Performance and Security of TEE-Based Threshold Cryptography

Marius Haller
upynn@student.kit.edu
KASTEL Security Research Labs
Karlsruhe Institute of Technology
Karlsruhe, Germany

Marc Leinweber
marc.leinweber@kit.edu
KASTEL Security Research Labs
Karlsruhe Institute of Technology
Karlsruhe, Germany

Tilo Spannagel
tilo.spannagel9@kit.edu
KASTEL Security Research Labs
Karlsruhe Institute of Technology
Karlsruhe, Germany

Markus Raiber
markus.raiber@kit.edu
KASTEL Security Research Labs
Karlsruhe Institute of Technology
Karlsruhe, Germany

Hannes Hartenstein
hannes.hartenstein@kit.edu
KASTEL Security Research Labs
Karlsruhe Institute of Technology
Karlsruhe, Germany

Abstract

Various Byzantine fault tolerant protocols rely on threshold cryptography to mitigate certain attack vectors or reduce communication complexity. Threshold cryptography schemes distribute secret key material and cryptographic operations such as decryption or signing, thereby making them resilient to partial system corruption. Trusted Execution Environments (TEEs) can accelerate such schemes, but introduce new security trade-offs. We analyze both performance gains and security implications of using TEEs for threshold cryptography. To this end, we implement and evaluate SplitTEE, a dedicated library for low-latency, non-interactive threshold cryptography in both the hybrid and the Byzantine fault model. We propose two variants: a common key variant, making extensive use of the TEE and achieving observed speedup factors of 1.4 to 2.2 for threshold encryption, of 20.7 to 27.5 for threshold signatures, and of 1.4 to 1.8 for common coins; and a split key variant that remains secure even against compromised TEEs via a forensic mechanism, while still achieving speedup factors between 1.2 and 2.5 for encryption, and between 2.2 and 4 for signatures.

CCS Concepts: • **Security and privacy → Distributed systems security**.

Keywords: Encryption, Signatures, Common Coin

ACM Reference Format:
Marius Haller, Marc Leinweber, Tilo Spannagel, Markus Raiber, and Hannes Hartenstein. 2026. Performance and Security of TEE-Based Threshold Cryptography. In *9th Workshop on System Software for Trusted Execution (SysTEX '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3805690.3805725

SysTEX '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2607-1/26/04
https://doi.org/10.1145/3805690.3805725

1 Introduction

Threshold cryptography distributes cryptographic operations, such as decryption, signing, or randomness generation, among a group of n parties of which each party only holds a *secret share* of the secret key [1]. Instead of computing the full result, each party can only compute an individual *result share*. Parties must combine at least $t+1$ distinct valid result shares, where $t < n$ represents the system's corruption threshold, to obtain the global result. Subsets smaller than $t+1$ parties cannot compute the global result or prevent honest parties from doing so. Threshold cryptography schemes are pivotal to modern distributed algorithms, aiming to eliminate single points of failure [2]. Threshold *encryption* can increase resilience in Byzantine fault tolerant (BFT) state machine replication by mitigating censorship [3] and request reordering [4]. It is also an essential building block in some secure e-voting schemes [5]. Threshold *signatures* are used to create irreversible commit certificates that reduce communication complexity, e.g., in HotStuff [6], Kauri [7], and CART [8]. Other use cases include the management of secure threshold wallets for cryptocurrencies [9] and the implementation of distributed certificate authorities for public key infrastructures [10]. Threshold *common coins*, as a shared source of randomness, are crucial for leader election in asynchronous consensus [3, 11, 12].

Established threshold protocols incur significant performance overheads as they typically rely on so-called pairing curves [13] or zero-knowledge proofs (ZKP) [1, 14]. For example, result share verification on a pairing curve takes ~ 4.5 ms on recent hardware [8] while a 'classical' signature verification on Curve25519 takes less than 0.05 ms [15]. Typically, the number of required executions scales in the number of parties *and* the number of handled requests, making cryptographic operations a non-negligible performance bottleneck

[2, 8, 16–18]. A well-known way to increase the performance of distributed algorithms is to build upon the hybrid fault model in which some computations are performed within a Trusted Execution Environment (TEE) (e.g., [19–21]). The TEE is assumed to only fail by crashing and can be used to limit the options available to potential attackers. Related work [22–25] has shown that the hybrid fault model allows us to avoid the use of expensive building blocks in threshold protocols. However, an analysis of the actual performance gain and the security level achieved is still missing. Moreover, using a TEE comes with a non-negligible trade-off: If the TEE assumptions do not hold [26–28], the security properties of the known proposals break.

In this paper, we investigate the actual performance gains of TEE-based threshold cryptography and the security trade-offs introduced. To this end, we present SpliTEE, a modular Rust library[1] specialized for TEE-based threshold cryptography. For each threshold protocol, SpliTEE offers a common key variant, a split key variant, and a non-TEE scheme. The *common key* variants, which generalize from related work, make extensive use of the TEE and focus on performance. The *split key* variants aim to optimize the security trade-off introduced by using TEEs; in this variant, the TEE merely serves as a first but performance-relevant line of defense.

The remainder of the paper is structured as follows. We give necessary background and an overview of related work in Section 2. We present the SpliTEE library including the TEE-based threshold schemes in Section 3. In Section 4, we provide a complexity analysis and a detailed empirical evaluation. We discuss our approach and conclude in Section 5.

2 Background and Related Work

2.1 Threshold Cryptography

Threshold cryptography schemes distribute secret key material among n parties, such that security properties hold even in the case where up to t parties are compromised. In addition, they often offer *robustness*, which guarantees that as long as $t + 1$ parties remain honest, they can successfully perform the desired actions. A naive way of achieving this, e.g., for signatures, would be for each party to have their own key pair and a signature being considered valid if there are at least $t + 1$ signatures under $t + 1$ distinct public keys. This approach leads to a scaling of the size of public key and signature material with the number of parties n, which significantly increases overhead. A signature size independent of the number of parties requires a key setup where correlated keys are generated: Each party obtains a share gsk_i of a secret-shared global secret key gsk, a corresponding global public key gpk, and a set of validity keys $VK := \{vk_i\}$ [29]. A validity key vk_i can be used to verify correctness of computations using gsk_i. To establish the described key material, parties can either rely on a trusted dealer or employ a Distributed Key Generation (DKG) protocol [30].

The parties must cooperate to calculate the result of a cryptopgraphic operation that is valid under the global public key gpk. The workflow of non-interactive threshold cryptography schemes unfolds as a three-step process. Parties use their secret share gsk_i to calculate a result share rs_i which is then broadcast. Honest parties verify received result shares with the respective validity key to achieve fault tolerance. Upon accumulating a set of $t + 1$ different valid result shares, parties combine them into the global result valid under the global public key gpk. The security notions of a threshold scheme depend on its specific type. For threshold encryption, security is defined by a threshold extension of the IND-CCA security game [1]. For threshold signatures, security is defined through an extension of the EUF-CMA security game [13]. Common coins are required to be unpredictable [14].

Most threshold schemes currently used operate over elliptic curves [31]. In this setting, validity keys typically are of the form $vk_i := g^{gsk_i}$, where g is a generator of the group used, and result shares are of the form k^{gsk_i} for some other, input-dependent generator k. Verifying correctness of a result share requires to check whether, given k^x and vk_i, it holds that $x = gsk_i$ without revealing gsk_i in the process. This is typically achieved by *a)* using zero-knowledge proofs (ZKP) or by *b)* working over groups that support bilinear pairings [32]. There is a large number of threshold protocols in cryptography literature (e.g., [1] for encryption, [13, 33] for signatures, and [14] for common coins). Barbaraci et al. [34] recently introduced Thetacrypt, a software library for state-of-the-art threshold cryptography schemes with a unified interface. Their benchmarks show that ZKP-based schemes generally perform better than pairing-based ones, and non-interactive protocols outperform interactive ones.

2.2 Hybrid Fault Model

In the hybrid fault model, parties are equipped with a Trusted Execution Environment (TEE) that is assumed to only show benign faults. Hybrid fault-tolerant protocols (e.g., [19, 21]) shift functionality into the TEE enabling simpler designs and more performance. A TEE commonly comprises a set of software and hardware protection mechanisms and can provide the ability to execute arbitrary code in a so-called enclave [35]. *Enclaved execution* guarantees confidentiality and integrity: data inside the enclave cannot be read from the outside; code and data inside the enclave cannot be altered without notice. Moreover, TEEs can provide an *attestation* feature, enabling the TEE to certify the identity of an enclave, i.e., the code that is currently being executed, to a remote party. Attestation can be used to bind arbitrary data, e.g., a public key or a nonce, to the enclave identity. We refer to protocol components executed within an enclave as the trusted part and those executed outside the enclave as the untrusted

[1] Available at https://github.com/abcperf/abcperf

part. The transition between these two parts is called context switch and required to interact with the trusted part.

To the best of our knowledge, there exist only a few proposals focusing on hybrid fault-tolerant threshold cryptography. NxBFT [25] and Fides [24] use a TEE-based common coin to implement asynchronous, hybrid-fault tolerant consensus. Both approaches commonly seed a pseudo-random number generator and safeguard its state using the TEE. Bebel and Ojha [23] sketch a hybrid fault-tolerant threshold encryption scheme based on a common secret key safeguarded by the TEE. The authors of Damysus [22] propose a so-called accumulator module that is similar to a threshold signature. We generalize the ideas behind these proposals with the common key variants, investigate the trade-off introduced by using TEEs, and, additionally, optimize the resilience to broken TEEs by proposing the split key variants.

3 SpliTEE Approach and Software Design

Threshold cryptography suffers from two inefficiencies in comparison to non-threshold cryptography: First of all, many efficient cryptographic protocols are incompatible to be efficiently instantiated in a threshold setting. Secondly, the result share verification required to tolerate adversarial input causes non-negligible overhead. We observe that threshold protocols can exploit TEEs in two different ways: In a *common key* approach, the TEE has access to the global secret key gsk and uses it to compute the global result once sufficient distinct parties approved it. In a *split key* approach, the TEE has only access to a share gsk_i of the global secret key and uses it to compute result shares with the attestation certifying that a correct gsk_i was used, and thereby omitting the need for pairings or ZKPs. The common key variant allows to solely rely on non-threshold cryptographic primitives that are more efficient than threshold-compatible primitives. In the case of a broken TEE, however, the global secret key is leaked and the security property violated. The split key variant requires to use threshold-compatible primitives with the benefit of protecting the global secret key in *any* case.

SpliTEE is a Rust software library implementing the common key and split key threshold cryptography schemes as well as conventional ("standard" in the following) threshold schemes for encryption, signatures, and common coins. Rust was chosen for its memory safety guarantees, high performance capabilities, and well-maintained ecosystem. The standard threshold schemes are selected as the best performing state-of-the-art schemes based on [34] and serve as the baseline for the performance evaluation as well as the base schemes for the split key variants. The standard threshold encryption is a hybrid encryption variant of [1]. Standard threshold signatures are based on BLS signatures [13]. The standard common coin is based on a distributed pseudo-random function (PRF) [14, Section 6]. The SpliTEE library is independent from an actual TEE implementation and split into the `trusted` and `untrusted` modules. To use SpliTEE with a TEE platform that matches our requirements (cf. Section 3.1), one must instantiate the trusted part within the TEE and provide the necessary interfaces such that other library components can seamlessly interact. Each type of threshold scheme in SpliTEE exposes an implementation-independent interface, allowing applications to change the used variant without effort. With the exception of BLS signatures, they operate on Curve25519. Trusted parts of both variants use EdDSA [36] to produce validity signatures.

3.1 System Model

We consider a distributed system of n parties, each equipped with a TEE offering enclaved execution and attestation; the TEE may only fail by crashing. Parties communicate via asynchronous secure channels with eventual delivery. We consider a Byzantine adversary in the hybrid fault model that may statically corrupt $t < n/2$ parties. We assume the Decisional Diffie-Hellmann (DDH) problem to be computationally hard in the used groups. For setup, we assume the existence of a public key infrastructure (PKI) and synchrony.

3.2 Common Key Schemes

The common key variants are generalizations of related work. Each party hosts a TEE which manages a copy of the entire global secret key gsk and a set of validity keys VK. To generate result shares, parties sign the input of the cryptographic operation using their validity secret key vsk_i and a non-threshold signature scheme; result share verification reduces to a simple signature verification. Upon accumulating $t + 1$ valid result shares, an honest party provides the result shares to the trusted part. The trusted part verifies that it received signatures according to $t + 1$ validity public keys and then computes the global result using a non-threshold cryptographic base scheme with the common global secret key gsk. The achieved security depends on the security of the used base scheme and the TEE. In case of a broken TEE, gsk is leaked and the security property is violated. Robustness, however, is not affected as malicious parties cannot prevent honest parties from producing valid global results.

The common key threshold encryption scheme is based on any IND-CCA secure public-key encryption scheme. To encrypt messages, parties or external clients use the encryption algorithm of the base scheme. We rely on a hybrid ElGamal encryption using ChaCha8Poly1305 [1, Section 2.5] as the base scheme. Parties generate decryption shares by signing the ciphertext with their validity secret keys vsk_i. The trusted part uses the base scheme's decryption algorithm together with the global secret key gsk to recover the plaintext message m of the ciphertext. The common key threshold signature scheme is based on any EUF-CMA secure digital signature scheme. We rely on EdDSA as the base scheme. Parties generate signature shares by signing a message m with their validity secret key vsk_i. The trusted part uses the

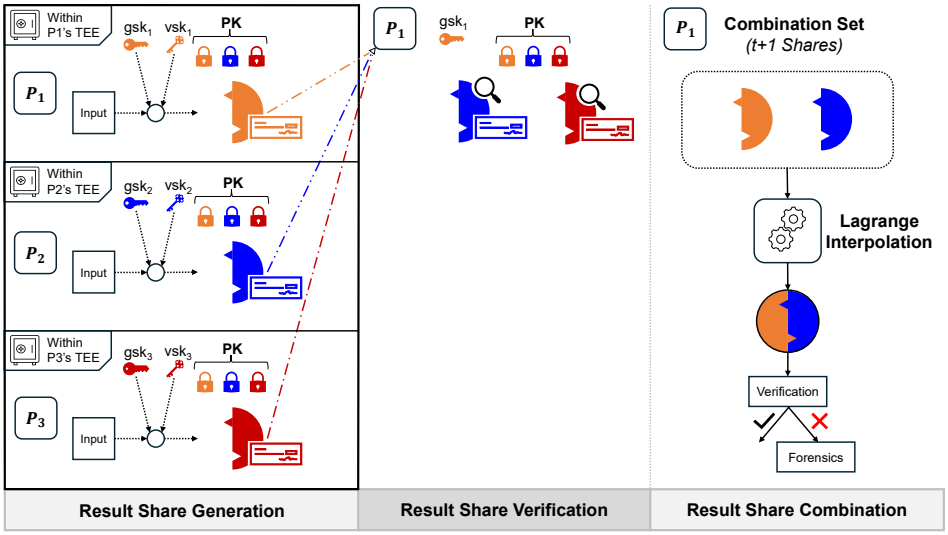


Figure 1. Protocol workflow (three steps from left to right) of the *split key* schemes at the example of three parties. Parties generate the result shares (colored semicircles, colors indicate the creator of a result share) of a standard threshold scheme within their TEE based on their secret key share gsk_i and produce a validity signature on the generated result share using their validity secret key vsk_i. Receiving parties verify the attached validity signature. Upon receiving $t + 1$ valid result shares, parties use the combination set to calculate the global operation result through Lagrange interpolation, perform a verification of the global result, and, if needed, trigger a forensic mechanism in which flawed result shares are identified.

signing algorithm of the base scheme with the global secret key gsk to compute the global signature σ on m. The common key common coin relies on SHA-512 as a local PRF. Parties generate coin shares by signing a toss identifier $c \in \mathbb{N}$ with their validity secret key vsk_i. The trusted part uses the global secret key gsk to evaluate the PRF at the toss identifier c yielding the global result.

3.3 Split Key Schemes

The split key variants are based on a standard, non-interactive *threshold* cryptography scheme that is combined with TEE-enabled share verification relying on non-threshold signature schemes. In contrast to standard threshold cryptography, only share generation and share verification are modified; only share generation is executed within the TEE. While the security property solely depends on the used base scheme, the robustness property depends on the integrity of the TEE: An adversary who breaks the TEE can generate incorrectly computed but validly signed result shares. Honest parties using a flawed result share during result share combination obtain an invalid global result. We design the split key *encryption* and split key *signature* schemes such that the use of flawed result shares does not yield a valid global result and propose a combinatorial forensic mechanism for the case of compromised TEEs. The best performing standard common coin [14, Section 6] relies on a distributed PRF for which result share verification is crucial and no function for global result verification can be defined, i.e., a process has to verify and combine the result shares itself to verify the global result. While it is possible to define a split key *common coin* based on [14, Section 6] with a slight performance gain (see Section 4.2), a forensic step cannot be supported and, therefore, we do not explain the split key common coin in detail.

The protocol flow of the split key schemes is depicted in Figure 1. Each party hosts a TEE which manages a share gsk_i of the global secret key and a validity secret key vsk_i. To generate result shares, a party's trusted part uses gsk_i to execute the share generation step of the base scheme and vsk_i to sign the result share. A valid signature certifies that the result was generated within the TEE; result share verification is reduced to a simple signature verification. Upon accumulating $t + 1$ valid result shares, parties employ the share combination algorithm of the base scheme to obtain the global result which is then verified. In the case of encryption, we rely on hybrid encryption with a symmetric encryption scheme that supports key commitment [37]; the symmetric key is secret shared between all n parties. The key commitment allows to detect if a flawed symmetric key was obtained during decryption share combination. In the case of signatures, the combined signature can be verified using the verification algorithm of the base threshold signature scheme. In both cases, if the verification of the global result fails, i.e., a party compromised their TEE, leaked vsk_i, and proposed an invalid result share that is validly signed, honest parties can identify the flawed result share by trying possible combinations of result shares (worst case $\binom{n}{t+1}$), reconstruct the correct global result, and punish the compromising party.

Strictly speaking, the split key encryption and signature schemes do neither require a TEE to maintain security nor to maintain robustness. The forensic mechanism, however, incurs immense computation cost. To this end, the TEE acts as a first line of defense to significantly reduce the probability that the combinatorial mechanism is triggered in the first place. Moreover, once a party proposed a flawed share, the forensic mechanism reveals the attacking party, which will then be excluded from the distributed system.

3.4 Setup

All eight implemented threshold schemes require pre-established cryptographic key material. The common key schemes must establish the global secret key and a common set of validity keys VK; the validity keys have to be bound to the TEE's execution context using the attestation feature. The split key schemes rely on a secret-shared secret key and a validity secret key; the validity secret key has to be bound to the TEE's execution context using the attestation feature. Moreover, one must ensure that all parties correctly initialize their TEE and that each party uses exactly *one* valid enclave. The setup is executed exactly once, i.e., there is no dependence on the number of executed threshold operations, making the corresponding overhead (e.g., due to attestation) negligible.

The setup protocol performs pairwise attestation in which the integrity of the TEE and the executed code is verified. In this step, each enclave generates a random secret key for confidential communication and authentication of computation results (authenticated encryption with associated data based on a Diffie-Hellman key exchange). The corresponding public key is part of the attestation certificate which binds the secret to the execution context. To prevent simulation attacks and to ensure robustness, all parties must agree on exactly one enclave identity per participant which is a consensus problem. In asynchronous networks, reaching (probabilistic) consensus requires a corruption threshold of at most $t < n/3$ [38]. Because the TEEs are not yet initialized, the hybrid fault model cannot be used to improve upon this lower bound. Hence, we must assume network synchrony during the setup phase to achieve the assumed corruption threshold of $t < n/2$ (cf. Section 3.1). To establish the required cryptographic key material, the setup protocol executes a DKG protocol. We adopt the synchronous construction by Gennaro et al. [30]. The split key schemes can directly use the result of the DKG as their cryptographic key material; the common key schemes derive their global secret key from the established public key. The confidential channels established in the first step of the setup ensure authenticity and confidentiality of the generated key material and transitively bind it to the TEE's execution context.

Table 1. Computational cost of threshold schemes regarding scalar multiplications (M) and pairing computations (P)

Scheme	Generation	Verification	Combination
Encryption			
Standard	$7M$	$8M$	$(t{+}1){\cdot}M{+}4M$
Common	$1M$	$2M$	$(2t{+}2){\cdot}M{+}M$
Split	$6M$	$2M$	$(t+1)\cdot M$
Signatures			
Standard	$1M$	$2P$	$(t+1)\cdot M$
Common	$1M$	$2M$	$(2t{+}2){\cdot}M{+}M$
Split	$2M$	$2M$	$(t+1)\cdot M$
Common Coin			
Standard	$3M$	$4M$	$(t+1)\cdot M$
Common	$1M$	$2M$	$(2t+2)\cdot M$

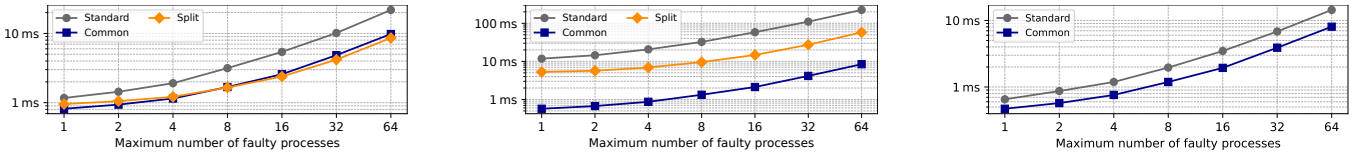
4 Evaluation

We conduct an analytical performance analysis by quantifying computationally expensive operations and validate these findings through an empirical evaluation in which we measure the average computation time of each scheme.

4.1 Analytical Assessment

The most expensive operations are bilinear pairings and scalar multiplications on elliptic curves. While TEE context switches do incur overhead [39], our microbenchmarks let us conclude that they (1) have a high variance and (2) they have no significant impact on the schemes' performances as they are at most in the range of a few microseconds. Table 1 presents the analytical cost of the TEE-based threshold schemes alongside the standard schemes, divided into the phases of share generation, share verification, and share combination. These three phases differ in their impact on the computational complexity of the overall threshold operation as both result share verification and result share combination have a linear dependency on the number of parties n.

The most critical phase is result share verification, since honest parties must verify *at least* $t + 1$ result shares. For the TEE-based variants, it is sufficient to verify an EdDSA signature which involves two scalar multiplications. Standard threshold schemes must either verify at least one ZKP [40], requiring four scalar multiplications, or evaluate two pairing functions, whose computational cost exceeds scalar operations significantly [8]. The share combination phase is the second most critical stage. Its computational complexity depends on the number of shares to be combined. The split key variants and the standard threshold schemes use Lagrange interpolation for combination, resulting in $t + 1$ scalar multiplications. The common key TEE-based constructions must re-verify all received result shares within the TEE. Since each share verification requires two scalar multiplications, the total cost of the combination phase amounts to



(a) Encryption **(b)** Signature **(c)** Common Coin

Figure 2. Average computation time of the of common key (blue), split key (orange), and standard (grey) schemes for corruption thresholds $t \in \{1, 2, 4, 8, 16, 32, 64\}$ and $n = 2t + 1$. All times are averaged over 15 runs with 30 executions each.

$2t + 2$ scalar multiplications. The share generation phase is not dependent on n and thus negligible.

4.2 Empirical Assessment

We conduct an empirical evaluation in which we measure the average computation time of the different variants and schemes. We use Intel SGX [35] and the Rust SGX SDK by the Apache Teclave project as the TEE platform. We use *ABCperf* [41], a Rust framework designed for empirical evaluation of distributed systems, to orchestrate our experiments on a heterogeneous server cluster comprising 20 servers (10 servers with *Intel Xeon E-2288G* CPUs, 10 servers with *Intel Xeon E-2388G* CPUs) that were interconnected via a 10 GBit/s interfaces. All servers ran Linux kernel version 6.8.0.

An experiment run executes a threshold operation 30 times for a fixed n and t. Each of the n parties first generates and broadcasts its result share, then collects and verifies $t + 1$ result shares, and finally executes the result share combination. Each party measures the time it takes to execute the full threshold operation, i.e., from generating the result share until obtaining the global result. We vary the corruption threshold $t \in \{1, 2, 4, 8, 16, 32, 64\}$ and choose $n = 2t + 1$. We determine the average time per operation and configuration of 15 runs; observed 95% confidence intervals are below 2%.

Figure 2a compares the performance of the threshold encryption schemes. The common key variant achieves a minimum speedup of 1.4× for $t = 1$ and a maximum speedup of 2.2× for $t = 64$. The split key variant achieves a minimum speedup of 1.2× for $t = 1$ and a maximum speedup of 2.5× for $t = 64$. Starting from $t = 16$, the split key scheme slightly outperforms the common key approach. Figure 2b compares the performance of the threshold signature schemes. The common key variant achieves a minimum speedup of 20.7× for $t = 1$ and a maximum speedup of 27.5× for $t = 16$. The split key variant achieves a minimum speedup of 2.2× for $t = 1$ and a maximum speedup of 4× for $t = 32$. Figure 2c compares the common key common coin with the standard common coin; we observe a minimum speedup of 1.4× for $t = 1$ and a maximum speedup of 1.8× for $t = 64$.[2]

[2]We implemented a split key common coin based on distributed PRFs [14, Section 6] without global result verification and forensic support that achieves for $t \geq 8$ a slightly higher speedup than the common key variant with 1.8×–2× (similar to split key encryption).

The observed behavior largely aligns with the results of the analytical complexity assessment. The most notable deviation occurs for the threshold signature schemes. Based on the analysis, we would not expect the split key approach to be an order of magnitude slower than the common key construction. However, the complexity assessment did not account for the fact that the split key approach must perform nearly half of the scalar multiplications on the pairing-friendly BLS12-381 curve. In contrast, the common key approach operates entirely on the highly optimized Curve25519, explaining the significant latency difference observed. Moreover, for increasing t, we would expect the split key encryption to significantly outperform the common key variant since it requires only half as many scalar multiplications during share combination. However, while the result share combination in the split key variant requires arbitrary scalar multiplications, the scalar multiplication with a fixed base point during EdDSA signature verification can be partially precomputed [36] reducing the expected performance advantage.

5 Discussion and Conclusion

The aim of this paper is to improve the understanding of TEE-based threshold cryptography in terms of performance and security, to increase the achieved resilience in the light of compromised TEEs, as well as to provide a comprehensive software library which simplifies the development of modern distributed algorithms. SpliTEE offers, with the common key variants, one consolidated form and, with the split key variant, one novel form of TEE-based threshold cryptography as well as state-of-the-art variants in the Byzantine fault model. The TEE-based variants can be deployed on any TEE platform offering enclaved execution and attestation. For the common key variants, it is further possible to optimistically omit the result share verification phase, typically used to detect flawed messages and malicious behavior as early as possible, without compromising the security achieved because the trusted part must re-verify each result share anyway. This optimistic optimization significantly increases the common key performance even further (~ 2×, cf. Tab. 1), eliminating any potential performance advantage of the split key variants for $t \geq 16$.

The security property of the common key variants breaks when the TEE assumption does not hold; the robustness

property is not violated. The split key variants maintain the security property but lose their robustness property. The robustness property can be broken down into a safety statement and a liveness statement: Parties are guaranteed to eventually reach agreement on the correct output value. The encryption and signature split key variants can efficiently detect if a global result would break robustness and abort, thus guaranteeing safety in any case. Moreover, they can perform a forensic step to regain liveness and to identify malicious parties, although this step is compute-intensive. If the split key common coin itself is built on top of a split key threshold signature scheme (see [14, Section 4.3]), the global result is a verifiable signature and a forensic step becomes possible. However, such a coin would be an order of magnitude slower than the best performing standard common coin (see split key variant of Figure 2b and standard variant of Figure 2c).

In conclusion, we deem SpliTEE to be a flexible and reliable building block in distributed systems research software. SpliTEE drastically decreases the computational workload of threshold cryptography schemes by utilizing TEEs. Therefore, it achieves significant throughput improvements in use cases where computational resources, in contrast to network resources, may become the bottleneck. Based on our findings, we make the following recommendations. For typical deployment scenarios, e.g., state machine replication, distributed ledgers, or blockchains, where the Byzantine fault model is assumed but actual faulty behavior is very costly for attackers and, thus, compromised TEEs are possible but *not frequent*, the split key variants are superior to standard threshold schemes: They achieve a significantly higher throughput without any security implications and a negligible probability for a forensic step. If the primary goal is maximum performance gain and successful attacks on TEEs are even less likely, the common key variants should be used. If the risk of the common key variants is not acceptable and a common coin is required, we recommend to use the standard scheme.

Acknowledgments

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF).

References

[1] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *J. Cryptol.*, vol. 15, no. 2, pp. 75–96, 2002. [Online]. Available: https://doi.org/10.1007/s00145-001-0020-9

[2] C. Berger, S. Schwarz-Rüsch, A. Vogel, K. Bleeke, L. Jehl, H. P. Reiser, and R. Kapitza, "SoK: Scalability techniques for BFT consensus," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2023, Dubai, United Arab Emirates, May 1-5, 2023*. IEEE, 2023, pp. 1–18. [Online]. Available: https://doi.org/10.1109/ICBC56567.2023.10174976

[3] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 31–42. [Online]. Available: https://doi.org/10.1145/2976749.2978399

[4] H. Zhang, L. Merino, Z. Qu, M. Bastankhah, V. Estrada-Galiñanes, and B. Ford, "F3B: A low-overhead blockchain architecture with per-transaction front-running protection," in *5th Conference on Advances in Financial Technologies, AFT 2023, Princeton, NJ, USA, October 23-25, 2023*, ser. LIPIcs, J. Bonneau and S. M. Weinberg, Eds., vol. 282. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 3:1–3:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.AFT.2023.3

[5] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *Eur. Trans. Telecommun.*, vol. 8, no. 5, pp. 481–490, 1997. [Online]. Available: https://doi.org/10.1002/ett.4460080506

[6] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, P. Robinson and F. Ellen, Eds. ACM, 2019, pp. 347–356. [Online]. Available: https://doi.org/10.1145/3293611.3331591

[7] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: BFT consensus with pipelined tree-based dissemination and aggregation," *ACM Trans. Comput. Syst.*, 2025. [Online]. Available: https://doi.org/10.1145/3769423

[8] A. Heß, F. J. Hauck, and E. Meißner, "Consensus-agnostic state-machine replication," in *Proceedings of the 25th International Middleware Conference, MIDDLEWARE 2024, Hong Kong, SAR, China, December 2-6, 2024*, J. Cao, Z. Jin, V. Schiavoni, and J. Edinger, Eds. ACM, 2024, pp. 341–353. [Online]. Available: https://doi.org/10.1145/3652892.3700776

[9] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," in *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, ser. Lecture Notes in Computer Science, M. Manulis, A. Sadeghi, and S. A. Schneider, Eds. Springer, 2016, pp. 156–174. [Online]. Available: https://doi.org/10.1007/978-3-319-39555-5_9

[10] A. P. Fournaris, "Distributed threshold cryptography certification with no trusted dealer," in *SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography, Seville, Spain, 18 - 21 July, 2011, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, J. López and P. Samarati, Eds. SciTePress, 2011, pp. 400–404.

[11] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, A. Miller, K. Censor-Hillel, and J. H. Korhonen, Eds. ACM, 2021, pp. 165–175. [Online]. Available: https://doi.org/10.1145/3465084.3467905

[12] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus," in *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, Y. Bromberg, A. Kermarrec, and C. Kozyrakis, Eds. ACM, 2022, pp. 34–50. [Online]. Available: https://doi.org/10.1145/3492321.3519594

[13] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004. [Online]. Available: https://doi.org/10.1007/s00145-004-0314-9

[14] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, G. Neiger, Ed. ACM, 2000, pp. 123–132. [Online]. Available: https://doi.org/10.1145/343477.343531

[15] Dalek Cryptography Developers, "Curve25519-dalek: Benchmarks," 2023, accessed 2025-10-16. [Online]. Available: https://github.com/dalek-cryptography/curve25519-dalek/tree/main/ed25519-dalek#benchmarks

[16] M. Camaioni, R. Guerraoui, M. Monti, P. Roman, M. Vidigueira, and G. Voron, "Chop Chop: Byzantine atomic broadcast to the network limit," in *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, A. Gavrilovska and D. B. Terry, Eds. USENIX Association, 2024, pp. 269–287. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/camaioni

[17] D. Hyland, J. Sousa, G. Voron, A. Bessani, and V. Gramoli, "Ten myths about blockchain consensus," in *Blockchains: A Handbook on Fundamentals, Platforms and Applications*, S. Ruj, S. S. Kanhere, and M. Conti, Eds. Springer International Publishing, 2024, pp. 3–24. [Online]. Available: https://doi.org/10.1007/978-3-031-32146-7_1

[18] R. von Seck, F. Rezabek, and G. Carle, "Thresh-Hold: Assessment of threshold cryptography in leader-based consensus," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, 2024, pp. 1–8. [Online]. Available: https://doi.org/10.1109/LCN60385.2024.10639786

[19] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Veríssimo, "Efficient byzantine fault-tolerance," *IEEE Trans. Computers*, vol. 62, no. 1, pp. 16–30, 2013. [Online]. Available: https://doi.org/10.1109/TC.2011.221

[20] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Trans. Computers*, vol. 68, no. 1, pp. 139–151, 2019. [Online]. Available: https://doi.org/10.1109/TC.2018.2860009

[21] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "OneShot: View-adapting streamlined BFT protocols with Trusted Execution Environments," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 2024, pp. 1022–1033. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00095

[22] ——, "DAMYSUS: streamlined BFT consensus leveraging trusted components," in *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, Y. Bromberg, A. Kermarrec, and C. Kozyrakis, Eds. ACM, 2022, pp. 1–16. [Online]. Available: https://doi.org/10.1145/3492321.3519568

[23] J. Bebel and D. Ojha, "Ferveo: Threshold decryption for mempool privacy in BFT networks," *IACR Cryptol. ePrint Arch.*, p. 898, 2022. [Online]. Available: https://eprint.iacr.org/2022/898

[24] S. Xie, D. Kang, H. Lyu, J. Niu, and M. Sadoghi, "Fides: Scalable censorship-resistant DAG consensus via trusted components," *CoRR*, vol. abs/2501.01062, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.01062

[25] M. Leinweber and H. Hartenstein, "Not eXactly Byzantine: Efficient and resilient TEE-based state machine replication," *CoRR*, vol. abs/2501.11051, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.11051

[26] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on Intel SGX," *CoRR*, vol. abs/2006.13598, 2020. [Online]. Available: https://arxiv.org/abs/2006.13598

[27] R. Buhren, H. N. Jacob, T. Krachenfels, and J. Seifert, "One glitch to rule them all: Fault injection attacks against AMD's secure encrypted virtualization," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 2875–2889. [Online]. Available: https://doi.org/10.1145/3460120.3484779

[28] J. D. Meulemeester, I. Verbauwhede, D. Oswald, and J. V. Bulck, "Battering ram: Low-cost interposer attacks on confidential computing via dynamic memory aliasing," in *47th IEEE Symposium on Security and Privacy (S&P)*, 2026. [Online]. Available: https://batteringram.eu/batteringram.pdf

[29] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007. [Online]. Available: https://doi.org/10.1007/s00145-006-0347-3

[31] N. Koblitz, A. Menezes, and S. A. Vanstone, "The state of elliptic curve cryptography," *Des. Codes Cryptogr.*, vol. 19, no. 2/3, pp. 173–193, 2000. [Online]. Available: https://doi.org/10.1023/A:1008354106356

[32] V. S. Miller, "The Weil pairing, and its efficient calculation," *J. Cryptol.*, vol. 17, no. 4, pp. 235–261, 2004. [Online]. Available: https://doi.org/10.1007/s00145-004-0315-8

[33] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 2567. Springer, 2003, pp. 31–46. [Online]. Available: https://doi.org/10.1007/3-540-36288-6_3

[34] M. Barbaraci, N. Schmid, O. Alpos, M. Senn, and C. Cachin, "Thetacrypt: A distributed service for threshold cryptography," *CoRR*, vol. abs/2502.03247, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.03247

[35] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, p. 86, 2016. [Online]. Available: http://eprint.iacr.org/2016/086

[36] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," *J. Cryptogr. Eng.*, vol. 2, no. 2, pp. 77–89, 2012. [Online]. Available: https://doi.org/10.1007/s13389-012-0027-1

[37] A. Albertini, T. Duong, S. Gueron, S. Kölbl, A. Luykx, and S. Schmieg, "How to abuse and fix authenticated encryption without key commitment," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 3291–3308. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/albertini

[38] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139. Springer, 2001, pp. 524–541. [Online]. Available: https://doi.org/10.1007/3-540-44647-8_31

[39] N. Weichbrodt, P. Aublin, and R. Kapitza, "sgx-perf: A performance analysis tool for Intel SGX enclaves," in *Proceedings of the 19th International Middleware Conference, Middleware 2018, Rennes, France, December 10-14, 2018*, P. Ferreira and L. Shrira, Eds. ACM, 2018, pp. 201–213. [Online]. Available: https://doi.org/10.1145/3274808.3274824

[40] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, ser. Lecture Notes in Computer Science, E. F. Brickell, Ed., vol. 740. Springer, 1992, pp. 89–105. [Online]. Available: https://doi.org/10.1007/3-540-48071-4_7

[41] T. Spannagel, M. Leinweber, A. Castro, and H. Hartenstein, "ABCperf: Performance evaluation of fault tolerant state machine replication made simple: Demo abstract," in *Proceedings of the 24th International Middleware Conference Demos, Posters and Doctoral Symposium, Bologna, Italy, December 11-15, 2023*. ACM, 2023, pp. 35–36. [Online]. Available: https://doi.org/10.1145/3626564.3629101