91] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham. “HotStuff: BFT Consensus with Linearity and Responsiveness”. In: *PODC.* we refer to the arxiv v6 long version. 2019.

A FURTHER FORMALIZATION AND OPTIMIZATION OF $\mathcal{M}\text{to}\mathcal{A}$

All ideas were conveyed in Sec. 4, we now put them in the formalism of Sec. 3.

Theorem 8. *Let* $\mathsf{M} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{iVf}, \mathsf{Cb}, \mathsf{Vf})$ *be any fNIM and* $|v| > 0$ *an integer, then the following scheme* $\mathsf{A} := (\mathsf{A.Kg}, \mathsf{A.Sign}, \mathsf{A.iVf}, \mathsf{Ag}, \mathsf{A.Vf})$ *is a one-time-tagged fNIA supporting messages with variable parts of bitlength* $|v|$.

- $\mathsf{A.Kg}()$: $(\mathsf{sk}_i^{j,b}, \mathsf{pk}_i^{j,b}) \leftarrow \mathsf{Kg}()\ \ \forall\big(j \in [|v|],\ b \in \{0,1\}\big)$;
  $\mathsf{sk_i} \leftarrow (\mathsf{sk}_i^{j,b})_{j \in [|v|], b \in \{0,1\}}$; $\mathsf{pk}_i \leftarrow (\mathsf{pk}_i^{j,b})_{j \in [|v|], b \in \{0,1\}}$
  output $(\mathsf{sk_i},\ \mathsf{pk}_i)$
- $\mathsf{A.Sign}(\mathcal{KG}, \mathsf{sk_i}, \tau_i, v_i)$: *decompose in bits* $v_i = (v_i^j)_{j \in [|v|]}$;
  output $\Sigma_\mathrm{i} \leftarrow \Big(\mathsf{Sign}(\mathcal{KG}, \mathsf{sk}_i^{j,v_i^j}, \tau_i)\Big)_{j \in [|v|]}$
- $\mathsf{A.iVf}(\mathcal{KG}, \mathsf{pk}_i, \tau_i, v_i, \Sigma_\mathrm{i})$: *decompose in bits* $v_i = (v_i^j)_{j \in [|v|]}$;
  output $\bigwedge_{j \in [|v|]} \mathsf{iVf}\Big(\mathcal{KG}, \mathsf{pk}_i^{j,v_i^j}, \tau_i, \Sigma_\mathrm{i}^\mathrm{j}\Big)$
- $\mathsf{Ag}\big(\mathcal{KG}, \tau, (\mathsf{pk}_i, \Sigma_\mathrm{i}, v_i)_{i \in [n]}\big)$:
  decompose in bits $v_i = (v_i^j)_{j \in [|v|]}\ \forall i \in [n]$;
  output $\mathsf{Cb}\Big(\mathcal{KG}, \tau, \big(\mathsf{pk}_i^{j,v_i^j}, \Sigma_\mathrm{i}^\mathrm{j}\big)_{j \in [|v|], i \in [n]}\Big)$
- $\mathsf{A.Vf}\big(\mathcal{KG}, \tau, (\mathsf{pk}_i, v_i)_{i \in [n]}, \Sigma\big)$:
  decompose in bits $v_i = (v_i^j)_{j \in [|v|]}\ \forall i \in [n]$;
  output $\mathsf{Vf}\Big(\mathcal{KG}, (\mathsf{pk}_i^{j,v_i^j})_{i \in [n], j \in [|v|]}, \tau, \Sigma\Big)$

Proof. Both individual completeness and robustness follow straightforwardly from the ones of M, let us prove unforgeability. Consider a forger $\mathcal{F}$ in the game a-uf of Figure 3, and the event where it wins the game. Namely, it outputs $(\mathcal{KG}, \tau^*, (\mathrm{pk}_i, v_i)_{i \in [n]}, \Sigma^*)$, such that there exists $(\dot{\mathrm{pk}}, v^*) \in (\mathrm{pk}_i, v_i)_{i \in [n]}$ such that no query on (τ^*, v^*) was ever responded by the oracle Sign. By assumption, Sign responded to at most one query prefixed by τ^*. Without loss of generality, we can assume that exactly one such query was made and responded to: $\Sigma \leftarrow (\tau^*, v')$. By the above $v' \neq v^*$, thus there is a bit index $j_0 \in [|v|]$ at which $b_0 := (v^*)^{j_0} \neq (v')^{j_0}$, thus no signature of $\dot{\mathrm{pk}}^{j_0,b_0}$ on τ was ever delivered by Sign. Since the other keys of the honest signer: $(\dot{\mathrm{pk}}^{j,b})_{(j,b)\neq(j_0,b_0)}$ are generated independently from $\dot{\mathrm{pk}}^{j_0,b_0}$, the intuitive conclusion is that Σ constitutes a M-forgery on the message τ with respect to the signers $(\mathrm{pk}_i^{j,v_i^j})_{i \in [n], j \in [|v|]}$ and target key $\dot{\mathrm{pk}}^{j_0,b_0}$.

Of course the indices (j_0, b_0) are not known in advance. So to make this argument rigorous and build from $\mathcal{F}$ a forger against M, the reduction must choose at random an index $(j_0, b_0) \in [|v|] \times \{0, 1\}$, and embed its target key in $\dot{\mathrm{pk}}^{j_0,b_0}$. Thus we have at most $2|v|$ loss compared to the $(2|v|n)$-unforgeability of M. □

Optimization for variable parts of variable lengths. In the case of messages with shorter variable bitlengths than the maximum: $|v'| < |v|$, is very easy to enable multisignatures with respect to the shorter set of keys $(\mathrm{pk}_i^{j,b})_{j \in [|v'|], b \in \{0,1\}})$ for each signer. A global adjustment consists in encoding $|v'|$ in the tag for all messages. It is possible to achieve further flexibility and let each signer i adjust the length of its signature depending on the length of its variable part v_i. To this end, add to each public key a list of $|v|$ keys, where the j-th key is used for signing the end-of-string at position $j \in [|v|]$.

B DETAILS FOR APPLICATIONS OF dms

To Blockchain Consensus. The main elementary operation in all such protocols, e.g., [58, 88, 67, 79, 36], is: one (or several) designated Combiner(s) wait(s) to receive a sufficiently large number of signatures, say N, on the same message content: m, then combine the signatures into Σ. Then it multicasts Σ to all the participants to the consensus, dubbed the *processes.* Moreover, Σ is often meant and verified by billions of external clients, since in most cases Σ attests validity of a block. Since in addition Σ is stored on-chain, it is therefore a first-class requirement that Σ be both small and fast to verify. As shown in Table 7, pairing-based multisignatures schemes are the most advantageous instantiation of Σ with this respect, since they take close to $N\times$ less storage space than a naive concatenation of Schnorr signatures. Furthermore, at least for $N \geqslant 3073$, they have 3× smaller Verifier runtime. The last hurdle to their adoption, as stressed in [89], was the runtime of $O(n)$ pairings required to verify pairing-based PoPs. This hurdle is now removed by dms. Still, surprisingly, a number of implementations of consensus protocols instantiate fNIMs as mere concatenations of signatures [58, 88, 67, 38], instead of bls-based multisignatures. The reason invoked [77, 38] is the verification time of an individual bls signature [26, 37](algorithm iVf), which takes 2 pairings. We observe that there exists known ways around this potential runtime gap. First, in case the Combiner would receive individual bls signatures faster than it can verify individually, then it can simply combine them and check them as a multisignature: this was empirically confirmed by [38]. In the rare events where one ill-formed signature would make this batch verification fail, the cheater which issued it would be publicly identified so this is a strong deterrence. Second, there is a more or less known method enabling a faster iVf, which is proposed in [56, 34]. The signer, in addition to its signature: $\sigma_i = \mathrm{sk_i}.\mathrm{H}(m)$, appends to it a "Chaum-Pedersen" proof: π, of knowledge of a common discrete logarithm: $\mathrm{sk_i}$ between σ_i and the public key $X_i := \mathrm{sk_i}.G_2$. Then the verification algorithm: $\mathrm{iVf}^{\mathrm{DLEQ}}$ verifies only π against σ_i and X_i, not anymore σ_i against X_i. Our implementation of $\mathrm{iVf}^{\mathrm{DLEQ}}$, with the same configuration as in Sec. 7 (same machine, gnark-crypto, BLS-377 curve), shows a runtime of 0.785ms, down from 1.9ms for the iVf of a standard bls signature (Table 7). So this reduces the gap w.r.t. our verification time of a Schnorr signature on secp256k1, which is of 0.220ms.

To Threshold Signatures. The recent weighted threshold signature schemes [43, 57] are constructed on the top of bls multisignatures, they both operate as follows. The list of published keys is denoted $(X_i)_{i \in [N]}$ and their weights $(w_i)_{i \in [N]}$. The Combiner collects valid individual bls signatures: $(\Sigma_\mathrm{i})_{i \in I}$ one some message m, issued by a subset $I \subset [N]$, totalizing some desired weight: $w := \sum_{i \in I} w_i$. It outputs the public multisignature $\Sigma := \sum_{i \in I} \Sigma_\mathrm{i}$ and the public weight w. It also outputs a proof of knowledge π of $I \subset [N]$, encoded as a N-sized binary vector $(b_i)_{i \in [N]}$, verifying the following (bi)linear relations: (i) $w = \sum_{i \in [N]} b_i w_i$ and (ii) $e(\Sigma, G_2) == e(\mathrm{H}(m), \sum_{i \in [N]} b_i X_i)$. Note that (i) and (ii) together ensure that Σ passes the bls multisignature verification (m-blsVf*) against the public keys $(X_i)_{i \in [N]}$, and that they totalize weight w. However, the sole passing of (m-blsVf*) does not guarantee unforgeability: as the reader knows well, some processing of the group setup must be done on the group of keys. In [43] the authors suggest using $\mathcal{MSP}$*-pop* [86, 22], where the Verifier verifies pairing-based PoPs appended to the published keys $(X_i)_{i \in [N]}$. Using instead dms, i.e., Schnorr-based PoPs, divides by > 5× this latter runtime, as demonstrated in Table 6.

C DETAILS FOR APPLICATION OF $\mathcal{M}$to$\mathcal{A}$ TO CONSENSUS

In Figure 9 we further recall the consensus Diem21 [44] among n_C processes. Diem21 was used in production by Meta, today by Aptos, and should not be confused with previous versions of Diem, as presented in [58, Fig. 1], which instead followed Hotstuff [91]. Then in Figure 10 we formalize how $\mathcal{M}$to$\mathcal{A}$ can be straightforwardly plugged: either in place of naive concatenation of $(2/3)n_\mathrm{C}$ signatures (in [44]) or in place of the BGLS [23] aggregate signature (in the production version [5]).

Diem21 proceeds by iterations called rounds, each with a designated process called the *leader.* We borrow freely from the terminology of [58, 36]. Although our presentation follows Jolteon [58] (very recently fixed [59]), we stick to the unusual specification of a new-round appearing in Diem21 [44]. We highlight it (in red) in Figure 9. The reason for not choosing the mainstream specification ([62, 59, 79]) of a new-round message, is that the latter mainly consists of a signature on a quorum certificate (QC). Since a QC is typically a multisignature, these objects are not efficiently aggregatable.

We however depart from Diem21 in that our model abstracts-out the view-synchronizer. Let us recall in more detail that a *view-synchronizer* [75] is a protocol enabling players to advance their local round numbers: $r_{new} \longleftarrow r$ in two ways. Either (i) upon receiving a round-($r_{new} - 1$)-quorum certificate from the consensus protocol, which we left explicit in the protocol; or, (ii) upon receiving an abstract signal denoted $\textsc{NextRound}(r_{new})$ for $r_{new} > r$, of which the implementation is the purpose of a view-synchronizer protocol. An *implementation* of a view-synchronizer should guarantee that, eventually, honest players are in the same round for a sufficiently long time, and that this happens infinitely often. Liveness of a consensus protocol is then conditioned to this guarantee. We consider the hybrid model, where processes go to the next round upon receiving a signal NextRound from a black box view-synchronizer. In this hybrid model, Diem21 enjoys a linear number of messages per round since communications are star-shaped around leaders.

Why abstracting-out view-synchronization ? Since the main claim of the Hotstuff consensus [91], i.e., linear communication complexity and responsiveness, is stated in the hybrid model of an abstract view-synchronizer, we choose the same model in order to make an apples-to-apples comparison. Even though an unproven implementation of view-synchronizer was suggested in [91], under the name “Pacemaker”, an attack breaking its liveness was recently shown in [68]. Our choice is also motivated by readability, since our contributions are orthogonal from view-synchronization. Last, even though the protocol Diem21 [44], which we use as baseline, innovated with a nice “Bracha” view-synchronizer, it is now advocated by specialists ([39]) to instead abstract-out view synchronizers, and delegate their implementation to recent dedicated papers with thorough proofs and tight performances [82, 76, 33, 83, 32, 75, 76]. The one of [76] has communication complexity in nf', where f' is the actual number of faults in the execution. As a side-remark, it is actually not hard to imagine how to divide the complexity of the view-synchronizer of Diem21 [44] as follows. Instead of appending their highest QC to the new-view message which they multicast, processes need only appending it to one which they send to the next leader. Thus, provided an implementation of QCs with mere concatenation of signatures, the total communication complexity and verification complexity would both drop from $O(n^3)$ down to $O(n^2)$.

C.0.1 Terminology. Multicast is the instruction to send a message to all, so nothing prevents processes from receiving different messages if the sender is corrupt. A signature of player i on a message m is denoted $\langle m \rangle_i$. When it is an individual signature which is meant to be combined with other signatures on m, i.e., in a multi or threshold signature scheme, then it is instead denoted $\{m\}_i$. Then, a $(2f+1)$-multisignature on m is denoted $\{m\}$. Namely, $\{m\}$ is a tuple consisting of m, a $(2f+1)$-sized subset $J \subset [n]$, and a multisignature Σ on m which is valid w.r.t. the public keys of J.

- *Round Number.* The protocol runs in sequential iterations called rounds $r = 1, 2, 3, \ldots$ where each player starts in round $r = 1$. Note that each player may advance through rounds at a different speed, and at any given time, two players may be in two different rounds due to network delay (since we are in the partially synchronous setting). As local state, each player $\mathcal{P} \in (\mathcal{P}_i)_{i \in [n]}$ keeps track of which round r it is currently in (formerly denoted ~~r_{cur}~~ in [58]). It also stores all of the certified blocks that it has seen thus far, to be defined below. Additionally, we assume that each round r has a pre-determined block proposer called leader: $\mathsf{lead}(r) \in (\mathcal{P}_i)_{i \in [n]}$. It may be randomly or deterministically chosen ahead of time, e.g., [40, 49]; this is referred to as a leader election oracle.
- *Block format.* A block is formatted as $b = [id, qc, rc, r, txn]$ where:
  - $id = H(qc, r, v, txn)$ is the unique hash digest of (qc, r, v, txn);
  - qc is a quorum certificate (QC: defined below) of the parent block of b;
  - r is the round number of b;
  - rc is either (1) a round-($r-1$) new-round (see below), or (2) $rc = \bot$ if qc is a round-($r-1$)-QC, i.e., $qc.r = r - 1$;
  - txn is a batch of new transactions;

  Note that when describing the protocol, it suffices to specify qc and r for a new block, since txn and id follow the definitions. We will use $b.x$ to denote the element x of b.
- *Quorum certificate (QC).* A QC: qc for a block b is a multi (or threshold) signature for the message $(b.id, b.r)$, produced by combining the individual signatures $\{b.id, b.r\}$ from any set of $2f+1$ players. The round number of a QC: qc for a block b is denoted by $qc.r$ which is equal to $b.r$. QCs are ranked by their round numbers, hence, we abuse notation and shorten as $qc \geqslant qc'$ the relation $qc.r \geqslant qc'.r$. Since the QC contained in a block determines its unique parent block, and since the genesis block is the common ancestor of all blocks, the total data structure forms a tree of blocks. A branch is called a “blockchain”. We use $b \longleftarrow b'$ to denote a *2-chain*, i.e., a block b' of which the parent is the block b, i.e., such that $b'.qc$ is a QC of b. Each player stores the highest quorum certificate: qc_{high}, which is the QC with the highest round number, which it ever received or formed. For convenience we denote $r_{high} := qc_{high}.r$ its round number.
- *New-round certificate (RC).* Jolteon & Diem21, [58, 44] make use of data structures called a Timeout Message (tmo) and a Timeout Certificate (TC). We present their definitions as appearing in Diem21 [44] . We rename them as a new-round message and a new-round certificate (RC), for at least two reasons.
  - First, for compatibility with the syntax of view-synchronizers, we specify that players send their round-r new-round message just after entering the new round r, instead of, in [58, 44], sending their round-($r-1$) timeout message upon timing-out in the old round $r-1$. This difference is merely syntactical, processes take the same actions. Namely: both in [58, 44] and in our presentation, players do not either cast anymore round-($r-1$) votes after sending their timeout message.
  - Second, because these data structures do not play anymore a role in the implementation of the view-synchronizer, as they did in [58, 44], since in our model the view-synchronizer is abstracted-out.

  A round-r new-round message by a player i consists of the players's qc_{high}, and of its individual signature on the pair (r, r_{high}): $\langle r, r_{high} \rangle_i$, where we recall $r_{high} := qc_{high}.r$. A round-r RC is meant to be formed out of $2f+1$ new-rounds. In the written specifications [44] (recalled in Figure 9), a round-r RC consists of the naive concatenation of $2f+1$ signed pairs from distinct issuers: $rc \leftarrow \left[\langle r,\ r_j \rangle_j \ :\ j \in J \right]$, where $J \subset [n]$ is a $(2f+1)$-sized subset:

we follow this . Whereas in the production version [5], the RC is the BGLS aggregate signature $rc \leftarrow \mathrm{BGLS.Ag}\Big[\langle r,\ r_j\rangle_j\ :\ j\in J\Big]$. Note that this presentation simplifies the slightly more compact encoding in [44], where the common message prefix r is factored out of the $2f+1$ signed messages. Following [85], note that rc plays the role of a *proof of non-supermajority* because it guarantees that no set of $f+1$ honest players, upon entering round r, could have previously voted for a block containing a QC of round strictly higher than $r_{max} := \max(r_j,\ j\in J)$.

Diem21 with black-box view-synchronizer

Instructions for each player $\mathcal{P}_i$, $i\in[n]$ in (local) round r (denoted ~~r_{curr}~~ in Jolteon). It keeps the highest voted round r_{vote}, the highest quorum certificate qc_{high} and the highest locked round $r_{high} := qc_{high}.r$. Players initialize $r_{vote}=0$, $r_{high}=0$, qc_{high} as the QC of the genesis block of round 0, and enter round $r=1$. The leader of round r is denoted lead(r).

Propose Upon entering round r, if $\mathcal{P}_i$ is the leader lead(r), then it waits until the first of the following two events happens:

- receiving or forming a round-$(r-1)$ QC, i.e., $qc_{high}.r = r-1$ ▹ e.g., if it entered round r upon receiving qc_{high}. Then it sets $rc \leftarrow \perp$; or
- receiving $2f + 1$ valid round-r new-round messages: $\Big[\big(\langle r,\ r_j\rangle_j\ ,\ qc_{high,j}\big)\ :\ j\in J\Big]$, where $J\subset[n]$ is of size $2f+1$. In this case it sets:
  $$rc \leftarrow \Big[\langle r,\ r_j\rangle_j\ :\ j\in J\Big]$$ ▹ proof of non-supermajority
  ▹ In the production version [5] rc is the BGLS aggregate.

Then it multicasts a block $b=[id, qc_{high}, rc, r, txn]$.

Vote Upon receiving the first proposal $b=[id, qc,\ rc, r, txn]$ from lead(r) while in round r execute **Lock**, and **Advance Round**, and then **Commit**, as instructed below. If $r > r_{vote}$ and $\Big\{$ either (1): $r = qc.r+1$; or (2) $rc = \big[\langle r,\ r_j\rangle_j\ :\ j\in J\big]$ with $|J| = 2f+1$ and $qc.r \geqslant \max\{r_j \mid j\in J\}$ $\Big\}$, then vote for b by sending the individual signature $\langle id, r\rangle$ to lead($r+1$), and update $r_{vote}\leftarrow r$.

Lock Upon receiving or forming a quorum certificate: qc, update $qc_{high}\leftarrow \max(qc_{high}, qc)$ (and thus $r_{high}\leftarrow\max(r_{high}, qc.r)$).

Commit (2-chain commit rule) Whenever there exists two adjacent certified blocks $b\leftarrow b'$ in the chain with consecutive round numbers, i.e., $b'.r = b.r+1$, commit b and all its ancestors.

Advance Round ▹ The dotted box reminds the view-synchronizer model:

For any $r_{new} > r$, update the current round number $r\leftarrow r_{new}$ *iff*
▹ implying that it stops voting for round- $\leqslant (r-1)$ proposals

- either upon receiving or forming a round-$(r_{new}-1)$ QC: qc ;
- or, upon NextRound(r_{new}) interrupt;

in this latter case, i.e., NextRound(r_{new}), it then sends to lead(r) a round-r new-round message: $\Big(\langle r,\ r_{high}\rangle_i\ ,\ qc_{high}\Big)$
▹ where $\langle r,\ r_{high}\rangle_i$ is a (standalone) signature of $\mathcal{P}_i$ on $(r,\ r_{high})$,
▹ and where we recall $r_{high} := qc_{high}.r$.

~~**Timer and Timeout** (Implemented NextRound() signals)~~

Figure 9: Differences with Jolteon: new-round messages and RCs (highlighted). Differences with both Jolteon & Diem21: the black-box view-synchronizer (dotted-boxed), which replaces the ~~explicit implementation of NextRound() from timeout certificates in Diem21/Jolteon~~.

Diem21 + $\mathcal{M}$to$\mathcal{A}$

Instructions for each process $\mathcal{P}_i$. Same initialization as in Diem. We consider A := (Kg, Sign, iVf, Ag, Vf) a one-time-tagged aggregate signature scheme, e.g., obtained from $\mathcal{M}$to$\mathcal{A}$.

Propose Upon entering round r, if $\mathcal{P}_i$ is the leader lead(r), then it waits until the first of the following two events happens:

- receiving or forming a round-($r-1$) QC: qc_{high}. Then $rc \leftarrow \bot$;
- or, receiving $2f+1$ round-r new-round messages: $\left[\left(\langle r,\ r_j\rangle_j\ ,\ qc_{high,j}\right)\ :\ j \in J\right]$. Then it aggregates the signatures w.r.t. the fixed tag r :

$$rc \leftarrow \left[r,\ (r_j)_{j\,\in\,J},\ J,\ \mathsf{A.Ag}\left(r\ ,\ \langle\, r_j \rangle_{j\,\in\,J}\right)\right]$$

▷ proof of non-supermajority

Then it multicasts a block $b = [id,\ qc_{high},\ rc,\ r,\ txn]$.

Vote Upon receiving the first proposal $b = [id,\ qc,\ rc,\ r,\ txn]$ from lead(r) while in round r, **Lock**, and **Advance Round**, and then **Commit**. If $r >\ r_{vote}$ and $\Big\{$ either (1): $r = qc.r + 1$; or (2) $r = rc.r + 1$ and $rc = \left[r,\ (r_j)_{j\,\in\,J}\ ,\ J,\ \Sigma\right]$ with $|J| = 2f+1$ and Σ a valid A-signature on the tagged messages $\left(r,\ r_j\right)_{j\,\in\,J}$ w.r.t. the public keys of J and $qc.r \geqslant\ \max\{r_j \mid j \in J\}$ $\Big\}$, then vote for b by sending the individual signature $\langle id, r\rangle$ to lead($r+1$), and update $r_{vote} \leftarrow r$.

Lock, Advance round, Commit Same as in Diem21.

New-round Upon receiving a call NextRound(r) from the view-synchronization mechanism, advance the current round number $r \leftarrow r+1$. Send to lead(r) a new-round message:

$$\left(\{r,\ r_{high}\}_i\ ,\ qc_{high}\right)$$

where $\{r,\ r_{high}\}_i \leftarrow \mathsf{A.Sign}(\mathsf{sk}_i, r,\ r_{high})$ is an individual signature of player $\mathcal{P}_i$, and where we recall $r_{high}\ :=\ qc_{high}.r$.

Figure 10: Differences with Figure 9 are highlighted.

D FURTHER DETAILS ON RELATED WORKS

4.1 $\mathcal{AS}$*-3 ([15]).* We recall below the verification algorithm, tagged ($\mathcal{AS}$-3.Vf), of the fNIA called $\mathcal{AS}$-3 in [15]. $\mathcal{AS}$-3 is a verifier-unrestricted variant of the seminal BGLS [23]. $\mathcal{AS}$-3 is defined over a bilinear group $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with public generators G_1, G_2, and for a given hash-to-curve H. The X_i are the public keys.
($\mathcal{AS}$-3.Vf)

$$(X_i, m_i)_{i\,\in\,[n]},\ \Sigma\ \longrightarrow\ e\big(\Sigma, G_2\big) == \sum_{i\,\in\,[n]} e\Big(\mathrm{H}(X_i|m_i)\,,X_i\Big)\ \wedge\ \big(\Sigma \in \mathbb{G}_1\big)$$

4.2 Pairing-based fNIM of Boldyreva [20]. We recall below its verification formula, tagged (m-blsVf*). Recall that this fNIM is secure only in the kosk model. The formula is the same as the one of dms (Sec. 5), without the checks of PoPs on public keys (the function which we called kVf).
(m-blsVf*)

$$(X_i)_{i\,\in\,[N]},\ m\ ,\ \Sigma\ \longrightarrow\ e\big(\Sigma\,,G_2\big)\ ==\ e\Big(\mathrm{H}(m)\ ,\ \sum_{i\,\in\,[N]} X_i\Big) \wedge \Sigma \in \mathbb{G}_1$$

4.3 SMSKR ([9]). The key published by each member of the group of potential signers, consists of a raw $\mathbb{G}_2$ element: $\mathsf{pk} = X \leftarrow x.G_2$. Once all keys of the group have been published: $\mathcal{KG}$, each signer re-randomizes its secret key:

$$x \leftarrow \mathrm{H}(\mathcal{KG}|X).x\ \ . \tag{23}$$

Hence, the individual signatures which it will generate with its re-randomized secret key x, will be valid w.r.t. the re-randomized public key:

$$X \leftarrow \mathrm{H}(\mathcal{KG}|X).X\ \ \forall X \in \mathcal{KG}\ . \tag{24}$$

Likewise, combined signatures will be verified against the sum of the re-randomized keys of the subgroup of signers. Hence, the Verifier must compute the rerandomized public keys. Namely, the processing of the group setup consists of computing Eq. (24) for each $X \in \mathcal{KG}$.

4.4 $\mathcal{MSP}$*-pop [86, 24][22, §6], and batch verification of group setup.* Each key $\mathsf{pk}_i = (X_i, \pi_i)$ comes appended with a PoP equal to a BLS signature on X_i: $\Pi_i \leftarrow x_i.X_i$, where $X_i \leftarrow x_i.G_2$. Their verification cost is dominated by two pairings for each key:

$$\mathsf{kVf}(\mathsf{pk}_i)\!\longrightarrow \big(e(\Pi_i, G_2) == e(\mathrm{H}(X_i), X_i)\big)\ \wedge\ X_i \in \mathbb{G}_2 \wedge\ \Pi_i \in \mathbb{G}_1\ . \tag{25}$$

In our benchmarks (Table 6), we first used the 2× speedup of [35] for batch verification of BLS signatures: the Verifier samples random numbers $(e_i)_{i\,\in\,[N]} \xleftarrow{\$} \mathbb{Z}_p^{[N]}$, then checks

$$e\Big(\sum_{i\,\in\,[N]} e_i.\Pi_i\ ,\ G_2\Big) == \sum_{i\,\in\,[N]} e\Big(e_i.\mathrm{H}(X_i)\ ,\ X_i\Big)\ . \tag{26}$$

Second, we sped-up the right-hand sum with the optimized implementation of products of pairings in gnark-crypto, inherited from [65].

4.5 dms*: Batch verification of group setup.* In our benchmarks (Table 6) we sped-up the verification of the PoPs of dms, i.e., $\mathsf{kVf}(\mathsf{pk}_i)$ $\forall i \in [N]$, using the method of [14] for batch verification of Schnorr signatures. Namely: parse $(X_i, \pi_i) \leftarrow \mathsf{pk}_i$ and $(R_i, z_i) \leftarrow \pi_i\ \forall i \in [N]$; $c_i \leftarrow \mathrm{H}_{\mathrm{pop}}(X_i, X_i, R_i)$; sample $(e_i)_{i\,\in\,[N]} \xleftarrow{\$} \mathbb{Z}_p^{[N]}$; output

$$X_i \in \mathbb{G}_2 \forall i \in [N] \wedge \Big(\sum_{i\,\in\,[N]} e_i\, z_i\Big)G_2\ ==\ \sum_{i\,\in\,[N]} e_i R_i + \sum_{i\,\in\,[N]} (e_i\, c_i).X_i \tag{27}$$

4.6 SIG_1 *[27].* The recent (non-dynamic) fNIM called SIG_1 [27] has a processing of the group setup runtime which is one order of magnitude higher than the three previous fNIMs, i.e., SMSKR, $\mathcal{MSP}$*-pop* and dms. Verification of N keys (each N-sized) requires $O(N^2)$ pairings and $O(N^2)$ group membership tests, instead of $N+1$ pairings in $\mathcal{MSP}$*-pop* and $O(N)$ group membership tests in all three previous fNIMs. Then, computing the verification key of a group $\mathcal{KG}$ of N keys takes N multi-additions, each with N terms.