DSig: Breaking the Barrier of Signatures in Data Centers

Marcos K. Aguilera, *VMware Research Group;* Clément Burgelin, Rachid Guerraoui, and Antoine Murat, *École Polytechnique Fédérale de Lausanne* (*EPFL*)*;* Athanasios Xygkis, *Oracle Labs;* Igor Zablotchi, *Mysten Labs*

https://www.usenix.org/conference/osdi24/presentation/aguilera

This paper is included in the Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-40-3

Open access to the Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation is sponsored by

DSig: Breaking the Barrier of Signatures in Data Centers

Marcos K. Aguilera[1], Clément Burgelin[2], Rachid Guerraoui[2], Antoine Murat[2], Athanasios Xygkis[3], and Igor Zablotchi[4]

[1]*VMware Research Group*
[2]*École Polytechnique Fédérale de Lausanne (EPFL)*
[3]*Oracle Labs*
[4]*Mysten Labs*

Abstract

Data centers increasingly host mutually distrustful users on shared infrastructure. A powerful tool to safeguard such users are digital signatures. Digital signatures have revolutionized Internet-scale applications, but current signatures are too slow for the growing genre of microsecond-scale systems in modern data centers. We propose DSig, the first digital signature system to achieve single-digit microsecond latency to sign, transmit, and verify signatures in data center systems. DSig is based on the observation that, in many data center applications, the signer of a message knows most of the time who will verify its signature. We introduce a new hybrid signature scheme that combines cheap single-use hash-based signatures verified in the foreground with traditional signatures pre-verified in the background. Compared to prior state-of-the-art signatures, DSig reduces signing time from 18.9 to 0.7 µs and verification time from 35.6 to 5.1 µs, while keeping signature transmission time below 2.5 µs. Moreover, DSig achieves 2.5× higher signing throughput and 6.9× higher verification throughput than the state of the art. We use DSig to (a) bring auditability to two key-value stores (HERD and Redis) and a financial trading system (based on Liquibook) for 86% lower added latency than the state of the art, and (b) replace signatures in BFT broadcast and BFT replication, reducing their latency by 73% and 69%, respectively.

1 Introduction

Digital signatures are used in many distributed protocols that have revolutionized the Internet through many use cases, such as enabling digital certificates [83], bootstrapping authentication protocols [30, 89], securing and auditing transactions [22, 71], tolerating Byzantine failures [4, 6, 18], and verifying software authenticity [28, 57]. Signatures are irrefutable proofs that someone produced a message, and these proofs can be verified by third parties. This property distinguishes signatures from message authentication codes (MACs) and authenticated symmetric encryption (e.g., SSL/TLS) [54]. Today's signatures are however too expensive for the growing genre of microsecond-scale systems in data centers. Even the fastest signature scheme, EdDSA [17, 50], accounts for 79–95.6% of the latency of applications such as auditable key-value stores, BFT broadcast, and BFT replication (Figure 1).

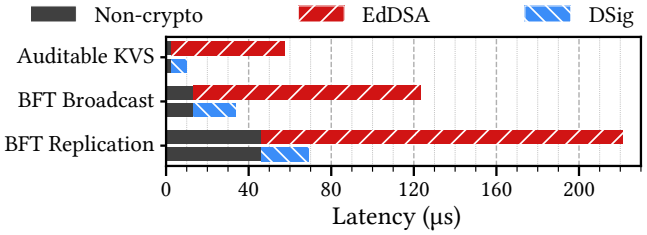

Figure 1: Median latency breakdown of an auditable key-value store (based on HERD [51], §6), a BFT broadcast primitive (CTB [3], §6), and a BFT replication system (uBFT [3], §6) when processing small requests using either EdDSA [50] (state of the art) or DSig. DSig reduces the cryptographic overhead by 86%, 82%, and 87%, respectively, and reduces the overall latency by 83%, 73%, and 69%, respectively.

We propose DSig, the first digital signature system to achieve single-digit microsecond performance for data centers. A key insight underlying the design of DSig is that, in many data center applications, signatures are issued and verified by parties known a priori in the common case, so signers and verifiers can exchange useful information beforehand and do part of the computation before knowing the messages to be signed, thereby reducing the latency of subsequent signature generation and verification. We use this observation to introduce a new hybrid online-offline signature system [37]. Hybrid schemes combine a traditional signature scheme that is slow but can sign many messages, with a hash-based signature scheme (HBSS) that is fast but can sign only one or a few messages. The traditional signature is used to validate a batch of HBSS key pairs, each of which signs one or a few messages. Hybrid signatures have been studied extensively in theory, but practical work has focused only on improving throughput for low-compute devices with limited bandwidth,

Table 1: Comparison of EdDSA and DSig in terms of latency to sign, transmit (tx) and verify; per-core sign and verify throughput; signature size; and background network traffic per signature with a single verifier.

	Latency (µs)			Tput (Kops)		Sig size	Bg Net
	Sign	Tx	Verify	Sign	Verify	(B)	(B/Sig)
EdDSA	18.9	1.1	35.6	53	28	64	0
DSig	0.7	2.0	5.1	131	193	1,584	33

on low-security signatures, or on tiny messages (§9).

Using hybrid signatures to achieve low latency and high throughput in data centers poses a number of challenges. First, the traditional part of hybrid signatures is compute-heavy and can impact latency. Second, hybrid signatures involve frequent key pair generation, which can exhaust compute resources and impact throughput. Third, the actual performance of hybrid signatures deviates from their theoretical models, as real performance requires careful consideration of microarchitectural effects (e.g., caching, prefetching) rather than simply the amount of computation. Fourth, to perform well, hybrid signatures need to be configured with an appropriate HBSS whose thousands of parameter combinations provide complex trade-offs between number of hash computations, signature size, and frequency of key pair generation; most parameter choices exceed our performance goals, the available network bandwidth, the computational resources, or all of the above.

We address these challenges as follows. First, we use hints about who will verify a message in the common case to preprocess the compute-heavy traditional signatures. Second, we use traditional signatures to sign and verify batches of HBSS public keys, thus amortizing the cost of their authentication, while hiding the latency introduced by batching from the critical path. Third, we study the real performance of HBSSs to determine the best schemes to use, and we discover non-intuitive cases where fewer hash computations actually harm performance. Fourth, we identify two promising HBSSs to use in DSig, W-OTS$^+$ [46] and HORS [84], and we explore their parameters in depth to understand how they affect latency, throughput, and resource usage; we give a recommended configuration of DSig that strikes a good trade-off.

We integrate DSig with five applications: two key-value stores (HERD [51] and Redis [87]), a financial trading system (based on Liquibook [73]), a BFT broadcast primitive (CTB [3]), and a BFT replication system (uBFT [3]). We use DSig to provide auditability through a signed security log in HERD, Redis, and Liquibook; and to replace the signatures used in CTB and uBFT to thwart Byzantine behavior.

We evaluate DSig and its applications. We find that DSig can sign, transmit, and verify a signature in 7.7 µs total, which is 7.2× faster than EdDSA [50], the fastest traditional signature scheme [17] (Table 1). DSig achieves 2.5× and 6.9× higher throughput than EdDSA for signing and verifying. DSig benefits applications significantly. In HERD, Redis, and Liquibook, DSig brings auditability with an added latency of less than 8 µs per operation, a reduction of 86% in overhead compared to EdDSA. In CTB, DSig reduces the broadcast latency by 73%, from 123 µs to 34 µs. In uBFT, DSig reduces the replication latency by 69%, from 221 µs to 69 µs.

The price for using DSig is as follows. First, to get the best performance, DSig needs a priori knowledge of where and when signatures are verified (DSig still works without such knowledge, but it is slower). Second, DSig requires extra bandwidth and space to transmit and store its larger ≈1.5 KiB signatures. This is a small cost in data-center systems, which have low-latency high-bandwidth networks and abundant storage, but DSig could be ill-suited for other settings, such as some wide-area systems.

In summary, our contributions are the following:

- We propose DSig, a new digital signature system targeted at microsecond-scale applications in data centers. DSig combines hash-based signatures, traditional signatures, and novel techniques to reduce latency in the critical path while achieving high throughput.
- We analyze and evaluate DSig's large parameter space for low latency at high throughput, and identify a configuration that best fits most scenarios.
- We implement DSig and integrate it into several applications: two key-value stores, a financial trading system, BFT broadcast, and a BFT replication system.
- We evaluate DSig and its applications. DSig significantly improves signature performance compared to EdDSA, the state of the art. These enhancements directly benefit the applications by providing better end-to-end latency and throughput, and by bringing auditability to the microsecond scale.

DSig is open source, available at `https://github.com/LPD-EPFL/dsig`.

2 Setting and Goals

Setting. We target microsecond-scale applications [2, 3, 11, 19, 31, 41, 76, 80–82] with a few tens of servers within the same data center—a scale that addresses the computing needs of many enterprises. These systems have a network with low latency (≈1 µs) and high bandwidth (100s of Gbps or even Tbps [74]).

Goals. Our goal is to achieve faster digital signatures to broaden their usability. We do not seek general-purpose signatures for all settings (wide area networks, mobile networks, embedded systems), but rather seek schemes that provide the right trade-offs in modern data centers. We seek signatures that provide the industry-standard level of security (128 bits).

Digital signatures are important because they are *transferable*: if Alice signs a message to Bob, Bob can prove to Carol that Alice indeed signed it (§3). This property makes

signatures more powerful than mechanisms such as SSL/TLS or MACs, which provide only symmetric authenticated channels between Alice and Bob [54], which do not suffice for the applications we consider (§6). Signatures help tackle distrustful parties in distributed protocols for a wide variety of use cases: securing transactions, enabling digital certificates, bootstrapping authentication of users and services [30, 89], verifying software authenticity [28, 57], providing integrity of audit logs [22, 71], tolerating Byzantine failures [4, 6, 18], etc. Many of these use cases apply to microsecond-scale applications, as such systems increasingly bring together mutually distrustful users on shared infrastructure [48, 88]. For example, microservices can benefit from Byzantine fault tolerance [3]; signed transactions can provide auditability in high-frequency trading systems; and signed logs can provide a legal trail in high-stakes settings.

Threat model. Our threat model is standard for digital signatures [54]. Malicious entities can intercept, store, inject, delay, and alter messages. We assume the security of standard cryptography building blocks: traditional digital signature schemes (Ed25519 [50]), hash-based signature schemes (W-OTS$^+$ [46] and HORS [84]), and cryptographic hashes (SHA256 [77], Haraka (v2) [55], and BLAKE3 [75]).

3 Background

3.1 Digital Signature Schemes

A digital signature scheme (DSS) has a key pair consisting of a public key *PK* and a secret key *SK*. A signer *s* uses *SK* to sign a message *m*, producing a signature σ for *m*. The signature σ allows a party who knows *PK* (and knows that *PK* belongs to *s*) to verify that *m* was signed by *s*.

DSSs provide *authenticity*, *integrity*, *public verifiability* and *non-repudiation* of messages [54]. Authenticity means that a party with a message and its signature can verify the identity of the message's signer. Integrity means that the party can verify that the message matches the message that was signed. Public verifiability means that only *m*, σ, and *PK* are needed to verify the authenticity and integrity of *m*. Signatures are transferable: a party can use σ and *m* to convince anyone who knows *PK* that *m* is authentic (and typically *PK* is published, so everyone knows *PK*). This property differentiates digital signatures from other mechanisms, such as message authentication codes (MACs), vectors of MACs, authenticated channels (e.g., SSL/TLS), and symmetric encryption (e.g., AES). Non-repudiation means that *s* cannot deny the signing of *m* once its signature σ is known. Non-repudiation implies that signatures are non-forgeable: without knowing *SK*, it is computationally infeasible to produce a signature σ which passes verification with *PK*.

3.2 The Cryptographic Barrier

After DSSs were proposed [33], many schemes followed: RSA [85], ECDSA [49], EdDSA [50], etc. These schemes rely on the hardness of certain problems (factoring, discrete logarithms) under sophisticated arithmetic (e.g., modular on elliptic curves). They seek to provide strong security and small time to sign and verify. For example, the state-of-the-art 128-bit-secure EdDSA takes 19 µs to sign and 36 µs to verify a small message on modern CPUs (Table 1).

State-of-the-art DSSs are too slow for modern data centers: even the fastest schemes are an order of magnitude slower than network latencies [17] due to the use of sophisticated arithmetic, which consumes CPU and cannot be parallelized. This slowness makes traditional DSSs prohibitive for distributed protocols, microservices, and applications that run at the microsecond scale, which need to check the signatures of messages before acting upon them. For example, signature-based BFT protocols must check signatures before taking safety-critical steps such as computing a quorum intersection, voting, vouching for a message, deciding on a majority value, etc; similarly, auditable applications must check signatures before executing requests to provide auditability (§6).

Signing or verifying messages in batches can improve the throughput of DSSs, but batching further increases latency and is thus ill-suited for latency-critical applications.

3.3 Hash-Based and Hybrid Signatures

Hash-based signature schemes (HBSSs) were proposed by Lamport [59]. They are DSSs that avoid advanced arithmetic by using only cryptographic hashes. Hashes are advantageous because they can be computed quickly: modern algorithms (e.g., Haraka [55] and BLAKE3 [75]) can hash a small message in less than 100 ns on modern CPUs. In some HBSSs (e.g., HORS [84], W-OTS$^+$ [46]), signature generation and verification execute at the microsecond scale, as they require few hash computations.

To explain HBSSs, we overview the HORS scheme (Figure 2), which, whilst simple, illustrates the key ideas of HBSSs. The secret key *SK* for signing is an array of *t* random secrets (*t* is a parameter), while the public key *PK* for verify-

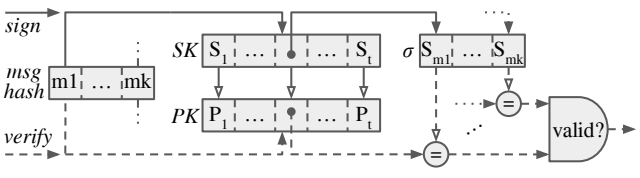

Figure 2: The HORS hash-based signature scheme. Solid lines convey the path taken to sign a message, while dashed lines convey the path to verify a signature. Hollow arrows indicate cryptographic hashes.

ing is the concatenation of the hash of each secret in SK. To sign a message, the signer hashes it with a salt into a string m, splits m into k substrings (k is a parameter), treats each substring as an index into SK, and concatenates the indexed secrets to obtain the signature σ. A verifier hashes the message with the salt and uses the substrings to index into PK. Then, the verifier hashes each secret in σ and checks that they match the indexed elements of PK. This scheme is secure because it is hard to (1) find messages that index the same secrets or (2) reveal secrets without being the signer. Other fast HBSSs, such as W-OTS$^+$ [46], are similar to HORS in that they sign by revealing a subset of the private key; as a result, they are limited to signing one or a few messages.

To overcome this limitation and sign an unlimited number of messages, hybrid signature schemes [37] combine HBSSs with traditional schemes. To sign a message m, a hybrid scheme concatenates an HBSS signature on m with the HBSS public key signed using a traditional signature. To verify a signature, the scheme verifies the HBSS signature of m and the traditional signature of the HBSS public key.

3.4 Challenges

Hybrid signatures were studied extensively in theory, but their application focused either on improving throughput in low-compute low-bandwidth devices, or on low-security signatures, or on tiny messages with only a few bits (§9). To use them in a high-performance data center setting, we must tackle several challenges.

Efficient signature verification. To verify a hybrid signature, we must check both its HBSS signature and its traditional signature. Traditional signatures, however, have high verification latency. We need new mechanisms to avoid the traditional signature verification in the critical path.

Frequent key generation. Because an HBSS key pair can be used only once or a few times, hybrid schemes need to frequently generate and sign new HBSS key pairs. This can become a bottleneck as it impairs signature throughput and, ultimately, its latency. We need new techniques to improve throughput while minimizing latency on the critical path.

Practical performance. We evaluate the performance of hybrid schemes and find that it does not match their theoretical analysis. The latter is based on simple metrics, namely the size of signatures and the number of hash calculations in the critical path. However, due to microarchitectural effects (e.g., CPU cache size, prefetching), optimal configurations in theory perform suboptimally in practice, and optimizations that target solely the theoretical metrics sometimes do not work.

Large parameter space. Hybrid signature schemes have many configuration options: the choice of the traditional scheme, choice of the HBSS, and their respective parameters. As a result, we are faced with thousands of options that provide different trade-offs in network bandwidth, computational resources, throughput, and latency characteristics.

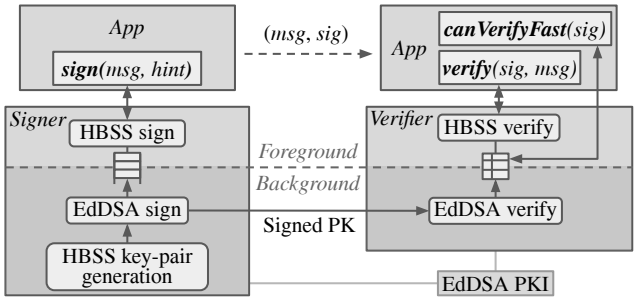

Figure 3: Architecture of DSig.

4 Design of DSig

We present the architecture of DSig (short for "Data center Signatures"), highlighting its extended interface and computing planes (§4.1). We then describe DSig's hybrid signature scheme (§4.2), its security (§4.3), and throughput optimizations (§4.4).

4.1 Architecture

Figure 3 depicts the architecture of DSig. Each process has a public-private key pair of a traditional signature scheme, where the public key is made available to other parties via a public key infrastructure (PKI). For the traditional signature scheme, we choose EdDSA [50] because it is the fastest such scheme [17]. The PKI can be as simple as an administrator pre-installing the keys, or it can be a full-fledged system.

DSig augments the interface of digital signatures (*sign* and *verify* functions) in two ways. First, *sign* takes a hint with the set of processes that will likely verify the signature. The hint is optional: if omitted, it defaults to all known processes. The hint does not restrict who can verify a signature—parties not indicated in the hint can still verify the signature, albeit at a lower performance. Second, a new *canVerifyFast* function returns whether verification of a given signature will be fast. This function can mitigate denial-of-service attacks by letting applications prioritize the handling of fast signatures (§6).

Internally, DSig has two planes: *foreground* and *background*. The foreground plane provides the user library with a synchronous API to sign and verify messages, while the background plane asynchronously pre-generates and propagates new HBSS keys to be used by the foreground plane. DSig's general design can be used with a wide range of HBSSs (e.g., Lamport's [59], HORS [84], W-OTS [34], W-OTS$^+$ [46]). We provide a specific recommendation based on an extensive performance study (§5).

Foreground plane. To sign a message, the signer uses a fresh HBSS key pair and returns to the application a DSig signature, which includes the resulting HBSS signature and the

Algorithm 1: Signers' Pseudocode

```
1  # Signer's setup
2  verifier_groups = { ... } # Provided
3  for group in verifier_groups:
4    signed_keypairs[group] = Queue()

6  # Signer's background plane
7  whenever ∃ group | signed_keypairs[group].size < S:
8    sk, pk = hbss.generate_keypair()
9    pk_σ = {pk: pk, sig: eddsa.sign(pk)}
10   multicast <HBSS_PUBKEY, pk_σ> to group
11   signed_keypairs[group].push((sk, pk_σ))

13 # Signer's foreground plane
14 def sign(msg, hint):
15   group = smallest group containing hint
16   sk, pk_σ = signed_keypairs[group].pop()
17   hbss_sig = hbss.sign(msg, sk)
18   return <SIG, hbss_sig, pk_σ>
```

Algorithm 2: Verifiers' Pseudocode

```
19 # Verifier's setup
20 verified_pks = Cache()

22 # Verifier's background plane
23 upon deliver <HBSS_PUBKEY, pk_σ> from process p:
24   if eddsa.verify(pk_σ, eddsa_pk_of(p)):
25     verified_pks.add((pk_σ, p))

27 # Verifier's foreground plane
28 def verify(msg, <SIG, hbss_sig, pk_σ>, p):
29   if (pk_σ, p) not in verified_pks:
30     if not eddsa.verify(pk_σ, eddsa_pk_of(p)): # Slow
31       return false
32   return hbss.verify(msg, hbss_sig, pk_σ.pk)

34 def canVerifyFast(<SIG, _, pk_σ>, p):
35   return (pk_σ, p) in verified_pks
```

corresponding HBSS public key signed with EdDSA. Signing with EdDSA is slow, so the HBSS public key is pre-signed in the background plane. On the other side, the verifier checks the authenticity of the message using the HBSS signature and the included HBSS public key. The authenticity of the HBSS public key is checked by the background plane.

Background plane. The signer generates HBSS key pairs and EdDSA-signs them before forwarding them to the foreground plane. It also sends the EdDSA-signed HBSS public keys to the background plane of the likely verifiers. The latter verifies the authenticity of the HBSS public keys.

The background plane hides the latency of two slow steps: (1) HBSS key pair generation, and (2) EdDSA-signing and EdDSA-verifying the HBSS public keys.

Note that DSig preserves the transferability of signatures irrespective of the background plane. Because DSig hybrid signatures are self-standing (as they include the EdDSA-signed HBSS public key), the only requirement for signature verification is knowledge of the signer's EdDSA public key. The background plane merely boosts performance when a hint is correct, by letting a verifier pre-check an HBSS public key before it receives a signature that includes it.

4.2 Signing and Verifying in DSig

The logic of a DSig signer is shown in Algorithm 1. Each signer is configured with a list of *verifier groups*—groups of processes that are likely to verify the same signatures on their critical path (line 2). This list has a default group that contains all the processes in the system, but is otherwise application-dependent. In the applications we examined (§6), the list was small and obvious (e.g., individual groups containing one process each). Each verifier group is associated with a key-pair queue (lines 3–4).

In the background plane, whenever a group's queue size is below a threshold S (line 7), the signer generates a new HBSS key pair (line 8), and signs the public key using EdDSA (line 9). Empirically, we found that S=512 works well while consuming only 3 MiB of memory per group. Then, the signer multicasts the signed public key to the processes in the group (line 10). The signer next appends the private key with the EdDSA-signed public key to the queue for consumption in the foreground plane (line 11).

To sign a message, the signer chooses the verifier group that matches the hint; if no group matches the hint, the signer picks the smallest group containing the hint (line 15). Then, it gets a new HBSS key pair from this group's queue (line 16). Next, the signer computes an HBSS signature of the message using the private key obtained from the queue (line 17). The DSig signature comprises the HBSS signature of the message together with its EdDSA-signed HBSS public key (line 18).

The logic of a DSig verifier is shown in Algorithm 2. In the background plane, the verifier receives EdDSA-signed HBSS public keys (line 23), which it verifies (line 24) and stores in a cache (line 25). In our applications, we found that having the cache store the latest $2\times S$=1024 HBSS public keys from each signer worked well while consuming only 100 KiB of memory per signer. In the foreground plane, the verifier first consults its cache to see if it has a verified entry for the HBSS public key (line 29). If so, the verifier proceeds with checking the HBSS signature using the HBSS public key. In this code path, the verifier checks signatures as fast as the underlying HBSS verify (line 32), which is fast. Otherwise, the verifier also checks the EdDSA signature of the HBSS public key (line 30), so the verifier can operate even without the background plane. The verifier's *canVerifyFast* function (§4.1) simply checks whether a signed HBSS public key has already been verified (lines 34–35).

As with other signature schemes, DSig can support key revocation through revocation lists that applications check prior to signing or verifying messages [54].

4.3 Security

For preciseness of argument, we show the security of our recommended DSig configuration, which uses W-OTS$^+$ [46] as its underlying HBSS (§5). Specifically, we show that this configuration of DSig is Existentially Unforgeable under Chosen-Message Attacks (EUF-CMA) [39] and that it provides 128-bit security, which is safe by today's standards [7].

EUF-CMA security. We consider Chosen-Message Attacks (CMA) in which the attacker can query the target to sign arbitrary messages. More precisely, we consider *adaptive* CMA in which the attacker can query the target based on public key(s) and previously obtained signatures. We consider Existential Unforgeability (EUF), which means it should be computationally infeasible for an attacker to forge a signature on any message, except for messages that have already been signed by the target.

DSig is as secure as its parts. To forge a signature in DSig, an attacker must find a combination of message (not previously signed), W-OTS$^+$ public key, EdDSA signature, and W-OTS$^+$ signature that passes the `verify` function (Algorithm 2). We assume that EdDSA provides EUF-CMA security, as proved by Brendel *et al.* [20], and show how DSig's security reduces to the security of W-OTS$^+$:

1. The verifier's background plane caches only correctly EdDSA-signed public keys (Alg. 2 lines 23–25). From the EUF-CMA security of EdDSA, and since a correct signer EdDSA-signs only its own public keys, (Alg. 1 lines 8–9), for any correct signer s, the verifier caches only the W-OTS$^+$ public keys s generates.

2. If a public key is not cached, the verifier EdDSA-verifies it on the critical path (Alg. 2 lines 29–31). As in (1) above, for any correct signer s, this verification only succeeds for public keys s generates. Thus, for any correct signer s, `verify` cannot return *true* for public keys s does not generate.

3. Since, for any correct signer s, an attacker can only use a W-OTS$^+$ public key generated by s, forging a DSig signature reduces to forging a W-OTS$^+$ signature.

W-OTS$^+$ with Haraka and BLAKE3. Hülsing proved that W-OTS$^+$ is EUF-CMA-secure when using a hash-function family that is second-preimage resistant, undetectable, and one-way [46]. Like SPHINCS+ [16], we pick the Haraka [55] hash-function family which satisfies those properties and relies on the battle-tested AES round function. Similarly to SPHINCS+, we reduce the signed messages to 128-bit digests by hashing them salted with the W-OTS$^+$ public key and a random nonce. We do so using the well-established BLAKE3 [75] hashes. Finally, we tune W-OTS$^+$'s parameters to provide 128 bits of security when signing said 128-bit digests. More precisely, we set the size of secrets and public key elements to 144 bits, which, together with a W-OTS$^+$ depth of 4 (§5), provides a security level of 133.9 bits [46].

DSig's security level. Breaking DSig can be reduced to breaking either EdDSA, W-OTS$^+$, Haraka, or BLAKE3. The EdDSA signature scheme Ed25519 provides 128-bit security under practical attacks [14], and our configuration of W-OTS$^+$ provides 133-bit security. The security of both Haraka and BLAKE3 relies on well-studied components [7, 55] and to date, no attack has compromised their security.

4.4 Optimizing Throughput

DSig has a few throughput optimizations that do not significantly impact latency.

Speeding up key pair generation. Generating an HBSS key pair requires producing hundreds to thousands of secrets for the private key, and then hashing each secret for the public key. To produce secrets quickly, DSig collects entropy from the hardware at startup to get a truly random 256-bit seed, which DSig then salts with the key index and hashes using BLAKE3 to generate a digest with the size of the private key. To produce the public key quickly, DSig hashes the secrets using Haraka, which has a high-throughput implementation that optimizes instruction pipelining to compute multiple hashes efficiently.

Amortizing the cost of EdDSA signatures. EdDSA-signing every HBSS public key is slow and becomes a throughput bottleneck as each EdDSA sign-verify computation takes $\approx$55 µs [17]. DSig EdDSA-signs batches of HBSS public keys [86]. However, batching naively would increase the size of signatures, since the entire batch of HBSS public keys should be included in every DSig signature to make their EdDSA signature self-standing. DSig addresses this issue by arranging the batch of HBSS public keys into a Merkle tree [67] and EdDSA-signing its root. As a result, a DSig signature is composed of an HBSS signature, an HBSS public key, a Merkle inclusion proof, and the EdDSA signature of the Merkle root. The Merkle proof is a space-efficient way of proving that the included HBSS public key is part of the EdDSA-signed batch. As Merkle proofs require collision-resistant hashes, we use (the efficient) BLAKE3. The space efficiency of Merkle proofs comes at the computational cost of generating and verifying them. DSig moves most of this cost to the background plane of both signers and verifiers, which precompute and cache the full Merkle tree associated with a batch. Then, on the critical path, signing a key merely requires copying the subset of the tree that constitutes the Merkle proof, while verifying the Merkle proof consists of simple string comparisons. The efficiency of this scheme depends on the batch size, which we determine in §8.7.

Speeding up bulk verification. Verifying many signatures with no assistance from the background plane (e.g., when checking an audit log) requires checking the same EdDSA signatures many times. To speed up this process, EdDSA

signatures verified in the foreground plane are cached. A cache entry takes only ≈33 bytes, a tiny overhead, but saves ≈36 µs of computation on our hardware (Table 3).

Reducing background bandwidth. Sending signed public keys both ahead of time and within signatures consumes significant networking bandwidth. To nearly halve the bandwidth usage, DSig batches, EdDSA-signs, and sends only BLAKE3 digests of the public keys in the background plane. This optimization requires computing the public key digest during signature verification, which adds only ≈1.3 µs of latency.

5 Choice of HBSS

DSig can be used with many HBSSs, but its performance heavily depends on the actual HBSS used and how this HBSS is configured. In this section, we explain which HBSS we choose for DSig and how we configure it.

5.1 HBSS Requirements

Our choice of HBSS is guided by the following requirements.

Security. To provide 128-bit security, DSig needs an HBSS with the same or stronger security.

Low sign and verify latency. DSig executes HBSS *sign* and *verify* operations on the critical path. These operations must have microsecond-scale latency. This latency depends on the choice of the hash function, on the number of hashes they involve, and on microarchitectural effects.

Short signatures. At the microsecond scale, signatures cannot exceed a few KiB in length, as larger signatures incur significant transmission latency: we experimentally find that when sending small messages each extra KiB takes approximately an extra microsecond on a 100 Gbps network. Furthermore, large signatures significantly increase the bandwidth consumption when applications send small messages.

Compressible public keys. Recall that DSig signatures must include an HBSS public key in order to be self-standing (§4.2). However, the combination of an HBSS signature and its public key can be in the KiB range, which is undesirable. We thus seek HBSSs for which this combination can be compressed, leading to smaller DSig signatures.

Lightweight key generation. HBSS key generation mainly involves hash computations, the number of which depends on the HBSS and can bottleneck DSig's throughput. HBSS that use few hashes for key generation are thus desirable.

5.2 Analysis

HBSSs can be grouped in two classes: HBSSs with keys that can sign only one or a few messages [34, 46, 59, 84], and HBSSs with keys that can sign many messages [9, 15, 16, 21, 47, 58]. Only the first class provides low latency (the second focuses on quantum resistance). Within that class, we focus on the fastest HBSSs: HORS [84] and W-OTS$^+$ [46].

HORS. Recall that a HORS signature reveals a subset of its private key secrets determined by the message being signed (§3.3). Verifying a signature requires hashing the revealed secrets, checking that they match the public key, and checking that all the secrets mandated by the signed message were revealed. HORS has two parameters: the number k of secrets revealed in a signature and the number of times r that a key pair can be used. The size of HORS keys is proportional to r, so picking $r \geq 2$ presents no benefits and we set $r=1$. Smaller values of k, however, lead to fewer hash computations, and thus to lower latency in theory, but they require larger HORS public keys for the same security level. Large HORS public keys require compression to fit our signature size budget. We thus devise two compression techniques, described next.

The first technique shortens the embedded HORS public key by removing the elements that can be deduced from the HORS signature (top of Figure 4). We analyze this approach in the first part of Table 2, which shows that configurations with few hashes ($k<32$) have large signatures (tens of KiB) that exceed our budget.

To use HORS signatures with small k while staying within our signature size budget, we devise another compression technique inspired by SPHINCS [15]. This technique is based on the observation that verifying a HORS signature merely requires checking that the few revealed secrets match the public key; knowledge of the full public key is unnecessary. We enable such checks using Merkle inclusion proofs: we arrange all public key elements in a Merkle forest, and EdDSA-sign its roots. Such DSig signatures replace their HORS public

Table 2: Analytical comparison of a DSig signature using either HORS or W-OTS$^+$ as its HBSS for various configurations with EdDSA batches of 128 public keys.

Conf	# Critical Hashes	Signature Size (B)	# BG Hashes	BG Traffic (B/Verifier)
Using HORS with factorized PKs				
k=8	8	8Mi	512Ki	33
k=16	16	64Ki	4Ki	33
k=32	32	8,552	512	33
k=64	64	4,456	256	33
Using HORS with merklified PKs				
k=8	8	4,712	1Mi	8Mi
k=16	16	4,968	8Ki	64Ki
k=32	32	5,480	1Ki	8Ki
k=64	64	6,504	510	4Ki
Using W-OTS$^+$				
d=2	≈68	2,808	136	33
d=4	≈102	1,584	204	33
d=8	≈161	1,188	322	33
d=16	≈263	990	525	33
d=32	≈434	864	868	33

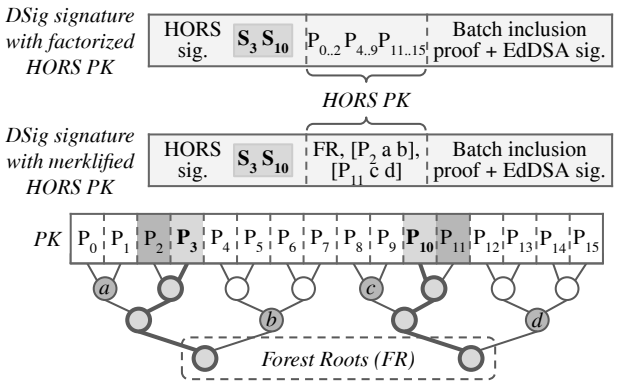

Figure 4: DSig signatures when using HORS with either factorized or merklified public keys. For illustration purposes, we depict a toy configuration of HORS with 2-secret signatures and 16-element keys.

Figure 5: Layout of DSig signatures when using W-OTS$^+$.

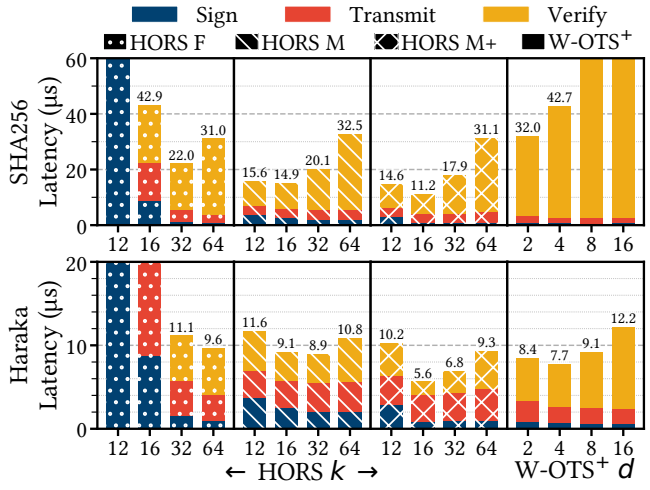

Figure 6: Sign-transmit-verify latency of DSig for 8 B messages using different hashing schemes (SHA256 (top) and Haraka (bottom)), and different HBSS configurations: HORS with factorized PK (HORS F), HORS with merklified PK (HORS M) and prefetched keys (HORS M+), and W-OTS$^+$.

key with the forest roots and the inclusion proofs of the revealed secrets (bottom of Figure 4). To avoid the overhead of checking Merkle proofs (i.e., computing around a hundred BLAKE3 hashes) on the critical path, we use a latency-hiding technique similar to the one in §4.4: signers send their complete public keys ahead of time to verifiers (by disabling background bandwidth reduction (§4.4)); verifiers precompute Merkle forests in their background plane, so Merkle proof verification on the critical path becomes mere string comparisons. We analyze this approach in the second part of Table 2, which shows that configurations with very few hashes ($k \leq 16$) have tractable signature sizes, but come at the expense of significant background traffic and many background hashes.

W-OTS$^+$. W-OTS$^+$ differs from HORS in two main ways. First, W-OTS$^+$ secrets are hashed $d-1$ times to obtain the public key, where d is a depth parameter. Second, to sign, each secret is hashed a different number of times, as determined by the message to be signed, before being included in the signature. We lower sign latency by caching these hashes upon computation of the public key so that signing reduces to string copying. To verify a signature, we hash each element the required number of times to get to depth d, as determined by the signed message, and verify that the obtained results match the public key. Note that W-OTS$^+$ does not require embedding the public key in the DSig signature (Figure 5). A downside of W-OTS$^+$ versus HORS is that W-OTS$^+$ needs many more hashes on the critical path.

We analyze W-OTS$^+$ within DSig in the last part of Table 2, which shows that W-OTS$^+$ configurations with small d satisfy our requirements regarding signature size, background processing, and bandwidth consumption. Moreover, although they all require more hashes than HORS on the critical path, their signatures are smaller than the smallest HORS ones. Note that as d gets bigger, the gain in signature size is outweighed by the drastic increase in hash computations.

Conclusion. Our analysis points to three general configurations for further experimental evaluation: (1) HORS with factorized PK and k close to 64, (2) HORS with merklified PK and k close to 16, and (3) W-OTS$^+$ with d close to 4.

5.3 Evaluation

We measure the latency of signing an 8 B message, transmitting it along with its DSig signature, and verifying the signature for the sensible HBSS configurations presented in §5.2. Our experimental setup is detailed in §8. We consider three hash functions: SHA256 [77] (the slowest), BLAKE3 [75], and Haraka [55] (the fastest). Figure 6 shows the results for SHA256 and Haraka (BLAKE3 stands in between).

When using Haraka (bottom of Figure 6), HORS with factorized public keys (denoted *HORS F*) achieves its best end-to-end latency for k=64. For k<64, its latency increases in spite of having fewer hashes on the critical path due to the transmission time of larger signatures.

Surprisingly, HORS with merklified public keys (*HORS M*) signs and verifies only marginally faster despite its far lower number of hashes. This disappointing performance is a microarchitectural effect of the size of the Merkle forests. Indeed, for the assembly and verification of precomputed Merkle proofs to be fast, the relevant elements of their associated Merkle forest should be present in local cache (L1/L2).

However, CPU prefetching is inefficient due to the inherent randomness of Merkle proofs.

To demonstrate the benefit of having the Merkle proofs in the local cache, we modify DSig to prefetch public and private keys into the processor cache right before signing and verifying. The modified system (*HORS M+*) achieves an end-to-end latency of as little as 5.6 µs (signing in 0.9 µs, verifying in 1.5 µs and transmitting in 3.3 µs) for k=16. For smaller k, the keys do not entirely fit in the local cache, hence the performance degradation.

W-OTS$^+$ achieves its best latency of 7.7 µs for d=4 (signing in 0.7 µs, verifying in 5.1 µs and transmitting in 2.0 µs) where it strikes a good balance between few hashes (low d) and short signatures (high d). We omit prefetching W-OTS$^+$ keys, as it has negligible latency benefit (not shown).

When using SHA256 (a slower hash function, top of Figure 6), HORS with factorized public keys (HORS F) sees its verification time vastly increase for large k, while small k still has long transmission time. HORS with merklified public keys (HORS M) has better latency for smaller k, which differs from HORS M with Haraka. Indeed, it is preferable to have fewer SHA256 computations (small k) than smaller Merkle forests (large k) since SHA256 is considerably more expensive than cache misses, which is not the case of Haraka. Finally, the large number of hashes (68 in expectation) of W-OTS$^+$ makes it perform worse than the best configuration of each presented HORS variant.

5.4 Recommended Configuration

From our analytical (§5.2) and experimental (§5.3) studies, we recommend using W-OTS$^+$ with d=4 and Haraka. This configuration offers single-digit sign-transmit-verify latency, tractable 1,584 B signatures, and requires little background computation and networking. Although HORS with merklified keys can achieve lower latency, it is too costly (in compute, bandwidth, and CPU cache pollution) and its superior performance requires modifying applications to prefetch keys into the processor cache, which can be impractical. Note, however, that the choice of HBSS depends on the relative performance of hardware and software: in the future, if cache misses become cheaper and hashing becomes relatively more expensive, low-k HORS configurations could be appealing.

6 Applications

We apply DSig to key-value stores, a financial trading system, BFT broadcast, and BFT replication.

Key-value stores (HERD [51] and Redis [87]). State-of-the-art key-value stores provide microsecond-level performance and form the backbone of many data center applications, microservices, and cloud services. Key-value stores are used to keep security-sensitive information such as service configuration, session management data, cached queries, access control data, chat sessions, etc. Yet, most key-value stores lack *auditability*—the ability for a third-party to check a log of all operations (reads and writes) executed on the key-value store. More precisely, in an auditable key-value store, the server keeps a log of executed operation such that, for any operation *op* in the log, the server can prove to a third party that *op*'s client requested its execution. For example, the third party may be a forensics specialist or a prosecutor, who wants proof that a client requested access or modification to some data. The threat model is that clients may wish to bypass the audit (i.e., execute an operation undetected), while the server is honest. The proofs provided by the server are operations signed by clients and the key-value store must ensure that (a) if an operation signed by client C is in the audit log, then it was executed by the key-value store for client C, and (b) if an operation is executed by the key-value store for client C, then it appears in the audit log as an operation signed by C.

To provide auditability, a key-value store requires all requests to be signed by clients and logged by the server. The server must check the client signature before executing a request (otherwise a client could send a request with a bogus signature, which the server would execute without later being able to prove it), so traditional digital signatures significantly increase the latency of microsecond-scale key-value stores.

We use DSig to add auditability to two key-value stores: HERD and Redis. HERD is highly optimized for the RDMA networks present in data centers, while Redis is popular among web application developers. HERD provides simple `GET` and `PUT` operations on key-value pairs, while Redis also provides higher-level operations on common data structures, such as lists, maps, sets, etc. We modify both systems so that clients use DSig to sign all operations, and servers log and verify the signed operations before executing them. This logging requires 1.5 KiB of storage per operation due to DSig's signatures. Key-value stores have predictable signing and verifying processes: clients simply set their signatures hints to the server process. Logs can be persisted at the microsecond scale using persistent memory. This is not currently implemented due to lack of hardware, but data from Yang *et al.* [91] indicate that persistence would take less than 4 µs, and this latency can be masked by storing in parallel with signature verification. Vanilla HERD takes ≈2.5 µs to `GET` or `PUT` a key-value pair, while vanilla Redis takes ≈12 µs.

Financial trading system (Liquibook [73]). Liquibook provides an order-matching engine for financial trading—it matches buy and sell limit orders from clients. We consider a trading system built using Liquibook and RDMA for fast client-server communication. We use DSig to enhance the system and provide auditability, as we did for key-value stores. Signature hints are identical to key-value stores. Without auditability, the trading system takes ≈3.6 µs to process orders, of which ≈2 µs are spent on communication.

BFT broadcast (CTB [3]). Byzantine Fault Tolerance (BFT) is becoming more relevant in data centers, due to the need

Table 3: Configuration details of machines.

CPU	2× 8c/16t Intel Xeon Gold 6244 @ 3.60GHz
NIC/Switch	Mlnx CX-6 MT28908 / MSB7700 EDR 100 Gbps
Software	Linux 5.4.0-167-generic / Mlnx OFED 5.3-1.0.0.1

to tolerate failures beyond simple crashes [42, 44, 45, 69, 70]. Consistent broadcast is a core BFT primitive that prevents equivocation [23] and has many uses, as in money transfer [27, 40], consensus [1, 13, 25], multi-party computation [10] and decentralized learning [35, 38] protocols. We consider uBFT's state-of-the-art implementation of Consistent Tail Broadcast (CTB) and replace its signatures with DSig's to improve performance. Signing hints are simple, as each signature is verified by all processes running the protocol.

BFT replication (uBFT [3]). State machine replication (SMR) is the standard approach for fault-tolerance [2, 18, 66]. We consider uBFT, a microsecond-scale BFT SMR system for data centers. BFT SMR protocols, including uBFT, often employ signed messages to guard against Byzantine replicas. uBFT recognizes the high cost of digital signatures and follows a fast/slow path approach. The fast path avoids signatures and has a latency of 5 µs. The slow path uses signatures, with a latency of ≈220 µs. The slow path is triggered even without Byzantine behavior (e.g., due to process slowness), leading to latency fluctuations between its two modes of operation. We use DSig to replace the digital signatures in uBFT and improve its performance. Signing hints are simple, as each signature is verified by all processes running the protocol. Moreover, we use DSig's DoS-mitigation mechanism (§4) to prevent a malicious attack from triggering superfluous EdDSA verifications. More precisely, because uBFT is a quorum system, it can make progress with $n-f$ responses (n is the number of replicas, f is the maximum number that can be Byzantine). We make a small modification to uBFT to use the *canVerifyFast* function to prioritize handling of messages that do not incur the EdDSA signature check. As a process gets at least $n-f$ messages from non-Byzantine processes, it ignores the slow-to-check messages from Byzantine players.

7 Implementation

Our implementation of DSig has 3,019 lines of C++17 (CLOC [32]). We use our own implementation of HORS [84] and W-OTS$^+$ [46], the official implementations of BLAKE3 [75] and Haraka [55], and Dalek's implementation of EdDSA (Ed25519 [65]). BLAKE3 and Dalek's EdDSA use AVX2 for high performance. We use uBFT's framework [3], which provides fast point-to-point communication.

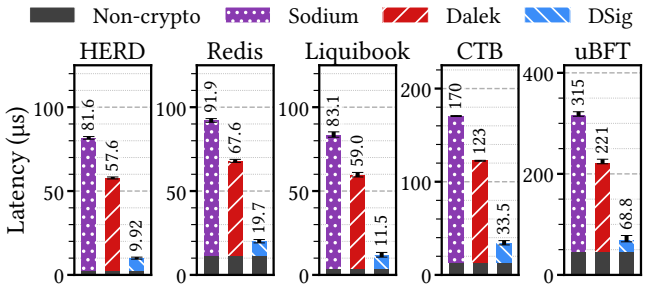

Figure 7: End-to-end latency of different applications using Sodium, Dalek or DSig for signatures. Printed values show the median latency; whiskers show the 10th and 90th %-iles.

8 Evaluation

We evaluate the performance of DSig and verify its suitability as a microsecond-scale signature system. We aim to answer the following:

- How do microsecond-scale applications that use signatures benefit from DSig's low latency (§8.1)?
- How does DSig's signing and verification latency compare to traditional signatures (§8.2, §8.3)?
- What is the throughput of DSig (§8.4)?
- How do DSig's higher bandwidth requirements impact its scalability (§8.5, §8.6)?
- How do we set DSig's EdDSA batch size (§8.7)?

Testbed. Our testbed is a cluster with 4 servers configured as shown in Table 3. The dual-socket machines have an RDMA NIC attached to the first socket. Our experiments execute on cores of the first socket using local NUMA memory. We accurately measure time using the TSC [43] via `clock_gettime` with the `CLOCK_MONOTONIC` parameter.

DSig configuration. We configure DSig per §5: in all experiments, we use W-OTS$^+$ with a depth d=4 as its HBSS. We dedicate a single core to DSig's background plane, which provides a high-enough throughput for our applications (§8.4). Unless specified otherwise, we use an EdDSA batch size of 128 (§8.7) and provide correct verifier hints when signing.

Baselines. We compare DSig against two well-known signature libraries: Sodium [8] (written in C) and Dalek [65] (written in Rust). Both implement the EdDSA signature scheme Ed25519—the fastest traditional scheme to date [17].

8.1 Application Latency

We configure applications with different signature schemes (Sodium, Dalek, DSig) and measure their latency. For the key-value stores, we use 16 B keys and 32 B values, 20% of `PUT` requests and 80% of `GET`s, where 90% of `GET`s return a value. For Liquibook, 50% of the requests are `SELL`s and 50% are `BUY`s. For CTB, we broadcast 8 B messages. Finally, for

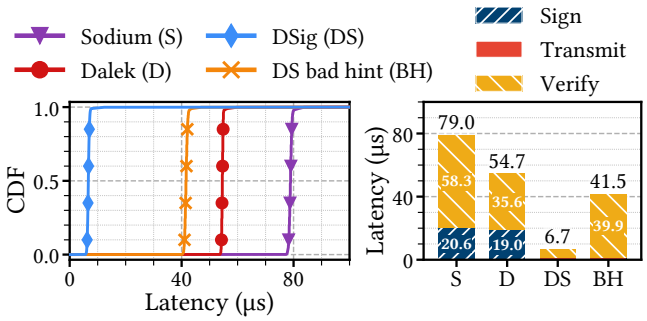

Figure 8: (Left) CDF of latency when signing, transmitting, and verifying the signature of 8 B messages using Sodium, Dalek and DSig with/without correct hints. (Right) Median latency breakdown. Transmission overhead is barely visible.

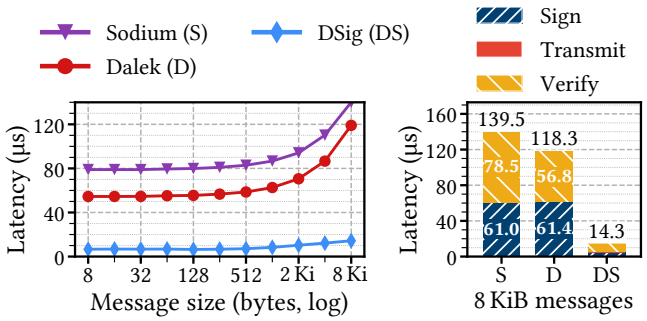

Figure 9: (Left) Latency to sign, transmit, and verify the signature of various-size messages using Sodium, Dalek and DSig with correct hints. (Right) Median latency breakdown for 8 KiB messages. Transmission overhead is invisible.

uBFT, we consider SMR operations of 8 B. We issue 10,000 requests one at a time to each application, measure the end-to-end latency, and report the 10th-, 50th-, and 90th-percentiles.

Figure 7 shows the results. For the three applications on the left, DSig provides auditability for a small cost: an increase of less than 7.9 µs in end-to-end latency. Sodium and Dalek add ≈79 µs and ≈55 µs, respectively, which is 10× and 7.0× DSig's overhead. In CTB, replacing Sodium (resp. Dalek) with DSig reduces the median cryptographic overhead by 87% (resp. 82%), and reduces the median end-to-end latency by 80% (resp. 73%). In uBFT, DSig reduces the median cryptographic overhead by 91% (resp. 87%), and reduces the median end-to-end latency by 78% (resp. 69%) compared to Sodium (resp. Dalek). DSig provides similar latency gains at the 90th percentile. In summary, across the tested applications, DSig significantly reduces cryptographic overheads and improves latency over the state of the art.

8.2 Latency of DSig

We study the latency to sign a message, transmit a signature, and verify a signature using DSig. We also consider the latency of incorrectly hinted DSig signatures for which EdDSA signatures are verified on the critical path. This represents the worst-case scenario for DSig. In each experiment, a process signs an 8 B message and transmits the signed message to a second process, which verifies the signature. We run each experiment 10,000 times for each signature scheme. The signature transmission latency is the incremental cost of adding the signature to a message, computed as the difference between transmitting a message with and without a signature. We estimate message transmission time as half of the round-trip time for ping-ponging the message.

Figure 8 shows the results. All signature schemes have stable latency up to the 99.9th percentile. Sodium and Dalek have similar signing median latency of 20.6 µs and 18.9 µs, respectively. While Sodium verifies in 58.3 µs, Dalek verifies in only 35.6 µs (39% faster) thanks to the use of AVX2 instructions. The (incremental) network latency is less than 100 ns for both since their signatures are merely 64 B. With correct hints, DSig takes 0.7 µs to sign and 5.1 µs to verify. This is 27× and 7.0× faster than Dalek, respectively. Interestingly, even though DSig's larger signatures lead to a 1.0 µs transmission overhead (more than 10× Dalek's), it has limited impact on its latency which is dominated by verification. Overall, DSig is 8.2× faster than Dalek. With incorrect hints, DSig's signature verification requires verifying both HBSS and EdDSA signatures, so verification latency increases to 39.9 µs (4.3 µs more than Dalek's). Signature generation, however, is not impacted as signers still benefit from background EdDSA precomputation and the total latency, although rising to 41.5 µs, is still 24% lower than Dalek's. Even with incorrect hints, DSig has much lower combined sign-transmit-verify latency than the state of the art.

8.3 Effect of Message Size on Latency

We study the effect of message size on the latency of DSig by running the experiments of §8.2 with varying message sizes.

Figure 9 (left) shows the results. With larger messages, DSig's total latency increases gradually but remains below 15 µs. Sodium's and Dalek's latencies are much higher. They also increase faster because they use a slower hash function than DSig (SHA256).[1] Figure 9 (right) shows the latency breakdown for the largest size (the breakdown for the smallest size is in §8.2). In all schemes, the split is about half-half to sign and verify, with negligible transmission time.

8.4 Throughput

We study DSig's throughput. In an experiment, a process signs small 8 B messages repeatedly with either a constant or

[1] Most signature schemes hash the input to operate on a fixed-size string.

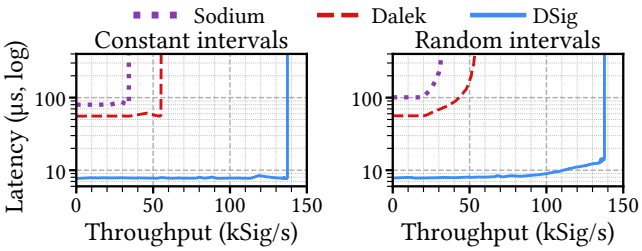

Figure 10: Latency-throughput graphs for Sodium, Dalek and DSig. Signatures are issued at constant or exponentially distributed intervals. All three use two cores on both sides; DSig dedicates one of them to its background plane.

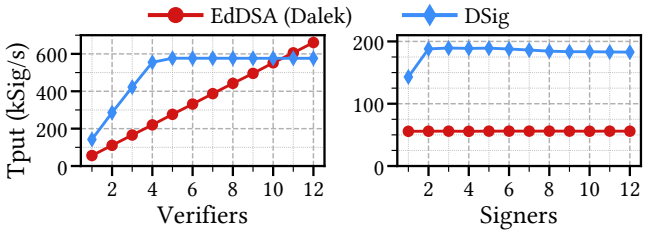

Figure 11: (Left) DSig's throughput with a signer sending the same signature to multiple verifiers. (Right) DSig's throughput with a verifier receiving different signatures from multiple signers. The NICs' bandwidth is limited to 10 Gbps.

an exponentially distributed random interval between signatures. The signer transmits the signature over the network to the verifier, which verifies it. We measure the total latency (sign plus transmit plus verify) and throughput of DSig, and compare it against Sodium's and Dalek's. In all experiments, the signer and the verifier use two cores each. DSig dedicates one core to each of its planes so that background events minimally impact foreground operations, while Sodium and Dalek use all cores to handle multiple messages in parallel.

Figure 10 shows the results as latency-throughput graphs with median latency and average throughput. With constant signature interval, all three systems exhibit stable latency until reaching maximum throughput. Sodium maintains a latency of ≈80 µs up to a throughput of 34 kSig/s where it is bottlenecked by verification time (58 µs). Dalek maintains a latency of ≈56 µs up to a throughput of 56 kSig/s where it is also bottlenecked by verification time (36 µs). DSig maintains a latency of ≈7.8 µs up to a throughput of 137 kSig/s where it is bottlenecked by the signer's background plane, which takes 7.4 µs to generate a new public key. We separately measured the verifier's background plane; it achieves a throughput of 3.6 MSig/s, so it is not a bottleneck. With a random signing interval, queuing occurs gradually, so the respective bottlenecks are less abrupt, causing a smoother latency degradation.

We run another experiment to measure the per-core throughput of DSig by running both of its planes on one core, and compare it to the per-core throughput of Dalek. While Dalek achieves 53 kSig/s signature generations (resp. 28 kSig/s verifications) per core, DSig achieves 131 kSig/s signature generations (resp. 193 kSig/s verifications) per core.

In summary, DSig sustains significantly higher throughput at much lower latency than EdDSA-based systems.

8.5 One-to-Many, Many-to-One Performance

We now study DSig's scalability and bottlenecks in one-to-many and many-to-one scenarios. In *one-to-many*, one signer signs a message and sends the signature to many verifiers; this pattern is common in distributed protocols. In *many-to-one*, many signers sign different messages and send them to the same verifier; this pattern is common in client-server applications. We run experiments where the signer(s) and verifier(s) use one foreground and one background core to work as fast as possible. We measure the aggregate verification throughput, and report the average. We consider a scenario where most of the network bandwidth (≈90%) is consumed by other activities, by limiting our NICs' bandwidth to 10 Gbps. This of course makes it harder for DSig to operate since it consumes more network bandwidth than other schemes. We compare the scalability of DSig to a two-core system based on Dalek.

Figure 11 shows the results. In one-to-many (left of figure), DSig's throughput increases until 577 kSig/s with 5 verifiers; at this point, the signer saturates its link to the verifiers, with the 1,584 B signatures and their 33 B background data accounting for ≈7 Gbps. Dalek scales more slowly than DSig with the number of verifiers, but it is not affected by bandwidth, as it continues to scale beyond 11 verifiers, at which point it surpasses DSig's throughput with 603 kSig/s using merely ≈300 Mbps to transmit 64 B signatures.

In many-to-one (right of figure), two signers are enough to achieve DSig's maximum throughput of 190 kSig/s as they saturate the verifier's foreground plane, which we set to run on a single core. As signing with Dalek is faster than verifying, Dalek does not scale beyond 1 verifier and achieves a maximum throughput of 53 kSig/s.

Overall, DSig's main scalability bottleneck compared to Dalek is its larger signatures.

8.6 Effect of Larger Signatures

We study how DSig's larger signatures affect application performance. In each experiment, we run a synthetic application where a server receives signed requests of a given size, checks their signature, spends some given processing time, and sends a 16 B unsigned reply. Similarly to §8.5, we limit the NICs' bandwidth to 10 Gbps, and compare the same application when using DSig or EdDSA. For fairness, EdDSA uses Dalek and pre-hashes messages with BLAKE3. In addition, we run an experiment with signatures disabled. The application runs with 4 cores: DSig uses 1 core for the background plane and

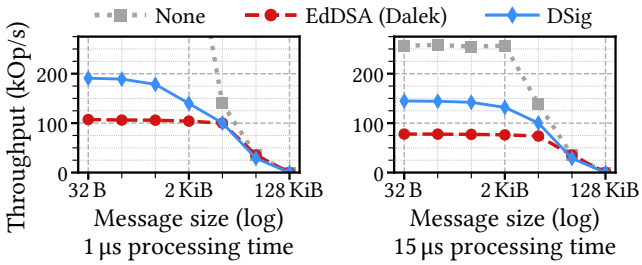

Figure 12: Request throughput of an application using signatures when NICs' bandwidth is constrained to 10 Gbps, for different request sizes and request processing times.

3 cores to handle requests, while the others use 4 cores to handle requests. We run enough clients to saturate the server. We consider 7 request sizes (32 B, 128 B, 512 B, 2 KiB, 8 KiB, 32 KiB and 128 KiB) and 2 processing times (1 µs and 15 µs).

Figure 12 shows the results. For both processing times, DSig outperforms EdDSA up to 8 KiB, after which it performs similarly to EdDSA. For small messages (32 B–512 B), the limited bandwidth has no impact on either scheme, so DSig significantly outperforms EdDSA thanks to its lower computational cost. With 2 KiB messages and 1 µs processing time, bandwidth impacts DSig while EdDSA is almost unaffected. Relative to 512 B messages, DSig's throughput decreases by 22%, while EdDSA's decreases by only 1.9%. Higher processing time offsets DSig's bandwidth bottleneck closer to 8 KiB messages. Beyond these points, the throughput of both DSig and EdDSA converges to that of the application that does not use signatures, as network bandwidth bottlenecks all three systems, making the overhead of signatures negligible.

In summary, DSig's higher per-core throughput lets applications reach higher throughput than with EdDSA even with limited network bandwidth, up to moderate-size messages.

8.7 EdDSA Batch Size

To set the size of EdDSA-signed key batches (§4.4), we run the same experiment as in §8.2 for different batch sizes, and we measure the latency and the per-core throughput. To take into account the impact of larger batches on low-end networks, we limit our NICs' bandwidth to 10 Gbps, as in §8.5 and §8.6.

Figure 13 shows the results, where a batch size of 1 means no batching. We see that batch sizes do not affect latency much (left of figure).[2] Throughput is different (right of figure): initially, batching improves throughput a lot for both signing and verifying. The gain dwindles when the amortized EdDSA cost per signature becomes a diminishing fraction of the overall computation as batches get larger. The best signing throughput is 135 kSig/s for batches of 32 keys, while the best verifying throughput is 206 kSig/s for batches of 4,096 keys. We pick a batch size of 128 as a balance.

[2]The transmission latency differs from §8.2 due to the 10 Gbps NIC limit.

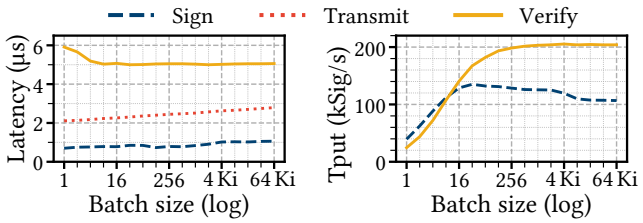

Figure 13: (Left) Median latency to sign, transmit, and verify a signature on an 8 B message with DSig for different EdDSA batch sizes. (Right) Single-core throughput of signing and verifying for different EdDSA batch sizes.

9 Related Work

HBSSs. HBSSs are well studied and prior work has proposed different implementations of them, many of which are variants of HORS [60,63,72,90]. Li *et al.* [63] proposed a variant targeted at smart grids with limited storage that reduces key and signature size but increases computation costs. Wang *et al.* [90] proposed a scheme with small signatures and microsecond performance, but it is limited to providing low ≈ 50-bit security. HORSE [72] reduces the cost of few-time signatures by repeatedly hashing the private key secrets, creating a matrix whose last row is the public key; however, it restricts the order in which applications can reveal public keys. W-OTS$^+$ [46] was proposed by Hülsing as a variant of W-OTS [34] with reduced signature and key sizes.

Online/offline signature schemes. The concept of online/offline digital signatures, in which heavy computation is done prior to knowing the message to sign, was first introduced by Even *et al.* [37]. So far, practical applications of the theoretical concept (including hybrid signature schemes) have targeted low-compute devices and/or wide area networks, with a focus on improving signature throughput or reducing bandwidth [53,62,64,78,92,93]. Recently, Esiner *et al.* [36] also recognized the importance of low-latency signatures, yet their solution is tailored for industrial control systems with tiny messages (25 bits), and does not provide self-standing signatures. No prior work addresses hybrid signatures in data centers with microsecond-scale performance.

Merkle-based signatures. Prior work proposes schemes that rely exclusively on HBSSs to sign (virtually) infinitely many messages, with the goal of attaining quantum resistance. Most of this work is based on XMSS [21], such as SPHINCS [15] and variants [16,47,58]. Instead of distributing keys regularly, these schemes efficiently pack an infinitude of one-time public keys using Merkle inclusion proofs [68]. These proofs need to be checked during signature verification, thus making the performance of such schemes be in the milliseconds.

Signature-like schemes. The cost of signatures has fueled alternatives for different scenarios. Message authentication codes (MACs) provide authentication and integrity of mes-

sages, but lack transferability, as parties use a shared secret to communicate. While MACs are widely used in networked systems to provide authenticated channels between two parties, they are not substitutes for signatures, as they provide weaker properties, they are harder to use, and they are more susceptible to protocol mistakes. In particular, using MAC-based mechanisms in BFT protocols has several drawbacks: (1) These mechanisms are ad-hoc and highly dependent on the protocol: some require MAC vectors [24] others require MAC matrices [5]; others explicitly prefer or mandate signatures over MACs for critical messages [3,4,26]. (2) These mechanisms add complexity to the BFT protocols, e.g., by requiring a fast-slow path approach where the fast path avoids signatures but the slow path (or view change) still uses them [3,4,24,26,29,56]; this added complexity increases their attack surface [26]. (3) These mechanisms often add messages and roundtrips to the protocols [5, 29, 79], and/or lower their resilience to failures [5, 79].

Some systems make extra assumptions to provide MAC with some form of transferability. TESLA [79] assumes clock synchrony and has time windows during which MACs are generated and transmitted; afterward, the MAC secrets are revealed to check previously seen MACs. This idea provides only a limited form of transferability and increases the latency of verification. Using trusted hardware [12, 52, 61], such as trusted execution environments (TEEs), one can provide MACs with transferability by hiding the secret and computing the MACs in the TEE so that every TEE owner can verify the MACs but only a designated TEE can create them.

10 Conclusion

DSig is the first digital signature system for microsecond-scale applications. DSig achieves single-digit microsecond latency for signing and verifying messages—27× and 7× faster than the prior state of the art—while achieving higher throughput. To achieve that, DSig introduces a new hybrid signature scheme that uses knowledge of where signatures are issued and verified in the common case. DSig can bring auditability to latency-critical applications with a small latency overhead, or replace other signature schemes in applications that use them. Ultimately, we believe that DSig makes digital signatures fast enough to broaden their use in data centers as a powerful security building block.

Acknowledgments

We thank the anonymous reviewers and our shepherd Steve Hand for their valuable comments, as well as the anonymous artifact evaluators for reviewing our implementation. We also thank Dariia Kharytonova, Ed Bugnion, Jean-Philippe Aumasson, Khashayar Barooti, Phillip Gajland, Pierre-Louis Roman, and Serge Vaudenay for their feedback.

References

[1] Ittai Abraham, Marcos K. Aguilera, and Dahlia Malkhi. Fast asynchronous consensus with optimal resilience. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing*, pages 4–19, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[2] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J. Marathe, Athanasios Xygkis, and Igor Zablotchi. Microsecond consensus for microsecond applications. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, Berkeley, CA, USA, 2020. USENIX Association.

[3] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. uBFT: Microsecond-scale BFT using disaggregated memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '23, page 862–877, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Dalia Papuc, Athanasios Xygkis, and Igor Zablotchi. Frugal Byzantine computing. In *Proceedings of the 35th International Symposium on Distributed Computing*, volume 209 of *DISC '21*, pages 3:1–3:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] Amitanand S. Aiyer, Lorenzo Alvisi, Rida A. Bazzi, and Allen Clement. Matrix signatures: From MACs to digital signatures in distributed systems. In Gadi Taubenfeld, editor, *Distributed Computing*, pages 16–31, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th European Conference on Computer Systems*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Jean-Philippe Aumasson. Too much crypto. *Cryptology ePrint Archive*, Paper 2019/1492, 2019.

[8] Jean-Philippe Aumasson, Daniel J. Bernstein, Alex Biryukov, Tung Chou, Frank Denis, Daniel Dinu, Romain Dolbeau, Jason A. Donenfeld, Niels Duif, Adrien

Gallouet, Jack Grigg, Mike Hamburg, Dmitry Khovratovich, Tanja Lange, Adam Langley, Isis Lovecruft, Andrew Moon, Samuel Neves, Yoav Nir, Colin Percival, Alexander Peslyak, Thomas Pornin, Bart Preneel, Peter Schwabe, George Tankersley, Henry de Valence, Filippo Valsorda, Zooko Wilcox-O'Hearn, Christian Winnerlein, Hongjun Wu, and Bo-Yin Yang. The Sodium cryptography library, 2022. Accessed: April 14, 2023.

[9] Jean-Philippe Aumasson and Guillaume Endignoux. Improving stateless hash-based signatures. In *Topics in Cryptology – The Cryptographers' Track at the RSA Conference*, CT-RSA '18, pages 219–242, Cham, Zug, Switzerland, 2018. Springer International Publishing.

[10] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. Asynchronous MPC with a strict honest majority using non-equivocation. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, page 10–19, New York, NY, USA, 2014. Association for Computing Machinery.

[11] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the killer microseconds. *Communications of the ACM*, 60(4):48–54, March 2017.

[12] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: SGX-based high performance BFT. In *Proceedings of the 12th European Conference on Computer Systems*, EuroSys '17, page 222–237, New York, NY, USA, 2017. Association for Computing Machinery.

[13] Naama Ben-David, Benjamin Y. Chan, and Elaine Shi. Revisiting the power of non-equivocation in distributed protocols. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, page 450–459, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. Ed25519: high-speed high-security signatures, 2017. Accessed: April 14, 2023.

[15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *Advances in Cryptology – Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '15, pages 368–397, Berlin, Germany, 2015. Springer-Verlag.

[16] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2129–2146, New York, NY, USA, 2019. Association for Computing Machinery.

[17] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems, 2022. Accessed: April 14, 2023.

[18] Alysson Bessani, João Sousa, and Eduardo E. P. Alchieri. State Machine Replication for the masses with BFT-SMART. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, page 355–362, NW Washington, DC, USA, June 2014. IEEE Computer Society.

[19] Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. Putting the "micro" back in microservice. In *Proceedings of the 2018 USENIX Annual Technical Conference*, USENIX ATC '18, page 645–650, Berkeley, CA, USA, 2018. USENIX Association.

[20] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*, SP '21, pages 1659–1676, New York, NY, USA, 2021. IEEE.

[21] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS – A practical forward secure signature scheme based on minimal security assumptions. In *Proceedings of the 4th International Workshop on Post-Quantum Cryptography*, PQCrypto '11, pages 117–129, Berlin, Germany, 2011. Springer-Verlag.

[22] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. Accessed: April 14, 2023.

[23] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer Publishing Company, Incorporated, Berlin, Germany, 2nd edition, 2011.

[24] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[25] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, page 301–308, New York, NY, USA, 2012. Association for Computing Machinery.

[26] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '09, page 153–168, USA, 2009. USENIX Association.

[27] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38, New York, NY, USA, 2020. IEEE.

[28] Victor Costan and Srinivas Devadas. Intel SGX explained. *Cryptology ePrint Archive*, Paper 2016/086, 2016.

[29] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. Hq replication: a hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, page 177–190, USA, 2006. USENIX Association.

[30] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1773–1788, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Alexandros Daglis, Mark Sutherland, and Babak Falsafi. RPCValet: NI-driven tail-aware balancing of μs-scale RPCs. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '19, page 35–48, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Al Danial. CLOC: Count Lines of Code, 2022.

[33] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[34] Chris Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In *Proceedings of the 10th IMA International Conference on Cryptography and Coding*, IMACC '05, pages 96–115, Berlin, Germany, 2005. Springer-Verlag.

[35] El Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. Collaborative learning in the jungle (decentralized, byzantine, heterogeneous, asynchronous and nonconvex learning). In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25044–25057, New York, NY, USA, 2021. Curran Associates, Inc.

[36] Ertem Esiner, Utku Tefek, Hasan S. M. Erol, Daisuke Mashima, Binbin Chen, Yih-Chun Hu, Zbigniew Kalbarczyk, and David M. Nicol. LoMoS: Less-Online/More-Offline signatures for extremely time-critical systems. *IEEE Transactions on Smart Grid*, 13(4):3214–3226, July 2022.

[37] Shimon Even, Oded Goldreich, and Silvio Micali. On-Line/Off-Line digital signatures. In *Advances in Cryptology – Proceedings of the 9th Annual International Cryptology Conference*, CRYPTO '89, page 263–275, Berlin, Germany, 1989. Springer-Verlag.

[38] Sadegh Farhadkhani, Rachid Guerraoui, Nirupam Gupta, Lê Nguyên Hoang, Rafael Pinot, and John Stephan. Robust collaborative learning with linear gradient overhead. In *Proceedings of the 40th International Conference on Machine Learning*, ICML '23, jmlr.org, 2023. The Journal of Machine Learning Research.

[39] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[40] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 307–316, New York, NY, USA, 2019. Association for Computing Machinery.

[41] Rachid Guerraoui, Antoine Murat, Javier Picorel, Athanasios Xygkis, Huabing Yan, and Pengfei Zuo. uKharon: A membership service for microsecond applications. In *Proceedings of the 2022 USENIX Annual Technical Conference*, USENIX ATC '22, pages 101–120, Berkeley, CA, USA, 2022. USENIX Association.

[42] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Andrew Baptist, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage*, 14(3), October 2018.

[43] Red Hat. RHEL for real time timestamping, 2020. Accessed: April 14, 2023.

[44] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. Cores that don't count. In *Proceedings of the 18th Workshop on Hot Topics in Operating Systems*, HotOS '21, page 9–16, New York, NY, USA, 2021. Association for Computing Machinery.

[45] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray Failure: The Achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, page 150–155, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Andreas Hülsing. W-OTS+ – Shorter signatures for hash-based signature schemes. In *Progress in Cryptology – Proceedings of the 6th International Conference on Cryptology in Africa*, AFRICACRYPT '13, pages 173–188, Berlin, Germany, 2013. Springer-Verlag.

[47] Andreas Hülsing, Christoph Busold, and Johannes Buchmann. Forward secure signatures on smart cards. In *Proceedings of the 19th International Conference on Selected Areas in Cryptography*, SAC '12, pages 66–80, Berlin, Germany, 2012. Springer-Verlag.

[48] Illumina. Advancing research and clinical genomic data solutions, 2023. Accessed: April 14, 2023.

[49] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, August 2001.

[50] Simon Josefsson and Ilari Liusvaara. Edwards-Curve digital signature algorithm (EdDSA). RFC 8032, January 2017.

[51] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 295–306, New York, NY, USA, 2014. Association for Computing Machinery.

[52] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. CheapBFT: Resource-efficient Byzantine Fault Tolerance. In *Proceedings of the 7th European Conference on Computer Systems*, EuroSys '12, page 295–308, New York, NY, USA, 2012. Association for Computing Machinery.

[53] Jayaprakash Kar. Provably secure online/off-line identity-based signature scheme for wireless sensor network. *Cryptology ePrint Archive*, Paper 2012/162, 2012.

[54] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, Boca Raton, FL, USA, 2nd edition, 2014.

[55] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 – efficient short-input hashing for post-quantum applications. *IACR Transactions on Symmetric Cryptology*, 2016(2):1–29, February 2016.

[56] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. *SIGOPS Oper. Syst. Rev.*, 41(6):45–58, oct 2007.

[57] Platon Kotzias, Srdjan Matic, Richard Rivera, and Juan Caballero. Certified PUP: Abuse in authenticode code signing. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 465–478, New York, NY, USA, 2015. Association for Computing Machinery.

[58] Mikhail A. Kudinov, Andreas Hülsing, Eyal Ronen, and Eylon Yogev. SPHINCS+C: Compressing SPHINCS+ with (almost) no cost. *Cryptology ePrint Archive*, Paper 2022/778:48, 2022.

[59] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, Menlo Park, CA, USA, October 1979.

[60] Jaeheung Lee and Yongsu Park. HORSIC+: An efficient post-quantum few-time signature scheme. *Applied Sciences*, 11(16), August 2021.

[61] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. TrInc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '09, page 1–14, Berkeley, CA, USA, 2009. USENIX Association.

[62] Fagen Li, Di Zhong, and Tsuyoshi Takagi. Practical identity-based signature for wireless sensor networks. *IEEE Wireless Communications Letters*, 1(6):637–640, December 2012.

[63] Qinghua Li and Guohong Cao. Multicast authentication in the smart grid with one-time signature. *IEEE Transactions on Smart Grid*, 2(4):686–696, December 2011.

[64] Joseph K. Liu, Joonsang Baek, Jianying Zhou, Yanjiang Yang, and Jun Wen Wong. Efficient online/offline identity-based signature for wireless sensor network. *International Journal of Information Security*, 9(4):287–296, August 2010.

[65] Isis Agora Lovecruft and Henry De Valence. ed25519-dalek: Fast and efficient rust implementation of ed25519 key generation, signing, and verification in rust, 2022.

[66] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Mencius: Building efficient replicated state machines for WANs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '08, page 369–384, Berkeley, CA, USA, 2008. USENIX Association.

[67] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, April 1978.

[68] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology – Proceedings of the 9th Annual International Cryptology Conference*, CRYPTO '89, pages 218–238, Berlin, Germany, 1989. Springer-Verlag.

[69] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, page 177–190, New York, NY, USA, 2015. Association for Computing Machinery.

[70] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A large scale study of data center network reliability. In *Proceedings of the 2018 Internet Measurement Conference*, IMC '18, page 393–407, New York, NY, USA, 2018. Association for Computing Machinery.

[71] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Accessed: April 14, 2023.

[72] William D. Neumann. HORSE: An extension of an r-time signature scheme with fast signing and verification. In *International Conference on Information Technology: Coding and Computing, Volume 1*, ITCC '04, pages 129–134 Vol.1, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[73] Eric Newhuis. Liquibook: Open source order matching engine, 2022.

[74] NVIDIA. NVIDIA ConnectX-6 DX datasheet, 2022. Accessed: April 14, 2023.

[75] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3, 2022.

[76] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '19, page 361–377, Berkeley, CA, USA, 2019. USENIX Association.

[77] Wouter Penard and Tim van Werkhoven. *On the secure hash algorithm family*, pages 1–18. Wiley, New York, NY, USA, 2008.

[78] Cong Peng, Min Luo, Li Li, Kim-Kwang Raymond Choo, and Debiao He. Efficient certificateless online/offline signature scheme for wireless body area networks. *IEEE Internet of Things Journal*, 8(18):14287–14298, September 2021.

[79] Adrian Perrig and J. D. Tygar. *TESLA Broadcast Authentication*, pages 29–53. Springer, Boston, MA, USA, 2003.

[80] George Prekas, Marios Kogias, and Edouard Bugnion. ZygOS: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*, SOSP '17, page 325–341, New York, NY, USA, 2017. Association for Computing Machinery.

[81] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 342–355, New York, NY, USA, 2015. Association for Computing Machinery.

[82] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. Arachne: Core-Aware thread management. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, page 145–160, Berkeley, CA, USA, 2018. USENIX Association.

[83] Eric Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, Boston, MA, USA, 2001.

[84] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference Information Security and Privacy*, ACISP '02, pages 144–153, Berlin, Germany, 2002. Springer-Verlag.

[85] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[86] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, page 93–100, New York, NY, USA, 1999. Association for Computing Machinery.

[87] Salvatore Sanfilippo. Redis, 2022.

[88] Synopsys. Synopsys and Amazon Web Services, October 2023. Accessed: April 14, 2023.

[89] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, WOEC '96, Berkeley, CA, USA, 1996. USENIX Association.

[90] Qiyan Wang, Himanshu Khurana, Ying Huang, and Klara Nahrstedt. Time valid one-time signature for time-critical multicast data authentication. In *Proceedings of the 28th IEEE International Conference on Computer Communications*, INFOCOM '09, pages 1233–1241, New York, NY, USA, 2009. IEEE.

[91] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, FAST '20, page 169–182, Berkeley, CA, USA, 2020. USENIX Association.

[92] Andrew Chi-Chih Yao and Yunlei Zhao. Online/Offline signatures for low-power devices. *IEEE Transactions on Information Forensics and Security*, 8(2):283–294, February 2013.

[93] Ping Yu and Stephen R. Tate. Online/Offline signature schemes for devices with limited computing capabilities. In *Topics in Cryptology – The Cryptographers' Track at the RSA Conference*, CT-RSA '08, pages 301–317, Berlin, Germany, 2008. Springer-Verlag.