Towards Scalable Threshold Cryptosystems

*Alin Tomescu, †Robert Chen, †Yiming Zheng,
‡Ittai Abraham, ‡§Benny Pinkas, ‡Guy Golan Gueta, *Srinivas Devadas

**MIT CSAIL*, †*MIT PRIMES & Lexington High School*, ‡*VMware Research*, §*Bar Ilan University*

***Abstract*—The resurging interest in Byzantine fault tolerant systems will demand more scalable threshold cryptosystems. Unfortunately, current systems scale poorly, requiring time quadratic in the number of participants. In this paper, we present techniques that help scale threshold signature schemes (TSS), verifiable secret sharing (VSS) and distributed key generation (DKG) protocols to hundreds of thousands of participants and beyond. First, we use efficient algorithms for evaluating polynomials at multiple points to speed up computing Lagrange coefficients when aggregating threshold signatures. As a result, we can aggregate a 130,000 out of 260,000 BLS threshold signature in just 6 seconds (down from 30 minutes). Second, we show how "authenticating" such multipoint evaluations can speed up proving polynomial evaluations, a key step in communication-efficient VSS and DKG protocols. As a result, we reduce the asymptotic (and concrete) computational complexity of VSS and DKG protocols from quadratic time to quasilinear time, at a small increase in communication complexity. For example, using our DKG protocol, we can securely generate a key for the BLS scheme above in 2.3 hours (down from 8 days). Our techniques improve performance for thresholds as small as 255 and generalize to any Lagrange-based threshold scheme, not just threshold signatures. Our work has certain limitations: we require a trusted setup, we focus on synchronous VSS and DKG protocols and we do not address the worst-case complaint overhead in DKGs. Nonetheless, we hope it will spark new interest in designing large-scale distributed systems.**

***Index Terms*—polynomial commitments, polynomial multipoint evaluation, distributed key generation, verifiable secret sharing, threshold signatures, BLS**

I. Introduction

Due to the popularity of cryptocurrencies, interest in Byzantine fault tolerant (BFT) systems has been steadily increasing [1]–[9]. At the core of BFT systems often lie simpler threshold cryptosystems such as threshold signature schemes (TSS) [10], [11], verifiable secret sharing (VSS) protocols [12]–[14] and distributed key generation (DKG) protocols [15]–[17]. For example, TSS and DKG protocols are used to scale consensus protocols [3], [5], [18]. Furthermore, DKG protocols [16] are used to securely generate keys for TSS [19], to generate nonces for interactive TSS [20], [21], and to build proactively-secure threshold cryptosystems [22], [23]. Finally, VSS is used to build multi-party computation (MPC) protocols [24], random beacons [6], [9], [25] and is the key component of DKG protocols.

Despite their usefulness, TSS, VSS and DKG protocols do not scale well in important settings. For example, BFT systems often operate in the *honest majority* setting, with n total players where $t > n/2$ players must be honest. In this setting, *t-out-of-n threshold cryptosystems*, such as TSS, VSS and DKG, require time quadratic in n [10], [12], [14], [26]. This is because of two reasons. First, reconstruction of secrets, a key step in any threshold cryptosystem, is typically implemented naively using $\Theta(t^2)$ time polynomial interpolation, even though faster algorithms exist [27]. This makes aggregating threshold signatures and reconstructing VSS or DKG secrets slow for large t. Second, either the dealing round, the verification round or the reconstruction phase in VSS and DKG protocols require $\Theta(nt)$ time. Fundamentally, this is because current polynomial commitment schemes require $\Theta(nt)$ time to either compute or verify all proofs [12], [14], [26]. In this paper, we address both of these problems.

Contributions. Our first contribution is a BLS TSS [10] with $\Theta(t \log^2 t)$ aggregation time, $\Theta(1)$ signing and verification times and $\Theta(1)$ signature size (see §III-A). In contrast, previous schemes had $\Theta(t^2)$ aggregation time (see §I-A1). We implement our fast BLS TSS in C++ and show it outperforms the naive BLS TSS as early as $n \geq 511$ and scales to n as large as 2 million (see §IV-A). At that scale, we can aggregate a signature $3000\times$ faster in 46 seconds compared to 1.5 days if done naively. Our fast BLS TSS leverages a $\Theta(t \log^2 t)$ time *fast Lagrange* interpolation algorithm [27], which outperforms the $\Theta(t^2)$ time *naive Lagrange* algorithm.

Our second contribution is a space-time trade-off for computing evaluation proofs in *KZG polynomial commitments* [14]. KZG commitments are quite powerful in that their size and the time to verify an evaluation proof are both constant and do not depend on the degree of the committed polynomial. We show how to compute n evaluation proofs on a degree t polynomial in $\Theta(n \log t)$ time. Each proof is of size $\lfloor \log t \rfloor - 1$ group elements. Previously, each proof was just one group element but computing all proofs required $\Theta(nt)$ time. Our key technique is to authenticate a polynomial multipoint evaluation at the first n roots of unity (see §II-4), obtaining an *authenticated multipoint evaluation tree (AMT)*. Importantly, similar to KZG proofs, our AMT proofs remain homomorphic (see §III-D1), which is useful when we apply them to distributed key generation (DKG) protocols.

Our third contribution is AMT VSS, a scalable VSS with a $\Theta(n \log t)$ time sharing phase, an $O(t \log^2 t + n \log t)$ time reconstruction phase, $\Theta(1)$-sized broadcast (during dealing round) and $\Theta(n \log t)$ overall communication. AMT VSS improves over previous VSS protocols which, in the worst case, incur $\Theta(nt)$ computation. However, this improvement

comes at the cost of slightly higher verification times and communication (see Table I). Nonetheless, in §IV, we show AMT VSS outperforms eVSS [14], the most communication-efficient VSS, as early as $n = 63$. Importantly, AMT VSS is highly scalable. For example, for $n \approx 2^{17}$, we reduce the best-case end-to-end time of eVSS from 2.2 days to 8 minutes.

Our fourth contribution is AMT DKG, a DKG with a $\Theta(n \log t)$ time sharing phase (except for its quadratic time complaint round), an $O(t \log^2 t + n \log t)$ time reconstruction phase, a $\Theta(1)$-sized broadcast (during dealing round) and $\Theta(n \log t)$ per-player dealing communication. AMT DKG improves over previous DKGs which, in the worst case, incur $\Omega(nt)$ computation. Once again, this improvement comes at the cost of slightly higher verification times and communication (see Table I). Nonetheless, in §IV, we show AMT DKG outperforms eJF-DKG [17], the most communication-efficient DKG, as early as $n = 63$. For $n \approx 2^{17}$, we reduce the best-case end-to-end time of eJF-DKG from 2.4 days to 4 minutes.

Our last contribution is an open-source implementation:

`https://github.com/alinush/libpolycrypto`

Limitations. Our work only addresses TSS, VSS and DKG protocols secure against *static* adversaries. However, *adaptive security* can be obtained, albeit with some overheads [26], [28]–[31]. We only target *synchronous* VSS and DKG protocols, which make strong assumptions about the delivery of messages. However, recent work [32] shows how to instantiate such protocols using the Ethereum blockchain [2]. Our VSS and DKG protocols require a *trusted setup* (see §V-1). Our evaluation only measures the computation in VSS and DKG protocols and does not measure network delays that would arise in a full implementation on a real network. Our techniques slightly increase the communication overhead of VSS and DKG protocols from $\Theta(n)$ to $\Theta(n \log t)$. However, when accounting for the time savings, the extra communication is worth it. Still, we acknowledge communication is more expensive than computation in some settings. Finally, we do not address the worst-case quadratic overhead of complaints in DKG protocols. We leave scaling this to future work.

A. Related Work

1) Threshold signature schemes (TSS): Threshold signatures and threshold encryption were first conceptualized by Desmedt [33]. Since then, many threshold signatures based on *Shamir secret sharing* (see §II-C) have been proposed [**?**], [10], [11], [20], [21], [34]–[37]. To the best of our knowledge, none of these schemes addressed the $\Theta(t^2)$ time required for polynomial interpolation. Furthermore, all current BLS TSS [10] implementations seem to use this quadratic algorithm [3], [38]–[40] and thus do not scale to large t. In contrast, our work uses $\Theta(t \log^2 t)$ fast Lagrange interpolation and scales to $t = 2^{20}$ (see §III-A).

An alternative to a TSS is a *multi-signature scheme (MSS)*. Unlike a TSS, an MSS does not have a unique, constant-sized public key (PK) against which all final signatures can be verified. Instead, the PK is dynamically computed given the contributing signers' IDs and their public keys. This means that a t-out-of-n MSS must include the t signer IDs as part of the signature, which makes it $\Omega(t)$-sized. Furthermore, MSS verifiers must have all signers' PKs, which are of $\Omega(n)$ size. To fix this, the PKs can be Merkle-hashed but this now requires including the PKs and their Merkle proofs as part of the MSS [41]. On the other hand, an MSS is much faster to aggregate than a TSS. Still, due to its $\Omega(t)$ size, an MSS does not always scale.

2) Verifiable secret sharing (VSS): VSS protocols were introduced by Chor et al. [13]. Feldman proposed the first efficient, non-interactive VSS with computational hiding and information-theoretic binding [26]. Pedersen introduced its counterpart with information-theoretic hiding and computational binding [12]. Both schemes require a $\Theta(t)$-sized broadcast during dealing. Kate et al.'s eVSS reduced this to $\Theta(1)$ using constant-sized polynomial commitments [14]. eVSS also reduced the verification round time from $\Theta(t)$ to $\Theta(1)$. However, eVSS's $\Theta(nt)$ dealing time scales poorly when $t \approx n$. Our work improves eVSS to $\Theta(n \log t)$ dealing time at the cost of $\Theta(\log t)$ verification round time. We also increase communication from $\Theta(n)$ to $\Theta(n \log t)$ (see Table I).

3) Publicly verifiable secret sharing (PVSS): Stadler proposed publicly verifiable secret sharing (PVSS) protocols [42] where any *external verifier* can verify the VSS protocol execution. As a result, PVSS is less concerned with players individually and efficiently verifying their shares, instead enabling external verifiers to verify all players' (encrypted) shares. Schoenmakers proposed an efficient (t, n) PVSS protocol [43] where dealing is $\Theta(n \log n)$ time and external verification of all shares is $\Theta(nt)$ time, later improved to $\Theta(n)$ time by Cascudo and David [25]. Unfortunately, when the dealer is malicious, PVSS still needs $\Theta(nt)$ computation during reconstruction. Furthermore, PVSS might not be a good fit in protocols with a large number of players. In this setting, it might be better to base security on a *large*, threshold number of honest players who individually and efficiently verify their own share rather than on a *small* number of external verifiers who must each do $\Omega(n)$ work. Indeed, recent work explores the use of VSS within BFT protocols *without* external verifiers [44]. Nonetheless, our AMT VSS protocol can be easily modified into a PVSS since an AMT for all n proofs can be batch-verified in $\Theta(n)$ time (see §III-C3).

4) Distributed key generation (DKG): DKG protocols were introduced by Ingemarsson and Simmons [45] and subsequently improved by Pedersen [12], [15]. Gennaro et al. [16] noticed that if players in Pedersen's DKG refuse to deal [15], they cannot be provably blamed and fixed this in their new JF-DKG protocol. They also showed that secrets produced by Pedersen's DKG can be *biased*, and fixed this in their New-DKG protocol. Neji et al. gave a more efficient way of debiasing Pedersen's DKG [46]. Gennaro et al. also introduced the first "fast-track" or optimistic DKG [24]. Canetti et al. modified New-DKG into an adaptively-secure DKG [28]. So far, all DKGs required a $\Theta(t)$-sized broadcast by each player.

Kate's eJF-DKG [17] reduced the dealer's broadcast to $\Theta(1)$

TABLE I
PER-PLAYER WORST-CASE ASYMPTOTIC COMPLEXITY OF (t, n) VSS/DKG PROTOCOLS.

Scheme	Dealing round time	Verification round time	Complaint round time	Reconstr. time (no interpol.)	Dealing commun. (broadcast)	Dealing commun. (private)
Feldman VSS [26]	$n \log n$	t	t^2	nt	t	n
JF-DKG [16]	$n \log n$	nt	t^3	nt	t	n
eVSS [14]	nt	1	t	n	1	n
eJF-DKG [17]	nt	n	t^2	n	1	n
AMT VSS	$n \log t$	$\log t$	$t \log t$	$n \log t$	1	$n \log t$
AMT DKG	$n \log t$	$n \log t$	$t^2 \log t$	$n \log t$	1	$n \log t$

via constant-sized polynomial commitments [14]. eJF-DKG also reduced verification time from $\Theta(nt)$ per player to $\Theta(n)$ but at the cost of $\Theta(nt)$ dealing time per player. All DKGs so far require $\Theta(nt)$ computation per player (in the worst case), while our AMT DKG requires $\Theta(n \log t)$. Furthermore, these DKGs assume a synchronous communication model between players, which can be difficult to instantiate. Recently, ETHDKG [32] surpasses this difficulty using Ethereum [2]. Kate et al. introduced *asynchronous* DKG protocols [17], [19] based on bivariate polynomials. We have not investigated if our techniques apply there.

5) Polylogarithmic DKG: Canny and Sorkin present a polylogarithmic time DKG [47], a beautiful result that unfortunately has limitations. In certain settings, their protocol only requires $\Theta(\log^3 n)$ computation and communication per player. The key idea is that each player only talks to a *group* of $\log n$ other players, leading to a $\Theta(\log^3 n)$ per-player complexity. Unfortunately, their protocol centralizes trust in a dealer who must "permute" the players before the protocol starts. The authors argue the dealer can be distributed amongst the players, but it is unclear how to do so securely while maintaining the $\Theta(\log^3 n)$ per player complexity.

Furthermore, their protocol does not efficiently support all thresholds (t, n). Instead, it only supports $((1/2 + \varepsilon)n, n)$ thresholds and tolerates $(1/2 - \varepsilon)n$ failures, where $\varepsilon \in (0, 1/2)$. Thus, their protocol can tolerate more failures only if ε is made very small. Unfortunately, a smaller ε causes the group size to increase, driving up the per-player complexity (see Appendix D). As a result, their protocol only scales in settings where a small fraction of failures is tolerated (e.g., 1/5) and a larger fraction of players is required to reconstruct (e.g., 4/5). Nonetheless, for their protocol to be truly distributed, the trusted dealer must be eliminated as a single point of failure.

6) DKG implementations: Finally, the increasing popularity of BLS threshold signatures [10] has led to several DKG implementations. For example, recent works implement a DKG on top of the Ethereum blockchain [32], [48], [49]. Cryptocurrency companies such as DFINITY and GNOSIS implement a DKG as well [50], [51]. Finally, Distributed Privacy Guard (DKGPG) [52] implements a DKG for ElGamal threshold encryption [53] and for DSS threshold signatures [28]. All current implementations are based on Feldman [26] or Pedersen commitments [15] and require $\Theta(nt)$ time per player.

II. PRELIMINARIES

In this section we introduce some notation, our cryptographic assumptions and the communication and adversarial model for the distributed protocols in this paper. Then, we give background on TSS, polynomial commitments, VSS, DKG and polynomial multipoint evaluations.

1) Notation: Let $\mathbb{F}_p$ denote the finite field "in the exponent" associated with a group $\mathbb{G}$ of prime order p with generator g. We use multiplicative notation for all algebraic groups in this paper. Let $1_{\mathbb{G}}$ denote the identity element of a group $\mathbb{G}$. Let $s \in_R S$ denote sampling an element s uniformly at random from some set S. Let $\log x$ be shorthand for $\log_2 x$. Let $[i, j] = \{i, i+1, \ldots, j-1, j\}$ and $[n] = [1, n]$ and. Let $\deg \phi$ denote the degree of a polynomial ϕ. We say a polynomial ϕ has *degree-bound* m if $\deg \phi < m$. Let $\mathsf{PP}_q(g; \tau) = \langle g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^q} \rangle$ denote public parameters used in the q-SBDH assumption (see Appendix B).

2) Cryptographic assumptions: Our work relies on the use of *pairings* or *bilinear maps*, first introduced by Menezes et al. [54]. Recall that a bilinear map $e(\cdot, \cdot) : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ has useful algebraic properties: $e(g^a, g^b) = e(g^a, g)^b = e(g, g^b)^a = e(g, g)^{ab}$. To simplify exposition, throughout the paper we assume symmetric (Type I) pairings, but our results can be re-stated in the setting of the more efficient asymmetric (Type II and III) pairings in a straightforward manner. Our schemes from §§III-C and III-D rely on the q-SBDH [55] and q-polyDH [14] assumptions, both defined in Appendix B.

3) Communication and adversarial model: DKG and VSS protocols assume a *broadcast channel* for all actors to reliably communicate with each other [13], [15]. (In practice, this can be implemented using BFT protocols [32].) In addition, some protocols need *private and authenticated channels* between actors [14]–[17], [26]. We focus on *synchronous* VSS and DKG protocols, where parties communicate in *rounds*. Within a round, each party performs some computation, (possibly) sends private messages to other players and broadcasts a message to everybody. By the end of the round, each party receives all messages sent in that round by other players (whether privately or via broadcast). We assume computationally-bounded adversaries $\mathcal{A}$ that control up to $t - 1$ players. We restrict ourselves to *static* $\mathcal{A}$'s who fix the set of $< t$ corrupted players before the protocol starts. We assume $\mathcal{A}$ can be *rushing* and can wait to hear all messages from all honest players in a round before privately sending or broadcasting his own message

within that same round. The protocols in this paper are *robust*: there are always t honest players who can reconstruct the secret. In the synchronous setting, robustness holds for all $t-1 < n/2$ [16].

4) FFT and Lagrange interpolation: We use the *Fast Fourier Transform (FFT)* to multiply and divide polynomials in $\mathbb{F}_p[X]$ of degree-bound $N = 2^k$ in $\Theta(N \log N)$ time [56], [57]. For this, we need a primitive Nth *root of unity* in $\mathbb{F}_p$, which we denote by ω_N [56]. Finally, given $(x_i, y_i = \phi(x_i))_{i\in[n]}$, interpolating ϕ takes $\Theta(n\log^2 n)$ time using *fast Lagrange interpolation* [27]. Specifically, recall that $\phi(x) = \sum_{i\in[n]} \mathcal{L}_i^{[n]}(x) y_i$ where $\mathcal{L}_i^{[n]}(x) = \prod_{j\in[n] j\neq i} \frac{x-x_j}{x_i-x_j}$ is called a *Lagrange polynomial* [58]. Note that $\mathcal{L}_i^{[n]}$ is defined with respect to the set of points $\{x_i\}_{i\in[n]}$. Throughout this paper, this set will typically be $\{x_i\}_{i\in T}$ where $T \subset [n]$, with either $x_i = i$ or $x_i = \omega_N^{i-1}$ and the Lagrange polynomial will be denoted $\mathcal{L}_i^T(x)$.

A. Threshold Signature Schemes (TSS)

A (t, n)-threshold signature scheme (TSS) is a protocol amongst n *signers* where *only* subsets of size $\geq t$ can produce a *digital signature* [59] on a message m. Many signature schemes can be turned into a TSS, such as RSA [11], [59], Schnorr [20], [60], [61], ElGamal [35]–[37], [62], ECDSA [21] and BLS [10], [63]. In this paper, we focus on the BLS TSS because of its simplicity and efficiency.

1) (Threshold) BLS signatures: A normal BLS signature on a message $m \in \{0,1\}^*$ is $\sigma = H(m)^s$ where $s \in_R \mathbb{F}_p$ is the *secret key* and $H : \{0,1\}^* \to \mathbb{G}$ is a hash function modeled as a random oracle. To verify the signature against the *public key* g^s, a bilinear map e is used to ensure that $e(H(m), g^s) \stackrel{?}{=} e(\sigma, g) \Leftrightarrow e(H(m), g)^s \stackrel{?}{=} e(H(m)^s, g)$.

To obtain a (t, n) BLS TSS [10], the secret key s is split amongst the n signers using (t, n) Shamir secret sharing (see §II-C). Specifically, each signer i has a *secret key share* s_i of s along with a *verification key* g^{s_i}. To produce a signature on m, each i computes a *signature share* $\sigma_i = H(m)^{s_i}$. Then, all σ_i's are sent to an *aggregator* (e.g., one of the signers). Since some signers are malicious, their σ_i might not be valid. Thus, the aggregator verifies each σ_i by checking if $e(g^{s_i}, H(m)) \stackrel{?}{=} e(\sigma_i, g)$. (This works because σ_i is a normal BLS signature that should verify under g^{s_i}.) This way, the aggregator finds a subset T of t signers who produced a valid signature share σ_i. Now, the aggregator can compute the final signature as $\sigma = \prod_{i\in T} \sigma_i^{\mathcal{L}_i^T(0)} = H(m)^{\sum_{i\in T} s_i \mathcal{L}_i^T(0)} = H(m)^s$ via Lagrange interpolation (see §II-4). Importantly, aggregation never exposes the secret key s, which is interpolated "in the exponent." The time to *aggregate* the signature is $\Theta(t^2)$, dominated by the time to (naively) compute the $\mathcal{L}_i^T(0)$'s.

B. Constant-sized Polynomial Commitments

Kate, Zaverucha and Goldberg introduced constant-sized polynomial commitments, often called *KZG commitments* [14]. Their scheme requires ℓ-SDH [64] public parameters $\mathsf{PP}_\ell(g;\tau) = (g^{\tau^i})_{i\in[0,\ell]}$ where τ denotes a trapdoor. (These parameters are computed via a *trusted setup*; see §V-1.) Their scheme is *computationally-hiding* (see Definition A.5) under the discrete log assumption and *computationally-binding* [14] under ℓ-SDH. Unlike Pedersen commitments [12], KZG can only commit to polynomials of maximum degree ℓ.

Let ϕ denote a polynomial of degree $d \leq \ell$ with coefficients $c_0, c_1, \ldots, c_d$ in $\mathbb{F}_p$. A KZG commitment to ϕ is a single group element $C = \prod_{i=0}^d \left(g^{\tau^i}\right)^{c_i} = g^{\sum_{i=0}^d c_i \tau^i} = g^{\phi(\tau)}$. Note that committing to ϕ takes $\Theta(d)$ time. To compute an *evaluation proof* that $\phi(a) = y$, KZG leverages the polynomial remainder theorem, which says:

$$\phi(a) = y \Leftrightarrow \exists q, \phi(x) - y = q(x)(x-a) \tag{1}$$

The proof is just a KZG commitment to q: a single group element $\pi = g^{q(\tau)}$. Computing the proof takes $\Theta(d)$ time. To verify π, one checks (in constant time) if $e(C/g^y, g) = e(\pi, g^\tau/g^a) \Leftrightarrow e(g,g)^{\phi(\tau)-y} = e(g,g)^{q(\tau)(\tau-a)}$.

1) Batch proofs and homomorphism: Given a set of points S and their evaluations $\{\phi(i)\}_{i\in S}$, KZG can prove all evaluations with *one* constant-sized *batch proof* rather than $|S|$ individual proofs [14]. The prover computes an *accumulator polynomial* $a(x) = \prod_{i\in S}(x-i)$ in $\Theta(|S|\log^2|S|)$ time and computes ϕ/a in $\Theta(d \log d)$ time, obtaining a quotient q and remainder r. The batch proof is $\pi = g^{q(\tau)}$. To verify π against $\{\phi(i)\}_{i\in S}$ and C, the verifier first computes a from S and interpolates r such that $r(i) = \phi(i), \forall i \in S$ in $\Theta(|S|\log^2|S|)$ time. Next, he computes $g^{a(\tau)}$ and $g^{r(\tau)}$ commitments. Finally, he checks if $e(C/g^{r(\tau)}, g) = e(g^{q(\tau)}, g^{a(\tau)})$. We stress that batch proofs are only useful when $|S| \leq d$. Otherwise, if $|S| > d$, we can interpolate ϕ directly from the evaluations, which makes verifying any evaluation trivial.

Finally, KZG proofs have a *homomorphic* property. Suppose we have two polynomials ϕ_1, ϕ_2 with commitments C_1, C_2 and two proofs π_1, π_2 for $\phi_1(a)$ and $\phi_2(a)$, respectively. Then, a commitment C to the sum polynomial $\phi = \phi_1 + \phi_2$ can be computed as $C = C_1 C_2 = g^{\phi_1(\tau)} g^{\phi_2(\tau)} = g^{\phi_1(\tau)+\phi_2(\tau)} = g^{(\phi_1+\phi_2)(\tau)}$. Even better, a proof π for $\phi(a)$ w.r.t. C can be aggregated as $\pi = \pi_1\pi_2$. This homomorphism is necessary in KZG-based protocols such as eJF-DKG (see §II-D).

C. (Verifiable) Secret Sharing

A (t, n) *secret sharing* scheme allows a *dealer* to split up a secret s amongst n *players* such that *only* subsets of size $\geq t$ players can reconstruct s. Secret sharing schemes were introduced independently by Shamir [65] and Blakley [66]. *Shamir's secret sharing (SSS)* is split into two phases. In the *sharing phase*, the dealer picks a degree $t-1$, random, univariate polynomial ϕ, lets $s = \phi(0)$ and distributes a *share* $s_i = \phi(i)$ to each player $i \in [n]$. In the *reconstruction phase*, any subset $T \subset [n]$ of t honest players can reconstruct s by sending their shares to a *reconstructor*. For each $i \in T$, the reconstructor computes a *Lagrange coefficient* $\mathcal{L}_i^T(0) = \prod_{j\in T, j\neq i} \frac{0-j}{i-j}$. Then, he computes the secret as $s = \phi(0) = \sum_{i\in T} \mathcal{L}_i^T(0) s_i$ (see §II-4).

Algorithm 1 eVSS: A synchronous (t, n) VSS

Sharing Phase

Dealing round:

1) The dealer picks $\phi \in_R \mathbb{F}_p[X]$ of degree $t-1$ with $s = \phi(0)$, computes all shares $s_i = \phi(i)$, and commits to ϕ as $c = g^{\phi(\tau)}$.
2) Computes KZG proofs $\pi_i = g^{q_i(\tau)}$, $q_i(x) = \frac{\phi(x)-\phi(i)}{x-i}$, $\forall i \in [n]$.
3) *Broadcasts* c to all players. Then, sends (s_i, π_i) to each player $i \in [n]$ over an *authenticated, private* channel.

Verification round:

1) Each player $i \in [n]$ verifies π_i against c by checking if $e(c/g^{s_i}, g) = e(\pi_i, g^{\tau - i})$. If this check fails (or i received nothing from dealer), then i broadcasts a *complaint* against the dealer.

Complaint round:

1) If the size of the set S of complaining players is $\geq t$, the dealer is *disqualified*. Otherwise, the dealer reveals the correct shares with proofs by broadcasting $\{s_i, \pi_i\}_{i \in S}$.
2) If any one proof does not verify (or dealer did not broadcast), the dealer is disqualified. Otherwise, each $i \in [n]$ now has his correct share s_i.

Reconstruction Phase

Given commitment c and shares $(i, s_i, \pi_i)_{i \in T}$, $|T| \geq t$, the *reconstructor*:

1) Verifies each s_i, identifying a subset V of t players with valid shares.
2) Interpolates $s = \sum_{i \in V} \mathcal{L}_i^V(0) s_i = \phi(0)$.

Algorithm 2 eJF-DKG's Sharing Phase

Dealing round: Each player i:

1) Picks $f_i \in_R \mathbb{F}_p[X]$ of degree $t-1$, sets $z_i = f_i(0)$ and $c_i = g^{f_i(\tau)}$.
2) Computes $g^{z_i} = g^{f_i(0)}$, a KZG proof $\pi_{i,0}$ for $f_i(0)$ and a NIZKPoK π_i^{DLog} for $g^{f_i(0)}$ and *broadcasts* $(c_i, g^{z_i}, \pi_{i,0}, \pi_i^{\mathsf{DLog}})$.
3) Computes shares $s_{i,j} = f_i(j)$ and KZG proofs $\pi_{i,j}$ and sends $(s_{i,j}, \pi_{i,j})$ to each $j \in [n]$ over an *authenticated, private* channel.

Verification round: For each $(c_i, g^{z_i}, \pi_{i,0}, \pi_i^{\mathsf{DLog}}, s_{i,j}, \pi_{i,j})$ from i, each j:

1) Verifies $\pi_{i,0}$ by checking $e(c_i/g^{z_i}, g) = e(\pi_{i,0}, g^{\tau - 0})$ and verifies the π_i^{DLog} NIZKPoK.
2) Verifies its share $s_{i,j}$ using $e(c_i/g^{s_{i,j}}, g) = e(\pi_{i,j}, g^{\tau - j})$.
3) If any of these checks fail (or nothing was received from i), then j broadcasts a complaint against i.

Complaint round:

1) Let S_i be the set of players complaining against i. If $|S_i| \geq t$, then i is marked as *disqualified* by all honest players. Otherwise, i broadcasts $\{s_{i,j}, \pi_{i,j}\}_{j \in S_i}$.
2) If any one proof does not verify (or i did not broadcast), then i is disqualified. Otherwise, each $j \in S_i$ now has his correct share $s_{i,j}$.
3) Let Q denote the set of players that were *not* disqualified. The agreed-upon (unknown) secret key $s = \sum_{j \in Q} z_j$. Each i sets $c = \prod_{j \in Q} c_j$, sets the *public key* $g^s = \prod_{j \in Q} g^{z_j}$, sets his share $s_i = \sum_{j \in Q} s_{j,i}$, and sets his KZG proof $\pi_i = \prod_{j \in Q} \pi_{j,i}$.

Unfortunately, SSS does not tolerate malicious dealers who distribute invalid shares, nor malicious players who might send invalid shares during reconstruction. To deal with this, *Verifiable Secret Sharing (VSS)* protocols enable players to verify shares from a potentially-malicious dealer [12]–[14], [26]. Furthermore, VSS also enables the reconstructor to verify the shares before interpolating the (wrong) secret. Loosely speaking, VSS protocols must offer two properties against any adversary who compromises the dealer and $< t$ players: *secrecy* and *correctness*. Secrecy guarantees that no adversary learns the secret s when the dealer is honest, since a malicious one can simply reveal s. Correctness guarantees that, after the sharing phase, either any set of $\geq t$ honest players can always reconstruct s or the dealer is *disqualified*. We refer the reader to [14] for more formal VSS definitions.

1) Kate et al.'s eVSS*:* At a high-level, eVSS follows the style of previous VSS protocols [12], [26]. In the *dealing round*, the dealer commits to ϕ and sends each player their share and *proof* that their share is correct. In the *verification round*, each player verifies the proof for his share and, if incorrect, broadcasts a *complaint*. Finally, in the *complaint round*, the dealer resolves complaints (if any) by broadcasting the correct share of each complaining player. We give a detailed description of eVSS in Algorithm 1 and its asymptotic complexity in Table I.

From Algorithm 1, eVSS's *overall communication* complexity is $\Theta(n)$ (since at most $2n + (t-1)$ shares and proofs are sent while dealing, complaining and reconstructing). eVSS's reconstruction phase is $O(t \log^2 t + n)$ time, since at most n shares have to be verified before the secret can be interpolated fast in $\Theta(t \log^2 t)$ time [27]. eVSS's dealing round is $\Theta(nt)$ time, since n KZG proofs must be computed. The verification round is $\Theta(1)$ time (per player). If S is the set of complaining players, the complaint round takes $\Theta(|S|)$ time and communication for the dealer to send $|S|$ shares with proofs and $\Theta(|S|)$ time for each player to verify them.

D. Distributed Key Generation (DKG)

TSS protocols pose a key generation problem: if one party P splits s to the n signers (via SSS), P would know s and could sign on behalf of the group. This would make the TSS insecure and thus motivates *distributed key generation* (DKG) protocols [15], [16]. A (t, n) DKG protocol for discrete log cryptosystems allows n players to jointly generate a secret key $s \in_R \mathbb{F}_p$ with public key $g^s \in \mathbb{G}$ such that *only* subsets of size $\geq t$ can reconstruct s.

Unlike VSS protocols, where the dealer knows s (see §II-C), DKG protocols guarantee nobody learns s during the execution of the protocol. Typically, DKG protocols achieve this by having each player i secret-share its own secret z_i and setting the *final secret* s to be $\sum_i z_i$. Similar to VSS, DKG protocols must offer two security properties against any adversary who compromises $< t$ players: *secrecy* and *correctness*. Informally, secrecy guarantees that no adversary can learn any information about s beyond what is leaked by g^s. Correctness guarantees that all honest players agree on g^s and any subset with $\geq t$ honest players can reconstruct s.

1) Kate's eJF-DKG*:* In this paper, we focus on improving eJF-DKG which, at a high-level, consists of n parallel executions of eVSS. We describe only its sharing phase in Algorithm 2, since it has the same reconstruction phase as eVSS. Note that eJF-DKG makes use of *non-interactive zero-knowledge proofs of knowledge (NIZKPoKs)* [67]. Although eJF-DKG is *biasable* and produces an s that is not guaranteed to be uniform, computing discrete logs on g^s is still hard [17], [61]. Also, debiasing DKG protocols is possible [16], [32], [46].

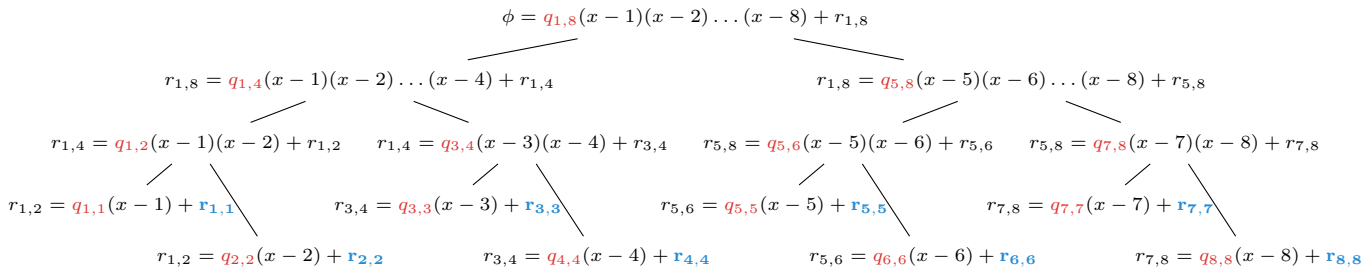


Fig. 1. A multipoint evaluation of polynomial ϕ at points $[8] = \{1, 2, \dots, 8\}$. Each node is expressed as $a = q \cdot b + r$: i.e., a polynomial a is being divided by b, resulting in a *quotient* q and a *remainder* r. In the root node, ϕ is divided by the root *accumulator* $\prod_{i \in [8]}(x - i)$, obtaining a quotient $q_{1,8}$ and a remainder $r_{1,8}$. Then, the root's left child divides $r_{1,8}$ by $(x-1)\cdots(x-4)$ while the right child divides it by $(x-5)\cdots(x-8)$. The process is repeated recursively on the resulting $r_{1,4}$ and $r_{5,8}$ remainders. The remainders $r_{i,i}$ in the leaves are the evaluations $\phi(i)$.

E. Polynomial Multipoint Evaluation

We build upon *polynomial multipoint evaluation* techniques [27]. Given a degree t polynomial ϕ, naively evaluating it at $n > t$ points $x_1, \dots, x_n$ requires $\Theta(nt)$ time. This is fast when t is very small relative to n but can be slow when $t \approx n$, as is the case in many instantiations of threshold cryptosystems. Fortunately, a multipoint evaluation reduces this time to $O(n \log^2 n)$ using a divide and conquer approach. Specifically, one first computes $\phi_L(x) = \phi(x) \bmod (x - x_1)(x - x_2) \cdots (x - x_{n/2})$ and then $\phi_R(x) = \phi(x) \bmod (x - x_{n/2+1})(x - x_{n/2+2}) \cdots (x - x_n)$ Then, one simply recurses on the two *half-sized* subproblems: evaluating $\phi_L(x)$ at $x_1, x_2, \dots, x_{n/2}$ and $\phi_R(x)$ at $x_{n/2+1}, x_{n/2+2}, \dots x_n$. Ultimately, the leaves of this recursive computation store $\phi(x) \bmod (x - x_i)$, which is exactly $\phi(i)$ by the polynomial remainder theorem (see Figure 1).

For example, consider the multipoint evaluation of ϕ at $\{1, 2, \dots, 8\}$, which we depict in Figure 1. We start at the root node ε. Here, we divide ϕ by the *accumulator polynomial* $(x-1)(x-2)\dots(x-8)$ obtaining a *quotient polynomial* $q_{1,8}$ and *remainder polynomial* $r_{1,8}$. Then, its left and right children divide $r_{1,8}$ by the left and right "half" of $(x-1)(x-2)\dots(x-8)$, respectively. This proceeds recursively: each node w divides $r_{\mathsf{parent}(w)}$ by its accumulator a_w, obtaining a quotient q_w and remainder r_w such that $r_{\mathsf{parent}(w)} = q_w a_w + r_w$. Note that all accumulator polynomials a_w can be computed in $O(n \log^2 n)$ time by starting with the $(x - i)$ monomials as leaves of a binary tree and "multiplying up the tree." Since division by a degree-bound n accumulator takes $O(n \log n)$ time, the total time is $T(n) = 2T(n/2) + O(n \log n) = O(n \log^2 n)$ [27].

III. Scalable Threshold Cryptosystems

First, we show how to speed up and scale threshold signature aggregation as well as secret reconstruction in any Lagrange-based threshold cryptosystem (see §III-A). Then, we introduce *authenticated multipoint evaluation trees* (AMTs), a new technique for precomputing logarithmic-sized evaluation proofs much faster in KZG commitments (see §III-B). Last, we use AMTs to speed up and scale Kate et al.'s eVSS and Kate's eJF-DKG (see §§III-C and III-D).

A. Scalable Threshold Signatures

In this section, we show how to reduce the time to aggregate a (t, n) BLS threshold signature from $\Theta(t^2)$ to $\Theta(t \log^2 t)$. Although we focus on BLS, our techniques can be used in any threshold cryptosystem (not just signatures) whose secret key lies in a prime-order field $\mathbb{F}_p$. This includes ElGamal signatures [35]–[37], ElGamal encryption [53] and Schnorr signatures [20], [61] (but not RSA-based schemes, whose secret key does not lie in a prime-order field [11]).

Recall from §II-A that BLS TSS aggregation has two phases: (1) computing Lagrange coefficients and (2) exponentiating signature shares by these coefficients. Unfortunately, as t gets large, naively computing Lagrange coefficients in $\Theta(t^2)$ time dominates exponentiating the shares (see Figure 2a). In fact, current descriptions and implementations of threshold schemes all seem to use this inefficient scheme, which we dub *naive Lagrange* [10], [38]–[40], [68]. We make three contributions. First, we adapt the *fast polynomial interpolation* from [27] to compute just the Lagrange coefficients $\mathcal{L}_i^T(0)$ fast in $\Theta(t \log^2 t)$ time. We call this scheme *fast Lagrange*. Second, we speed up this scheme by using roots of unity rather than $\{1, 2, \dots, n\}$ as the signer IDs. Third, we implement a BLS TSS based on fast Lagrange and show it outperforms the naive one as early as $n = 511$ (see §IV-A).

1) Fast Lagrange-based BLS: Recall from §II-4 that a *Lagrange polynomial* $\mathcal{L}_i^T(x)$ is defined as $\mathcal{L}_i^T(x) = \prod_{\substack{j \in T \\ j \neq i}} \frac{x-j}{i-j}$. Let us define $N(x) = \prod_{i \in T}(x - i)$. Then, let $N_i(x) = \frac{N(x)}{x-i} = \prod_{j \in T, j \neq i}(x - j)$ be the numerator and let $D_i = N_i(i) = \prod_{j \in T, j \neq i}(i - j)$ be the denominator. Now, we can rewrite $\mathcal{L}_i^T(x) = \frac{N_i(x)}{D_i}$.

Our goal is to quickly compute $\mathcal{L}_i^T(0)$ for each signer ID $i \in T$. In other words, we need to quickly compute all $N_i(0)$'s and all D_i's. First, given the set of signer IDs T, we interpolate $N(x)$ in $\Theta(t \log^2 t)$ time by starting with the $(x - i)$'s as leaves of a tree and "multiplying up the tree." Second, we can compute all $N_i(0) = N(0)/(-i)$ in $\Theta(t)$ time. (Note that $N(0)$ is just the first coefficient of $N(x)$.) However, computing $D_i, \forall i \in T$ appears to require $\Theta(t^2)$ time. Fortunately, the *derivative* $N'(x)$ of $N(x)$ evaluated at i is exactly equal to

D_i [27]. Thus, a $\Theta(t \log^2 t)$ multipoint evaluation of $N'(x)$ at all $i \in T$ can efficiently compute all D_i's!

To see why $N'(i) = D_i$, it is useful to look at the closed form formula for $N'(x)$ obtained by applying the product rule of differentiation (i.e., $(fg)' = f'g + fg'$). For example, for $N(x) = (x-1)(x-2)(x-3)$:

$$\begin{aligned} N'(x) &= (x-2)(x-3) + (x-1)(x-3) + (x-1)(x-2) \\ &= N_1(x) + N_2(x) + N_3(x) \end{aligned}$$

In general, we can prove that $N'(x) = \sum_{i \in T} N_i(x)$, where $\deg N' = t - 1$. Since $N_j(i) = 0$ for all $i \neq j$, it follows that $N'(i) = N_i(i) + 0 = D_i$. Lastly, computing $N'(x)$ only takes $\Theta(t)$ time via polynomial differentiation. (i.e., $N = (c_t, c_{t-1}, \ldots, c_1, c_0) \Rightarrow N' = (t \cdot c_t, (t-1)c_{t-1}, \ldots, 2c_2, c_1)$)

To summarize, given a set T of signer IDs, we can compute the Lagrange coefficients $\mathcal{L}_i^T(0) = N_i(x)/N'(i)$ by (1) computing $N(x)$ in $\Theta(t \log^2 t)$ time, (2) computing all $N_i(0)$'s in $\Theta(t)$ time, (3) computing $N'(x)$ in $\Theta(t)$ time and (4) evaluating $N'(x)$ at all $i \in T$ in $\Theta(t \log^2 t)$ time. This reduces the time to compute all $\mathcal{L}_i^T(0)$'s from $\Theta(t^2)$ to $\Theta(t \log^2 t)$.

2) Further speed-ups via roots of unity: The fast Lagrange technique works for any threshold cryptosytem whose secret key s lies in prime-order field $\mathbb{F}_p$. However, for fields that support roots of unity, further speed-ups are possible. (A caveat is that pairings on the underlying elliptic curve can be up to $2\times$ slower.) Without loss of generality, assume the total number of signers n is a power of two and let ω_n denote a primitive nth root of unity in $\mathbb{F}_p$. If we replace the $\{1, \ldots, n\}$ signer IDs with roots of unity $\{\omega_n^{i-1}\}_{i \in [n]}$, then $N'(x)$ can be evaluated at any subset of signer IDs with a single Fast Fourier Transform (FFT). This is much faster than a polynomial multipoint evaluation, which performs many polynomial divisions, each involving many FFTs. Our fast Lagrange implementation from §IV-A takes advantage of this optimization. Furthermore, we use roots of unity to compute inverses faster in both our naive and fast Lagrange implementations (see §IV-A). For example, in naive Lagrange, we compute $N(0) = \prod_{i \in T}(0 - \omega_n^i)$ much faster as $(-1)^{|T|} \cdot \omega_n^{\sum_{i \in T} i}$.

B. Authenticated Multipoint Evaluation Trees (AMTs)

In this section, we improve KZG's $\Theta(nt)$ time for computing n proofs for a degree-bound t polynomial to $\Theta(n \log t)$ time. Our key technique is to commit to the quotients in a polynomial multipoint evaluation (see §II-E), obtaining an *authenticated multipoint evaluation tree* (AMT). However, our new AMT evaluation proofs are logarithmic-sized, whereas KZG proofs are constant-sized. As a result, when we apply AMTs to scale VSS and DKG later in §§III-C and III-D, we slightly increase communication complexity and reconstruction time. Nonetheless, in §IV, we demonstrate that the time saved in proof computation more than makes up for these smaller increases. Throughout this section, we restrict ourselves to computing AMTs at points $\{1, 2, \ldots, n\}$ on polynomials of degree $t - 1 < n$, since this is the VSS/DKG setting, (In §V-3, we discuss generalizing to any set of points.) Finally, in Appendix C, we show AMT evaluation proofs are secure under q-SBDH. In contrast, KZG proofs are secure under a weaker assumption called q-SDH [64].

1) Computing AMT proofs: KZG evaluation proofs leverage the *polynomial remainder theorem*: $\forall i \in \mathbb{F}_p, \exists q_i$ of degree $t-1$ such that $\phi(x) = q_i(x)(x - i) + \phi(i)$. Specifically, a constant-sized KZG proof for $\phi(i)$ is just a commitment to the quotient polynomial q_i (see §II-B) and takes $\Theta(t)$ time to compute. Thus, computing KZG proofs for each $i \in [n]$ takes $\Theta(nt)$ time. We improve on this by looking at $\phi(x)$ from the lens of a polynomial multipoint evaluation [27].

For example, consider the multipoint evaluation of ϕ at all $i \in [8]$ from Figure 1. Note that every node in the multipoint evaluation tree stores a quotient and a remainder obtained by dividing the parent node's remainder by its accumulator polynomial (see §II-E). The first key idea is that, for any evaluation point $i \in [8]$, $\phi(x)$ can be expressed as $\phi(i)$ plus a linear combination of quotients and accumulator polynomials along the path to $\phi(i)$'s leaf in the multipoint evaluation tree. For example, consider $i = 1$, which has the left-most path in tree. Start with the root node in Figure 1, which says:

$$\phi(x) = q_{1,8}(x)(x-1)\ldots(x-8) + r_{1,8}(x)$$

Then, expand $r_{1,8}(x)$ by going left in the tree (down towards $\phi(1)$'s leaf), obtaining:

$$\begin{aligned} \phi(x) = q_{1,8}(x)&(x-1)(x-2)(x-3)(x-4)\cdots(x-8) \\ &+ q_{1,4}(x)(x-1)(x-2)(x-3)(x-4) + r_{1,4} \end{aligned}$$

Repeat this process recursively by replacing $r_{1,4}(x)$ and then $r_{1,2}(x)$ to get:

$$\begin{aligned} \phi(x) = q_{1,8}(x)&(x-1)(x-2)(x-3)(x-4)\cdots(x-8) \\ &+ q_{1,4}(x)(x-1)(x-2)(x-3)(x-4) \\ &+ q_{1,2}(x)(x-1)(x-2) \\ &+ q_{1,1}(x)(x-1) + \phi(1). \end{aligned}$$

Note that $\phi(x)$ can be re-expressed similarly for any other points $i \in [2, n]$. Importantly, note that there are only $\Theta(n)$ quotient and accumulator polynomials shared by all such expressions of $\phi(i)$.

Our second key idea follows naturally: we commit to all these $\Theta(n)$ quotient polynomials in the multipoint evaluation of ϕ. This gives us logarithmic-sized evaluation proofs for any point $i \in [n]$. We call these proofs *AMT proofs*. For example, in Figure 1, the AMT proof for $\phi(4)$ would be $\{g^{q_{1,8}(\tau)}, g^{q_{1,4}(\tau)}, g^{q_{3,4}(\tau)}, g^{q_{4,4}(\tau)}\}$, where τ denotes the trapdoor used in KZG commitments (see §II-B).

2) Verifying AMT proofs: The next question is how to verify our new logarithmic-sized AMT proofs. Recall that, given any point i, $\phi(x)$ can be expressed as:

$$\phi(x) = \phi(i) + \sum_{w \in \mathsf{path(i)}} q_w(x) a_w(x) \tag{2}$$

where $\mathsf{path}(i)$ is the set of nodes along the path from the root to $\phi(i)$ and q_w and a_w denote the quotient and accumulator polynomials stored at node w in the multipoint

evaluation tree (see Figure 1). How can we verify a proof $\pi_i = \left(g^{q_w(\tau)}\right)_{w \in \mathsf{path}(i)}$ for $\phi(i) = y_i$? We simply use a bilinear map to check that Equation (2) holds at $x = \tau$:

$$e(g^{\phi(\tau)}, g) \stackrel{?}{=} e(g^{y_i}, g) \prod_{w \in \mathsf{path}(i)} e(g^{q_w(\tau)}, g^{a_w(\tau)}) \Leftrightarrow \quad (3)$$

$$e(g,g)^{\phi(\tau)} \stackrel{?}{=} e(g,g)^{y_i} \prod_{w \in \mathsf{path}(i)} e(g,g)^{q_w(\tau) a_w(\tau)} \Leftrightarrow$$

$$e(g,g)^{\phi(\tau)} \stackrel{?}{=} e(g,g)^{y_i + \sum_{i \in \mathsf{path}(w)} q_w(\tau) a_w(\tau)} \Leftrightarrow$$

$$\phi(\tau) \stackrel{?}{=} y_i + \sum_{w \in \mathsf{path}(i)} q_w(\tau) a_w(\tau)$$

This is reminiscent of how KZG proofs are verified by checking that $\phi(x) = q_i(x)(x - i) + \phi(i)$ holds at $x = \tau$ (see §II-B). However, note that *the verifier needs to have the $g^{a_w(\tau)}$ accumulator commitments*, which are not part of the AMT proof. This implies AMT verifiers must have $\Theta(n)$ public parameters, whereas KZG verifiers only need $\{g^\tau\}$ as their public parameters (see §II-B). Fortunately, in §III-B4 we reduce the verifers' public parameters to just $\Theta(\log t)$.

3) Better AMTs using roots of unity: Instead of evaluating ϕ at points $\{1, 2, 3, \ldots, n\}$, we assume $n = 2^m$ and evaluate ϕ at all n nth roots of unity in $\mathbb{F}_p$. Specifically, we compute $\phi(\omega_n^{i-1})$ rather than $\phi(i)$, where ω_n is a primitive nth root of unity. (We can generalize to any n by using the first n Nth roots of unity, where $N = 2^m$ is the smallest value such that $N \geq n$.) The main benefit of using roots of unity is they give rise to simpler accumulator polynomials of the form $(x^{2^k} + c)$ in the multipoint evaluation tree (for some c). This speeds up the multipoint evaluation (see Appendix A), since dividing degree-bound $2n$ polynomials by $(x^n + c)$ can be done in $\Theta(n)$ rather than $\Theta(n \log n)$ time. In Appendix A, we show this new, optimized AMT proof is $\lfloor \log(t-1) \rfloor + 1$ group elements and computing an AMT takes $\Theta(n \log t)$ time.

The $(x^{2^k} + c)$ form of the accumulators is best illustrated with an example. Let $n = 8$ and ω_8 denote a primitive 8th root of unity. Previously, in Figure 1, the evaluation points $\{1, 2, \ldots, 8\}$ were ordered as $\langle (x-1), (x-2), \ldots, (x-8) \rangle$ monomials in the leaves. Then, the accumulators were computed by multiplying "up the tree," culminating in the root accumulator $\prod_{i \in [8]} (x - i)$. In our case, the evaluation points are $\{\omega_8^{i-1}\}_{i \in [8]}$ but we reorder them using a bit-reversal permutation [69] as $\langle (x - \omega_8^0), (x - \omega_8^4), (x - \omega_8^2), (x - \omega_8^6), (x - \omega_8^1), (x - \omega_8^5), (x - \omega_8^3), (x - \omega_8^7) \rangle$. This ordering ensures that, as we multiply "up the tree," all accumulators are of the form $(x^{2^k} + \omega_8^j)$ for some j.

Let us see exactly how this happens. The parent accumulator of the first two leaves $(x - \omega_8^0)$ and $(x - \omega_8^4)$ is their product $(x - \omega_8^0)(x - \omega_8^4) = x^2 - \omega_8^4 x - \omega_8^0 x + \omega_8^0 \omega_8^4$. Since $\omega_n^i \omega_n^j = \omega_n^{(i+j) \bmod n}$ [56], this equals $x^2 - x(\omega_8^4 + \omega_8^0) + \omega_8^4$. Since $\omega_n^{k+n/2} = -\omega_n^k$ [56], this equals $(x^2 + \omega_8^4)$. The remaining accumulators after $(x^2 + \omega_8^4)$ on this level are $\{(x^2 + \omega_8^0), (x^2 + \omega_8^6), (x^2 + \omega_8^2)\}$. Recursing on the next level, its accumulators are $\langle (x^4 + \omega_8^4), (x^4 + \omega_8^0) \rangle$. Finally, the root will be $(x^8 - \omega_8^0) = (x^8 - 1) = \prod_{i=0}^{7} (x - \omega_8^i)$.

4) Do AMTs need extra public parameters?: Recall that in KZG, given $(t-1)$-SDH public parameters, one can commit to any degree-bound t polynomials and compute *any number* of KZG evaluation proofs. In contrast, computing an AMT at $n > t - 1$ points seems to require committing to degree $n > t - 1$ accumulator polynomials (e.g., to the root accumulator $(x^n - 1)$). Yet this is not possible given only $(t-1)$-SDH parameters, as ensured by the $(t-1)$-polyDH assumption (see Appendix B). Fortunately, when computing an AMT, divisions by accumulators of degree $> t - 1$ always give quotient zero (see Appendix A). This means that, when pairing such quotients with their accumulators during proof verification, the result will always be $1_{\mathbb{G}_T}$ (see Equation (3)). In other words, such pairings need never be computed and so their corresponding accumulators (of degree $> t - 1$) need never be committed to. Furthermore, quotients are not problematic since they always have degree $< \deg \phi = t - 1$ (or are equal to zero).

Second, AMT verifiers only need a logarithmic number of $g^{\tau^{2^k}}$ powers to recreate any accumulator commitment $g^{a_w(\tau)}$. (This is a bit worse than KZG verifiers, who only need g^τ.) Specifically, given a subset $\{g^{\tau^{2^k}} \mid 0 \leq k \leq \lfloor \log(t-1) \rfloor\}$ of the $(t-1)$-SDH parameters, the verifier can commit to any degree-bound t accumulator of the $(x^{2^k} + c)$ form. Thus, we impose no additional overhead in the trusted setup. In contrast, if we evaluated ϕ at $\{1, 2, \ldots, n\}$, verifiers would need all $(t-1)$-SDH public parameters to reconstruct the accumulators.

C. Scalable Verifiable Secret Sharing

In this section, we scale (t, n) VSS protocols to large n in the difficult case when $t > n/2$. Specifically, we reduce eVSS's dealing time from $\Theta(nt)$ to $\Theta(n \log t)$ by replacing KZG proofs with AMT proofs. We call this new VSS protocol AMT VSS and describe it below.

1) Faster dealing: The difference between AMT VSS and eVSS is very small. First, players' shares are computed as $s_i = \phi(\omega_N^{i-1})$ (rather than $\phi(i)$ as in eVSS), where N is the smallest power of two $\geq n$. Second, instead of using (slow) KZG proofs, the dealer computes an AMT for ϕ at points $\{\omega_N^{i-1}\}_{i \in [n]}$, obtaining the shares s_i for free in the process. Then, as in eVSS, the dealer sends each player i its share s_i but now with an AMT proof π_i (see §III-B1). The verification round, complaint round and reconstruction phase remain the same, except they all use AMT proofs now.

AMT VSS's dealing time is $\Theta(n \log t)$, dominated by the time to compute an AMT. This is a significant reduction from eVSS's $\Theta(nt)$ time, but comes at a small cost due to our larger AMT proofs. First, the verification round time increases from $\Theta(1)$ to $\Theta(\log t)$. Second, the complaint round complexity increases from $O(t)$ to $O(t \log t)$ time and communication (but we improve it in §III-C2). Third, the reconstruction phase time increases from $\Theta(t \log^2 t + n)$ to $O(t \log^2 t + n \log t)$ (but we improve it in §III-C3). Finally, the overall communication increases from $\Theta(n)$ to $\Theta(n \log t)$. Nonetheless, in §IV-B, we show AMT VSS's end-to-end time is much smaller than eVSS's, which makes these increases justifiable.

2) Faster complaints: Kate et al. previously point out that KZG batch proofs (see §II-B1) can be used to reduce the communication and the *concrete* computational complexity of eVSS's complaint round [14]. Suppose S is the set of complaining players. Without batch proofs, the dealer only has to broadcast $|S|$ previously-computed KZG proofs and each player has to verify them by computing $2|S|$ pairings. With batch proofs, the dealer spends $\Theta(|S|\log^2|S| + t\log t)$ time to compute the batch proof and each player spends $\Theta(|S|\log^2|S|))$ to verify it.

While batch proofs increase asymptotic complexity for the dealer and players, the concrete complexity decreases, since players now only compute two pairings rather than $2|S|$. Furthermore, the communication complexity decreases, since only 1 proof rather than $|S|$ needs to be broadcast. Thus, AMT VSS can also use batch proofs and maintain the same performance as eVSS during the complaint round. (However, in Table I, we do not assume this optimization.)

3) Efficient reconstruction: In some cases, we can reduce the number of pairings computed during AMT VSS's reconstruction phase. In this phase, the reconstructor is given anywhere from t to n shares and their AMT proofs. His task is to find a subset of t valid shares and interpolate the secret. Let us first consider the *best case*, where all submitted shares are *valid*. In this case, if the reconstructor naively verifies any t AMT proofs, he spends $\Theta(t\log t)$ time. But he would be computing the same quotient-accumulator pairings multiple times (as in Equation (3)), since proofs with intersecting paths will share quotient commitments. By memoizing these computations, the reconstructor can verify the t proofs in $\Theta(t)$ time. Alternatively, this can be sped up by exposing a g^s public key during dealing (as in DKG protocols; see §III-D3).

Now let us consider the *worst case*, where $n-t$ shares are invalid and t shares are valid. The reconstructor wants to find the t valid shares as fast as possible. Once again, he can memoize the quotient-accumulator pairings that are part of successfully validated proofs. This way, for the t valid proofs, only $\Theta(t)$ pairings need to be computed. Thus, at most $\Theta((n-t)\log t)$ pairings could possibly be computed for the invalid proofs. The worst-case reconstruction time remains $\Theta(n\log t)$ but, in practice, the number of pairings is reduced significantly by the memoization.

4) Public parameters: The AMT VSS dealer needs $(t-1)$-SDH public parameters, just like in eVSS. This is because committing to accumulator polynomials of degree $\geq t$ is not necessary, as discussed in §III-B4. In fact, adding more public parameters for committing to degree $\geq t$ polynomials would break the *correctness* of eVSS and thus of AMT VSS [14]. Specifically, if the dealer commits to a degree $\geq t$ polynomial ϕ, then different secrets could be reconstructed, depending on the subset of players whose shares are used. This is why the $(t-1)$-polyDH assumption (see Definition A.3) is needed in both protocols. Finally, AMT VSS players (and the reconstructor) need $\Theta(\log t)$ public parameters to verify AMT proofs, an increase from eVSS' $\Theta(1)$ (i.e., g^τ).

D. Scalable Distributed Key Generation

In this section, we scale (t,n) DKG protocols to large n in the difficult case when $t > n/2$. We start from eJF-DKG, where each player acts as an eVSS dealer (see Algorithm 2), taking $\Theta(nt)$ time to compute n KZG evaluation proofs and $\Theta(t)$ time to compute one KZG proof for $g^{f_i(0)}$ (see Algorithm 2). We simply replace eVSS with AMT VSS in eJF-DKG, obtaining a new protocol we call AMT DKG with smaller $\Theta(n\log t)$ per-player dealing time. Importantly, we keep the same KZG proof for $g^{f_i(0)}$.

Compared to eJF-DKG, AMT DKG has slightly larger communication (see §IV-C5), larger proof verification times and a slower complaint round (see Table I). Fortunately, when using KZG batch proofs (see §III-C2), the complaint round can be made more efficient in both eJF-DKG and AMT DKG. Furthermore, we show AMT DKG players can verify their shares much faster under certain conditions (see §III-D2). Finally, in §IV-C, we show that our smaller dealing time more than makes up for these increases.

1) Homomorphic AMT proofs: At the end of eJF-DKG's sharing phase, each player must aggregate all his shares, commitments and KZG proofs from the set of qualified players into a final share, commitment and proof (see Algorithm 2). But for this to work in AMT DKG, AMT proofs must be *homomorphic*: $\forall a \in \mathbb{F}_p$, a proof for $f_1(a)$ and a proof for $f_2(a)$ must be *aggregated* into a proof for $(f_1+f_2)(a)$.

The key observation is that "adding up" the multipoint evaluation trees of two polynomials ϕ and ρ at the same points (i.e., at $X = \{\omega_N^{j-1}\}_{j\in[n]}$) results in a multipoint evaluation tree of their sum $\phi+\rho$ (also at X). In more detail, let $q_{w,[\psi]}$ denote the quotient polynomial at node w in ψ's multipoint evaluation tree (at X). Then, one can show that $q_{w,[\phi+\rho]} = q_{w,[\phi]} + q_{w,[\rho]}$ and that $g^{q_{w,[\phi+\rho]}(\tau)} = g^{q_{w,[\phi]}(\tau)+q_{w,[\rho]}(\tau)} = g^{q_{w,[\phi]}(\tau)}g^{q_{w,[\rho]}(\tau)}$. In other words, given an AMT for ϕ and an AMT for ρ, we can obtain an AMT for $\phi+\rho$ by multiplying quotient commitments at each node. It follows that a proof for $f_1(a)$ and one for $f_2(a)$ can be aggregated into a proof for $(f_1+f_2)(a)$ by multiplying commitments at each node.

2) Fast-track verification round: During the verification round, each player j must receive and verify shares from all players $i \in [n]$, including himself (see Algorithm 2). Specifically, each player i gives j: (1) a KZG commitment c_i of i's polynomial f_i, (2) a share $s_{i,j} = f_i(\omega_N^{j-1})$ with an AMT proof $\pi_{i,j}$ and (3) $g^{f_i(0)}$ with a NIZKPoK and KZG proof. Next, player j must verify each $s_{i,j}$ and $g^{f_i(0)}$ against their c_i. With *naive verification*, this takes $\Theta(n\log t)$ pairings for all $s_{i,j}$'s (since $\pi_{i,j}$'s are AMT proofs), and $\Theta(n)$ pairings for the $g^{f_i(0)}$'s. We show how *batch verification* can do this faster, with anywhere from $\Theta(\log t)$ to $\Theta(n\log t)$ pairings, depending on the number of valid shares. (We will not address the $\Theta(n)$ work required to verify all NIZKPoKs.)

First, consider the *best case* when all $s_{i,j}$'s are valid. The key idea is player j will verify just one *aggregated* share $s_j = \sum_{i=1}^n s_{i,j}$ against an aggregated commitment $c_{\mathsf{all}} = \prod_{i=1}^n c_i$ and aggregated proof π_j from all $\pi_{i,j}$'s (as

explained in §III-D1). (We ignore the $g^{f_i(0)}$'s for now.) This takes $\Theta(n \log t)$ aggregation work but only takes $\Theta(\log t)$ pairings. If successful, j has a valid share s_j on $f_{\mathsf{all}} = \sum_{i=1}^{n} f_i$. The same aggregation can be done on the $g^{f_i(0)}$'s and their KZG proofs. This way, the number of pairings is reduced significantly to $\Theta(\log t)$ for the shares and $\Theta(1)$ for the $g^{f_i(0)}$'s. (Again, j still does $\Theta(n)$ work to verify the NIZKPoKs individually, which we will not address.)

Since players can be malicious, let us consider an *average case* when a small number of b shares are bad. In this case, j can identify the b shares faster via *batch verification* [10]. Specifically, j starts with the shares, proofs and commitments as leaves of a binary tree, where every node aggregates its subtree's shares, proofs and commitments. As a result, the root will contain $(c_{\mathsf{all}}, s_j, \pi_j)$. If verification of the root fails, j proceeds recursively down the tree. Whenever a node verifies, shares in its subtree will no longer be checked individually, saving work for j. In this fashion, j only computes $\Theta(b \log t)$ pairings if $\leq b$ shares are bad.

Unfortunately, in the *worst case* (i.e., $t-1$ bad shares), batch verification computes $\approx (2n-1)\log t$ pairings, which is slower than the $\approx n \log t$ pairings when done naively. Thus, as pointed out by previous work [70], j should abort and verify naively after too many nodes fail verification. To summarize, j can compute fewer pairings by batch-verifying optimistically to see if he is in the best or average case and downgrading to naive verification otherwise. We stress that j still does $\Theta(n \log t)$ work to build the tree and $\Theta(n)$ work to verify all NIZKPoKs, but fewer (expensive) pairings are computed.

3) Optimistic reconstruction: DKG protocols have the advantage that g^s must be exposed to all players and the reconstructor. Thus, the reconstructor can optimistically interpolate s from any t shares (without verifying them) and check the result against g^s. In the best case, when all or most shares are valid, this will recover the correct s very fast (see §IV-C3). (Note that AMT VSS and eVSS do not expose g^s but they could be easily modified to do so and speed up the reconstruction in the best case, at a very small increase in dealing time.) In the worst case, AMT DKG's reconstruction time is the same as AMT VSS's (see §III-C3).

IV. Evaluation

In this section, we demonstrate the scalability of our proposed cryptosystems. Our experiments focus on the difficult case when $t > n/2$, specifically $t = f+1$ and $n = 2f+1$. We benchmark TSS, VSS and DKG cryptosystems for thresholds $t \in \{2^1, 2^2, 2^3, \ldots, 2^{20}\}$. Although we did not benchmark other thresholds, similar performance gains would have been observed for other sufficiently large values of t (e.g., $t = f+1$ and $n = 3f+1$). However, we acknowledge that, for sufficiently small t, eVSS's and eJF-DKG's $\Theta(nt)$ dealing would outperform ours. Similarly, in this small t setting, naive Lagrange interpolation would outperform fast Lagrange. Our experiments show that:

- Our BLS TSS scales to $n \approx 2$ million signers and outperforms the naive scheme as early as $n = 511$ (see Figure 2a).
- AMT VSS scales to hundreds of thousands of participants, and outperforms eVSS as early as $n = 255$ (see Figure 2f).
- AMT DKG scales to $n \approx 65{,}000$ players and outperforms eJF-DKG at $n = 1023$ (see Figure 2i).

Importantly, our VSS and DKG speed-ups come at the price of a modest increase in communication (see Figure 2c). For example, for $n \approx 65{,}000$, a DKG player's communication during dealing increases by 4.11× from 18 MiB in eJF-DKG to 74 MiB in AMT DKG. However, since the worst-case end-to-end time decreases by 32× from 16.76 hrs in eJF-DKG to 30.83 mins in AMT DKG, the extra communication should be worth it in many applications.

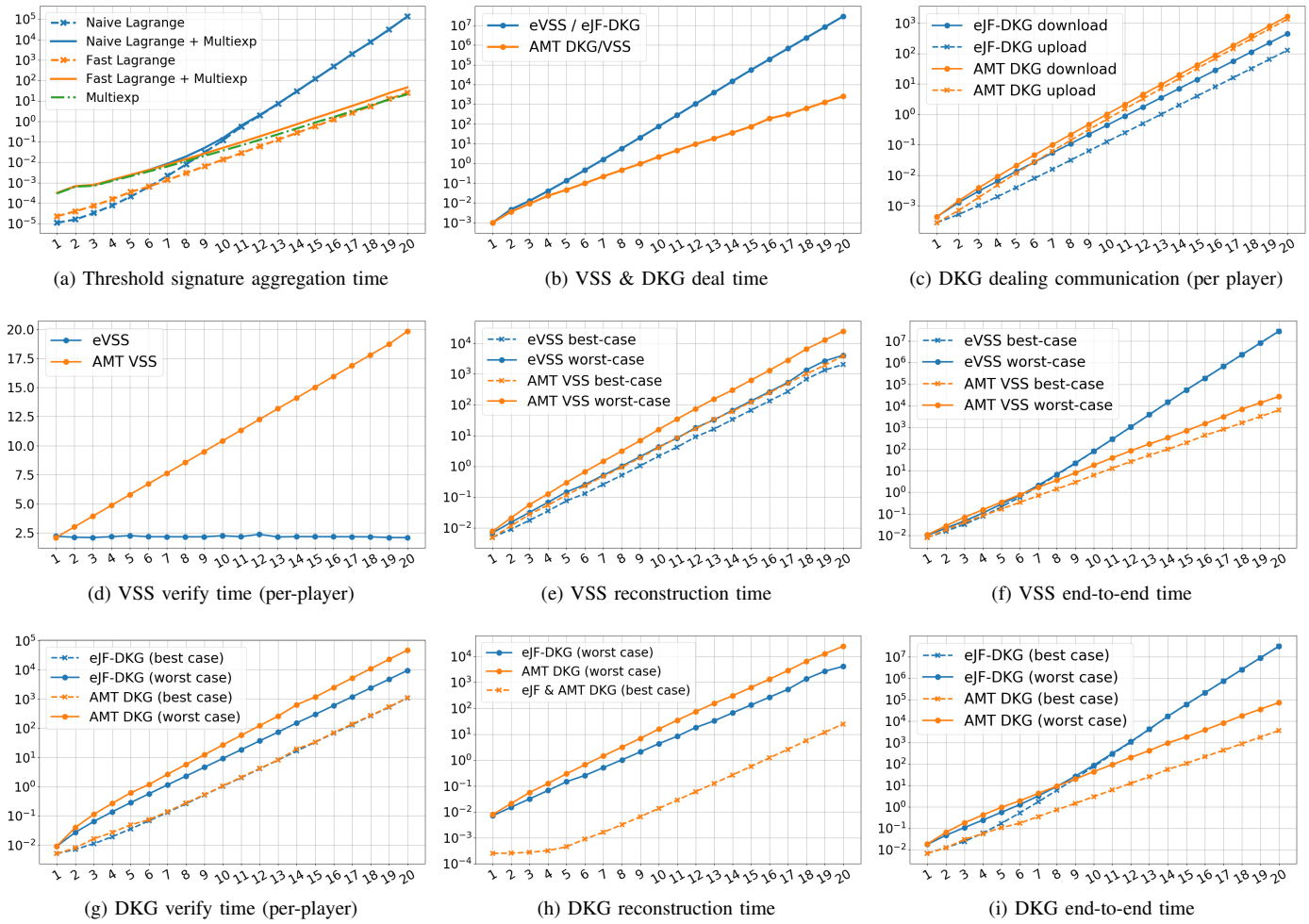
For prohibitively-slow experiments with large t, we repeat them fewer times than experiments with smaller t. For brevity, we specify the amount of times we repeat an experiment for each threshold via a *measurement configuration*. For example, the measurement configuration of our efficient BLS threshold scheme is $\langle 7 \times 100, 13 \times 10 \rangle$. This means that for the first 7 thresholds $t \in \{2^1, 2^2, \ldots, 2^7\}$ we ran the experiment 100 times while for the last 13 thresholds we ran it 10 times.

1) Codebase and experimental setup: We implemented (1) our BLS threshold signature scheme from §III-A, (2) eJF-DKG [17] and AMT DKG and (3) eVSS [14] and AMT VSS in 5700 lines of C++. We used a 254-bit Barretto-Naehrig curve with a Type III pairing [71] from Zcash's `libff` [72] elliptic curve library. We used `libfqfft` [73] to multiply polynomials fast using FFT. All experiments were run on an Intel Core i7 CPU 980X @ 3.33GHz with 12 cores and 20 GB of RAM, running Ubuntu 16.04.6 LTS (64-bit version). Since all benchmarked schemes would benefit equally from multi-threading, we did not implement it.

2) Limitations: Our DKG and VSS evaluations do not account for network delays. This is an important limitation. Our focus was on the computational bottlenecks of these protocols. Nonetheless, scaling and evaluating the broadcast channel of VSS and DKG protocols is necessary, interesting future work. In particular, ideas from scalable consensus protocols [4] could be used for this. Finally, our VSS and DKG "worst case" evaluations do not fully account for malicious behavior. Specifically, they do not account for the additional communication and computational cost associated with complaint broadcasting. We leave this to future work (see §V-2).

A. BLS Threshold Signature Experiments

First, we sample a random subset of t signers T with valid signature shares $\{\sigma_i\}_{i \in T}$. Second, we compute Lagrange coefficients $\mathcal{L}_i^T(0)$ w.r.t. points $x_i = \omega_N^{i-1}$ (see §II-4) using both fast and naive Lagrange. Third, we compute the final threshold signature $\sigma = \prod_{i \in T} \sigma_i^{\mathcal{L}_i^T(0)}$ using a multi-exponentiation. The measurement configuration for fast Lagrange is $\langle 7 \times 100, 13 \times 10 \rangle$ while for naive Lagrange is $\langle 8 \times 100, 6 \times 10, 8, 4, 2, 1, 1, 1 \rangle$. We plot the average aggregation time in Figure 2a and observe that our scheme beats the naive scheme as early as $n = 511$. We do not measure the time to identify valid signature shares via batch verification [10], which our techniques leaves unchanged.



(a) Threshold signature aggregation time

(b) VSS & DKG deal time

(c) DKG dealing communication (per player)

(d) VSS verify time (per-player)

(e) VSS reconstruction time

(f) VSS end-to-end time

(g) DKG verify time (per-player)

(h) DKG reconstruction time

(i) DKG end-to-end time

Fig. 2. All benchmarked threshold cryptosystems have threshold $t = f + 1$ out of $n = 2f + 1$. The x-axis always indicates $\log_2 t$. The y-axis is in seconds, except in Figure 2c it is in MB and in Figure 2d it is in milliseconds.

Our results show that our fast Lagrange interpolation drastically reduces the time to aggregate when $t \approx n/2$. Specifically, for $n \approx 2^{21}$, we aggregate a signature in 46.26 secs, instead of 1.59 days if aggregated via naive Lagrange (2964× faster). The benefits are not as drastic for smaller thresholds, but remain significant. For example, for $n \approx 2^{15}$, we reduce the time by 41× from 29.74 secs to 719.65 ms. For $n = 4095$, we see a 6.6× speed-up from 636.6 ms to 96.17 ms. For $n = 2047$, we see a 3× speed-up from 155.62 ms to 50.74 ms.

B. Verifiable Secret Sharing Experiments

In this section, we benchmark eVSS and AMT VSS. We do not benchmark the complaint round since, when implemented with KZG batch proofs, it remains the same (see §III-C2).

1) VSS dealing: For eVSS dealing, the measurement configuration is $\langle 10 \times 10, 3, 2, 2, 1, 1, 0, 0, 0, 0, 0\rangle$. For large $t \geq 2^{16}$, eVSS dealing is too slow, so we extrapolate it from the previous dealing time (i.e., we multiply by 3.5). For AMT VSS dealing, the measurement configuration is $\langle 12 \times 100, 50, 22, 10, 5, 3, 2, 1, 1\rangle$. In eVSS, we compute the shares s_i "for free" as remainders of the $\phi(x)/(x - i)$ divisions. We plot the average dealing time in AMT VSS and eVSS as a function of n in Figure 2b. Our results show that AMT VSS's $\Theta(n \log t)$ dealing scales much better than eVSS's $\Theta(nt)$ dealing. For example, for $n \approx 65,000$, eVSS takes 15.1 hrs while AMT VSS takes 1.24 mins. For very large $n \approx 2^{21}$, eVSS takes a prohibitive 330 days while AMT VSS takes 42 mins. We find that AMT VSS's dealing outperforms eVSS's as early as $n = 31$.

2) VSS verification round: In Figure 2d, we plot the time for one player to verify its share. The measurement configuration is $\langle 20 \times 1000\rangle$ for both schemes. In eVSS, verification requires two pairings and one exponentiation in $\mathbb{G}_1$, taking on average 2.15 ms. In AMT VSS, verification requires $\lfloor \log(t-1) \rfloor + 1$ pairings and one exponentiation in $\mathbb{G}_1$, ranging from 2.07 ms ($n = 3$) to 19.85 ms ($n \approx 2^{21}$).

3) VSS reconstruction: In Figure 2e, we plot the time to reconstruct the secret. We consider the *best-case* and *worst-case* times, as detailed in §III-C3. For eVSS, "best case" means the first t share verifications are successful and "worst case" means the first $n - t$ are unsuccessful (see §III-C3). The measure-

ment configuration is $\langle 5 \times 1000, 500, 250, 120, 60, 30, 15, 5 \times 10, 8, 4, 2, 1\rangle$ for eVSS and $\langle 9{\times}100, 4{\times}10, 4, 2, 5{\times}1\rangle$ for AMT VSS. In both protocols, the (fast) Lagrange interpolation time is insignificant compared to the time to verify shares during reconstruction (e.g., for $n \approx 2^{21}$ in eVSS, interpolation is only 25 secs out of the total 34 mins worst-case time).

AMT VSS's best-case is very close to eVSS's worst-case. This is because, with the help of memoization, AMT VSS's best case only computes $\leq 2n-1$ pairings (i.e., the number of nodes in a full binary tree with n leaves). This closely matches the $2n$ pairings in eVSS's worst case. (In practice, we replace n of these pairings and $\mathbb{G}_1$ exponentiations by n $\mathbb{G}_T$ exponentiations, which are slightly faster.) AMT VSS's worst case is 1.12× to 6× slower than eVSS's. But we show next that our faster dealing more than makes up for this. Finally, eVSS's best-case time is half its worst-case time, as expected.

4) VSS end-to-end time: Finally, we consider the *end-to-end time*, which is the sum of the sharing and reconstruction phase times. (Again, a limitation of our work is ignoring the overhead of the complaint round in the worst case.) Figure 2f gives the best- and worst-case end-to-end times. The key takeaway is that AMT VSS's smaller dealing time makes up for the increase in its verification round and reconstruction phase times. AMT VSS outperforms eVSS's worst-case time at $n \geq 255$ and its best-case time at $n \geq 63$. For example, for large $n = 16,383$, we reduce the worst-case time from 1.1 hrs to 2.9 mins and the best-case time from 1.1 hrs to 51.48 secs. The best case improvement ranges from 1.26× ($n = 63$) to 4484× ($n \approx 2^{21}$). The worst case improvement ranges from 1.26× ($n = 255$) to 1055× ($n \approx 2^{21}$). Thus, we conclude AMT VSS scales better than eVSS.

C. Distributed Key Generation Experiments

Our DKG experiments mostly tell the same story as our VSS experiments: AMTs drastically reduce the dealing time of DKG players, which more than makes up for the slight increase in verification and reconstruction time. However, AMT DKG has a 1.2× to 5.2× communication overhead during dealing. Still, we believe this is worth the drastic reduction in end-to-end times (see §IV-C4).

1) DKG dealing: DKG dealing time is equal to VSS dealing time (see §IV-B1) plus the time to compute a KZG proof and a NIZKPoK for $g^{f_i(0)}$. However, as n increases, the time to compute these two proofs pales in comparison to the time to compute the n evaluation proofs. Thus, in Figure 2b, we treat DKG dealing times as equal to VSS dealing times. As a result, the same observations apply here as in §IV-B1: AMTs drastically reduce dealing times.

2) DKG verification round: We consider both the *best case* and the *worst case* verification time, as discussed in §III-D2. In our best-case experiment, each player j aggregates all its shares as $s_j = \sum_{i\in[n]} s_{i,j}$ and their evaluation proofs as π_j. Then, j verifies s_j against π_j. Similarly, j aggregates and efficiently verifies all its $g^{f_i(0)}$'s and their KZG proofs. In the worst-case experiment, j individually verifies the $s_{i,j}$ shares and the $g^{f_i(0)}$'s. Importantly, in both experiments, j individually verifies all n NIZKPoKs for $g^{f_i(0)}$ in $\Theta(n)$ time. The two experiments are meant to bound the time of a realistic implementation that carefully uses *batch verification* [10], [70] to not exceed the worst-case time too much.

The best-case eJF-DKG measurement configuration is $\langle 8 \times 100, 50, 25, 12, 9 \times 10\rangle$ and the worst-case is $\langle 5 \times 100, 50, 25, 12, 12{\times}10\rangle$. For AMT DKG, the best-case configuration is $\langle 12{\times}100, 80, 40, 20, 16, 8, 4, 3, 2\rangle$ and the worst-case is $\langle 5{\times}100, 4{\times}80, 40, 20, 8, 4, 2, 6{\times}1\rangle$. The average per-player verification times are plotted in Figure 2g. In the best case, both schemes perform roughly the same, since the verification of the n NIZKPoKs quickly starts dominating the aggregated proof verification. In the worst case, AMT DKG time ranges from 8.96 ms ($n = 3$) to 12.92 hrs ($n \approx 2^{21}$). In contrast, eJF-DKG time ranges from 8.92 ms to 2.59 hrs (1.5× to 5× faster). Nonetheless, eJF-DKG remains slower overall due to its much slower dealing (see §IV-C4). Both best- and worst-case times can be reduced by batch-verifying NIZKPoKs, which resemble Schnorr signatures [60] and are amenable to batching [74].

3) DKG reconstruction: Here the measurement configuration is $\langle 4 \times 1000, 200, 50, 25, 13 \times 10\rangle$ and times are plotted in Figure 2h. The best case is very fast in both eJF-DKG and AMT DKG, taking only 24.71 secs for $t = 2^{20}$, since both schemes interpolate the secret s without verifying shares and check it against g^s (see §III-D3). For the worst case, the time is the sum of (1) the (failed) best-case reconstruction time and (2) the worst-case time to identify t valid shares from n shares. Since the best case is very fast, the DKG worst-case time (see Figure 2h) looks almost identical to its VSS counterpart (see Figure 2e). Note that the same AMT VSS speed-up techniques for finding t valid shares apply in AMT DKG (see §III-C3). AMT DKG's worst case is anywhere from 1.1× to 6× slower than eJF-DKG's, much like AMT VSS. However, as we show next, AMT DKG's faster dealing more than makes up for this.

4) DKG end-to-end time: Similar to the VSS experiments in §IV-B4, we consider the *end-to-end time*. Figure 2i plots the best- and worst-case end-to-end times and shows that AMT DKG outperforms eJF-DKG starting at $n \geq 63$ (in the best case) and at $n \geq 1023$ (in the worst case). This is a direct consequence of AMT VSS outperforming eVSS, since the DKG protocols use these VSS protocols internally. For example, for large $n = 16,383$, we reduce the worst-case end-to-end time from 1.19 hrs to 7.12 mins and the best-case time from 1.16 hrs to 25.45 secs. The improvement in best-case end-to-end time ranges from 1.6× ($n = 63$) to 8607× ($n \approx 2^{21}$) and, in the worst case, from 1.3× to 427×. Thus, we conclude AMT DKG scales better than eJF-DKG.

5) DKG communication: We estimate each player's upload and download during the dealing round. For upload, each eJF-DKG and AMT DKG player i has to broadcast a KZG commitment $g^{f_i(\tau)}$ (32 bytes) and a commitment $g^{f_i(0)}$ with a NIZKPoK and a KZG proof (32 + 64 + 32 bytes). Then, i has to send each $j \in [n]$ its share (32 bytes) with an evaluation proof (32 bytes for KZG or $(\lfloor \log(t-1) \rfloor + 1) \cdot 32$ bytes for AMT). For download, each player i, has to download $n-1$ shares, each with their KZG commitment and evaluation proof,

plus $n-1$ $g^{f_j(0)}$'s, each with their NIZKPoK and KZG proof. Note that AMT DKG uses KZG proofs for $g^{f_i(0)}$ to minimize its communication overhead.

We plot the upload and download numbers for both schemes in Figure 2c. eJF-DKG's per-player upload ranges from 288 bytes to 128 MiB while download ranges from 448 bytes to 448 MiB. AMT VSS's upload overhead ranges from 1.0× to 10.5× and its download overhead ranges from 1.0× to 3.7×. Overall, AMT VSS's upload-and-download overhead ranges from 1.0× to 5.2×. Thus, we believe the 8607× and 427× reductions in best- and worst-case end-to-end times are sufficiently large to make up for this overhead.

V. Discussion and Future Work

1) Generating public parameters: Similar to eVSS and eJF-DKG, our protocols require a *trusted setup* to generate ℓ-SDH public parameters. Fortunately, this setup needs to be done only once and can be securely implemented via MPC protocols [75], [76]. In fact, currently deployed systems have already demonstrated the practicality of this approach. In 2018, approximately 200 participants used an MPC [76] to generate new public parameters for the Sapling version of Zcash [77]. The MPC protocol allowed anyone to participate and only required one honest party, making it a very good candidate.

2) Sortitioned DKG: To further reduce communication and computation, we propose a *sortitioned DKG* where only a small, random *committee* of $c < n$ players deal. The key question is where does the randomness to pick the committee come from? When a DKG runs many times, this randomness could come from previous DKG runs (e.g., DKGs for Schnorr TSS nonces). To bootstrap securely, the first DKG run would be with a full committee of size $c = n$. When a DKG runs only once, such as when distributing the secret key of a (t, n) TSS, the c players could be a decentralized cothority [41] different than the TSS signers. The cothority would run the DKG dealing round while the n signers would run the DKG verification round (see Algorithm 2). The complaint round would be split: accused cothority members would compute the KZG batch proofs (see §III-C2) while the n signers would receive and verify those proofs. Importantly, our AMT technique would help cothority members deal much faster to the n signers. We leave defining and proving the security of sortitioned DKGs to future work.

3) Arbitrary points: AMTs can be generalized to any set of points $\{x_i\}_{i\in[n]}$ (not just $x_i = \omega_N^{i-1}$) *for which verifiers do not have the necessary accumulator commitments*. The accumulators $g^{a_w(\tau)}$ can be included as part of the proof *but* along with (1) a subset proof w.r.t. the parent accumulator and (2) an "extractable" counterpart $g^{\alpha a_w(\tau)}$, where α is another trapdoor. The asymptotic proof size remains the same but will increase in practice by 4x (with Type III pairings). Furthermore, this construction will need extra public parameters of the form $(g^{\alpha\tau^i})_{i\in[0,\ell]}$. On the other hand, proof verifiers now need $\Theta(1)$ rather than $\Theta(\log n)$ public parameters (see §§III-B4 and III-C4). We leave proving this construction secure under ℓ-PKE [78] to future work.

4) Information-theoretic hiding AMTs: We can devise an information-theoretic hiding version of our AMT proofs that is compatible with information-theoretic hiding KZG commitments [14]. This version of AMTs can be used to speed up the unbiasable New-DKG protocol [16]. Let $h = g^\kappa$ be another generator of $\mathbb{G}$ such that nobody knows the discrete log $\kappa = \log_g(h)$. Assume that, in addition to $\mathsf{PP}_q(g;\tau)$, we also have public parameters $\mathsf{PP}_\ell(h;\tau)$. An information-theoretic hiding KZG commitment to ϕ of degree d is $c = g^{\phi(\tau)}h^{r(\tau)} = g^{\phi(\tau)+\kappa r(\tau)}$ where r is a random, degree d polynomial [14]. Note that c is just a commitment to the polynomial $\psi(x) = \phi(x) + \kappa r(x)$. As a consequence, all we have to do is build an AMT for ψ. For this, we compute an AMT for ϕ with public parameters $\mathsf{PP}_\ell(g;\tau)$ and one for r but with parameters $\mathsf{PP}_\ell(h;\tau)$. By homomorphically combining these two AMTs we get exactly the AMT for ψ (see §III-D1). We leave proving this construction is information-theoretic hiding to future work.

5) Vector commitments (VCs): AMTs naturally give rise to a VC scheme with logarithmic-sized proofs [79]. Similar to the multivariate polynomial-based VC from [80], this scheme would also support efficiently updating proofs and updating VC digests after vector updates. Thus, our VC could also be used for building stateless cryptocurrencies [80], [81]. Our VC can be extended with zero-knowledge-like properties using the information-theoretic variant of AMTs (see §V-4).

6) Batch AMT verification: The efficient reconstruction techniques from §III-C3 reduced the number of pairings when verifying an AMT, but still required $\Theta(t+(n-t)\log t)$ pairings in the worst case. At the cost of doubling the prover time and proof size, this can be reduced to $\approx 2n-1$ pairings, independent of how many proofs are valid. The key idea is to also include commitments to the *remainder polynomials* from the multipoint evaluation tree in the AMT (see Figure 1). This way, an entire AMT tree can be verified node-by-node, top-to-bottom by checking that the division at each node is correct. We leave proving this approach secure to future work.

VI. Conclusion

We introduced new techniques that both speed up and scale threshold cryptosystems. First, we showed how computing Lagrange coefficients efficiently can drastically reduce threshold signature aggregation time. We believe our fast BLS threshold signature scheme can be used to design simple, large-scale, decentralized random beacons. Second, we introduced a quasilinear time technique for precomputing proofs in KZG polynomial commitments. When applied to VSS and DKG protocols, this technique drastically reduces computation without increasing communication too much. We left scaling the broadcast channel and the complaint round to future work.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," https://bitcoin.org/bitcoin.pdf, 2008, Accessed: 2017-03-08.

[2] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," http://gavwood.com/paper.pdf, Accessed: 2016-05-15.

[3] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A Scalable and Decentralized Trust Infrastructure," in *DSN'19*.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *ACM SOSP'17*.

[5] T. Hanke, M. Movahedi, and D. Williams, "DFINITY Technology Overview Series, Consensus System," *CoRR*, vol. abs/1805.04548, 2018. [Online]. Available: http://arxiv.org/abs/1805.04548

[6] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol," in *CRYPTO'17*.

[7] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain," in *EUROCRYPT'18*.

[8] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability," in *ACM CCS'18*.

[9] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable Bias-Resistant Distributed Randomness," in *IEEE S&P'17*.

[10] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme," in *PKC'03*.

[11] V. Shoup, "Practical Threshold Signatures," in *EUROCRYPT'00*.

[12] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *CRYPTO'91*.

[13] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," in *IEEE FOCS'85*.

[14] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-Size Commitments to Polynomials and Their Applications," in *ASIACRYPT'10*.

[15] T. P. Pedersen, "A Threshold Cryptosystem without a Trusted Party," in *EUROCRYPT'91*.

[16] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, 2007.

[17] A. Kate, "Distributed Key Generation and Its Applications," Ph.D. dissertation, Waterloo, Ontario, Canada, 2010.

[18] C. Cachin, K. Kursawe, and V. Shoup, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography," *Journal of Cryptology*, vol. 18, no. 3, Jul 2005.

[19] A. Kate and I. Goldberg, "Distributed Key Generation for the Internet," in *IEEE ICDCS'09*.

[20] D. R. Stinson and R. Strobl, "Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates," in *ACISP'01*.

[21] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security," in *ACNS'16*.

[22] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage," in *CRYPT0'95*.

[23] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive Public Key and Signature Systems," in *ACM CCS'97*.

[24] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography," in *ACM PODC'98*.

[25] I. Cascudo and B. David, "SCRAPE: Scalable Randomness Attested by Public Entities," in *ACNS'17*.

[26] P. Feldman, "A Practical Scheme for Non-interactive Verifiable Secret Sharing," in *IEEE FOCS'87*.

[27] J. von zur Gathen and J. Gerhard, "Fast polynomial evaluation and interpolation," in *Modern Computer Algebra*, 3rd ed. Cambridge University Press, 2013, ch. 10.

[28] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Adaptive Security for Threshold Cryptosystems," in *CRYPTO'99*.

[29] M. Abe and S. Fehr, "Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography," in *CRYPTO'04*.

[30] Y. Frankel, P. MacKenzie, and M. Yung, "Adaptively-Secure Distributed Public-Key Systems," in *Algorithms - ESA'99*.

[31] S. Jarecki and A. Lysyanskaya, "Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures," in *EUROCRYPT'00*.

[32] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "ETHDKG: Distributed Key Generation with Ethereum Smart Contracts," Cryptology ePrint Archive, Report 2019/985, 2019, https://eprint.iacr.org/2019/985.

[33] Y. Desmedt, "Society and Group Oriented Cryptography: A New Concept," in *CRYPTO'87*.

[34] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures," in *CRYPTO'91*.

[35] L. Harn, "Group-oriented (t, n) threshold digital signature scheme and digital multisignature," *IEE Proceedings - Computers and Digital Techniques*, vol. 141, no. 5, Sep. 1994.

[36] C. Park and K. Kurosawa, "New ElGamal Type Threshold Digital Signature Scheme," 1996.

[37] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust Threshold DSS Signatures," in *EUROCRYPT'96*.

[38] Chia Network, "BLS signatures in C++ using the RELIC toolkit," https://github.com/Chia-Network/bls-signatures, Accessed: 2019-05-06.

[39] DFINITY, "go-dfinity-crypto," https://github.com/dfinity/go-dfinity-crypto, Accessed: 2019-05-06.

[40] Mitsunari Shigeo, "BLS threshold signature," https://github.com/herumi/bls/, Accessed: 2019-05-06.

[41] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning," in *IEEE S&P'16*.

[42] M. Stadler, "Publicly Verifiable Secret Sharing," in *EUROCRYPT'96*.

[43] B. Schoenmakers, "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting," in *CRYPTO'99*.

[44] S. Basu, A. Tomescu, I. Abraham, D. Malkhi, M. K. Reiter, and E. G. Sirer, "Efficient Verifiable Secret Sharing with Share Recovery in BFT Protocols," in *ACM CCS'19*.

[45] I. Ingemarsson and G. J. Simmons, "A Protocol to Set Up Shared Secret Schemes Without the Assistance of a Mutually Trusted Party," in *EUROCRYPT'90*.

[46] W. Neji, K. Blibech, and N. Ben Rajeb, "Distributed key generation protocol with a new complaint management strategy," *SCN'16*.

[47] J. Canny and S. Sorkin, "Practical large-scale distributed key generation," in *EUROCRYPT'04*.

[48] Philipp Schindler, "Ethereum-based Distributed Key Generation Protocol," https://github.com/PhilippSchindler/ethdkg, Accessed: 2019-05-07.

[49] "Orbs Network: DKG for BLS threshold signature scheme on the EVM using Solidity," https://github.com/orbs-network/dkg-on-evm, 2018, Accessed: 2019-02-15.

[50] DFINITY, "Distributed Key Generation in JS," https://github.com/dfinity/dkg, Accessed: 2019-05-07.

[51] GNOSIS, "Distributed Key Generation," https://github.com/gnosis/dkg, Accessed: 2019-05-07.

[52] Heiko Stamer, "Distributed Privacy Guard," https://www.nongnu.org/dkgpg/, Accessed: 2019-05-07.

[53] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *CRYPTO'89*.

[54] A. Menezes, S. Vanstone, and T. Okamoto, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field," in *ACM STOC'91*.

[55] V. Goyal, "Reducing Trust in the PKG in Identity Based Cryptosystems," in *CRYPTO'07*.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[57] J. von zur Gathen and J. Gerhard, "Newton iteration," in *Modern Computer Algebra*, 3rd ed. Cambridge University Press, 2013, ch. 9.

[58] J. Berrut and L. Trefethen, "Barycentric Lagrange Interpolation," *SIAM Review*, vol. 46, no. 3, 2004.

[59] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Commun. ACM*, vol. 21, no. 2, Feb. 1978.

[60] C. P. Schnorr, "Efficient Identification and Signatures for Smart Cards," in *CRYPTO'89*.

[61] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Applications of Pedersen's Distributed Key Generation Protocol," in *CT-RSA'03*.

[62] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, July 1985.

[63] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *ASIACRYPT'01*.

[64] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *Journal of Cryptology*, vol. 21, no. 2, pp. 149–177, 2008.
[65] A. Shamir, "How to Share a Secret," *Commun. ACM*, vol. 22, no. 11, Nov. 1979.
[66] G. R. Blakley, "Safeguarding cryptographic keys," in *International Workshop on Managing Requirements Knowledge (AFIPS)*, 1979. [Online]. Available: doi.ieeecomputersociety.org/10.1109/AFIPS.1979.98
[67] J. Camenisch and M. Stadler, "Proof Systems for General Statements about Discrete Logarithms," ETH Zurich, Tech. Rep., 1997.
[68] VMware, "threshsign library," https://github.com/vmware/concord-bft/tree/master/threshsign, Accessed: 2019-05-06.
[69] Wikipedia contributors, "Bit-reversal permutation — Wikipedia, The Free Encyclopedia," 2019, Accessed: 2019-07-24. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bit-reversal_permutation&oldid=883335942
[70] L. Law and B. J. Matt, "Finding Invalid Signatures in Pairing-Based Batches," in *Cryptography and Coding*, S. D. Galbraith, Ed., 2007.
[71] P. S. L. M. Barreto and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order," in *Selected Areas in Cryptography*, 2006.
[72] SCIPR Lab, "libff," https://github.com/scipr-lab/libff, 2016, Accessed: 2018-07-28.
[73] ——, "libfqfft," https://github.com/scipr-lab/libfqfft, 2016, Accessed: 2018-07-28.
[74] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, Sep 2012.
[75] S. Bowe, A. Gabizon, and M. D. Green, "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK," in *FC'18*, 2018.
[76] S. Bowe, A. Gabizon, and I. Miers, "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model," Cryptology ePrint Archive, Report 2017/1050, 2017, https://eprint.iacr.org/2017/1050.
[77] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification," https://github.com/zcash/zips/blob/master/protocol/protocol.pdf, Accessed: 2017-11-17.
[78] J. Groth, "Short Pairing-Based Non-interactive Zero-Knowledge Arguments," in *ASIACRYPT'10*.
[79] D. Catalano and D. Fiore, "Vector Commitments and Their Applications," in *PKC'13*.
[80] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A Cryptocurrency with Stateless Transaction Validation," Cryptology ePrint Archive, Report 2018/968, 2018, https://eprint.iacr.org/2018/968.
[81] D. Boneh, B. Bünz, and B. Fisch, "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains," in *CRYPTO'19*.
[82] A. Kate, G. M. Zaverucha, and I. Goldberg, "Polynomial commitments," Tech. Rep., 2010. [Online]. Available: https://pdfs.semanticscholar.org/31eb/add7a0109a584cfbf94b3afaa3c117c78c91.pdf

APPENDIX

A. AMT Prover Time and Proof Sizes

We will restrict ourselves to our $n = 2^m$ and $\deg \phi = t-1 < n$ setting. We first show that computing our optimized, roots-of-unity-based AMT takes $O(n \log t)$ time (see §III-B3). The key observation is that, when computing the AMT, divisions at higher levels (i.e., closer to the root) in the tree are *trivial* and need not be performed. Specifically, at sufficiently high levels, the degree of the divisors (i.e., accumulators) are larger than the degrees of the dividends (i.e., remainders), and always give quotients equal to zero. Since zero quotients can be easily recreated by verifiers, their commitments need not be included in the proof. We expand on this next.

Let us number levels differently, from $\log n$ (the root) to 0 (the leaves), so that level i has $n/2^i$ nodes, each with an accumulator of degree 2^i. Now, let k be the smallest value such that $2^k \leq \deg \phi < 2^{k+1}$. In other words, k is the level at which accumulator degrees are $\leq \deg \phi$ and thus divisions are non-trivial. Put differently, each node on level k will be the root node of an (authenticated) multipoint evaluation (sub)tree. We argue that the time to compute any one such subtree is $O(2^k \log 2^k)$ and, since there are $n/2^k$ such subtrees, the final AMT takes $O(n \log 2^k) = O(n \log t)$ time since $2^k \leq t-1 = \deg \phi$. We prove this inductively next.

At the root node of a level k subtree, the dividend $d_k = \phi$ has $\deg d_k < 2^{k+1}$ (by definition of k above). The accumulator a_k has $\deg a_k = 2^k$. Thus, the quotient $q_k = d_k / a_k$ will have $\deg q_k = \deg d_k - \deg a_k < 2^{k+1} - 2^k = 2^k$ and the remainder $r_k = d_k \bmod a_k$ will have $\deg r_k < \deg a_k = 2^k$. The division at this level will only take $O(\deg d_k) = O(2^{k+1})$ time, thanks to the $(x^{2^k} + c)$ form of a_k. Committing to the quotient will take $O(2^k)$ time. To summarize, at level k we are doing $O(2^{k+1})$ work and $\deg d_k < 2^{k+1}, \deg a_k = 2^k, \deg q_k < 2^k, \deg r_k < 2^k$. Next, we argue that the amount of work *per node* on level $k-1$ is half the work per node at level k. This is because (1) the dividend d_{k-1} is set to the remainder r_k from the parent, so $\deg d_{k-1} < 2^k$, (2) $\deg a_{k-1} = 2^{k-1}$, (3) $\deg q_{k-1} = \deg d_{k-1} - \deg a_{k-1} < 2^k - 2^{k-1} = 2^{k-1}$ and (4) $\deg r_{k-1} < \deg a_{k-1} = 2^{k-1}$. Thus, at level $k-1$, the division takes $O(2^k)$ time and committing to the quotient takes $O(2^{k-1})$ time. As a result, the time to compute the subtree can be expressed as $T(2^{k+1}) = 2T(2^{k+1}/2) + O(2^{k+1}) = O(2^k \log 2^k)$.

Finally, an AMT proof is $O(\log t)$-sized. Recall that quotients in the AMT are non-zero only at levels k and below, where $2^k \leq t-1 < 2^{k+1}$. Thus, an AMT proof will only have non-zero quotients at levels $k, k-1, k-2, \ldots, 1, 0$. Since $k = \lfloor \log_2(t-1) \rfloor$ the exact proof size is $\lfloor \log_2(t-1) \rfloor + 1$ group elements.

B. Cryptographic Assumptions

Let $\mathsf{poly}(\cdot)$ denote any function upper-bounded by some univariate polynomial.

Definition A.1 (Bilinear pairing parameters)**.** Let $\mathcal{G}(\cdot)$ be a randomized polynomial algorithm with input a security parameter λ. Then, $\langle \mathbb{G}, \mathbb{G}_T, p, g, e \rangle \leftarrow \mathcal{G}(1^\lambda)$ are called *bilinear pairing parameters* if $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order p where discrete log is hard, $\mathbb{G}$ has generator g and if e is a bilinear map, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that $\mathbb{G}_T = \langle e(g,g) \rangle$.

Definition A.2 (ℓ-Strong Bilinear Diffie-Hellman (SBDH) Assumption)**.** Given as input security parameter 1^λ, bilinear pairing parameters $\langle \mathbb{G}, \mathbb{G}_T, p, g, e \rangle \leftarrow \mathcal{G}(1^\lambda)$, public parameters $\mathsf{PP}_q(g;\tau) = \langle g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^\ell} \rangle$ where $\ell = \mathsf{poly}(\lambda)$ and τ is chosen uniformly at random from $\mathbb{Z}_p^*$, no probabilistic polynomial-time adversary can output a pair $\langle c, e(g,g)^{\frac{1}{\tau+c}} \rangle$ for some $c \in \mathbb{Z}_p$, except with probability negligible in λ.

Definition A.3 (ℓ-Polynomial Diffie-Hellman (polyDH) Assumption)**.** Given as input security parameter 1^λ, bilinear pairing parameters $\langle \mathbb{G}, \mathbb{G}_T, p, g, e \rangle \leftarrow \mathcal{G}(1^\lambda)$, public parameters $\mathsf{PP}_q(g;\tau) = \langle g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^\ell} \rangle$ where $\ell = \mathsf{poly}(\lambda)$

and τ chosen uniformly at random from $\mathbb{Z}_p^*$, no probabilistic polynomial-time adversary can output $(\phi(x), g^{\phi(\tau)}) \in \mathbb{Z}_p[X] \times \mathbb{G}$, such that $2^\lambda > \deg \phi > \ell$, except with probability negligible in λ.

C. AMT Proofs are Computationally Hiding and Binding

Recall from [14] that a polynomial commitment scheme consists of six algorithms: Setup, Commit, Open, VerifyPoly, CreateWitness, VerifyEval. We show our modified KZG scheme with AMT proofs satisfies *computational hiding* (see Definition A.5) under the discrete log (DL) assumption and *evaluation binding* (see Definition A.4) under the ℓ-Strong Bilinear Diffie-Hellman (ℓ-SBDH) assumption. These properties were originally defined in [14]. We prove these properties hold for a more general scheme that builds AMTs for an arbitrary set X of n points (rather than just for the set of roots of unity). For this scheme, Setup returns not only ℓ-SDH public parameters, but also the accumulator commitments necessary to verify AMT proofs. In other words, given an evaluation point $x^* \in X$, verifiers have access to accumulators $\{g^{a_w(\tau)}\}_{w \in \mathsf{path}(x^*)}$ necessary to verify x^*'s AMT proof.

Definition A.4 (Evaluation binding)**.** $\exists$ negligible function $\mathsf{negl}(\cdot)$, $\forall$ security parameters λ, $\forall \ell > 0, \forall$ adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \ell), \\ (c, x_0, \phi(x_0), \pi, \phi'(x_0), \pi') \leftarrow \mathcal{A}(\mathsf{pp}) : \\ \mathsf{VerifyEval}(\mathsf{pp}, c, x_0, \phi(x_0), \pi) = 1 \wedge \\ \mathsf{VerifyEval}(\mathsf{pp}, c, x_0, \phi'(x_0), \pi') = 1 \wedge \\ \phi(x_0) \neq \phi'(x_0) \end{array}\right] = \mathsf{negl}(\lambda)$$

Definition A.5 (Computational hiding)**.** Given pp randomly generated via $\mathsf{Setup}(1^\lambda, d)$, $c \in \mathbb{G}$, $\phi \in_R \mathbb{F}_p[X]$ of degree d and $(x_i, \phi(x_i), \pi_i)_{i=1}^d$ for distinct x_i's such that $\mathsf{VerifyEval}(\mathsf{pp}, c, x_i, \phi(x_i), \pi_i) = 1, \forall i \in [d]$, no adversary $\mathcal{A}$ can output $\phi(\hat{x})$ for any $\hat{x}$ where $\hat{x} \neq x_i, \forall i \in [d]$, except with probability negligible in the security parameter λ.

Computational hiding proof. Suppose there exists an adversary $\mathcal{A}$ that breaks computational hiding and outputs $\phi(\hat{x})$ for an unqueried $\hat{x} \neq x_i, \forall i \in [d]$. Then, we show how to build an adversary $\mathcal{B}$ that takes as input a random discrete log instance (g, g^a) and uses $\mathcal{A}$ to break it and output a. (Our proof is in the same style as Kate et al.'s proof for $\mathsf{PolyCommit}_{\mathsf{DL}}$'s computational hiding [82].)

$\mathcal{B}$ runs $\mathsf{Setup}(1^\lambda, d)$ which picks $\tau \in_R \mathbb{F}_p$ and outputs public parameters $\mathsf{pp} = \mathsf{PP}_d(g; \tau)$. Importantly, since $\mathcal{B}$ runs the setup he will know the trapdoor τ. Then, $\mathcal{B}$ picks random points $(x_i, y_i) \in X \times \mathbb{F}_p, \forall i \in [0, d]$ with distinct x_i's, except he sets $y_0 = a$. (Since $\mathcal{B}$ does not know a, he just sets $g^{y_0} = g^a$.) Note that $(x_i, y_i)_{i=0}^d$ determines a degree d polynomial ϕ where $\phi(x_i) = y_i, \forall i \in [0, d]$. Since $\mathcal{B}$ does not know a (only g^a), it will interpolate ϕ's commitment $g^{\phi(\tau)}$ "in the exponent" as:

$$g^{\phi(\tau)} = \prod_{i \in [0,d]} (g^{y_i})^{\mathcal{L}_i^T(\tau)}$$

Here $T = \{x_i\}_{i \in [0,d]}$ and recall that $\mathcal{L}_i^T(\tau) = \prod_{j \in T, j \neq i} (\tau - x_j)/(x_i - x_j)$ (see §II-4). To summarize, $\mathcal{B}$ "embeds" the (g, g^a) challenge in an (unknown-to-$\mathcal{B}$-but-determined) polynomial ϕ with commitment $c = g^{\phi(\tau)}$.

Next, $\mathcal{B}$ has to simulate AMT proofs π_i for $y_i = \phi(x_i), \forall i \in [d]$. To do this, recall that at each node w in the AMT, we have quotient and remainder polynomials q_w, r_w such that $r_{\mathsf{parent}(w)} = q_w a_w + r_w$ (see Figure 1). Also, recall that $\mathcal{B}$ knows τ so he can compute accumulator evaluations $a_w(\tau), \forall$ nodes w in the AMT. Now, $\mathcal{B}$ can simulate proofs as follows.

For the root node $w = \varepsilon$, we have $r_{\mathsf{parent}(\varepsilon)} = \phi$, so $\mathcal{B}$ picks a random $r_\varepsilon(\tau) \in \mathbb{F}_p$, and computes the root quotient commitment as $g^{q_\varepsilon(\tau)} = (g^{\phi(\tau)}/g^{r_\varepsilon(\tau)})^{\frac{1}{a_\varepsilon(\tau)}}$. At the next level, consider the children nodes u and v of the root ε. For each child $w \in \{u, v\}$, $\mathcal{B}$ must commit to a quotient q_w that satisfies $r_\varepsilon(\tau) = q_w(\tau) a_w(\tau) + r_w(\tau)$ for some r_w. So $\mathcal{B}$ proceeds similarly: for each child $w \in \{u, v\}$, he picks a random $r_w(\tau)$ and computes a commitment $g^{q_w(\tau)} = (g^{r_\varepsilon(\tau)}/g^{r_w(\tau)})^{\frac{1}{a_w(\tau)}}$. $\mathcal{B}$ will do this recursively until it reaches leaf nodes in the AMT. For each leaf l, instead of picking $r_l(\tau)$ randomly, $\mathcal{B}$ will set it to the y_i corresponding to that leaf. This way, $\mathcal{B}$ can simulate quotient commitments $\{g^{q_w(\tau)}\}_{w \in \mathsf{path}(x_i)}$ for all $i \in [d]$ that pass the AMT proof verification in Equation (3).

Next, $\mathcal{B}$ calls $\mathcal{A}$ with $(\mathsf{pp}, c, (x_i, y_i, \pi_i)_{i=1}^d)$ as input, hoping that $\mathcal{A}$ outputs another point $\hat{x}$ and its evaluation $\phi(\hat{x})$. Since $\mathcal{A}$ breaks computational hiding, this happens with non-negligible probability. (Note that $\mathcal{B}$ can check $\mathcal{A}$ succeeds by interpolating $g^{\phi(\hat{x})}$ "in the exponent".) When $\mathcal{A}$ succeeds, if $\hat{x} = x_0$, then $a = \phi(\hat{x})$, so $\mathcal{B}$ breaks discrete log on (g, g^a). Otherwise, $\mathcal{B}$ uses the first d points $(x_i, y_i = \phi(x_i))_{i \in [d]}$ and this new distinct $(\hat{x}, \phi(\hat{x}))$ point to interpolate ϕ and as a result obtain $a = \phi(x_0)$. (Recall that, by Definition A.5, we have $\hat{x} \neq x_i, \forall i \in [d]$.) As a result, $\mathcal{B}$ breaks discrete log on (g, g^a).

Evaluation binding proof. Suppose there exists an adversary $\mathcal{A}$ that outputs a commitment c, with two contradicting proofs π, π' attesting that $\phi(k)$ is equal to v and v', respectively. We show how to build another adversary $\mathcal{B}$ that breaks q-SBDH. First, $\mathcal{B}$ runs $\mathcal{A}$ to get $(c, \pi, \pi', \phi(k), v, v')$. Let $W = \mathsf{path}(k)$ denote the nodes along k's path in the AMT. Let $(\pi_i)_{i \in W}$ denote the quotient commitments in π. Similarly, let $(\pi_i')_{i \in W}$ denote the quotient commitments in π'. Since both proofs verify, we have:

$$e(c, g) = e(g^v, g) \prod_{i \in W} e(\pi_i, g^{a_i(\tau)})$$
$$e(c, g) = e(g^{v'}, g) \prod_{i \in W} e(\pi_i', g^{a_i(\tau)})$$

Dividing the first equation by the second, we get:

$$1_{\mathbb{G}_T} = \frac{e(g^v, g)}{e(g^{v'}, g)} \frac{\prod_{i \in W} e(\pi_i, g^{a_i(\tau)})}{\prod_{i \in W} e(\pi_i', g^{a_i(\tau)})} \Leftrightarrow$$
$$1_{\mathbb{G}_T} = e(g^{v - v'}, g) \prod_{i \in W} \frac{e(\pi_i, g^{a_i(\tau)})}{e(\pi_i', g^{a_i(\tau)})} \Leftrightarrow$$
$$e(g^{v' - v}, g) = \prod_{i \in W} e(\pi_i / \pi_i', g^{a_i(\tau)})$$

Now, recall that one of the accumulators $(a_i(x))_{i\in W}$ is the monomial $(x-k)$, and all the other $a_i(x)$'s contain $(x-k)$ as a term, which means it can be factored out of them. Thus, since $(x-k)$ perfectly divides all $a_i(x)$'s, let $r_i(x) = a_i(x)/(x-k), \forall i \in W$. Importantly, the adversary $\mathcal{B}$ can compute all $r_i(x)$'s in polynomial time, since it can reconstruct all the accumulator polynomials $(a_i(x))_{i\in W}$. As a result, $\mathcal{B}$ can compute all commitments $(g^{r_i(\tau)})_{i\in W}$. Then, $\mathcal{B}$ breaks ℓ-SBDH as follows:

$$e(g^{v'-v}, g) = \prod_{i\in W} e(\pi_i/\pi_i', g^{r_i(\tau)(\tau-k)})$$

$$e(g^{v'-v}, g) = \prod_{i\in W} e(\pi_i/\pi_i', g^{r_i(\tau)})^{(\tau-k)}$$

$$e(g^{v'-v}, g) = \left[\prod_{i\in W} e(\pi_i/\pi_i', g^{r_i(\tau)})\right]^{(\tau-k)}$$

$$e(g,g)^{\frac{1}{\tau-k}} = \left[\prod_{i\in W} e(\pi_i/\pi_i', g^{r_i(\tau)})\right]^{\frac{1}{v'-v}}$$

D. Polylogarithmic DKG Configurations

As discussed in §I-A, Canny and Sorkin presented a *sparse matrix* DKG with $O(m^3)$ time and communication per player, where m is a group size [47]. Depending on the desired threshold (t, n) and the number f of malicious nodes tolerated, the group size m can be as small as $\Theta(\log^3(n))$. Unfortunately, for f sufficiently close to t, the group size becomes too large, approaching $n/2$ (see Table II).

TABLE II
THE GROUP SIZE m AND FOR VARIOUS (t, n) SPARSE MATRIX DKGS WITH f FAILURES TOLERATED.

ε	Group size m	$f = (1/2-\varepsilon)n$	$t = (1/2+\varepsilon)n$	n
0.1	50,598	26,214	39,321	65,536
0.15	14,222	22,937	42,598	65,536
0.2	5,691	19,660	45,875	65,536
0.25	2,735	16,384	49,152	65,536
0.3	1,475	13,107	52,428	65,536
0.33	1,057	11,141	54,394	65,536
0.4	505	6,553	8,982	65,536
0.05	445,909	471,859	576,716	1,048,576
0.1	53,022	419,430	629,145	1,048,576
0.15	14,949	367,001	681,574	1,048,576
0.2	5,977	314,572	734,003	1,048,576
0.25	2,871	262,144	786,432	1,048,576
0.3	1,548	209,715	838,860	1,048,576
0.33	1,107	178,257	870,318	1,048,576
0.4	527	104,857	943,718	1,048,576