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

Matthieu Rambaud
Télécom Paris

Christophe Levrat
Inria Saclay

dms

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 verification time of a 3000-wise aggregate signature of BGLS (Eurocrypt’03), takes 300x longer verification time than verification of a 3000-wise pairing-based multisignature. (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 approach easily fills a gap in this 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, 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 $\mathcal{M}_{to,A}$ 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.

I. Introduction

In an aggregate signature scheme [23, 15, 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 $\mathcal{A}$ which, roughly, takes as input any multiset of triples of public key - message - valid individual signature: $(pk_i, m_i, \Sigma_i)_{i \in [n]}$, and outputs a single
aggregate signature \( \Sigma \). Finally, there is a public verification algorithm \( \text{Vf} \) which takes as input a purported aggregate signature \( \Sigma \) with respect to a multiset of pairs of public key-message: \((pk_i, m_i)\), \( i \in [n] \), and outputs a bit denoting acceptance or rejection. Such schemes have the unforgeability property, that acceptance implies that for any pair \((pk_i, m_i)\) in the multiset such that the key \( X \) was generated by some honest process, then it must have signed the corresponding message \( m_i \). 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: \( \text{Cb} : m_i, (pk_i, \Sigma_i) \), \( i \in [N] \to \Sigma \). 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 8 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 \( 300 \times \) 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 \( 6.5 \times \) 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 fNIMs [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 \( n_{GC} \) 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 Apts. It proceeds by iterations called “rounds”. Under good conditions, the leader of a new round in Diem21 only has to combine \((2/3)n_C\) 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 \((2/3)n_C\) signatures over different so-called “timeout messages”. It multicasts the aggregate signature, called a “timeout certificate”. Aggregation is done by Apts ([5]) with BGLS. Hence, already for \((2/3)n_C = 129\), we observe in Table 8 that verification of a timeout certificate is \( 49 \times \) 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 \( K_G \). The Verifier must then process \( K_G \), we call this task processing of the group setup.

In a first category of fNIMs (\( \text{MSP} \)-pop [86, 22, 24], Pixel and \( \text{ASMP} \)-pop [86, 47, 22, 24]), each signer incorporates a so-called proof of possession (PoP): \( \pi \) into its public key: \( pk = (X, \pi) \). 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 the kosk. The fNIM of Boldyrev [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 \( \pi \) 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: \( K_G \) 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 8, this time is orders of magnitude longer than the online verification of a multi-signature.

In a second category of fNIMs (\( \text{ASMP} \)-pop [22], SMSKR [9] and \( \text{SIG}_1 \) [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: \( K_G \) In \( \text{ASMP} \) and \( \text{SIG}_1 \) 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: \( K_G \). 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 8, 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: \( K_G \).

A last category of fNIMs (the blog version \( \text{MSP-blog} \) [21, 56]) does not require processing of the group setup tasks, beyond verification of membership of the keys in the subgroup \( G_2 \). 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 \( N \)-sized hashes).

**Goal (iii): achieving provable security for use with the curves used in practice.** In Table 7 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 \( \text{MSP-pop} \) [86, 22, 24] (IETF draft standard 2022), we find that the proven
bound on the probability of a forgery after $2^{20}$ 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-Namprempren-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 $\mathcal{SIG}_1$ of [27], but which has a quadratic Verifier’s processing of keys (see Sec. VII and below). So this proves 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].

A. An Efficient Multi-to-Aggregate Compiler

We address Goal (i) by introducing $\mathcal{MtoA}$: 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 $\tau$, 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. $\mathcal{MtoA}$ operates as follows. Each signer prepares and publishes $2|v|$ public verification keys. To sign a message $m_i = (\tau, v_i)$, it signs $|v|$ 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 $\tau$, 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 $\tau$ (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) Performance and Applications of $\mathcal{MtoA}$, and related works: We resume the example of the production implementation [5] of Diem21 [44] by Apto. A timeout message of a process $P_j$ is formatted as the signed message: $\{r, r_j\}$. The tag $r$ is the current round number, while $r_j < r - 1$ is roughly the highest round number in which $P_j$ saw a QC: $q^{2/3}_{\text{high,j}}$. 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 $\mathcal{MtoA}$ is efficient. In Table 8 we report on the Verifier’s runtime for an $\mathcal{MtoA}$ 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. This is $39 \times$ 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 $\mathcal{MtoA}$). By contrast, $\mathcal{MtoA}$ has post-quantum security whenever applied to any post-quantum fNIM, e.g. to [51]. Another use-case where $\mathcal{MtoA}$ 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.

B. 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.

a) 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-Oht-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. VII-2).

The main difficulty of our proof consists, upon being given $\pi_i, \pi_{i,j}$, in extracting from the forger all the secret keys, i.e., $\xi$ s.t. $X_i = \xi.G_2$, where $G_2$ is a public generator of the subgroup $G_2$ (except the one
of the challenge public key $X$). The last step from there is that the reduction builds a valid individual signature for $X$ 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 $X$, without the forger giving $\dot{x}$ to the reduction. The main difficulty is singled-out in Sec. VI 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 $X$). 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 $X$. 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. III) the adversary has the power to query such an oracle potentially before it chooses the set of keys of its forgery. This power models that the adversary could, in practice, engage in signing sessions with concurrent groups containing the same target key $X$, 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: $\{ (H(m_j), \dot{x}.H(m_j)) \}_{j \in X} = \dot{x}.G_1, G_2 \}, j$ 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].

b) 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 7 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 $5 \times$ 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 8, batch verification of BLS individual signatures (as used in MSP-pop, SMSKR, dms) is $3 \times$ 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 Sec. B. In Sec. B we also explain how dms can be advantageously plugged in the threshold signature schemes [43, 57], in place of MSP-pop.

II. 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 $A$ in a game $g$ [12], denoted $P(g^A = 1)$, to designate the probability that $A$ wins the game, i.e., that $g$ sets the flag $\text{win} \leftarrow 1$.

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

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 fortiiori a public uniform random string (URS). So $G_1, G_2$ could be fixed by seeding any public uniform sampling algorithm with the URS.

b) 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 $A$ executed in a security game ([17]) is called algebraic if for all group elements $Z \in G_i, i \in \{1, 2, T\}$ that $A$ outputs to any oracle of the game, it additionally provides a representation in terms of received group elements in $G_i$ and those from groups from which there is an efficient mapping to $G_i$. In particular for $Z \in \{G_1, G_2\}$: if $U_0, \ldots, U_\ell \in G_1$ and $V_0, \ldots, V_m \in G_2$ denote the group elements in $G_1$ and $G_2$ received so far by $A$, then it provides a list of coefficients in $\mathbb{Z}_p (\mu_{i,j})$ in $\{0, 1\}$, and possibly $(\zeta_{i,j})_j \in [m]$, such that, depending on the case:

- $Z \in G_1$ (Type 1 and 2): $Z = \sum_i \mu_i U_i + \sum_j \zeta_j \psi(V_j)$
- $Z \in G_2$ (Type 1): $Z = \sum_i \mu_i U_i + \sum_j \zeta_j V_j$
- $Z \in G_T$ (Type 2 and 3): $Z = \sum_j \zeta_j V_j$

c) 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 $\text{Sign}$ and verification $\text{Verf}$, the latter returning a bit denoting acceptance or rejection. They furthermore have the following properties. Correctness requires that $\forall (sk, pk) \leftarrow Kg(), \forall m,$
Vf(pk, m, Sign(sk, m)) = 1. Unforgeability requires that a polynomial adversary 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 exemplify 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: \(G_1 \in G_2, G_2 \in G_2\), and by a hash-to-curve map \(m \rightarrow H(m) \in G_2\) which we model as a random oracle.

- **BLS.Kg()**: sample \(x \in \mathbb{Z}_p\), output \((sk, pk) = (x, x.G_2)\);
- **BLS.Sign** : \(m \rightarrow sk.H(m)\);
- **BLS.Vf** : \(pk, m, \Sigma \rightarrow e(\Sigma, G_2) = e(H(m), X)\).

![Figure 1: EUF-CMA game for BLS](image)

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 reduction [26] to the problem called co-cdh: given \((Q, a.G_1, a.G_2)\) for \(a \in \mathbb{Z}_p\), compute \(a.Q\). Whereas over bilinear groups of type III, it has a loose reduction [37] to the problem called co-cdh*:\(a \in \mathbb{Z}_p\), compute \(a.Q\). Note that in BDN18 [22], co-cdh and co-cdh* are respectively renamed \(\psi\)-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 \(G_1\). We observe that over type II groups, a minor adaptation of the proof shows a tight reduction to the DL in \(G_2\) (use \(\psi\) to port the DL challenge in \(G_1\)). 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 \(a\).

### III. New Definitions

We first define fully non-interactive multisignature schemes (NIM), then fully non-interactive one-time-tagged aggregate signature schemes (NTA), 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 \(KG\), called the keys of the group (of potential signers). A scheme is called dynamic when this input plays no role. In turn, all \(KG\)-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 \(KG\) 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, ASMP [22, §4] and 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 \((pk_i, \Sigma_i) \in [N]\) or \((pk_i, v_i, \Sigma_i) \in [n]\), 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** (NIM). A fully non-interactive multisignature scheme consists of the following local algorithms.

- **Kg()** \rightarrow \((sk, pk)\) (key generation)
- **Sign(\(KG\), sk, i) \rightarrow \Sigma_i** (signing)
- **iVf(\(KG\), pk, m, \Sigma_i) \rightarrow 0/1** (individual signature verification)
- **Cb(\(KG\), m, (pk, \Sigma_i) \in [N]) \rightarrow \Sigma** (combination)
- **Vf(\(KG\), (pk, \Sigma_i) \in [N], m, \Sigma) \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 \(KG\) to which it belongs, must pass individual verification. Formally:

\[
\Pr\left(0 \rightarrow iVf(\{KG\}, pk, m, \Sigma) \right) = \text{negl}
\]

**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 \(KG\) containing all signers, as soon as they all pass individual verification. Formally, for any polynomial adversary \(A\):

\[
\Pr\left(0 \rightarrow Vf(\{KG\}, (pk, \Sigma_i) \in [N], m, \Sigma) \right) = \text{negl}
\]

Unforgeability states that any polynomial forger \(F\) has negligible advantage in the following game denoted \(mf\-uf\).

A challenger exhibits to \(F\) an honestly generated key \(pk\), then \(F\) commits to a group of public keys \(KG\), then it is granted access to a pk-signing oracle with respect to \(KG\). The forger wins if it can create a valid multi-signature on behalf of some subgroup of signers containing \(pk\), on a message never queried to the oracle.

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 \(KG\) of keys. So this non-dynamic restriction rules-out a forger which would concurrently interact with multiple groups. But this restriction appears
requires that aggregation must re-

group with respect to keys com-

on In all existing \( fNIM \) schemes, the ver-

vari

\( F \)

KG

\( \exists \sigma \in \Sigma^* \) requires that for any com-

a all

Moreover they should satisfy the following three properties.

(3)

\( \sigma \in \Sigma^* \) requires to regroup them in pairs:

\( \tau \),\( v \)

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.

- \( \text{Kg}() \rightarrow (\text{sk}, \text{pk}) \) (key generation)

- \( \text{Sign}(\text{KG}, \text{sk}, \tau, v) \rightarrow \Sigma_i \) (signing)

- \( \text{Vf}(\text{KG}, \text{pk}, \tau, v, \Sigma_i) \rightarrow 0/1 \) (individual signature verification)

- \( \text{Ag}(\text{KG}, \tau, (\text{pk}, v, \Sigma_i)) \rightarrow \Sigma \) (aggregation)

- \( \text{Vf}(\text{KG}, \tau, (\text{pk}, v, \Sigma_i)) \rightarrow 0/1 \) (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 \( \text{KG} \) to which it belongs, \( \text{must} \) pass individual verification. Formally:

\[
\Pr \left( 0 \leftarrow \text{Vf}(\text{KG}, \text{pk}, \tau, v, \Sigma) \right) = \neg \text{negl} \right)
\]

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 \( \text{KG} \) containing all signers, as soon as they all pass individual verification. Formally, for any polynomial adversary \( A \):

\[
\Pr \left( 0 \leftarrow \text{Vf}(\text{KG}, \tau, (\text{pk}, v_i), m, \Sigma) \right) = \neg \text{negl} \right)
\]

Unforgeability requires that any polynomial forger \( F \) has negligible advantage in the following game denoted \text{a-uf}.

A challenger exhibits to \( F \) an honestly generated key \( \text{pk} \), then \( F \) commits on a group of public keys \( \text{KG} \), then it is granted access to a \text{pk}-signing oracle with respect to \( \text{KG} \), 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 \((\text{pk}, v^*)\), for some tag \( v^* \), such that the oracle never delivered a signature on the tagged message \((\tau|v^*)\). Formally:

\[
\Pr \left( 0 \leftarrow \text{Vf}(\text{KG}, \tau, (\text{pk}, v_i), m, \Sigma) \right) = \neg \text{negl} \right)
\]

Comments on Definitions and Related Works

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

Key-aggregation. In all existing \text{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: \( \text{KAg} : (\text{KG}, (\text{pk}, i) \in [n]) \rightarrow \text{apk} \). Then, verification proceeds using only \text{apk}: \( \text{Vf-}\text{lazy}(\text{apk}, m, \Sigma) \rightarrow 0/1 \). We purposely 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 \text{Vf-}\text{lazy}, but it would be insecure to run such a Verifier algorithm without any
This is a property which we credit to the unforgeability definitions in [56, §3.3]. Of course solving this apparent issue would just require to specify that the RSMS-pop algorithm verifies PoPs, in a way or another.

Interactive setups. The issue of enabling verification of apks 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 MORO[80] and ASMP & 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 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 ASMP/ASMP-pop decreases linearly only in the total number of signers over all groups in which the target key pk is involved. Of course all these issues disappear in dynamic schemes.

Non-interactive schemes. In the scheme MSP [22], the Sign algorithm takes as input the exact list of the keys which are meant to sign the message: (pk_i)_{i ∈ [N]}. Upon receiving N signatures on some message m generated by the keys (pk_i)_{i ∈ [N]}, the combination Cb can combine them only if they were all generated with input (pk_i)_{i ∈ [N]}. Said in the other way: Cb cannot combine any signature on m generated with input (pk_i)_{i ∈ [N]}, unless receiving such signatures from all the intended signers (pk_i)_{i ∈ [N]}. Said otherwise: aggregation fails as soon as one of the intended signers aborts (this issue is lifted in the blog version 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.

IV. MtoA: MULTI TO AGGREGATE COMPILER

We convey all ideas of MtoA, further formalism and the (obvious) proof can be found in Sec. A. We consider any fnIM: M, and describe the resulting one-time-tagged fnIM, called A. Each potential signer i generates then outputs a list of 2|v| M-public keys: \{pk_i^{a,b} | j ∈ [v], b ∈ \{0, 1\}\}. To sign some (τ | v_i), the signer i parses v_i = (v_i^j)_{j ∈ [v]} as the bit decomposition of the variable part, then outputs a signature on τ under each public key pk_i^{a,v_j} for j ∈ [v_i]. 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)_{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 multisig- natures on the public keys, although it does shows-up in the two-round schemes ([46, 42, 13]): although it does appear in the blog version MSP-blog [21]. This limitation also tightly reduce to the hardness of the DL, in the AGM, by [53, §6]. The syntax of dms does not contain any group KG, hence, it is a dynamic fnIM. Noticeably, and unlike pairing-based PoPs, no subgroup membership in 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 G_2.

V. dms

dms is specified over any bilinear group \{(G_1, G_2, G_T, e)\} and operates with any hash-to-curve random oracle \(H : \{0, 1\}^n → G_1\) and random oracle \(H_{pop} : \{0, 1\}^n → \mathbb{Z}_p\). We will show that its security 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. V-A we state the main ingredient of unforgeability of dms. It states that whenever an algebraic adversary A outputs a public key X^a along with

\[ A \]
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. V-B we conclude the proof of unforgeability from this ingredient.

A. Extracting Schnorr despite correlations

We formalize the game called **ssc** in Figure 4, which stands for “Schnorr Straight-line extraction in presence of BLSS 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 \((X, \pi)\) to the adversary \(\mathcal{A}\). Then the adversary is given access to hash-into-\(G_1\) and hash-into-\(Z_p\) random oracles: \(H : m_i \rightarrow M_i\) and \(H_{\text{pop}}\); and to a BLS signing oracle for the honest secret key \(\text{SIGN} : m_i \rightarrow \Sigma_i := x.H(m_i)\).

All in all, up to delivering to \(\mathcal{A}\) both replies from \(\text{SIGN}\) and \(H\) for every queried \(m_i\), we have that \(\mathcal{A}\) is delivered a random string with a hidden structure: \((M_i, x.M_i) \in [q_H]\), in addition to the Schnorr proof of knowledge \(\pi\) for the same exponent: \(x\) of \(X\). In the game \(\text{ssc}\) 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 \(X^*\) and \(R^*\) in terms of all group elements received so far, of which all hashes \(M_i\) and signatures \(\Sigma_i\) returned by the oracles so far:

\[
\begin{align*}
X^* &= \alpha.G + \beta.X + \sum_{i \in [q_H]} \gamma_i^\prime \Sigma_i + \sum_{i \in [q_H]} \delta_i^* M_i \\
R^* &= \alpha'.G + \beta'.X + \sum_{i \in [q_H]} \gamma_i^\prime \Sigma_i + \sum_{i \in [q_H]} \delta_i^* M_i
\end{align*}
\]

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 \(\alpha\). 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 \(\text{DL}\). The proof is done in Sec. VI and is one of our main technical contributions.

**Theorem 4.** From any \(\text{ssc}\)-adversary \(\mathcal{A}\) with advantage \(\epsilon\) and making \(q_H\) RO queries, one can build an adversary \(\mathcal{B}\) against the discrete logarithm \(\text{DL}\) in \(G_2\) with advantage \(\epsilon' \geq (1 - \frac{q_H + 1}{p})(\epsilon - q_H/p)\) and with at most twice the running time.

B. Tight reduction of \(\text{dms}\) to \(\text{DL}\).

The proof follows from the following chain of tight reductions:

\[
\begin{align*}
\text{m-uf (dms)} &\xrightarrow{\text{Lem. 5}} \text{ssc} \wedge \text{BLS-uf} \xrightarrow{\text{Thm. 4}} \text{DL} \wedge \text{BLS-uf} &\xrightarrow{[53]} \text{DL}
\end{align*}
\]

We now outline each of the reductions. The game on the left is the unforgeability of \(\text{dms}\). 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 \(\text{m-uf}\) of \(\text{dms}\), with advantage \(\epsilon\), running time \(t\) and making at most \(q_H\) RO queries, then there exists a forger \(\mathcal{B}\) in the unforgeability game \(\text{BLS-uf}\) of BLS signatures, which has advantage \(\epsilon' \geq \epsilon - q_H/p - UB^{\text{ssc}}(t)\) and at most twice the running time \(t\), where \(UB^{\text{ssc}}(t)\) denotes the upper-bound on the advantage of a time-\(t\) adversary in game \(\text{ssc}\).

The third reduction, labelled by Thm. 4 is because Theorem 4 shows a tight reduction from \(\text{ssc}\) to \(\text{DL}\).

The last reduction, from BLS unforgeability to the hardness of \(\text{DL}\), is proven in [53, 56]. 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 \(\text{DL}\) problem to be considered is when the adversary is given for free all valid secret keys from \(\text{BLS}\), which is all the point of our work, is that instead of being “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 where \(\mathcal{A}\) would already have queried \(H_{\text{pop}}(X, X, R)\); then outputs \(\pi := (R := r.G_2, z)\). Then it gives \(pk \leftarrow (X, \pi)\) 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 \(\text{ssc}\). Whenever \(\mathcal{F}\) makes an \(\text{m-uf}\)-SIGN signature query on some message \(m\), \(\mathcal{B}\) simply makes the query on the same \(m\) to its own \(\text{BLS-uf}\)-SIGN oracle, then forwards the result to \(\mathcal{F}\). At some point, \(\mathcal{F}\) outputs a \(\text{dms}\) forgery: \(((pk)_i)_{i \in [N]}, m, \Sigma\), in particular such that \(m\) was not queried before to \(\text{SIGN}\).
In what follows we honestly generates the public key which is equal to $F_e(\Sigma)$, except with probability $UB^{SSC}(t)$, in at most twice the running time $t$ of $F$.

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

**Proof of the Claim.** Consider formally the adversary $F^\bot$, which is equal to $F$ except that the last two outputs of its forgery are removed: $(pk_i)_{i \in [N]}, \bot, \bot$. Then $F^\bot$ is a game-SSC adversary, which concludes the Claim.

**End of the proof.** Informally, the reduction $B$ removes, from the forgery $\Sigma$, 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, $B$ computes the individual signatures $\Sigma_i \leftarrow x_i, H(m)$, then outputs the forgery:

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

Let us formally verify that $\Sigma_X$ 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:

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

On the other hand, $\Sigma$ being a valid dms multi-signature, we have

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

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

$$ e \left( H(m), \sum_{i \in [N_1]} x_i, G_2 + (N - N_1)X, G_2 \right) - e \left( \sum_{i \in [N_1]} x_i, H(m), G_2 \right).$$

Finally, cancelling out all equalities $e(H(m), x_i, G_2) = e(x_i, H(m), G_2)$, we obtain:

$$ (N - N_1)e(\Sigma_X, G_2) = e \left( H(m), (N - N_1)X \right),$$

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

### VI. Proof of Theorem 4

As in most works on BLS [53, 7], we describe only the proof in the case of type I bilinear groups, so in what follows we identify $G := G_1 = G_2$ and $G' := G_1 = G_2$, except 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 $A$ submits at most one Schnorr signature to the game SSC, instead of $q_A$ 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 $A$ 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 $B$ against $DL$ receives a challenge $L$ and its goal is to output the exponent $t$, i.e., s.t. $L = tG$. To this end, it runs the SSC adversary $A$ and simulates SSC to it, as follows. We call $\mathcal{B}$ the master reduction because it tosses a coin and, depending on its value 1 or 0, behaves towards $A$ 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 $A$, is how the challenge $L$ is embedded. Both reductions use procedures, denoted $H$, $SIGN$ and $H_{pop}$, to simulate to $A$ the responses to its queries to oracles $H$, $SIGN$ and $H_{pop}$ 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 $\pi$ on $X$ following the standard technique, namely: samples $(r, z) \leftarrow Z_p^2$, programs the random oracle as $c \leftarrow H_{pop}(X, X, R)$, then outputs $\pi \leftarrow (R := rG_2, z)$. So its simulation of SSC is perfect, up to the $q_A$-probability event where $F$ would have queried $H_{pop}(X, X, R)$ before it was programmed on $c$.

Reduction D honestly generates the public key $X$ as $x \leftarrow Z_p, X \leftarrow xG_2$. It embeds the $DL$ challenge in the simulated hash-to-curve oracle $H$, using the following trick of [53, §6]. Upon queried a new message $m$, $H$ samples $(h_i, b_i) \leftarrow Z_p^2$ then returns $M_i \leftarrow b_iL + h_iG$. In particular, the output $M_i$ varies uniformly in $G$, so $H$ perfectly simulates a random oracle. In conclusion, it can perfectly simulate the signing oracle, as: $m \leftarrow xH(m)$. Note that since $D$ knows the secret key $x$, it could also honestly generate a Schnorr PoP as $r \leftarrow Z_p, R \leftarrow rG_2, c \leftarrow H_{pop}(X, X, R)$ then $\pi \leftarrow (rG_2, z + r + cX)$. So it would not have to program $H_{pop}$, 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 $H$, 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 $C^A = 1$ where $A$ outputs a winning triple $(X^*, R^*, z^*)$ to $C$. Namely, it passes verification of Schnorr proofs, i.e., s.t. for $e^* \leftarrow H_{pop}(X^*, X^*, R^*)$ then $z^*G = R^* + e^*X^*$. Being algebraic, $C$ 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 $A$ submitted when it outputted $X^*$ and $R^*$ for the first time, i.e., either to oracle $H_{pop}$ or directly to the main C procedure. Since $A$ won, then $\alpha$ 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. $xG = X$. To explain how C tries to find it efficiently, start from the relation $z^*G = R^* + e^*X^*$, substitute $R^*$ and $X^*$ by their decompositions in Eq. (5), then we obtain:

$$ (13) \quad z^*G = c^* \left( \alpha G + \beta X + \sum_{i} \gamma_i \Sigma_i + \sum_{i} \delta_i M_i \right)$$

\[9\]
Replacing the oracle responses by their values: $\Sigma_i h_i = 1$, $M_i = h_i G$, and substituting $X = xG$, we obtain:

\[
\varepsilon^* G = x \left( c^* \beta + c^* \sum_i \gamma_i h_i + \beta^* + \sum_i \gamma_i' h_i \right) + c^* \alpha + c^* \sum_i \delta_i h_i + \alpha^* + \sum_i \delta_i' h_i.
\]

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

\[
f^* := c^* \left( \beta + \sum_i \gamma_i h_i \right) + \beta^* + \sum_i \gamma_i' h_i
\]

... 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 hand-out by the adversary $A$ strictly before $c^*$ was sampled uniformly at random. Let us prove it on the example of C. Either A queried ($X^*, X^*, R^*$) to $H_{pop}$ before outputting ($X^*, R^*, z^*$) to the main C procedure, then $c^*$ was sampled by $H_{pop}$ just after. Or, A gave ($X^*, R^*, z^*$) to the main C procedure without having queried $H_{pop}(X^*, X^*, R^*)$ before, then C makes the query to its internal procedure $H_{pop}(X^*, X^*, R^*)$ just after, which then samples $c^*$.

**Lemma 6.** Consider, as before, the event (up to probability $q_h/p$) where \[ \text{no query } H_{pop}(X, X, R) \text{ was made before } H_{pop} \text{ was programmed as } \tilde{H}(X, X, R) \rightarrow C. \] Consider any query $H_{pop}(X, \tilde{X}, \tilde{R})$ made for the first time. Denote the decompositions of $\tilde{X}$ and $\tilde{R}$ as in (5)(6) (so we omit adding a dot above the coefficients $\alpha, \beta, \gamma, \ldots$), and the response $\tilde{c}$. Consider the indices $i = 1, \ldots, i_0$ of all queries $m_i$ to $\tilde{H}$ and $\text{SIGN}$ (responding $M_i$ and $\Sigma_i$, respectively) which were made before the query $H_{pop}(X, \tilde{X}, \tilde{R})$, i.e., before $\tilde{c}$ was sampled. Then:

$\tilde{c}$ is sampled independently from $(\alpha, \beta, (\delta_i, \gamma_i, \gamma_i', h_i))_{i \in [i_0]}$

and, for all $i > i_0$: $\delta_i = \delta_i' = \gamma_i = \gamma_i' = 0$.

The rest of the proof strategy is as follows. Let us consider the event where the adversary wins against the master reduction: $(\varepsilon^A = 1) := (C^A = 1 \wedge \beta^C = C) \vee (D^A = 1 \wedge \beta^D) = D$, where $\varepsilon^A = C$ and $\varepsilon^D = D$ denote the events where the coin was 1 or 0, i.e., where $\varepsilon^A$ behaves as C or D. We are going to consider the sub-event, denoted $\neg V \subset (\varepsilon^A = 1)$, defined as the non-vanishing of at least one of the coefficient in $f^*$ (Eq. (15)), i.e., $\neg V := \left( \beta + \sum_i \gamma_i h_i \neq 0 \right) \vee (\beta^* + \sum_i \gamma_i' h_i \neq 0)$. An immediate consequence of Lemma 6 is that, under C, in the sub-event $\neg V \wedge (C^A = 1)$, then the bad event ($f^* = 0$) (almost) never happens, and thus C (almost) always able to find the DL challenge $x = \ell$. The “almost” will be quantified later. So what remains to conclude the problem is to show that, in the complementary event $V \subset (\varepsilon^A = 1)$ which we call “very bad”, then the reduction D will (almost always) be able to find $\ell$ 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:

\[
V := \left\{ \beta X + \sum_i \gamma_i M_i = 0 \wedge \beta' X + \sum_i \gamma_i' M_i = 0 \right\}.
\]

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 $A$. A consequence is that the coin tossed by the master-reduction $\delta^C$, 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., $(\delta^C = 1)$, whatever $V$ or $\neg V$ is the most likely to happen, the master-reduction $\delta$ will have almost probability $1/2$ to extract the DL challenge $\ell$. 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 $q_h/p$ probability:

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

\[
\begin{align*}
P(DL^C = 1) &= (1 - q_h/p) \cdot P(C^A = 1 \wedge V) \\
P(DL^D = 1) &= (1 - q_h/p) \cdot P(D^A = 1 \wedge V)
\end{align*}
\]

Assuming Lemma 6, let us conclude Theorems 4. Since the master reduction $\delta^C$ behaves as C or D with probability $1/2$ each, we have $P(DL^C = 1) = 1/2 P(DL^C = 1) + 1/2 P(DL^D = 1)$. By Equations (17) and (18), it is in turn $\geq (1 - q_h/p) \left[ P(C^A = 1 \wedge V) + P(D^A = 1 \wedge V) \right]$.

Now, note that for any fixed A, we have $P(C^A = 1 \wedge V) = P(D^A = 1 \wedge V)$. Indeed if not, then an unlimited adversary $A$ could distinguish between C and D, a contraction. Substituting, we obtain: $P((DL)^B = 1) \geq 1/2 \left( 1 - q_h/p \right) P(C^A = 1)$. By the first claim of Lemma 7, since the view of $A$ against C is $q_h/p$-close to its view in SSC, we have that $P(C^A = 1) \geq \epsilon - q_h/p$. Replacing in the previous formula of $P((DL)^B = 1)$ yields the theorem.

**a) Proof of Lemma 7:** The first claim was 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 $H_{pop}(X, X, R)$ 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 $\delta^C = C$. By Lemma 6, for each query $H_{pop}(X, X, R)$ in the execution, and denoting $f$ as defined in the formula (15) (here w.r.t. $(X, X, R)$), we have $P(f = 0) = P(V) = 1/p$ 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 $\tilde{f}$ is equal to 0. In particular, for the specific $f^*$ of the winning triple, we thus have

$$\mathbb{P}(f^* = 0 | \forall V) \leq q_H/p.$$  

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 $\mathcal{E} = D$. Let us start from Eq. (13) and, since we assumed $V$, simplify by Eq. (16). Replacing $M_i = b_i, L + \tilde{h}_i, G$ and $L = \ell, G$ we obtain

$$z^*G = c^* \left( \alpha.G + \sum_{i} \delta_i (b_i, \ell.G + \tilde{h}_i, G) \right) + \alpha'.G + \sum_{i} \delta_i' (b_i, \ell.G + \tilde{h}_i, G)$$

Thus $D$ can efficiently recover $\ell$ by division by the scalar:

$$\lambda := c^* \sum_i \delta_i b_i + \sum_i \delta_i' b_i$$

... unless this scalar $\lambda$ is zero.

Let us assume that it is the case, then we cannot be in the event $W := \{ (\delta_i = 0 \land \delta_i' = 0) \forall i \land V \}$. Indeed, substituting in Eq. (5) those vanishings and those of Eq. (16), would yield $X^* = \alpha.G$, contradicting that $A$ wins. Hence, we must be in the event $\forall W \subset V$ where at least one of the coefficients $\delta_i$ or $\delta_i'$ is nonzero. To conclude, we apply the same kind of reasoning as in the proof of Eq. (17). Let us consider one query $\Pi_{\text{rep}}(X, X, R)$ in the execution. By Lemma 6, $\mathbb{P}(\tilde{c} = 0 | \forall W) = 1/p$ 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 $\tilde{c}$ is equal to 0. In particular, for the specific $c^*$ of the winning triple, we thus have

$$\mathbb{P}(c^* = 0 | \forall W) \leq q_H/p.$$  

In the event $c^* \neq 0$, we Claim that:

$$\mathbb{P}(\lambda = 0 | c^* \neq 0 | \forall W) \leq 1/p,$$

which concludes the proof. The Claim follows from the fact that, by construction, all $(b_i, \ell)_i$ are information-theoretically hidden from the adversary, hence they are independent of $(\delta_i, \delta_i')$, of which at least one is nonzero by definition of $\forall W$.

b) 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 $c^* \leftarrow \Pi(X^*, R^*, X^*)$ 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 $\delta_i$ 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 $(X^*, R^*, X^*)$ that could have been made to $\Pi_{\text{pop}}$ in order to find winning triples.

Although the proof immediately reduces to the case where $A$ 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 $G_1$ in the decompositions Equations (5) and (6) disappear (so the reduction needs not anymore program the hash-to-curve).

VII. 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, 1

1The code is available at https://anonymous.4open.science/r/MtoA-830A/.
offering a pairing of type III, for which the uncompressed size of a point in \(G_1\) is 768 bits and of a point in \(G_2\) is 1536 bits. Compressed points, i.e., their \(x\)-coordinate plus one bit, are twice smaller. Each number is the mean over 10 executions.

1) Comparing processing of the group setup runtimes: We dub “Verifier” the verification function \( Vf(KG, (pk_i)_{i \in [N]}, m, \Sigma) \rightarrow 0/1 \) 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: \( KG \); and online verification the remaining tasks performed upon learning the actual subset of signers \((pk_i)_{i \in [N]} \subset KG \) and the signed message \((m, \Sigma)\). Of course, in dynamic fNIMs such as \(MSP\)-\(pop\) \cite{[86],[22],[66]} and \(dms\), the Verifier does not take any group of public keys \(KG\) 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 \(KG\). Recall that in \(dms\) we formalized this task as the key verification function \(kVf(pk_i), \forall i \in [N] \).

In Table 6 we consider the three fNIMs which have the fastest online verification: SMSKR \cite{[9]}, \(MSP\)-\(pop\) and \(dms\). The online verification is identical in all of them, i.e., returns \(\Sigma \in G_1 \land e(\Sigma, G_2) = e(H(m), \sum_{i \in [N]} X_i)\). On the other hand, most of the runtime of the Verifiers in both \(MSP\)-\(pop\) and SSKMR (recalled in Sections D-3 and D-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 \(|KG| = 2702\) keys all-at-once. In what follows we denote \(N = |KG| = 2702\) for simplicity. This number was chosen as \(2702 = 2|v|.193\), for \(|v| = 7\). These numbers illustrate the use-case of the compiler \(MtoA\) applied to a group of \(n_C = 193\) 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 \cite{[35]} for reducing batch verification of \(N\) BLS signatures into a product of \(N\) pairings (recalled in Sec. D-4). Second, we computed this product using the optimized algorithm of gnark-crypto, inherited from \cite{[65]}. Analogously, we batched the verification of the \(N\) PoPs in \(dms\) by using the method of \cite{[14]} for batch verification of Schnorr signatures (recalled in Sec. D-5).

On the second line we consider the incremental processing of the group setup in the scenario where: there is a group \(KG\) of \(N = 2702\) keys, over which processing of the group setup was already performed, and then there are \(14\) new keys \((pk_i)_{i \in [14]}\) which join the group, in place of the old keys \((pk_i)_{i \in [14]}\). Note that this corresponds to the same use-case of \(MtoA\), for messages of variable parts \(|v| = 7\) bits, when one of the \(n_C = 193\) real group members leaves and is replaced by a new member. This results in the new group \(KG' = (KG \setminus (pk_i)_{i \in [14]}) \cup (pk_i)_{i \in [14]}\). Since the resulting group \(KG'\) is different from \(KG\), in SMSKR the Verifier needs to compute again all the \(2702\) rerandomized keys relatively to the new group \(KG'\), in addition to checking \(G_1\) membership of the \(14\) new keys. Hence, we see that the incremental processing of the group setup of SSKMR 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 \(MSP\)-\(pop\) and \(dms\), the incremental processing of the group setup only consists in verifying the PoPs of the \(14\) new keys (and the \(G_1/G_2\)-memberships).

<table>
<thead>
<tr>
<th>(\text{in ms})</th>
<th>SSKMR \cite{[9]}</th>
<th>(MSP)-(pop) \cite{[86],[22],[66]}</th>
<th>(dms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch (</td>
<td>KG</td>
<td>) keys</td>
<td>1134.9</td>
</tr>
<tr>
<td>(14) new keys</td>
<td>828.7</td>
<td>123.5</td>
<td>3.3</td>
</tr>
</tbody>
</table>

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

2) Comparison with previous provable securities: In Table 7 we state proven upper-bounds: UB\(_m\)-uf \cite{[22]}, on the probability to forge a multisignature, i.e., the advantage in the m-uf game.

- The first column states the formulas of UB\(_m\)-uf, not directly in terms of the running time \(t\) of the adversary, but instead in terms of: \(q_{\text{f}}\) the number of its random oracle requests (hash-to-curve); \(q_{\text{n}}\) the number of its signing requests; and of the upper-bounds: UB\(_B\), UB\(_co\)-cdh, UB\(_co\)-bdh and UB\(_M\)-uf \cite{[27]} on the advantage in the games of DL, co-cdh, co-bdh \cite{[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 \(+q_{\text{f}}\cdot t_{\text{exp}},\) where \(t_{\text{exp}}\) is a scalar multiplication, also known as exponentiation. In line with \cite[Thm 5]{[22]}, we corrected the bound for UB\(_M\)-pop displayed in \cite[Thm 4.1]{[86]}. In which \(q_{\text{f}}\) was replaced by the much smaller number \(q_{\text{n}}\) of signature queries. We also corrected a bug in the bounds of \cite[Thms 1–4]{[9]} (as confirmed by the authors on 26/1/2024).

- The second column states the models in which the formulas are proven: RO stands for random oracle, and “RMSS” is, for simplicity, the assumption that the “random modular subset sum” \cite[Def. 3]{[9]} is at least as hard as DL \cite[§C.3]{[9]}.

- The third column are numerical applications when assuming furthermore the AGM. Concretely, under the AGM then UB\(_3\)-bdh = UB\(_co\)-bdh = UB\(_d\) \cite{[11]}. We took \(q_{\text{f}} = 2^{80}\), the number of clock cycles \(t = 2^{30}\cdot t_{\text{exp}}\); \(q_{\text{n}} = 2^{30}\) (in line with \cite{[13]}); groups of size \(p \sim 2^{253}\), and the hardness of DL estimated as UB\(_d\) \sim t^2/p. The latter formula is shown for generic groups in \cite{[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 \cite{[71, 6]} for both the popular curves BLS12-377 (\(p \sim 2^{253}\)) used in Zexe \cite{[31]}, and BLS12-381 (\(p \sim 2^{253}\)), 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 \cite{[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 4606bits.

3) Comparing verification times of \(MtoA\) + multi-BLS: vs BGLS: In the first three lines of Table 8 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 \(\tau\) 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 $K_G$, the public keys of the subgroup of $n$ signers can be encoded as a $|K_G|$-sized array of bits. We approximated $|K_G| \approx n$, since $|K_G| = n_{\text{sec}} = (3/2)n$ 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 Sec. D-1). More precisely, we evaluated verification of the fastest variant of BGLS: $e(\Sigma, G_2) = \sum_{i=0}^{n-1} e(H(m_i), X_i)$. This variant, dubbed $\mathcal{A}S$-1 in [15], is restricted to pairwise different messages $m_i$. Hence, for a fair comparison with $\mathit{MtoA} + \mathit{dms}$, which is unrestricted and has tight security, we should have instead evaluated the costlier variant of BGLS called $\mathcal{A}S$-4 in [15]. The verification would then have taken even more time. The signature is a $G_1$ element, which has uncompressed size equal to 92 bytes = 768 bits.

Second line: $\mathit{MtoA}$ instantiated with any BLS-based fnIM with optimal online verification, i.e., either $\mathcal{MSP}$-pop, SMIKR or our $\mathit{dms}$. Namely, a $\mathit{MtoA}$ signature comes as a multisignature over $N = |\mathcal{V}|n = 7n$ keys, and its verification is as in $\mathit{dms}$ without verification of PoPs, i.e., as in [20], recalled in Sec. D-2. As in consensus protocols, we considered that the Verifier knows the tag $\tau$ in advance, and thus could precompute $H(\tau)$.

Third line: naive concatenation of $n$ Schnorr signatures. Since no pairing is necessary, we used the fastest curve secp256k1. Each signature is of the form $(R, z)$, where the $G_1$ element $R$ is now only of size 64 bytes, and the $G_2$-element $z$ is of size 32 bytes.

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

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

VIII. ACKNOWLEDGEMENTS

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

<table>
<thead>
<tr>
<th>Table 7: fNIMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
<tr>
<td>BLS aggregate sig.</td>
</tr>
<tr>
<td>$\mathit{MtoA}$ with BLS multisig.</td>
</tr>
<tr>
<td>Batch Schnorr</td>
</tr>
</tbody>
</table>

Table 8: Online verification times over $n$ messages (see above)

REFERENCES


APPENDIX A

FURTHER FORMALIZATION AND OPTIMIZATION OF MÀTA

All ideas were conveyed in Sec. IV, we now put them in the formalism of Sec. III.

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

- \( \mathbb{A.Kg}(): (sk^i_{j,b}, pk^i_{j,b}) \leftarrow \mathbb{Kg}() \forall (j \in [|v|], b \in \{0, 1\}) \); 
  \( sk^i \leftarrow (sk^i_{j,b})_{j \in [|v|], b \in \{0, 1\}}, pk^i \leftarrow (pk^i_{j,b})_{j \in [|v|], b \in \{0, 1\}} \).

- \( \mathbb{A.Ag}(): \mathbb{Sign}(\mathbb{KG}, sk^i, \tau_i, v_i): \text{decompose in bits } v_i = (v^i_j)_{j \in [|v|]} \); 
  \( \text{output } \Sigma_i \leftarrow \left( \mathbb{Sign}(\mathbb{KG}, sk^i_{j,v^i_j}, \tau_i) \right)_{j \in [|v|]} \);

- \( \mathbb{A.IVf}(\mathbb{KG}, pk^i, \tau_i, v_i, \Sigma_i): \text{decompose in bits } v_i = (v^i_j)_{j \in [|v|]} \); 
  \( \text{output } \bigwedge_{j \in [|v|]} \mathbb{IVf}(\mathbb{KG}, pk^i_{j,v^i_j}, \tau_i, \Sigma^i_j) \);

- \( \mathbb{A.Ag}(\mathbb{KG}, \tau, (pk^i_{j,v^i_j}, v^i_j)_{i \in [|n|], \Sigma^i_j}) \): 
  \( \text{decompose in bits } v_i = (v^i_j)_{j \in [|v|]} \forall i \in [|n|] \); 
  \( \text{output } \mathbb{Cb}(\mathbb{KG}, \tau, (pk^i_{j,v^i_j}, \Sigma^i_j)_{j \in [|v|], i \in [|n|]} \).

Proof: Both individual completeness and robustness follow straightforwardly from the ones of \( \mathbb{M} \), 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 \( (\mathbb{KG}, \tau^*, (pk^i_{j,v^i_j})_{i \in [|n|], \Sigma^i_j}) \), such that there exists \( (pk^i, v^i_j)_{i \in [|n|], j \in [|v|]} \) such that no query on \((\tau^*, v^*)\) was ever responded by the oracle \( \mathbb{Sign} \). By assumption, \( \mathbb{SIGN} \) responded to at most one query prefixed by \( \tau^* \). Without loss of generality, we can assume that exactly one such query was made and responded to: \( \Sigma^i \leftarrow (\tau^*, v^*) \). By the above \( v' \neq v^* \), there is a bit index \( j_0 \in [|v|] \) at which \( b_0 := (v^*)_{j_0} \neq (v')_{j_0} \), thus no signature of \( pk^i_{j_0, b_0} \) on \( \tau \) was ever delivered by \( \mathbb{SIGN} \). Since the other keys of the honest signer: \( (pk^i_{j,b})_{(j,b) \neq (j_0, b_0), \Sigma^i_j} \) are generated independently from \( pk^i_{j_0, b_0} \), the intuitive conclusion is that \( \Sigma \) constitutes a \( \mathbb{M} \)-forgery on the message \( \tau \) with respect to the signers \( (pk^i_{j,v^i_j})_{i \in [|n|], j \in [|v|]} \) and target key \( pk^i_{j_0, b_0} \).

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 \( \mathbb{M} \), the reduction must choose at random an index \( (j_0, b_0) \in [|v|] \times \{0, 1\} \), and embed its target key in \( pk^i_{j_0, b_0} \).

Thus we have at most \( 2|v| \) loss compared to the \((2|v|n)\)-unforgeability of \( \mathbb{M} \).

APPENDIX B

DETAILS FOR APPLICATIONS OF \( \mathbb{dm} \)

a) 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 \( \Sigma \). Then it multicasts \( \Sigma \) to all the participants to the consensus, dubbed the processes. Moreover, \( \Sigma \) is often meant and verified by billions of external clients, since in most cases \( \Sigma \) attests validity of a block. Since in addition \( \Sigma \) is stored on-chain, it is therefore a first-class requirement that \( \Sigma \) be both small and fast to verify. As shown in Table 8, pairing-based multisignatures schemes are the most advantageous instantiation of \( \Sigma \) with this respect, since they take close to \( N \times \) less storage space than a naive concatenation of Schnorr signatures. Furthermore, at least for \( N \geq 3073 \), they have \( 3 \times \) 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 \( \mathbb{dm} \). Still, surprisingly, a number of implementations of consensus protocols instantiate \( \mathbb{fnIMs} \) 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 \( \mathbb{IVf} \)), 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 \( \mathbb{IVf} \), which is proposed in [56, 34]. The signer, in addition to its signature: \( \sigma_i = sk_i, H(m) \), appends to it a “Chaum-Pedersen” proof: \( \pi \), of knowledge of a common discrete logarithm: \( sk_i \) between \( \sigma_i \) and the public key \( X_i := sk_i, G \). Then the verification algorithm: \( \mathbb{IVf}^{\mathbb{DLeq}} \) verifies only \( \pi \) against \( \sigma_i \) and \( X_i \), not anymore \( \sigma_i \) against \( X_i \). Our implementation of \( \mathbb{IVf}^{\mathbb{DLeq}} \), with the same configuration as in Sec. VII (same machine, gnaark-crypto, BLS-377 curve), shows a runtime of 0.785ms, down from 1.9ms for the \( \mathbb{IVf} \) of a standard BLS signature (Table 8). So this reduces the gap w.r.t. our verification time of a Schnorr signature on secp256k1, which is of 0.220ms.

b) 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: \((\Sigma_i)_i \in I\) one some message \(m\), issued by a subset \(I \subset [N]\), totaling some desired weight: \(w := \sum_{i \in I} w_i\). It outputs the public multisignature \(\Sigma := \sum_{i \in I} \Sigma_i\) and the public weight \(w\). It also outputs a proof of knowledge \(\pi\) 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) \(\epsilon(\Sigma, G_2) = \epsilon(H(m), \sum_{i \in [N]} b_i X_i)\). Note that (i) and (ii) together ensure that \(\Sigma\) passes the BLS multisignature verification \((m, \text{blsVF}^*)\) against the public keys \((X_i)_i \in [N]\), and that they totalize weight \(w\). However, the sole passing of \((m, \text{blsVF}^*)\) 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 \(\mathcal{MSP} - \text{pop}\) [86, 22], where the Verifier verifies pairing-based PoPs appended to the published keys \((X_i)_i \in [N]\). Using instead \(\text{dms}\), i.e., Schnorr-based PoPs, divides by \(> 5\times\) this latter runtime, as demonstrated in Table 6.

APPENDIX C
DETAILS FOR APPLICATION OF \(\mathcal{M}t\text{o}A\) TO CONSENSUS

In Figure 9 we further recall the consensus Diem21 [44] among \(n_p\) 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 \(\mathcal{M}t\text{o}A\) can be straightforwardly plugged: either in place of naive concatenation of \((2/3)n_p\) 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: \(r_{\text{new}} \leftarrow r\) in two ways. Either (i) upon receiving a round-\((t_{\text{new}} - 1)\) quorum certificate from the consensus protocol, which we left explicit in the protocol; or, (ii) upon receiving an abstract signal denoted \(\text{NEXTROUND}(t_{\text{new}})\) for \(t_{\text{new}} > r\), 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 \(\text{NEXTROUND}\) 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.

a) 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 ([59]) 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 \(O(n^2)\) down to \(O(n^2)\).

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 \(\{m_i\}\). 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 \(\{m\}_J\). Namely, \(\{m\}\) is a tuple consisting of \(m\), a \((2f + 1)\)-sized subset \(J \subset [n]\), and a multisignature \(\Sigma\) 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 \(P \in (\mathcal{P}_i)_i \in [n]\) keeps track of which round \(r\) it is currently in (formerly denoted \(r_{\text{sync}}\) 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 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) \(rc = \perp\) if \(qc\) is a round-\((r - 1)\)-QC, i.e., \(qc.r = r - 1\);
- $\text{txn}$ is a batch of new transactions;

Note that when describing the protocol, it suffices to specify $\text{qc}$ and $r$ for a new block, since $\text{txn}$ and $\text{id}$ follow the definitions. We will use $b.x$ to denote the element $x$ of $b$.

- **Quorum certificate (QC).** A QC: $\text{qc}$ for a block $b$ is a multi- (or threshold) signature for the message $(\text{id}, \text{br})$, produced by combining the individual signatures $\{b.i.d., b.r\}$ from any set of $2f+1$ players. The round number of a QC: $\text{qc}$ for a block $b$ is denoted by $\text{qc}.r$ which is equal to $b.r$. QCs are ranked by their round numbers, hence, we abuse notation and shorten as $\text{qc} \geq \text{qc}'$ the relation $\text{qc}.r \geq \text{qc}'.r$. 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 $b \leftrightarrow b'$ to denote a 2-chain, i.e., a block $b'$ of which the parent is the block $b$, i.e., such that $b'.q$ is a QC of $b$. Each player stores the highest quorum certificate: $\text{qc}_{\text{high}}$, which is the QC with the highest round number, which it ever received or formed. For convenience we denote $\text{r}_{\text{high}} := \text{qc}_{\text{high}}.r$ its round number.

- **New-round certificate (RC).** Jolteon & Diem21, [58, 44] make use of data structures called a Timeout Message (tm) 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 player’s $\text{qc}_{\text{high}}$ and of its individual signature on the pair $(r, \text{r}_{\text{high}})$, where we recall $\text{r}_{\text{high}} := \text{qc}_{\text{high}}.r$. A round-$r$ RC is meant to be formed out of $2f+1$ new-rounds. In the written specifications [44] (called in Figure 9), a round-$r$ RC consists of the naive concatenation of $2f+1$ signed pairs from distinct issuers: $\text{rc} \leftarrow [(r, r_j)_{j} : j \in J]$, 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 $\text{rc} \leftarrow \text{BGLS.Ag} [(r, r_j)_{j} : j \in J]$. 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 $\text{r}_{\text{max}} := \max(r_j, j \in J)$.
Instructions for each process $P_i$. Same initialization as in Diem.
We consider $A := (K_g, Sig_n, \text{IVf}, Ag, Vf)$ a one-time-tagged aggregate signature scheme, e.g., obtained from $\mathcal{MtoA}$.

**Propose** Upon entering round $r$, if $P_i$ 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: $q_{\text{high}}$. Then rc ← ⊥;
- or, receiving 2f + 1 round-r new-round messages:
  \[
  \left( (r, r_j), q_{\text{high}} \right) ; j \in J \right] 
  \]
  Then it aggregates the signatures w.r.t. the fixed tag $r$:
  \[
  rc ← \left( r, (r_j)_{j \in J}, J, A, \text{Ag} \left( r, (r_j)_{j \in J} \right) \right)
  \]
\[\triangleright \text{proof of non-supremacy}\]
Then it multicasts a block $b = [id, q_{\text{high}}, rc, r, txn]$. 

**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 $r > r_{\text{vote}}$ and $\{ (1): r = qc \cdot r + 1; \text{or} \} (2) r = rc \cdot r + 1$ and $rc = [r, (r_j)_{j \in J}, J, \Sigma]$ with $|J| = 2f + 1$ and $\Sigma$ a valid $A$-signature on the tagged messages $(r, r_j)\in J$ w.r.t. the public keys of $J$ and $qc \cdot r > \max(r_j, j \in J)$, then vote for $b$ by sending the individual signature $(id, r)$ to lead($r + 1$), and update $r_{\text{vote}} ← r$.

**Lock, Advance round, Commit** Same as in Diem21.

**New-round** Upon receiving a call $\text{NEXTROUND}(r)$ from the view-synchronization mechanism, advance the current round number $r ← r + 1$. Send to lead($r$) a new-round message:
\[
\left( (r, r_{\text{new}}), q_{\text{high}} \right)
\]
where $(r, r_{\text{new}}) ← A, \text{Sign}(sk_i, r, r, r_{\text{new}})$ is an individual signature of player $P_i$, and where we recall $r_{\text{high}} := q_{\text{high}}$. $r$.

**Figure 10:** Differences with Figure 9 are highlighted.

**APPENDIX D**

**Further details on related works**

1) $\mathcal{A}$:S:3 (15): We recall below the verification algorithm, tagged ($\mathcal{A}$:S:3, $\text{Vf}$) of the fNIA called $\mathcal{A}$:S:3 in [15]. $\mathcal{A}$:S:3 is a verifier-unrestricted variant of the seminal BGLS [23]. $\mathcal{A}$:S:3 is defined over a bilinear group $(g, G_1, G_2, G_T)$ with public generators $G_1, G_2$, and for a given hash-to-curve $H$. The $X_i$ are the public keys.

($\mathcal{A}$:S:3, $\text{Vf}$)\[
(X_i, m_i) \in [N], \Sigma \rightarrow e(\Sigma, G_2) = e(H(X_i|m_i), X_i) \wedge (\Sigma \in G_1)
\]

2) Pairing-based fNIM of Boldyreva [20]: We recall below its verification formula, tagged (m-bls$\text{Vf}^*$). Recall that this fNIM is secure only in the kock model. The formula is the same as the one of dms (Sec. V), without the checks of PoPs on public keys (the function which we called $\text{Vf}$).

(m-bls$\text{Vf}^*$) \[
(X_i)_{i \in [N]}, m \rightarrow e(\Sigma, G_2) = e(H(m), \sum_{i \in [N]} X_i) \wedge \Sigma \in G_1
\]

3) SMSKR (9): The key published by each member of the group of potential signers, consists of a raw $G_2$ element: $pk = X ← x.G_2$. Once all keys of the group have been published: $KG$, each signer re-randomizes its secret key:
\[
x ← H(KG)x.
\]

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:
\[
X ← H(KG)x. \forall X \in KG.
\]

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. (25) for each $X \in KG$.

4) $\mathcal{M}$:S$\text{P}$-pop [86, 24][22, §6], and batch verification of group setup: Each key $pk_i = (X_i, \pi_i)$ comes appended with a PoP equal to a BLS signature on $X_i$: $\Pi_i ← X_i.X_i$, where $X_i ← X_i.G_2$. Their verification cost is dominated by two pairings for each key:

\[
kVF(pk_i) → (e(\Pi_i, G_2) = e(H(X_i), X_i)) \wedge X_i \in G_2 \wedge \Pi_i \in G_1.
\]

In our benchmarks (Table 6), we first used the $2 \times$ speedup of [35] for batch verification of BLS signatures: The Verifier samples random numbers $(e_{i})_{i \in [N]} \in \mathbb{Z}_p^{[N]}$, then checks
\[
(27)\sum_{i \in [N]} e_{i}.\Pi_i, G_2 = \sum_{i \in [N]} e_{i}.H(X_i), X_i.
\]

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

5) dms: Batch verification of group setup: In our benchmarks (Table 6) we sped-up the verification of the PoPs of dms, i.e., $kVF(pk_i) \forall i \in [N]$, using the method of [14] for batch verification of Schnorr signatures. Namely: parse $(X_i, \pi_i) ← pk_i$ and $(R_i, z_i) ← \pi_i \forall i \in [N]$; $c_i ← H_{\text{pop}}(X_i, X_i, R_i)$; sample $(e_{i})_{i \in [N]} \in \mathbb{Z}_p^{[N]}$; output

\[
(28)X_i \in G_2 \forall i \in [N] \wedge \left( \sum_{i \in [N]} e_i.z_i \right)G_2 = \sum_{i \in [N]} e_i.R_i + \sum_{i \in [N]} (e_i.c_i).X_i
\]

6) SIG$_{1}$ [27]: The recent (non-dynamic) NIM called SIG$_{1}$ [27] has a processing of the group setup runtime which is one order of magnitude higher than the three previous NIMs, i.e., SMSKR, $\mathcal{M}$:S$\text{P}$-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 $\mathcal{M}$:S$\text{P}$-pop and $O(N)$ group membership tests in all three previous fNIMs. Then, computing the verification key of a group $KG$ of N keys takes N multi-additions, each with N terms.