Xoodyak, a final update

Joan Daemen[1], Seth Hoffert, Silvia Mella[1], Michaël Peeters[2],
Gilles Van Assche[2] and Ronny Van Keer[2]

[1] Radboud University
[2] STMicroelectronics

Date: September 30, 2022

In this document, we provide some new results on our candidate algorithm Xoodyak since its updated submission in May 2021. In particular, we cover the following topics:

- an update on security, including new third-party cryptanalysis, new trail bounds and a new generic attack;
- new third-party implementations;
- API flexibility, including an example of nonce-misuse resistant mode.

This document answers the call of the NIST LWC team to provide updates on the finalist algorithms.

1 Security

At the time of this writing, Xoodyak does not have any attacks beyond the generic ones. It was an explicit design goal to rigorously instantiate the full-state keyed duplex with a strong permutation. Consequently, the number of rounds, 12, is the same in all calls to the Xoodoo permutation. This approach makes Xoodyak very hard to attack.

1.1 New third-party cryptanalysis

Since the updated submission of Xoodyak in May 2021, the following papers have been published.

In [11], Yunwen Liu, Siwei Sun and Chiao Li revisit the rotational cryptanalysis from the perspective of differential-linear cryptanalysis. They apply this technique to find rotational differential-linear distinguishers with correlation 1 on Xoodoo reduced to 4 rounds.

In [1], Emanuele Bellini and Rusydi Makarim introduce functional cryptanalysis as a generalization of other techniques such as differential and rotational cryptanalysis. They show a functional distinguisher on 3 rounds of Xoodoo, which works better than a differential distinguisher.

In [18], Zimin Zhang, Wenying Zhang and Hongfang Shi attack a reduced-round version of Xoodyak using a genetic algorithm.

In [10], Kai Hu and Thomas Peyrin present a second-order differential-linear distinguisher for Xoodoo reduced to 4 rounds.

In [8], Orr Dunkelman and Ariel Weizman present differential-linear attacks on Xoodyak with 4 or 5 rounds. They can recover the key with practical complexities on these reduced-round instances.

Table 1: Lower bounds on the weight of differential and linear trails and the weight of best trails as a function of the number of rounds.

# rounds	Previous works lower bound	Previous works best known	[7] lower bound	[7] best known
1	2 [5]	2 [5]	-	-
2	8 [5]	8 [5]	-	-
3	36 [5]	36 [5]	-	-
4	74 [3]	-	80	80
5	94 [3]	-	98	120
6	108 [15]	-	132	168
8	148 [3]	-	176	288
10	188 [3]	-	220	440
12	222 [3]	-	264	624

1.2 Improved trail bounds

We have improved the bounds on the weights of differential and linear trails and reported on them in [7]. We summarize in Table 1 our new bounds and best trails found for up to 12 rounds and compare them with previously known bounds.

1.3 A new generic attack and its consequences

Via private communications [9], Henri Gilbert and co-authors informed us about a new generic attack they found against duplex-based authenticated encryption that breaks Claim 2 in [4]. The attack succeeds in key recovery and has expected data complexity 2^{148} ciphertext blocks, with extremely long forgery attempts, and computational complexity 2^{148} invocations of the underlying permutation, so it is academic in nature. We have investigated the attack and confirm that it does not contradict the proven bound for full-state keyed duplex in [6] that Claim 2 in [4] is based on.

The attack is not covered by Claim 2 because we underestimated the resource parameter Ω in [6]. We cite: $\Omega \leq M$ *is the number of blocks, in keyed mode, that overwrite the outer state (i.e., the first* R_{kout} *bytes of the state) and for which the adversary gets a subsequent output block. In particular, this counts the number of blocks processed by* DECRYPT$(\cdot)$ *for which the adversary can also get the corresponding key stream value or other subsequent output.*

In fact, the new generic attack exploits the outcome of a tag verification in forgery attempts following a block that overwrites the outer state. Claim 2 can be fixed by *including in* Ω *all the decryption blocks in forgery attempts*, regardless of whether or not the adversary gets a subsequent output block. The consequences for the claimed security can be expressed by adapting Corollary 2 from [4], that we will do hereunder.

If the nonce is not repeated and if forgery attempts do not leak unverified decrypted ciphertexts and only returns whether a tag is valid, then we still have $L = 0$ in Claim 2. In addition, if the nonce is globally unique in order to ensure $q_{\text{iv}} = 1$, we obtain the following simplified security claim.

New Corollary 2 *Assume that (1* XOODYAK *satisfies Claim 2; (2) this authenticated encryption scheme is fed with a single* κ*-bit key with* $\kappa \leq 192$*; (3) it is implemented such that the nonce is not repeated and the decryption only leaks whether a tag is valid. Then, it can be distinguished from an ideal scheme with an advantage whose dominating terms are:*

$$\frac{N}{2^{\kappa}} + \frac{N}{2^{184}} + \frac{N\Omega}{2^{192}} + \frac{\Omega^2}{2^{193}} + \frac{M^2}{2^{192+\kappa}} \ .$$

When $\Omega \lll 2^{96}$ and $M \lll 2^{96+\kappa/2}$ this translates into the following security strength levels assuming a t-byte tag (the complexities are in bits):

	security strength $(\log_2 N)$
plaintext confidentiality	$\min(184, 192 - \log_2 \Omega, \kappa, 8t)$
plaintext integrity	$\min(184, 192 - \log_2 \Omega, \kappa, 8t)$
associated data integrity	$\min(184, 192 - \log_2 \Omega, \kappa, 8t)$

2 New third-party implementations

Here we present new implementations from third-parties, of Xoodyak or the underlying permutation Xoodoo.

- In [14], Shelton et al. developed a code rewrite engine and leakage emulator, called Rosita and Elmo, that detect leakage in masked software implementations of cryptographic primitives, and automatically correct these implementations to remove leaky instructions. They applied this technique on several cryptographic primitives, including Xoodoo, which they choose for its simplicity and suitability to produce efficient implementations on 32-bit architectures. They corrected the original masked implementation of Xoodoo and successfully eliminated all leakage up to one million traces. The original code, running on the well-known Cortex-M0, ran in 637 cycles and the fixed one in 769 cycles.

- The ATHENa project [2], run by the Cryptographic Engineering Research Group at George Mason University, is currently evaluating all finalist algorithms submitted to the LWC competition. They provide several protected software and hardware implementations of Xoodyak using various masking methodologies, and also provide evaluation reports from side-channel security evaluation labs.

- In [16], Wakeland developed ASIC implementations of Xoodyak and AES-128 for comparison purposes. He presented area and throughput figures for both schemes using TSMC 5nm and 16nm technologies. As also summarized by Wakeland in a message sent to the LWC mailing list on February 11, when targeting high throughput, Xoodyak can perform 2.7x better than AES-128, still having a smaller area. While, when targeting small area, Xoodyak provides an area reduction of 28% in 5nm and 53% in 16nm, still achieving a higher throughput than AES. Note that he did not implement modes on top of AES, to support AEAD and have a fairer comparison in terms of functionality, since he observed that this could only have a negative impact on AES area and throughput. Therefore, the performance factors reported are only lower bounds.

3 API flexibility

All candidates submitted to the LWC competition have to follow a strict and well defined interface. Although this is necessary to evaluate all submissions on an equal basis, we think that API flexibility is an important asset for a lightweight cryptographic primitive. Xoodyak offers such flexibility along with a clear security contract that allows designers to better meet the security and/or performance targets of their application.

Here we remind our tweak in the previous round that exploits this flexibility, and then show how this flexibility can be used to offer a better resistance against nonce-misuse, as well as to mitigate side-channel attacks.

3.1 Performance

Between round 2 and round 3, we proposed a different way to absorb the nonce using the existing Cyclist interface. This saves one call to the Xoodoo[12] permutation in keyed mode, which has a significant impact for short messages. The improvement over the round 2 performance is cleary visible in the benchmarks [12, 17].

3.2 A nonce-misuse resistant mode

Cryptographic primitives are often analysed in a degraded form, either by reducing some security parameters or by using the primitives outside the recommended setting, in order to obtain practical results on the analysis. Although these results do not necessarily prevent a safe use of the primitives, they allow to better assess their strength or even to highlight some weaknesses in their design.

A common attack setting is the one of nonce-misuse, where the attacker is allowed to use the primitive with the same key and nonce repeatedly. Depending on the primitive and mode, a nonce-misuse may either have no impact, or may result in the leakage of one or several plaintext block(s), or leakage of the internal state, or even retrieval of the secret key.

The candidate we submitted to the LWC competition uses stream encryption and, as stated in our specification, nonce-misuse may lead to the leakage of at most one plaintext block. This is a generic result that is inherent to any stream encryption mode, and we confirm that our candidate is not vulnerable to any nonce-misuse attack leading to secret state or secret key retrieval.

Thanks to the generic API offered by Xoodyak, one may however improve this result and uses Xoodyak in a mode that prevents any plaintext leakage even in the case of nonce-misuse. There are several alternatives, and below we propose one of them that has a good trade-off between simplicity and performance. This mode reuses the idea of synthetic IV (SIV) [13], which assumes that the device has access to the entirety of the plaintext, which is often the case in embedded devices.

Although this mode has two passes, using it with Xoodyak benefits from the greater absorption rate of the plaintext in the first pass. The gain depends on the actual implementation, but the overall processing time of this mode will be less than twice the one of the regular AEAD mode (typically only 60% overhead in SW implementations).

$\textsc{Cyclist}(K, \text{nonce}, \epsilon)$
$\textsc{Absorb}(A)$
$\textsc{Absorb}(P)$
$T \leftarrow \textsc{Squeeze}(t)$
$\textsc{Cyclist}(K, \text{nonce}, \epsilon)$
$\textsc{Absorb}(T)$
$C \leftarrow \textsc{Encrypt}(P)$
return (C, T)

3.3 Protections against side-channel and fault attacks

Xoodyak was designed with protections against side channel attacks in mind. The submission document mentions various techniques to help prevent them. While we do not want to repeat these ideas here, we would like to stress that some of these techniques rely on the rich Cyclist interface and cannot be applied without reconsidering the NIST-defined API. For instance, the support of sessions or the use of rolling subkeys frustrate side-channel attacks by making the sensitive value a moving target; however, these techniques do not fit in the defined API. The trickled absorbing of the diversifier is another technique that

could possibly fit in the defined API, but the point of this technique is to cache previously computed values to avoid exposing them again. It is therefore recommended to keep a state associated to a given key and to pass it every time the key is used.

We also remind that Xoodyak provides a Ratchet() interface that can be used to prevent transforming any state recovery attack into a key recovery attack.

The opportunity of using one method or the other is best decided at protocol or application level. We feel that different users of lightweight cryptography can have varying requirements. Again, having a flexible API and leaving freedom in how the side-channel attacks are mitigated seems like a natural consequence of this.

References

[1] Emanuele Bellini and Rusydi H. Makarim, *Functional cryptanalysis: Application to reduced-round Xoodoo*, IACR Cryptol. ePrint Arch. (2022), 134.

[2] CERG, *ATHENa: Automated Tool for Hardware EvaluatioN – Lightweight Cryptography in Hardware and Embedded Systems*, https://cryptography.gmu.edu/athena/index.php?id=LWC, 2022.

[3] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, *Xoodyak, a lightweight cryptographic scheme*, IACR Trans. Symmetric Cryptol. **2020** (2020), no. S1, 60–87.

[4] ———, *Xoodyak, a lightweight cryptographic scheme*, Submission to NIST Lightweight Cryptography Standardization Process (round 3), May 2021, https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/xoodyak-spec-final.pdf.

[5] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer, *The design of Xoodoo and Xoofff*, IACR Trans. Symmetric Cryptol. **2018** (2018), no. 4, 1–38.

[6] J. Daemen, B. Mennink, and G. Van Assche, *Full-state keyed duplex with built-in multi-user support*, IACR Cryptology ePrint Archive **2017** (2017), 498.

[7] Joan Daemen, Silvia Mella, and Gilles Van Assche, *Tighter trail bounds for Xoodoo*, Cryptology ePrint Archive, Paper 2022/1088, 2022, https://eprint.iacr.org/2022/1088.

[8] Orr Dunkelman and Ariel Weizman, *Differential-linear cryptanalysis on Xoodyak*, NIST Lightweight Cryptography Workshop, 2022.

[9] Henri Gilbert, Rachelle Heim Boissier, Louiza Khati, and Yann Rotella, *Private communications*, 2022.

[10] Kai Hu and Thomas Peyrin, *Revisiting higher-order differential(-linear) attacks from an algebraic perspective – applications to Ascon, Grain v1, Xoodoo, and ChaCha*, NIST Lightweight Cryptography Workshop, 2022.

[11] Yunwen Liu, Siwei Sun, and Chao Li, *Rotational cryptanalysis from a differential-linear perspective - practical distinguishers for round-reduced FRIET, Xoodoo, and Alzette*, Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Anne Canteaut and François-Xavier Standaert, eds.), Lecture Notes in Computer Science, vol. 12696, Springer, 2021, pp. 741–770.

[12] Sebastian Renner, Enrico Pozzobon, and Jürgen Mottok, *NIST LWC software performance benchmarks on microcontrollers*, https://lwc.las3.de/, 2022.

[13] P. Rogaway and T. Shrimpton, *A provable-security treatment of the key-wrap problem*, Advances in Cryptology - EUROCRYPT 2006, Proceedings (S. Vaudenay, ed.), Lecture Notes in Computer Science, vol. 4004, Springer, 2006, pp. 373–390.

[14] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom, *Rosita: Towards automatic elimination of power-analysis leakage in ciphers*, Proceedings 2021 Network and Distributed System Security Symposium, Internet Society, 2021.

[15] The Keccak Team, *Updated bounds on differential and linear trails in Xoodoo*, https://keccak.team/2021/updated_bounds_xoodoo.html, 2021.

[16] Michael C. Wakeland, *Asic benchmarking for proposed lightweight cryptography standard xoodyak*, March 2022, https://calhoun.nps.edu/handle/10945/69718.

[17] Rhys Weatherley, *AVR/ARM microcontroller benchmarking*, https://rweather.github.io/lwc-finalists/performance.html, 2022.

[18] Zimin Zhang, Wenying Zhang, and Hongfang Shi, *Genetic algorithm assisted state-recovery attack on round-reduced Xoodyak*, Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II (Elisa Bertino, Haya Shulman, and Michael Waidner, eds.), Lecture Notes in Computer Science, vol. 12973, Springer, 2021, pp. 257–274.