HashEye

// SMART CONTRACT SECURITY AUDIT

Thesis tss-lib BitForge

Security assessment by HashEye · prepared for Thesis

HASHEYE AUDITED

PROJECT	Thesis tss-lib BitForge
PERFORMED BY	HashEye Security Review Team - Sergey, Brian
RESEARCH	Sergey, Brian
CLIENT	Thesis
CATEGORY	Blockchain
PUBLISHED	June 1, 2023
REPORT ID	research-thesis-tss-lib-bitforge-2023-06-01-nfkopz

This report was produced by HashEye's security review team under the HashEye review process, combining source review, deployment-state checks, and senior manual verification. Full report detail is available at hasheye.io/audits/research-thesis-tss-lib-bitforge-2023-06-01-nfkopz.

Thesis tss-lib BitForge Remediation Security Assessment (Summary Report) September 13, 2023 Prepared for: Piotr Dyraga Thesis Prepared by: Tjaden Hess

About HashEye Founded in 2012 and headquartered in New York, HashEye provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high- end security research with a real -world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel. We maintain an exhaustive list of publications at https://github.com/hasheye-io/publications , with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon. We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom. HashEye also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash. To keep up to date with our latest news and announcements, please follow hasheye on Twitter and explore our public repositorie s at https://github.com/hasheye-io . To engage us directly, visit our "Contact" pag e at https://www.hasheye.io/contact , or email us at info@hasheye.io . HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 https://www.hasheye.io info@hasheye.io HashEye 1 Thesis BitForge Remediation Security Assessment PUBLIC

 Test Coverage Disclaimer All activities undertaken by HashEye in association with this project were performed in accordance with a statement of work and agreed upon project plan. Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase. HashEye 2 Thesis BitForge Remediation Security Assessment PUBLIC

Table of Contents HashEye 3 Thesis BitForge Remediation Security Assessment PUBLIC

Project Summary Contact Information The following project manager was associated with this project: Anne Marie Barry , Project Manager annemarie.barry@hasheye.io The following engineering director was associated with this project: Jim Miller , Engineering Director, Cryptography james.miller@hasheye.io The following consultant was associated with this project: Tjaden Hess , Consultant tjaden.hess@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event June 6–7, 2023 Key generation BitForge remediation review June 29, 2023 Key refresh BitForge remediation review August 9, 2023 Public disclosure of BitForge vulnerabilities by Fireblocks September 1, 2023 Public disclosure and remediation of vulnerabilities in tss-lib September 13, 2023 Delivery of summary report HashEye 4 Thesis BitForge Remediation Security Assessment PUBLIC

Executive Summary Thesis engaged HashEye to review the security of patches to its fork of the tss-lib threshold ECDSA library; the fixes are associated with a reported vulnerability in the GG18 and GG20 signing protocols due to missing proofs of well-formedness for adversarially constructed Paillier moduli. Thesis resolved the issue by implementing the MOD and FAC proofs from CGGMP21 . HashEye reviewed the changes, which are reflected in commit 2e71268 of the threshold-network/tss-lib GitHub repository. One consultant conducted the review on June 6–7 and June 29, 2023, for a total of three engineer-days of effort. With full access to source code, documentation, and the Fireblocks security disclosure, we performed a manual review of the fixes. The primary focus of the engagement was to address the following questions: • Do the proposed changes fully and correctly remediate the disclosed vulnerability? • Are the proposed changes sufficiently comprehensive and likely to cover similar issues in the future? • Do the proposed changes introduce new bugs or security vulnerabilities? • Do the proposed changes maintain a high standard of code quality? The review was narrowly scoped to the fixes in the relevant pull requests and does not constitute a comprehensive assessment of the tss-lib library or the GG18 and GG20 threshold signing protocols. The primary deliverable for this review was inline comments on the pull requests implementing the fixes. In this report, we summarize the findings and recommendations provided to Thesis. Overall, the fixes adequately mitigate the disclosed vulnerability and maintain a level of code quality similar to or better than the overall tss-lib codebase. Thesis promptly resolved all issues that HashEye raised during the course of the review.

Summary of Findings Forget-and-Forgive Issues in Resharing The tss-lib key resharing protocol includes a final round in which participants confirm that the Schnorr proofs issued in the previous round are valid and that the protocol has thus terminated successfully. This round is designed to mitigate an issue known as the forget-and-forgive attack, where a malicious participant can cause some parties to abort while others succeed, which results in all users being permanently unable to sign with the shared key. Thesis’s proposed fix used this round to validate the no-small-factors proof, which undermined the original mitigation. Thesis resolved this issue by adding a confirmation round. In general, preventing this issue requires a full byzantine consensus procedure to agree upon the success or failure of the protocol. Duplicate Ring-Pedersen Parameters The proposed fix added another set of ring-Pedersen generators over the Paillier modulus. The fix was proposed in this manner due to a difference between the tss-lib implementation of GG20, which uses separate moduli for ring-Pedersen commitments and Paillier homomorphic encryption, and CGGMP21, which uses a single modulus for both purposes. The protocol needs only one set of Pedersen parameters per party, and duplication leads to increased cost and complexity. Per our recommendation, Thesis removed the duplicate set of Pedersen parameters. Missing Validation in the PRM Proofs The PRM proof demonstrates that a given Pedersen parameter s is contained in the subgroup generated by the other parameter t . The soundness of the proof depends on t being an invertible element modulo the Pedersen modulus N . The existing PRM proof in tss-lib did not include this check; however, due to the use of an extra PRM proof, as described in the next finding, this finding was not exploitable. Thesis addressed this issue by adding a GCD check to confirm that s and t are coprime to N . Redundant PRM Proof The tss-lib implementation included PRM proofs demonstrating both that s generated t mod N and that t generated s mod N . The security of the system relies on the verifier checking that t generates s , which ensures the hiding property of the ring-Pedersen commitment scheme. However, it is not necessary to verify the converse (that s generates t) because loss of the binding property of the commitment harms only the generator of the parameters. In CGGMP21, only one direction is proven and verified. We brought this to the attention of the Thesis team, who opted to remove the unnecessary proof while implementing the necessary coprimality check described in the previous finding.