Status Update on Elephant

Tim Beyne[1], Yu Long Chen[1], Christoph Dobraunig[2,3], and Bart Mennink[2]

[1] KU Leuven and imec-COSIC, Leuven, Belgium
[2] Radboud University, Nijmegen, The Netherlands
[3] Graz University of Technology, Austria
elephant@cs.ru.nl

September 17, 2020

1 Planned Tweak Proposal

Before discussing new proofs, analyses, and implementations, we will first discuss a tweak to the Elephant v1.1 mode that we are planning to apply. In a nutshell, the change consists of moving from a Wegman-Carter-Shoup style authenticator [16,18] in v1.1 to a protected counter sum style authenticator [2,13] in v2. In addition, the role of the masks has been changed slightly to enhance the efficiency of the new variant. In more detail, Elephant v1.1 is depicted in Figure 1a and Elephant v2 is depicted in Figure 1b. Note that the roles of the masks are now $(\cdot, 0)$ for associated data authentication (used to be encryption), $(\cdot, 1)$ for encryption (used to be ciphertext authentication), and $(\cdot, 2)$ for ciphertext authentication (used to be associated data authentication).

The two versions have a comparable security bound and a comparable level of efficiency. As a bonus, Elephant v2 achieves authenticity even under nonce-reuse. In other words, these two constructions achieve the following security properties:

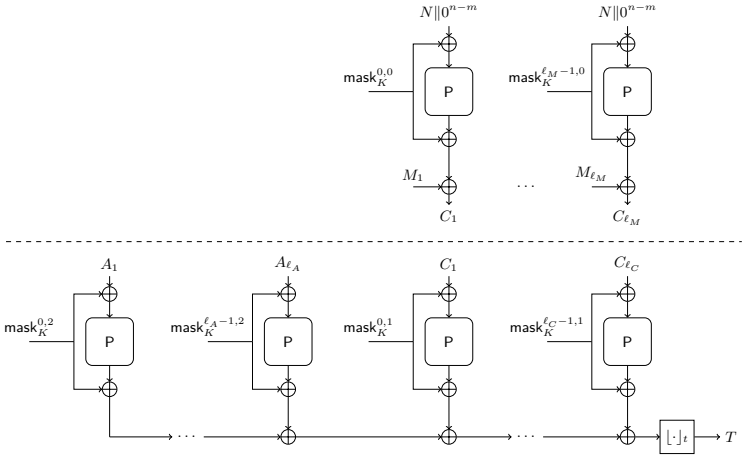
	Elephant v1.1		Elephant v2	
security	confidentiality	authenticity	confidentiality	authenticity
nonce-respecting	✓	✓	✓	✓
nonce-misuse	✗	✗	✗	✓

2 New Proofs Supporting the Security Claims

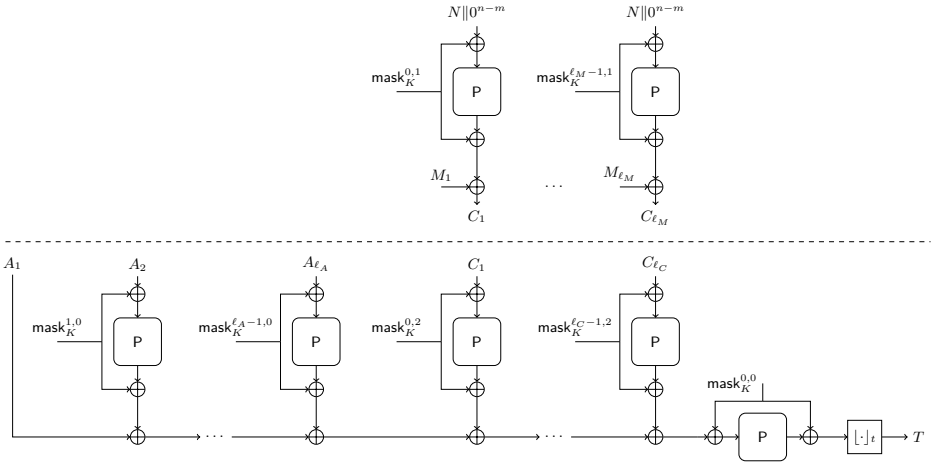
Elephant v1.1 already came with a tight generic security analysis. The security analysis has been published in [3]. We have already derived a tight generic security proof for Elephant v2.

3 New Third-Party Analysis

During the second round of the competition, various third-party analyses have appeared. We distinguish between dedicated analysis of Elephant in Section 3.1, and specific analysis on Spongent and Keccak in Section 3.2.



(a) Elephant v1.1



(b) Elephant v2

Fig. 1: Depiction of Elephant v1.1 and v2. For the encryption part (top): message is padded as $M_1 \dots M_{\ell_M} \xleftarrow{n} M$, and ciphertext equals $C = \lfloor C_1 \dots C_{\ell_M} \rfloor_{|M|}$. For the authentication part (bottom): nonce and associated data are padded as $A_1 \dots A_{\ell_A} \xleftarrow{n} N\|A\|1$, and ciphertext is padded as $C_1 \dots C_{\ell_C} \xleftarrow{n} C\|1$.

3.1 Dedicated Analysis

Zhou et al. [20] derived an interpolation attack against round-reduced Keccak-f, and applied it to Delirium. Their analysis targets a round-reduced version of the encryption of Delirium, where the number of rounds of the used variant of the Keccak permutation is reduced from the specified 18 rounds to 8 rounds. The time complexity of the attack is estimated as $2^{98.3}$ XOR operations, the memory complexity is 2^{70} bits, and the required amount of data 2^{70} blocks. The

attack makes use of the fact that an affine space of dimension 65 sums to zero after 6 rounds, which are then followed by 2 rounds for key recovery. Since the encryption of Elephant v1.1 and Elephant v2 are the same (the sole change is in the authentication), the analysis is applicable in a similar manner. We note that algebraic attacks on 8-round Delirium were anticipated in §5.3 of the Elephant specification.

3.2 Spongent and Keccak Permutation

One *new* article investigating the strength of the Spongent permutation has appeared [17]. This article only considers round-reduced versions of the Spongent permutation itself. It forms no threat to Dumbo and Jumbo (neither v1.1 nor v2). Furthermore, various works investigating the strength of the Keccak permutation have appeared [4,6–12,14,19]. These only consider round-reduced versions of the Keccak permutation or usage of this permutation in different modes. They form no threat to Delirium (neither for v1.1 nor for v2).

4 New Implementations

This section summarizes new results related to the implementation of Elephant. In Section 4.1, the impact of the proposed changes in Elephant v2 is discussed. Section 4.2 discusses a new reference implementation of Delirium which exploits parallelization. Third-party implementation results are discussed in Section 4.3.

4.1 Impact of the Changes in Elephant v2

The impact of the changes from Elephant v1.1 to Elephant v2 on existing implementations should be minimal. The final permutation call cannot be parallelized, which could have a limited impact on the performance for certain message lengths. This is only a concern for implementations that process the associated data and message in parallel when possible.

4.2 New Parallel Reference Implementation

As a proof of concept, we have released an additional parallelized reference implementation for Delirium.[4] A similar implementation will be included in the updated reference code package for Elephant v2. The new implementation processes up to eight blocks in parallel using an optimized parallel Keccak-f[200] implementation. The Keccak-f[200] implementation was generated using the *KeccakTools* package, by making suitable modifications to the Keccak-f[1600] parameters. Thus, the same strategy can be directly applied to obtain implementations with varying levels of parallelism suited to the target word size.

Unsurprisingly, the new parallel reference implementation is significantly faster than the standard reference implementation. For example, for moderately

[4] Available at https://github.com/TimBeyne/Elephant.

long messages, speedups between around 8 and 80 (depending on compilation options) over the standard reference implementation are realized. However, because only the implementation of the primitive has been optimized, we recommend against using the parallel reference implementation for benchmarking purposes. Especially for short messages, this would result in a distorted performance picture. Needless to say, the same applies to the serial reference implementation.

4.3 New Third-Party Implementation Results

A few third-party software implementations have appeared during the second round of the competition. We are not aware of any hardware-implementations of Elephant at this time.

Campos et al. [5] provided a C implementation of Delirium which exploits parallelism on platforms with 32-bit words. For messages longer than 64 bytes, their implementation achieves a speedup of about 1.5 to 2 over the original reference implementation. However, we remark that their parallel Keccak-f[200] implementation is based on a reference rather than an optimized implementation. Our own experiments with the new parallel reference implementation suggest that this can have a strong performance impact.

Belaïd et al. [1] introduced a tool to automatically generate masked (and unmasked) bitsliced implementations. They apply the tool to several primitives, including Spongent-π. Their work illustrates that, despite the hardware-oriented nature of Spongent-π, reasonable software performance can be obtained by means of bitslicing. Due to its inherent parallelism, the Elephant mode is a good match for such implementations. The implementation in [1] processes up to eight blocks in parallel, but the target word size (32 bits) would allow increasing this to up to 32 blocks. We also note that there is significant room for improvement in the efficiency of bitsliced implementations of Spongent-π, because alternative representations of Spongent-π could be exploited (as for PRESENT [15]).

5 Target Applications

In general, the target application of Elephant is lightweight cryptography with a small footprint, while still providing the option to speed up using parallel implementations. Note that, indeed, Elephant is the candidate in the competition with the *smallest cryptographic primitive*, while still achieving comparable security. This, together with the different levels of security that the mode achieves, additionally puts Elephant v2 at advantage over the current NIST standards.

References

1. Belaïd, S., Dagand, P., Mercadier, D., Rivain, M., Wintersdorff, R.: Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 311–341. Springer (2020)

2. Bernstein, D.J.: How to Stretch Random Functions: The Security of Protected Counter Sums. J. Cryptology 12(3), 185–192 (1999)
3. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. IACR Trans. Symmetric Cryptol. 2020(S1), 5–30 (2020)
4. Bi, W., Dong, X., Li, Z., Zong, R., Wang, X.: MILP-aided cube-attack-like cryptanalysis on Keccak Keyed modes. Des. Codes Cryptogr. 87(6), 1271–1296 (2019)
5. Campos, F., Jellema, L., Lemmen, M., Müller, L., Sprenkels, D., Viguier, B.: Assembly or Optimized C for Lightweight Cryptography on RISC-V? Cryptology ePrint Archive, Report 2020/836 (2020)
6. Chen, Y., Gao, X.: Quantum Algorithms for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems. Cryptology ePrint Archive, Report 2018/008 (2018)
7. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical Collision Attacks against Round-Reduced SHA-3. J. Cryptology 33(1), 228–270 (2020)
8. Kumar, R., Mittal, N., Singh, S.: Cryptanalysis of 2 Round Keccak-384. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 120–133. Springer (2018)
9. Kumar, R., Rajasree, M.S., AlKhzaimi, H.: Cryptanalysis of 1-Round KECCAK. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2018. LNCS, vol. 10831, pp. 124–137. Springer (2018)
10. Li, T., Sun, Y.: Preimage Attacks on Round-Reduced Keccak-224/256 via an Allocating Approach. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 556–584. Springer (2019)
11. Li, Z., Dong, X., Bi, W., Jia, K., Wang, X., Meier, W.: New Conditional Cube Attack on Keccak Keyed Modes. IACR Trans. Symmetric Cryptol. 2019(2), 94–124 (2019)
12. Liu, F., Cao, Z., Wang, G.: Finding Ordinary Cube Variables for Keccak-MAC with Greedy Algorithm. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 2019. LNCS, vol. 11689, pp. 287–305. Springer (2019)
13. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC Mode for Lightweight Block Ciphers. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 43–59. Springer (2016)
14. Rajasree, M.: Cryptanalysis of Round-Reduced KECCAK using Non-Linear Structures. Cryptology ePrint Archive, Report 2019/884 (2019)
15. Reis, T.B.S., Aranha, D.F., López-Hernández, J.C.: PRESENT Runs Fast - Efficient and Secure Implementation in Software. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 644–664. Springer (2017)
16. Shoup, V.: On Fast and Provably Secure Message Authentication Based on Universal Hashing. In: Koblitz, N. (ed.) CRYPTO '96. LNCS, vol. 1109, pp. 313–328. Springer (1996)
17. Sun, L., Wang, W., Wang, M.: MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. IET Inf. Secur. 14(1), 12–20 (2020)
18. Wegman, M.N., Carter, L.: New Hash Functions and Their Use in Authentication and Set Equality. J. Comput. Syst. Sci. 22(3), 265–279 (1981)
19. Zhou, H., Li, Z., Dong, X., Jia, K., Meier, W., Ashur, T.: Practical Key-Recovery Attacks On Round-Reduced Ketje Jr, Xoodoo-AE And Xoodyak. Comput. J. 63(8), 1231–1246 (2020)
20. Zhou, H., Zong, R., Dong, X., Jia, K., Meier, W.: Interpolation Attacks on Round-Reduced Elephant, Kravatte and Xoofff. Cryptology ePrint Archive, Report 2020/781 (2020)