



Preimage-type Attacks for Reduced Ascon-Hash: Application to Ed25519

Marcel Nageler¹ , Lorenz Schmid¹, and Maria Eichlseder¹ 

Graz University of Technology, Graz, Austria

marcel.nageler@tugraz.at, schmid.lorenz@protonmail.com,

maria.eichlseder@tugraz.at

Abstract. Hash functions and extendable output functions are some of the most fundamental building blocks in cryptography. They are often used to build commitment schemes where a committer *binds* themselves to some value that is also hidden from the verifier until the opening is sent. Such commitment schemes are commonly used to build signature schemes, e.g., Ed25519 via Schnorr signatures, or non-interactive zero-knowledge proofs. We specifically analyze the binding security when Ascon-Hash256 or Ascon-XOF128 is used inside of Ed25519, which is closely related to finding second preimages. While there is ample prior work on Ascon-XOF128 and Ascon-Hash256, none of it applies in this setting either because it analyzes short outputs of 64 or 128 bits or because the complexity is above the security claim and generic attack of 128 bits. We show how to exploit the setting of finding a forgery for Ed25519. We find that this setting is quite challenging due to the large 320-bit internal state combined with the 128-bit security level. We propose a second-preimage attack for 1-round Ascon-Hash256 with a complexity of 2^{64} Gaussian eliminations and a random-prefix-preimage attack (also known as Nostradamus attack) for 1-round Ascon-Hash256, for the Ed25519 setting, with complexity $2^{29.7}$ Gaussian eliminations.

Keywords: Digital Signatures · Hash Functions · Preimage Attacks

1 Introduction

The security of hash functions and eXtendable Output Functions (XOF) is commonly discussed by analyzing the classical security notions of preimage resistance, second-preimage resistance and collision resistance. Preimage resistance means that it is difficult to find a message M for a given hash h , such that $\mathcal{H}(M) = h$. Second-preimage resistance means that it is difficult to find a second message M' for a given M (and thus h) such that $\mathcal{H}(M') = \mathcal{H}(M) = h$. Finally, collision resistance means that it is difficult to find messages M, M' such that $\mathcal{H}(M) = \mathcal{H}(M')$. These security notions cover many use cases but are not always an exact fit.

For example, when building a commitment scheme from a hash function, the above notions are not enough. The standard construction is that the committer chooses some fixed-length random value R , and then generates the commitment h

for M by hashing $h = \mathcal{H}(R \parallel M)$ and sending h to the verifier. This construction is commonly used in the Fiat-Shamir transformation [FS86] which, in turn, is used for signature schemes like Schnorr signatures [Sch89] and Ed25519 [Nat23] as well as non-interactive zero-knowledge proofs. Later, the commitment can be opened by sending R, M to the verifier who checks whether $\mathcal{H}(R \parallel M) = h$. Such a scheme needs to be *binding*, i.e., it must be difficult for the committer to change the message M after sending h , and *hiding*, i.e., it must be difficult for the verifier to learn (information about) the message M . While binding security follows from collision resistance of \mathcal{H} , hiding security requires additional properties that ensure that the hash function does not leak some part of the message [HM96].

When a commitment scheme is used in a concrete construction, the security requirements might again change. For example, the above hash-based commitment scheme is used in Ed25519, a NIST-standardized deterministic variant of Schnorr signatures. For this setting Neven et al. [NSW09] define the security requirements for hash functions as random-prefix-preimage (**rpp**) resistance and random-prefix-second-preimage (**rpsp**) resistance. When formulated in terms of a commitment scheme, **rpp** security can be phrased as follows. The attacker picks a commitment h and is assigned a random prefix R . Then, the attacker has to find a message M such that $\mathcal{H}(R \parallel M) = h$. This attack is also called a *Nostradamus* or *chosen-target-forced-prefix* attack and has been applied generically to Merkle-Damgård hashing via the Herding attack [KK06], and in dedicated analysis to AES-based hashing [ZSWH23, DGL⁺24]. We analyze exactly this setting as it can be used to create forgeries for Schnorr-based signatures, like Ed25519.

The main motivation for this work is the recent standardization of the Ascon family for lightweight cryptography by NIST [Nat24]. The standard includes the hash function Ascon-Hash256 with fixed 256-bit output size and Ascon-XOF128, which is very similar in its specification, but can return outputs of arbitrary size while claiming up to 128 bits of security. These hashing functions are significantly more lightweight than the SHA-2 family, particularly in terms of their area footprint, which is very relevant for embedded devices and other constrained environments. This raises the question whether the currently used SHA-512 in Ed25519 can be replaced by Ascon-XOF128 without loss of security.

Related Work. The Ascon family has been thoroughly analyzed during the CAESAR and NIST LWC competitions, with the hash functions mainly in focus for the last few years. NIST’s standardized versions Ascon-XOF128 and Ascon-Hash256 [Nat24] have some minor differences compared to the submitted versions Ascon-XOF and Ascon-Hash [DEMS19a] that were the main target of the analysis, but the results are essentially directly applicable.

While there is ample work on finding preimages for Ascon-XOF128 and Ascon-Hash256, none of it is applicable to the specific setting of Ed25519. We believe this is due to 2 reasons. First, the security claim for Ascon-XOF128 with L -bit output is $\min(128, L)$ -bit preimage resistance. As Ascon’s rate is 64 bits, we primarily see preimage attacks on 64-bit outputs with complexity below 2^{64} , which can then often be generically extended to preimage attacks on 128-bit

outputs with complexity below 2^{128} . Attacking longer outputs is not appealing, since the attack becomes more difficult but the security claim stays 128 bits. The difficulty is primarily that more permutation calls (i.e., more rounds in total) need to be attacked to gain sufficient degrees of freedom and match longer output. However, in 2022 Lefevre and Mennink [LM22] found that the bound for preimage resistance (but not second preimage resistance) is not tight and can be increased to 192 bits for 256-bit outputs. This leads to some preimage attacks on **Ascon-Hash256** (with 256-bit output) targeting a complexity below 2^{192} . These resulting attacks have complexity higher than 2^{128} , above the standard’s security claim, and are thus worse than the generic random-prefix/second preimage attack.

We summarize existing results in Table 1. The designers of **Ascon** provide preliminary analysis, including linear equations obtained from the hash value for preimages on 2 rounds with 64 bit output and degree-based accelerated brute-force for 5- and 6-round **Ascon-XOF** with 64-bit output. Qin et. al. [QHD⁺23]

Table 1: Comparison of preimage attacks on **Ascon-XOF128** or **Ascon-Hash256** in related work. **rpp** denotes random-prefix preimage. GE: Gaussian eliminations.

Output	Setting	#R	Complexity	Strategy	Reference
64 bit	preimage	2	$2^{31.6}$	Linearize & Guess	[LHC ⁺ 23]
	preimage	2	2^{34}	Guess & Determine	[BKK24]
	preimage	2	2^{39} GE	Linearization	[DEMS19b]
	preimage	3	2^{51} GE	Guess & Determine	[FLYS23]
	preimage	3	2^{56} GE	Guess & Determine	[BKK24]
	preimage	4	2^{63}	Guess & Determine	[BKK24]
	preimage	5	$2^{58.9}$	Degree-Based	[DEMS19b]
	preimage	6	$2^{63.2}$	Degree-Based	[DEMS19b]
128 bit	preimage	2	2^{98}	Guess & Determine	[BKK24]
	preimage	3	2^{112}	Linearize & Guess	[LHC ⁺ 23]
	preimage	3	2^{120}	Guess & Determine	[BKK24]
	preimage	3	2^{120}	Differential-Linear	[NHS ⁺ 24]
	preimage	4	2^{124}	Linearize & Guess	[LHC ⁺ 23]
	preimage	4	2^{125}	Meet-in-the-Middle	[QHD ⁺ 23]
	preimage	4	2^{125}	Differential-Linear	[NHS ⁺ 24]
	preimage	4	2^{127}	Guess & Determine	[BKK24]
256 bit	preimage	3	2^{163}	Meet-in-the-Middle	[DZQ ⁺ 24]
	preimage	3	2^{184}	Differential-Linear	[NHS ⁺ 24]
	preimage	4	2^{185}	Meet-in-the-Middle	[DZQ ⁺ 24]
	preimage	4	2^{189}	Differential-Linear	[NHS ⁺ 24]
	preimage	5	2^{191}	Meet-in-the-Middle	[DZQ ⁺ 24]
any (n -bit)	2 nd preimage	1	2^{64} GE	Linearization	Section 3
	preimage	1	2^{64} GE + 2^{n-128}	Linearization	Section 3
	rpp preimage	1	$2^{29.7}$ GE $\approx 2^{35.3}$	Linearization	Section 4

apply the Meet-in-the-Middle attack to various sponge-based hash functions and find preimage attacks for Ascon-XOF with 128-bit output with complexity 2^{125} .

Several papers build upon the linearization approach, some of them also inspired by related attacks on Keccak [GLS16]. Li et al. [LHC⁺23] propose a linearize and guess approach, where they first guess some set of bits of the input and then create a linear equation system for the missing bits. Furthermore, they combine multiple ways of choosing the set of guessed bits to increase the effective degrees of freedom in their attack. They find preimage attacks with complexity $2^{31.6}$ and 2^{124} for 2-round Ascon-XOF with 64-bit output and 4-round Ascon-XOF with 128-bit output, respectively. Fu et al. [FLYS23] use MILP modeling to find a good set of guessed bits yielding a preimage attack on 64-bit 3-round Ascon-XOF with complexity 2^{51} . Baek et al. [BKK24] show how to guess bits to be able to linearize 1 output bit after 4 rounds, leading to attacks on 4-round Ascon-XOF with complexity 2^{63} and 2^{127} for 64-bit and 128-bit outputs, respectively.

Based on the improved bound by Lefevre and Mennink on the generic complexity for first preimage attacks of 2^{192} , two papers propose first preimage attacks with complexity below 2^{192} . Niu et al. [NHS⁺24] analyze the security of Ascon-Hash against preimages using high-probability differential-linear distinguishers and find preimage attacks for 4-round Ascon-Hash with complexity 2^{189} . Dong et al. [DZQ⁺24] use a Meet-in-the-Middle approach to find preimages for 5-round Ascon-Hash with complexity 2^{191} . Note that these works do not apply to random-prefix/second preimages, where the generic attack has complexity 2^{128} .

Our Contribution. We provide a dedicated analysis of Ascon hashing when used in Ed25519. We present the first preimage-type attacks on round-reduced Ascon-Hash256 and Ascon-XOF128 with output longer than 128 bits and complexity below the claimed 2^{128} . Our contribution can be summarized as follows.

- We propose a second-preimage attack on 1-round Ascon-Hash256 and Ascon-XOF128 with arbitrarily long outputs with complexity 2^{64} . We also extend this to a first-preimage attack on Ascon-XOF128 with n -bit output with complexity $2^{64} + 2^{n-128}$ which is below the 128-bit claim for $n \leq 255$.
- We show that the probability of finding a preimage in a random-prefix-preimage setting can be greatly amplified by choosing the target state according to good linear approximations.
- By combining the amplified probability with a new efficient (≈ 7.9 Gaussian eliminations) method to find an internal state state matching 128 conditions, we propose a random-prefix-preimage attack on 1-round Ascon-Hash256 and Ascon-XOF128 with arbitrarily long outputs with complexity $2^{29.7}$.

Outline. In Section 2, we discuss background on Ed25519 and the Ascon cipher suite. In Section 3, we present our second-preimage attack and how to extend it to a first-preimage attack. In Section 4, we show how we can optimize the techniques for a random-prefix-preimage attack. We conclude in Section 5.

2 Background

In this section, we discuss background on Ed25519 (Section 2.1), background on Ascon (Section 2.2) and discuss the combination of the two in Section 2.3.

2.1 EdDSA and Ed25519

The Edwards-curve Digital Signature Algorithm (EdDSA) is a digital signature algorithm based on a deterministic variant of Schnorr signatures [Sch89] which operates on twisted Edwards curves. It is standardized by NIST as part of the digital signature standard [Nat23] and also specified in RFC 8032 [JL17]. We show the signature generation and verification routines in Figure 1.

In contrast to Schnorr signatures, where an ephemeral private key (sometimes also called a nonce) is required, EdDSA generates this key using a hash over a private value and the message. There are two versions of EdDSA standardized by NIST: Ed25519 and Ed448, which provide approximately 128 or 224 bits of security, respectively. In this work, we focus on Ed25519.

The signature generation for Ed25519 proceeds as follows. First, the private key d is expanded using SHA-512: $h_1 \parallel h_2 = \text{SHA-512}(d)$. Then, the private scalar s is calculated by interpreting h_1 as a little endian integer and clearing bits 0, 1, 2, and 254, and setting bit 255. The ephemeral private key r is formed by hashing $h_2 \parallel M$ and interpreting the result as a little-endian integer. The signature is formed by $R \parallel S$. The ephemeral public key R is calculated as $R = r \cdot G$. Then, the signer commits to R , Q , and M by hashing $h = \text{SHA-512}(R \parallel Q \parallel M)$. Finally, S is calculated using $S = r + h \cdot s \bmod n$.

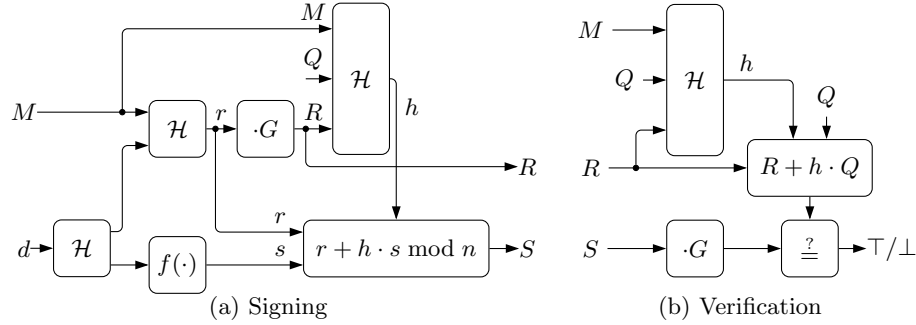


Fig. 1: Ed25519 signing and verification routines. G is the generator of the Elliptic Curve Group. $Q = s \cdot G$ is the public key for the private key d . The inputs of the hash function \mathcal{H} are processed bottom-to-top.

Security requirements for the hash function. The hash function is used in 3 different contexts with different security requirements when signing a message. First, for expanding the private key, the hash function needs to fulfill the

requirements of a secure pseudorandom function (PRF). Here, the capabilities of an attacker are rather limited. However, statistical biases in the output could lead to attacks. Next, for generating the ephemeral private key r , again a secure PRF is needed. Here, a statistical bias in the output could be used to recover the private key by solving an instance of the hidden number problem [BV96]. Finally, the hash function is used to commit to R , Q , and M . Here, it is sufficient (but not necessary) for the hash function to serve as a binding commitment scheme. In other words, for a given/chosen commitment h it must be computationally hard to find an opening for a value that was not committed to.

Forgeries exploiting non-binding commitment schemes. Neven et al. more carefully analyze necessary requirements for hash functions used in Schnorr signatures, and define random-prefix-preimage (**rpp**) resistance and random-prefix-second-preimage (**rpsp**) resistance [NSW09] as necessary requirements. Via a non-tight reduction in the generic group model, they also prove them to be sufficient requirements. The two requirements are formulated as follows: For **rpp**, the adversary picks a target hash H (corresponding to h in Ed25519), then a prefix R is chosen at random, finally, the adversary wins if they find a message M such that $\mathcal{H}(R \parallel M) = H$. For **rpsp**, the adversary picks a message M , then a prefix R is chosen at random, finally, the adversary wins if they find a message M' such that $\mathcal{H}(R \parallel M) = \mathcal{H}(R \parallel M')$. From Figure 1b, it is clear that breaking **rpp** resistance leads to a forgery: An adversary picks h and S to calculate R using the verification equation, then a message M with $\mathcal{H}(R \parallel Q \parallel M) = h$ is a forgery. The same idea works when breaking **rpsp** by requesting a signature for M .

2.2 The Ascon cipher suite and Ascon-XOF128

Ascon is a family of lightweight cryptographic algorithms for authenticated encryption and hashing. Originally proposed as a candidate in the CAESAR competition (2014–2019) [DEMS14] for authenticated encryption (AEAD) and selected as the winning “primary choice” for lightweight authenticated encryption, the family was later extended with various keyed and unkeyed hashing functions [DEMS19a,DEMS24]. The designers’ specification was published in the Journal of Cryptology [DEMS21]. In the NIST Lightweight Cryptography (LWC) Project, NIST selected Ascon as the winner of the corresponding LWC competition (2019–2023). Consequently, NIST standardized a selection of the family members in NIST SP 800-232, currently available as an initial public draft [Nat24]: the authenticated encryption scheme Ascon-AEAD128 and the unkeyed hashing functions Ascon-Hash256 (with fixed 256-bit output), Ascon-XOF128 (eXtensible Output Function with variable output length), and Ascon-CXOF128 (Customizable XOF with variable output length and supporting a customization string as an additional input). All of these are designed for a security level of 128 bits, or more specifically, $\min(128, L)$ -bit preimage resistance and $\min(128, L/2)$ -bit collision resistance of the XOF or CXOF with L -bit hash output. Compared to the designers’ specification, the NIST version incorporates some tweaks, including a switch from big-endian to little-endian specifications and the addition of the CXOF variant.

Specification of the Ascon- p permutation. All Ascon family members are based on the same lightweight permutation, Ascon- p , with different numbers of rounds. The round function consists of three steps which operate on a 320-bit state divided into 5 words S_0, S_1, S_2, S_3, S_4 of 64 bits each [Nat24]:

- **Round constant addition** p_C : XORS a round constant \mathcal{RC} to word S_2 . The constant is simply a concatenation of two 4-bit counters and can be found in the specification. While our attacks take the round constants into account, we do not always mention it explicitly to keep the descriptions simple.
- **Nonlinear substitution layer** p_S : applies the 5-bit S-box \mathcal{S} given in Figure 2a 64 times in parallel in a bit-sliced fashion (vertically, across words). Here, \oplus denotes XOR and \odot denotes AND.
- **Linear diffusion layer** p_L : XORS different rotated copies of each word (horizontally, within each word) as specified in Figure 2b.

We denote the state S , at the input to round i and thus to the round constant addition and S-box, as x^i , and the input to the linear layer as y^i . Accordingly, we index the words as x_0, \dots, x_4 in our analysis, similar to the designers’ notation [DEMS21]. Since we focus our analysis on single-round permutations, we use i to index permutation calls instead of rounds.

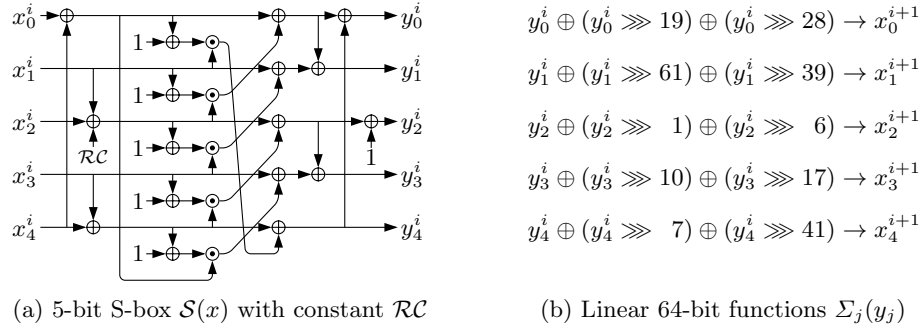


Fig. 2: Round function of Ascon- p in round i .

Specification of Ascon-XOF128. The eXtensible Output Function Ascon-XOF128 takes a message input M of arbitrary length and produces a hash output H of arbitrary length. Internally, it uses the 12-round permutation Ascon- p [12] in a sponge construction [BDPV07] with a rate of $r = 64$ bits and a capacity of $c = 256$ bits, as illustrated in Figure 3.

- **Initialization:** The state S is initialized with a fixed initial value, specifically

$$S \leftarrow \text{Ascon-}p[12](0x0000080000cc0003 \parallel 0^{256}).$$

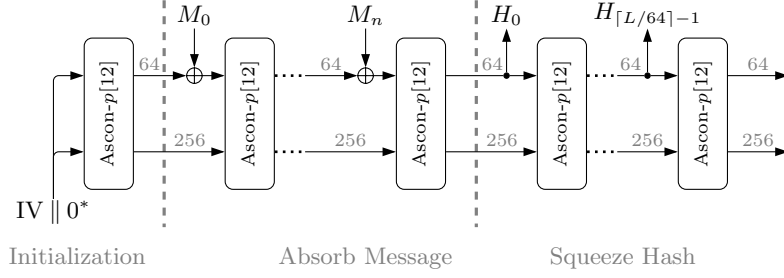


Fig. 3: Mode of operation for Ascon-XOF128.

- **Message Absorption and Padding:** The input message M is padded to a multiple of 64 bits by appending the bitstring $1 \parallel 0^*$; in the little-endian encoding of SP 800-232 and when considering byte-level messages, this corresponds to appending the byte $0x01$ followed by as many zero bytes as necessary. We remark that this means that when considering bit-level messages, any 64-bit bitstring except the all-zero string corresponds to a valid padded last message block; when considering byte-level messages, an 8-byte byte sequence is a valid padded last message block if it ends in $0x01$ (or $0x01 \ 0x00$, or $0x01 \ 0x00 \ 0x00$, etc).

The resulting padded message blocks are indexed as M_0, \dots, M_n when $64n \leq |M| \leq 64n + 63$. For each message block M_i , the state is updated by XORING the message block and applying the permutation:

$$\begin{aligned} S_0 &\leftarrow S_0 \oplus M_i \\ S &\leftarrow \text{Ascon-}p[12](S). \end{aligned}$$

- **Hash Squeezing:** If an L -bit hash output is requested, the XOF produces $\lceil L/64 \rceil$ output blocks H_j , truncating the last output block accordingly:

$$\begin{aligned} H_j &\leftarrow S_0 \\ S &\leftarrow \text{Ascon-}p[12](S). \end{aligned}$$

Generic security of Ascon-XOF128. XOFs aim to provide similar security properties as hash functions, but the specific security levels also depend on the output size. The security level typically increases with the output size according to generic bounds, but is capped at some target security level based on the internal primitive and construction. More specifically, Ascon-XOF128 with L -bit hash output claims $\min(128, L)$ -bit (first and second) preimage resistance and $\min(128, L/2)$ -bit collision resistance. The limit $128 = \frac{c}{2}$ corresponds to the birthday bound in the inner part of the state. The function also provides some additional properties, such as resistance to length-extension attacks. This follows from the design philosophy of the sponge construction [BDPV07], which is proven

to be indistinguishable from a random oracle in the ideal permutation model up to the birthday bound in the capacity.

In fact, it was recently shown that this bound is not completely tight: Lefevre and Mennink [LM22] found that while the collision and second preimage security bounds are tight, the preimage bound is not. For the **Ascon** hashing functions including **Ascon-XOF128**, they showed that up to 192-bit preimage resistance can be achieved. More specifically, the claim could be updated from $\min(128, L)$ to $\min(L, \max(L - r, \frac{c}{2}))$ in the ideal permutation model where $L \leq c$, which corresponds to 192 bits for $c = L = 256$ as in **Ascon-Hash256**. This increase is due to the difficulty of matching multiple squeezed output blocks. However, second preimage resistance remains at $\min(128, L)$. Note that the matching attack was already described by the Keccak team in 2011 [BDPV11].

This discrepancy is interesting in the context of random-prefix (second) preimage resistance (**rpp** and **rpsp**). In fact, we can show that both correspond to the second preimage resistance of $\min(128, L)$ bits. For **rpsp**, the matching attack works as follows. The adversary picks a message M and receives the random prefix R . They can now compute the intermediate state S^* right before squeezing starts by absorbing R (obtaining S^R) and then M . Then, they can build a 5-block second preimage by constructing an inner collision as illustrated in Figure 4. Using all $2^{c/2} = 2^{2r} = 2^{128}$ candidates of $M_0 \parallel M_1$ and all of $M_3 \parallel M_4$, they are likely to find a match in the inner part of the state S^M where M_2 is absorbed, while M_2 is chosen to complete the match on the outer part. For an **rpp** attack of the same complexity, the adversary can directly choose S^* to determine their chosen target hash H and then proceed as above. These attacks are tight (up to small constants), since an **rpp** or **rpsp** adversary can also produce collisions.

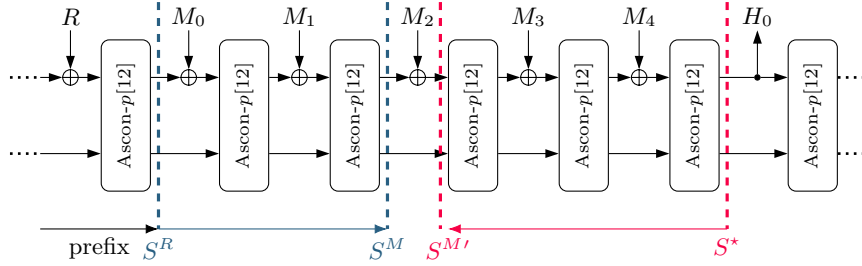


Fig. 4: Generic random-prefix (second) preimage (**rpp**, **rpsp**) attacks on **Ascon-XOF128** and other sponges.

Dedicated cryptanalysis of Ascon-XOF128. The analysis of **Ascon-XOF128** builds directly on the analysis of the closely related submitted candidate design **Ascon-XOF**; most results are directly transferable, except if they depend heavily on minor details such as the initial value and round constants, which are slightly

different. For an overview of dedicated preimage attacks relevant for Ascon-XOF128, we refer to Table 1 in the introduction.

2.3 Ascon-XOF128 and Ed25519

Ed25519 is currently specified for use with SHA-512 only. However, on constrained platforms and in applications where Ascon is already available, instantiating \mathcal{H} with Ascon-XOF128 instead as a more lightweight alternative has been brought up repeatedly as a discussion point in the NIST Lightweight Cryptography process, including the mailing list and workshops [MSP⁺23]. This is particularly relevant when the code size and memory requirements are constrained. It is worth noting that SHA-512, which can provide up to 256-bit security, is not chosen here due to its formal security level (Ed25519 aims for 128-bit security overall), but more due to the requirement of hashing the private key to 512-bit outputs to be used for different purposes, as well as the good software performance on 64-bit CPUs. While Ascon-XOF128 clearly satisfies the functional requirements to instantiate \mathcal{H} and brings in the necessary 128-bit security level, the concrete security implications with respect to the specific required security properties warrant further discussion.

One difficult aspect when discussing protocols and composite schemes involving multiple different primitives is how to concretely quantify their security level, specifically, which unit to use. Classically, the security level of symmetric primitives and schemes is quantified with respect to the offline and/or online query complexity of the primitive. For example, the k -bit key recovery security of a block cipher with k -bit key and n -bit block means that we expect that an adversary needs a time complexity of around $T = 2^k$ (offline) cipher evaluations to recover the key with a reasonable success probability ε (e.g., $T = 2^k$ for $\varepsilon = 1$, or $T = 2^{k-1}$ for $\varepsilon = \frac{1}{2}$, etc), and access to at least $D = 1$ (online) data produced by the target key. However, this is a primitive-dependent metric, and the same bit-level security for two different primitives does not necessarily correspond to equivalent effort for the adversary, as in the notable example of password hashing.

Moreover, for primitives in asymmetric cryptography, the best attacks are typically not generic attacks, but dedicated mathematical algorithms whose complexity is measured in other—more basic—operations, such as arithmetic operations. There are different approaches for handling this discrepancy and assigning bit-level security claims to asymmetric primitives, which relate arithmetic complexities either to the evaluation complexity of the asymmetric primitive or to a symmetric reference primitive such as AES, as in the case of NIST’s PQC security levels. A useful intermediate unit is bit operations. On the downside, any step from primitive-dependent metrics (e.g., cipher evaluations) towards more general metrics (e.g., bit operations, arithmetic operations) is typically also a step towards more implementation-specific and platform-specific metrics, which are often more volatile.

In case of Ed25519, the attack complexity based on solving the discrete logarithm problem on Curve25519 has been analyzed by Bernstein [Ber06]. He estimates the complexity to be about 2^{125} elliptic curve point additions, needing

around 2^{10} CPU cycles each. If we generously count each CPU cycle as a 64-bit operation, this totals to around 2^{141} bit operations. Other estimates are slightly higher, at around 2^{145} bit operations (for success probability very close to 1).

Comparing this to an **rpp/rps** attack on **Ascon-XOF128** as in Figure 4, the attack complexity is dominated by around 2^{128} permutation calls (absorbing M_1) and around 2^{128} inverse permutation calls (inversely absorbing M_3). One permutation call corresponds to 12 forward rounds of **Ascon-p**, each of which requires around $22 + 10 = 32$ word operations (XOR, AND, NOT; not counting rotations memory-moving operations, unlike the above estimate), totalling $12 \cdot 32 \cdot 64 \approx 2^{15}$ bit operations. Under the very conservative estimate that an inverse permutation call costs at least as much (while in reality, it is *much* more expensive), and not counting further overheads due to large memories or memoryless cycle-finding, we get an optimistic estimate of around 2^{144} bit operations overall, comparable to the cost to attack **Ed25519** via discrete logarithms. Thus, using **Ed25519** with **Ascon-XOF128** instead of **SHA-512** or **SHA3-512** is expected to provide the same overall security level, although **Ascon-XOF128** is more lightweight.

3 Forgeries via Second Preimages on Ascon-XOF

In this section, we outline our second-preimage attack on **Ascon-XOF128** which we use to create forgeries for **Ed25519+Ascon-XOF128**. In contrast to previous works which focused only on short outputs of 64 or 128 bits, our attack works for arbitrarily long outputs. Instead of targeting the arbitrarily long output in our attack, we instead target the internal **Ascon** state before the output is generated. This internal **Ascon** state is directly available to us, since we are performing a second-preimage attack. Alternatively, our attack can be viewed as a first preimage attack on the internal state of **Ascon**.

Our forgery attack proceeds in 3 steps as depicted in Figure 5. First, we show how to transform the problem of forging a signature into finding a preimage for an internal **Ascon** state (Section 3.1). Next, we show how to prepare the internal state of **Ascon** to enable the final part of the attack (Section 3.2). Finally, we show how to connect the prepared internal state to the target state (Section 3.3).

3.1 Forgeries based on second-preimage attacks

The more intuitive forgery attack based on a second preimage proceeds by obtaining one signature (R, S) for an arbitrary message M . Since the signature is verified by hashing $\mathcal{H}(R \parallel Q \parallel M) = h^*$, where Q is the public key, we can forge a message by finding an M' such that $\mathcal{H}(R \parallel Q \parallel M') = h^*$. Note that, strictly speaking, this scenario is not covered by a second-preimage attack, because we require a special prefix $R \parallel Q$. However, for **Ascon** and many other iterative hash functions, this prefix is equivalent to attacking the function with a modified initial value. Still, we require 1 known signature.

We can formulate a better attack, that requires no known signatures. As discussed in Section 2.1, we can create forgeries by breaking the commitment

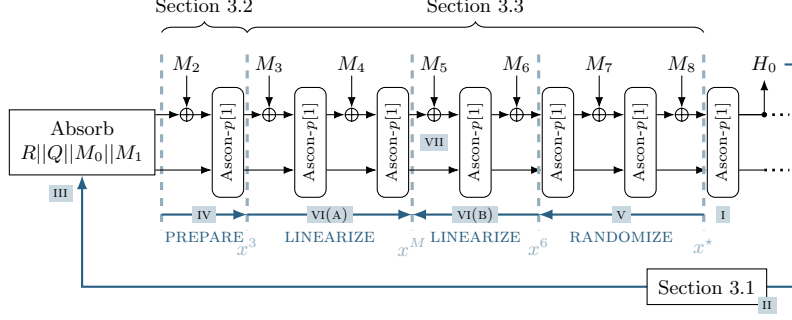


Fig. 5: Finding forgeries with second preimages on Ascon-XOF128.

scheme used for generating the signature. Concretely, we choose an arbitrary scalar value S and an arbitrary internal Ascon state x^* that leads to the output h^* . Concretely, we choose a suitable target hash h and an scalar value S . We find the first half of the signature as

$$R = S \cdot G - h^* \cdot Q,$$

where G denotes the elliptic curve generator and Q denotes the public key. Finally, we find a forgery by breaking the random-prefix preimage resistance of the hash function. That is, we find M such that

$$\mathcal{H}(R \parallel Q \parallel M) = h^*.$$

Note that we can apply our preimage attack that targets the internal state x^* and that we do not require any known signatures.

3.2 Preparing the internal state

As noted in previous works, linearizing the Ascon permutation works best if the diffusion of the message is limited in the first block. The two most effective conditions on the internal state are $x_1 = 0$ and $x_3 \oplus x_4 = 1^{64}$. As depicted in Figure 6, these two conditions affect the two AND gates that process the message and force the other input to be zero, hence preventing diffusion of the message through the AND gates. Note that after the S-box, the initial conditions lead to $y_4 = 1^{64}$; we will use this fact in Section 4. However, satisfying these two conditions simultaneously is not trivial as combined they correspond to 128 single-bit conditions.

Ensuring $x_1^3 = 0$. This step is relatively straight-forward. We start at state x^2 and want to ensure $x_1^3 = 0$. By examining the Ascon S-box (Figure 2a), we see that the message part (x_0) linearly affects y_1^2 after the S-box. Hence, we can temporarily set $M_2 = 0$ to find what y_1^2 would be: say $y_{1|M_2=0}^2 = \alpha$. Then we can set $M_2 = \alpha$, to ensure $y_1^2 = 0$ and $x_1^3 = \Sigma_1(y_1^2) = 0$.

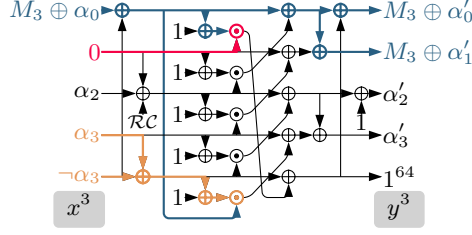


Fig. 6: Relevant initial conditions $x_1^3 = 0$ (■) and $x_3^3 = \neg x_4^3$ (■) limiting the diffusion of the message (■).

Ensuring $x_3^3 \oplus x_4^3 = 1^{64}$. To ensure all conditions are satisfied, we propose a brute-force approach. First, we pick two random messages M_0 and M_1 and process them. Next, we use M_2 to ensure $y_1^2 = 0$ as described above. If $x_3^3 \oplus x_4^3 = 1^{64}$ we are done; else, we try again. We expect to need about 2^{64} tries. In Section 4.2, we improve the time complexity for this step to ≈ 7.9 Gaussian eliminations.

3.3 Completing the second-preimage attack

To complete the attack, we need to gather enough degrees of freedom to generate a system of equations that has a solution with a high probability. Hence, we linearize the Ascon permutation both in the forward direction as well as in the backward direction. Due to the initial conditions, we can linearize two permutation calls/rounds in the forward direction as depicted in Figure 7. After the 2-round linearization in forward direction over the variables M_3 and M_4 , the words x_1^M , x_2^M , and x_3^M can be described as linear functions of M_3 and M_4 , while x_0^M and x_4^M can only be described as function of degree 2. For x_0^M , we do not really care, as this value is XORed with M_5 anyway. For x_4^M , we will exclude it from our equation system, and solve repeatedly until we find a solution that works for x_4^M . When linearizing the backward direction, we find that we can only reasonably linearize one permutation call as a function of M_6 . In this case, all expressions for x^M are linear as there is only one equivalent message bit per S-box. If we try to linearize more permutation calls, we find that we lose linear equations more quickly than we gain degrees of freedom by the additional message blocks.

We can formalize parts of the matching point x_1^M , x_2^M , and x_3^M as a linear function of M_3 and M_4 using the matrix A_f and in the backward direction as a function of M_6 using the matrix A_b :

$$\begin{aligned} x_1^M \parallel x_2^M \parallel x_3^M &= A_f \cdot (M_3 \parallel M_4) \oplus b_f, \\ x_1^M \parallel x_2^M \parallel x_3^M &= A_b \cdot M_6 \oplus b_b. \end{aligned}$$

Since the internal state needs to match, we write

$$(A_f \parallel A_b) \cdot (M_3 \parallel M_4 \parallel M_6) = b_f \oplus b_b,$$

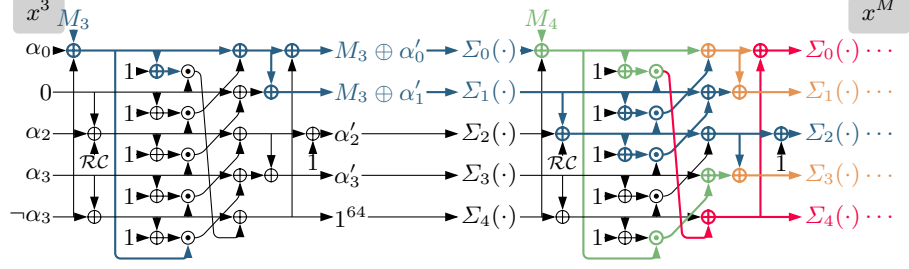


Fig. 7: Linearization over two 1-round permutation calls with initial conditions satisfied.

and solve for the message blocks. Note that we can deduce M_5 in post-processing, to make sure the value in x_0^M matches. Since the final matrix has dimensions 192×192 , we expect one solution on average. For each solution, we check whether x_4^M matches. Hence, we repeat this process about 2^{64} times. Further optimizations based on preprocessing the linear system are discussed in [Sch25].

Generating 2^{64} starting states x^3 would be too expensive for our attack. Instead, we add messages M_7 and M_8 (conforming to padding) before the target state x^* to generate random target states x^6 leading to new values for A_b and b_b .

Extension to preimage attack. To extend this attack to a first-preimage attack, we need to find the target state x^* based on a hash value / XOF output $h = (H_0 \parallel H_1 \parallel H_2 \parallel \dots)$. We can accelerate this process by determining the state x^{H_0} when H_0 is squeezed. We set $x^{H_0} = (H_0, \epsilon_1, \epsilon_2, \Sigma_0^{-1}(H_1) \oplus \epsilon_3, \epsilon_4)$, where ϵ_1 , ϵ_2 , and ϵ_4 are arbitrary constants and $\epsilon_3 = H_0 \cdot \epsilon_1 \oplus H_0 \oplus \epsilon_1 \cdot \epsilon_2 \oplus \epsilon_1 \cdot \epsilon_4 \oplus \epsilon_1 \oplus \epsilon_2$ is calculated in such a way that we deterministically match H_1 in the next output block. Any additional output blocks are matched probabilistically, leading to an overall runtime of $2^{64} + 2^{n-128}$ to find a preimage for an n -bit output. This is below the 128-bit security claim for messages up to 255 bits. We believe this can be improved by more careful analysis of linearization properties.

4 Forgeries via Random-Prefix Preimages on Ascon-XOF

In this section, we outline our attack on the Ed25519+Ascon-XOF128 signature scheme construction by attacking Ascon-XOF128 when used as a commitment scheme. That is, we choose an output for Ascon-XOF128 (commitment) and then find a preimage (opening) that fits the chosen output. Our attack proceeds in 3 steps as depicted in Figure 8. First, we transform the problem of forging a signature into finding a *random-prefix preimage* for Ascon-XOF128 (Section 4.1). Next, we prepare the internal state of Ascon-XOF128 (Section 4.2). We repeat this step to prepare many candidates for the final step. Finally, we connect the prepared internal state with the target hash by solving an equation system over 4 message blocks (Section 4.3). In Section 4.4, we experimentally verify our attack.

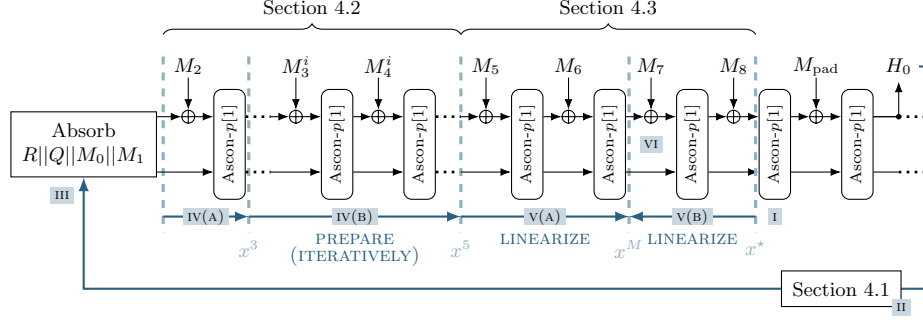


Fig. 8: Finding forgeries with random-prefix preimages on Ascon-XOF128.

4.1 Forgeries based on preimage attacks

For this attack, we make use of the same forgery attack from Section 3.1 where no known signatures are needed. However, instead of utilizing a second-preimage attack on Ascon-XOF128, we will now exploit the fact that we can freely choose the target hash h^* . Concretely, we will choose a suitable internal 320-bit Ascon state x^* , absorb the padding block $M_{\text{pad}} = 0x0000000000000001$ and generate the target hash h^* . We will see that by choosing a suitable x^* , we can amplify the probability that we find a solution by $2^{37.4}$ leading to a very practical attack.

4.2 Preparing the internal state

As discussed in Section 3.2, we require the conditions $x_1 = 0$ and $x_3 \oplus x_4 = 1^{64}$ on the internal state (x^5 , see Figure 8) to limit the diffusion of the message blocks. This leads to better linearization enabling the rest of the attack.

In Section 3.2, we only needed one state that fulfills the conditions, as we could generate many target states by prepending message blocks to the actual target state x^* . Compared to the second preimage attack, there is no incentive to randomize the target state x^* , as we specifically choose x^* to make the equation system easier to solve. Instead, we need to generate a new state that fulfills the conditions each time we want to try to solve the system. That is, the complexity contributes a multiplicative factor to the total complexity instead of an additive one. Note that we can apply this improved method also to the previous attack as well, although it does not affect the overall complexity.

Therefore, we propose a more efficient process: First, we use 1 message block to ensure $x_1 = 0$ like in Section 3.2. We expect a Hamming weight of 32 on average: $\text{HW}(x_3 \oplus x_4) \approx 32$. Next, we repeatedly use a linearization procedure over two message blocks to maintain $x_1 = 0$ while increasing $\text{HW}(x_3 \oplus x_4)$. We find that after 7.9 repetitions on average (see Section 4.4), all 128 single-bit conditions are satisfied. Next, we explain the main step of the iterative process.

Increasing the Hamming weight of $x_3 \oplus x_4$. At this point, we start with $x_1^3 = 0$, $x_2^3 = \alpha_2$, $x_3^3 = \alpha_3$, and $x_4^3 = \alpha_4$. For the first iteration, we expect

$\text{HW}(\alpha_3 \oplus \alpha_4) \approx 32$. We will now describe two permutation calls that process the messages M_3 and M_4 with a system of equations and solve the system to increase the Hamming weight. To prevent the diffusion of M_3 through the bottom AND gate (■ in Figure 6 on page 13) we set the relevant message bits to zero. While this reduces our overall degrees of freedom by $64 - \text{HW}(\alpha_3 \oplus \alpha_4)$, it ensures that M_3 only diffuses to the first two words of the Ascon state after the first permutation call. Then, after XORING the message M_4 , XORING the round constant \mathcal{RC} , after the S-box but before linear layer, we arrive at the state $(y_0^4, y_1^4, y_2^4, y_3^4, y_4^4)$ as depicted in Figure 9. The relevant part of the new state is calculated as follows:

$$\begin{aligned} y_1^4 &= M_3' \cdot \delta_{1a} \oplus M_4' \oplus \delta_{1b}, \\ y_3^4 &= M_3' \oplus M_4' \cdot \delta_{3a} \oplus \delta_{3b}, \\ y_4^4 &= M_3' \cdot (M_4' \oplus \delta_{4a}) \oplus \delta_{4b}, \end{aligned}$$

where $M_3' = \Sigma_1(M_3 \oplus \beta_1)$, $M_4' = M_4 \oplus \Sigma_0(M_3 \oplus \beta_0)$ and each lowercase Greek letter denotes some 64-bit constant and \cdot denotes bitwise AND. Note that y_4^4 is a nonlinear combination of M_3' and M_4' leading to a nonlinear system. However, since we require $\Sigma_1(y_1^4) = 0$ or equivalently $y_1^4 = 0$, we can replace $M_4' = M_3' \cdot \delta_{1a} \oplus \delta_{1b}$:

$$\begin{aligned} y_3^4 &= M_3' \cdot \delta'_{3a} \oplus \delta'_{3b}, \\ y_4^4 &= M_3' \cdot \delta'_{4a} \oplus \delta_{4b}, \end{aligned}$$

with $\delta'_{3a} = \delta_{1a} \cdot \delta_{3a} \oplus 1^{64}$, $\delta'_{3b} = \delta_{1b} \cdot \delta_{3a} \oplus \delta_{3b}$, and $\delta'_{4a} = \delta_{1a} \oplus \delta_{1b} \oplus \delta_{4a}$. Now, we want to solve $\Sigma_3(M_3' \cdot \delta'_{3a} \oplus \delta'_{3b}) \oplus \Sigma_4(M_3' \cdot \delta'_{4a} \oplus \delta_{4b}) = 1^{64}$. Note that the condition for $y_1^4 = 0$ is implicit in this system and will be satisfied when we calculate M_4' as a function of M_3' . However, we have already spent some degrees of freedom, so we have 64 equations and only $\text{HW}(\alpha_3 \oplus \alpha_4)$ variables. Hence, the system of equation most will likely not have a (full) solution. Finding M_3' such that the Hamming weight is maximized is an instance of decoding a linear code. In our case, we use a greedy algorithm to find a good, but not necessarily optimal, solution. Concretely, we create a new system of equations that is initially empty and then add each of the 64 equations that does not contradict this new system. This process can be thought of as a variant of Gaussian elimination where a row of the matrix is discarded should it lead to a contradiction. With this process, we can effectively increase the Hamming weight of $\alpha_3 \oplus \alpha_4$ while keeping $\alpha_1 = 0$.

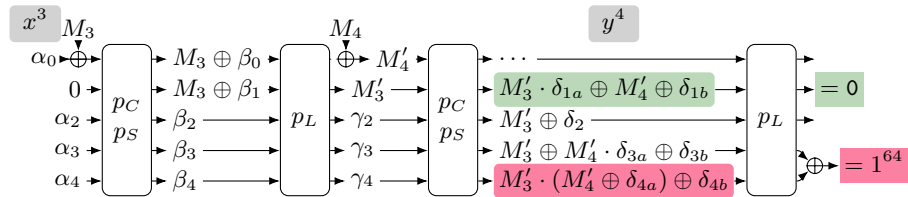


Fig. 9: Iteratively finding the initial conditions $x_1 = 0$ (■) and $x_3 = \neg x_4$ (■). Some bits of M_3 are used to compensate the missing initial conditions on $x_3 \oplus x_4$.

We find that by repeating it on average 7.9 times, we can fulfill all 128 conditions and enable the next step of the attack.

4.3 Completing the preimage attack

Now, we have an efficient routine to generate a state $x^5 = (\alpha_0, 0, \alpha_2, \alpha_3, \neg\alpha_3)$ using 7.9 Gaussian eliminations on average. However, even with all initial conditions, a linearization attack like in Section 3.3 would still need approximately 2^{64} repetitions to be successful. Hence, we analyze how we can improve the attack by controlling the target value x^* . We depict this part of the attack in Figure 10.

Choosing a Suitable Target Value. To choose the target value x^* , we first recall that $y_4^5 = 1^{64}$ due to the initial conditions (see Figure 6 on page 13). This implies that $x_4^6 = 1^{64}$ since $\Sigma_4(1^{64}) = 1^{64}$. Based on this fact, we can derive that the difference $y_2^6 \oplus y_3^6$ is biased (see Figure 10 on page 18):

$$y_2^6 \oplus y_3^6 = M'_6 \cdot \delta_{3a} \oplus 1^{64},$$

with δ_{3a} denoting a uniform 64-bit constant. So for those bits where the bit of δ_{3a} is zero, the difference is guaranteed to be 1, while for the other the difference is controlled by M'_6 . We can also explain this fact by examining the linear approximation table of the Ascon S-box which confirms that $x_{4,i} = y_{2,i} \oplus y_{3,i}$ with $p = 75\%$ and we know that $x_4^6 = 1^{64}$.

To exploit this property, we analyze the algebraic normal form of the inverse S-box when the rate bits are set to a given constant in Table 2. We find that for

Table 2: Algebraic normal form of component functions of the inverse S-box. The highlighted values ensure $x_2 \oplus x_3 = 1$ after adding the round constant \mathcal{RC} .

	x_0	x_1	x_2	x_3	x_4
$S^{-1}(y \parallel 0000)$	1	y	$y + 1$	0	0
$S^{-1}(y \parallel 0001)$	1	$y + 1$	y	1	0
$S^{-1}(y \parallel 0010)$	0	y	$y + 1$	1	1
$S^{-1}(y \parallel 0011)$	y	$y + 1$	$y + 1$	0	1
$S^{-1}(y \parallel 0100)$	0	0	0	y	y
$S^{-1}(y \parallel 0101)$	0	$y + 1$	y	0	1
$S^{-1}(y \parallel 0110)$	y	1	1	$y + 1$	0
$S^{-1}(y \parallel 0111)$	1	y	y	1	y
$S^{-1}(y \parallel 1000)$	y	$y + 1$	y	1	y
$S^{-1}(y \parallel 1001)$	y	y	$y + 1$	1	y
$S^{-1}(y \parallel 1010)$	$y + 1$	$y + 1$	1	0	$y + 1$
$S^{-1}(y \parallel 1011)$	0	y	0	0	$y + 1$
$S^{-1}(y \parallel 1100)$	$y + 1$	1	y	y	1
$S^{-1}(y \parallel 1101)$	$y + 1$	y	1	0	$y + 1$
$S^{-1}(y \parallel 1110)$	1	0	0	$y + 1$	$y + 1$
$S^{-1}(y \parallel 1111)$	$y + 1$	$y + 1$	$y + 1$	1	0

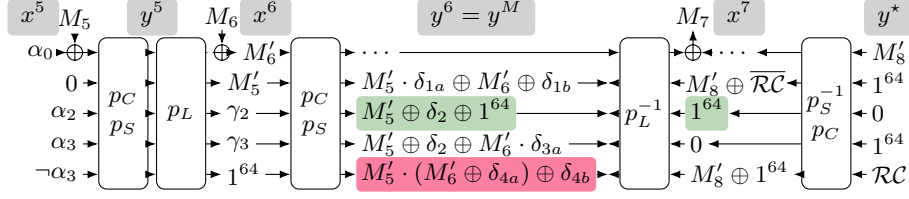


Fig. 10: Main part of the attack. The left side, satisfying the initial conditions x^5 , is connected to the fixed target state $y^* = p_L^{-1}(x^*)$. We express y_6 from the left and from the right and match the equations. By inserting the green equation (■) into the red equation (■), we get a linear system.

$\mathcal{S}^{-1}(y \parallel 1011)$, we get $x_2 = 1$ and $x_3 = 0$. Similarly, for $\mathcal{S}^{-1}(y \parallel 1101)$, we get $x_2 = 0$ and $x_3 = 0$. By combining these two patterns ($\mathcal{S}^{-1}(y \parallel 1011)$ for most bits and $\mathcal{S}^{-1}(y \parallel 1101)$ for those bits where the round constant \mathcal{RC} flips a bit of x_2), we can ensure that $x_3^7 = 1^{64}$ and $x_4^7 = 0$. Since these two constants are fixed points of the linear layer Σ_2/Σ_3 , we get the same constants before the linear layer at the state y^6 : $y_3^6 = 1^{64}$ and $y_2^6 = 0$. Consequently, we get $64 - \text{HW}(\delta_{3a}) \approx 32$ equations for free, greatly improving the probability that we get a solution.

With the target state selected, we can formulate the equation system:

$$\begin{aligned}\Sigma_1(M'_5 \cdot \delta_{1a} \oplus M'_6 \oplus \delta_{1b}) &= M'_8 \oplus \overline{\mathcal{RC}}, \\ \Sigma_2(M'_5 \oplus \delta_2 \oplus 1^{64}) &= 1^{64}, \\ \Sigma_3(M'_5 \oplus \delta_2 \oplus M'_6 \cdot \delta_{3a}) &= 0, \\ \Sigma_4(M'_5 \cdot (M'_6 \oplus \delta_{4a}) \oplus \delta_{4b}) &= M'_8 \oplus 1^{64},\end{aligned}$$

where each δ is a 64-bit constant and $\overline{\mathcal{RC}} = \mathcal{RC} \oplus 1^{64}$.

Simplifying the equation system by changing variables. At this point, the last word of the equation system is nonlinear, however, from the second equation we see that $M'_5 = \delta_2$. By simplifying, we get linear equations:

$$\begin{aligned}\Sigma_1(M'_6) \oplus M'_8 &= \overline{\mathcal{RC}} \oplus \Sigma_1(\delta_2 \cdot \delta_{1a} \oplus \delta_{1b}), \\ M'_6 \cdot \delta_{3a} &= 0, \\ \Sigma_4(M'_6 \cdot \delta_2) \oplus M'_8 &= 1^{64} \oplus \Sigma_4(\delta_{4a} \cdot \delta_2 \oplus \delta_{4b}).\end{aligned}$$

Here, we can already see the improvement by choosing a suitable target state y^* , as the equation $M'_6 \cdot \delta_{3a} = 0$ is a tautology for $64 - \text{HW}(\delta_{3a})$ bits. By substituting $M'_8 = \Sigma_1(M'_6) \oplus \overline{\mathcal{RC}} \oplus \Sigma_1(\delta_2 \cdot \delta_{1a} \oplus \delta_{1b})$, we can further simplify:

$$\begin{aligned}M'_6 \cdot \delta_{3a} &= 0, \\ \Sigma_4(M'_6 \cdot \delta_2) \oplus \Sigma_1(M'_6) &= \Sigma_4(\delta_{4a} \cdot \delta_2 \oplus \delta_{4b}) \oplus \Sigma_1(\delta_2 \cdot \delta_{1a} \oplus \delta_{1b}) \oplus \mathcal{RC}.\end{aligned}$$

Now, we have a linear equation system with 64 variables and $64 + \text{HW}(\delta_{3a})$ equations. After we solve the equation system with Gaussian elimination, we can

deduce M'_8 from M'_6 . Then, we find $M_5 = \Sigma_1^{-1}(M'_5) \oplus \alpha_0$, M_6 by seeing what value we need to XOR to get $x_0^6 = M'_6$, M_7 by seeing what value we need to XOR to get $x_0^7 = M'_8 \cdot \overline{\mathcal{RC}} \oplus \mathcal{RC}$, and $M_8 = \Sigma_0(M'_8)$.

Probability of finding a solution. The overall complexity of the attack depends on the probability that the constructed system of equations has at least one solution. The final linear system of equations has 64 variables and $X_{\#eq} \sim 64 + B(n = 64, p = 0.5)$ equations, where B denotes a binomial distribution for n trials with probability p . So, for a single equation system, the logarithm of the probability to find at least one solution is distributed according to a binomial distribution: $p_{sol} \sim 2^{-B(n=64, p=0.5)}$. To predict the overall complexity, we calculate the expected value of p_{sol} :

$$\mathbb{E}(p_{sol}) = \sum_{k=0}^{64} 2^{-k} \cdot \binom{64}{k} \cdot 0.5^k \cdot 0.5^{64-k} \approx 2^{-26.56},$$

which we later also verify experimentally. This is an improvement by a factor of $2^{37.4}$ compared to the previous 2^{-64} .

Overall complexity. We need 7.9 Gaussian eliminations on average to find a starting point and 1 Gaussian elimination to see whether we find a solution for this starting point. As we need to repeat this process on average $2^{26.56}$ times the expected complexity of the attack is $(7.9 + 1) \cdot 2^{26.56} \approx 2^{29.7}$.

4.4 Experimental evaluation

The code for this experimental evaluation is available online¹.

We implement the routine of Section 4.2 to find a state that matches the initial conditions. By trying 5000 different random states, we find that we need on average 7.9 Gaussian eliminations corresponding to 16.8 message blocks to find a state that fulfills all initial conditions. We show the distribution of the number of Gaussian eliminations needed in Figure 11.

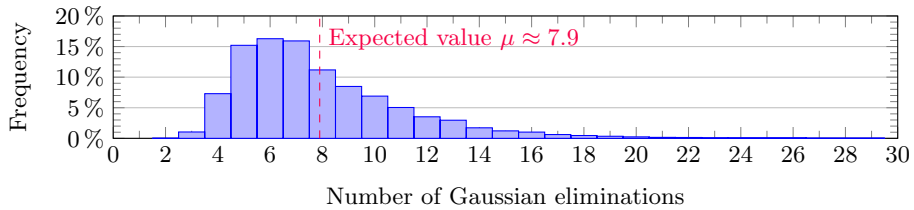


Fig. 11: Experimental complexity of finding initial state ($n = 5000$)

¹ <http://github.com/isec-tugraz/ascon-rpp-preimages>.

We implemented the random-prefix-preimage attack of Section 4 in C++ and performed it for the Ascon-Hash256 IV. Out of $2^{34.5}$ random prefixes $M_0 \parallel M_1$, we find that 294 of them lead to a successful preimage attack, corresponding to a rate of one solution every $2^{26.47}$ tries, which closely matches our statistical estimate. On our server with 2 AMD EPYC 9754 CPUs we find one random-prefix-preimage about every 27 seconds (3.2 core-hours). On the same hardware this corresponds to roughly $2^{35.3}$ brute-force attempts. Note that with a more optimized implementation of Gaussian elimination this number would be lower.

5 Conclusion

In this paper, we have shown second-preimage and random-prefix-preimage attacks on 1-round Ascon-XOF128. These are the first preimage attacks on round-reduced Ascon-XOF128 with 256-bit output and Ascon-Hash256 with complexity below the claimed 128-bit security. For second-preimage attacks and random-prefix-preimage attacks, the security claim is also matched by a generic attack. Both our attacks apply in settings where Ascon-XOF128 or Ascon-Hash256 are used as a commitment scheme as they allow us to break the binding property. This property is required by the Fiat-Shamir transformation [FS86], which is commonly used to transform interactive zero-knowledge proofs into non-interactive zero-knowledge proves or signature schemes like in Ed25519.

We propose a second-preimage attack on 1-round Ascon Hash with complexity 2^{64} , which we also extend to a first-preimage attack on Ascon-XOF with outputs up to 255 bit for a complexity below 2^{128} . We find that by specifically exploiting the fact that we can freely choose the hash output, our random-prefix-preimage attack is faster by a factor of $2^{34.3}$ compared to our second-preimage attack.

We believe our work can be built upon in future work. In particular, being able to choose the target state might allow improved cryptanalysis on other sponge-based hash functions as well using our techniques. Furthermore, increasing the number of attacked round is of high interest. However, since the core of our attack necessarily spans 3 permutation calls to get enough degrees of freedom, increasing the number of attacked rounds implies analyzing 6 rounds of Ascon over all capacity bits, which may be challenging given the 128-bit security level.

Acknowledgments. We would like to thank Fredrik Meisingseth for insightful discussions on probability theory. This research was funded in part by the European Research Council (ERC) Starting Grant KEYLESS (#101165216).

References

- BDPV07. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. ECRYPT Hash Workshop 2007, 2007. URL: <https://keccak.team/files/SpongeFunctions.pdf>.
- BDPV11. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. URL: <https://keccak.team/files/CSF-0.1.pdf>.

Ber06. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006. doi:10.1007/11745853_14.

BKK24. Seungjun Baek, Giyoon Kim, and Jongsung Kim. Preimage attacks on reduced-round Ascon-XOF. *Des. Codes Cryptogr.*, 92(8):2197–2217, 2024. doi:10.1007/s10623-024-01383-0.

BV96. Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *CRYPTO ’96*, volume 1109 of *LNCS*, pages 129–142. Springer, 1996. doi:10.1007/3-540-68697-5_11.

DEMS14. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  fer. Ascon v1. Submission to the CAESAR competition, 2014. URL: <https://competitions.cr.yp.to/round1/asconv1.pdf>.

DEMS19a. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  fer. Ascon v1.2. Submission to the NIST Lightweight Cryptography competition, 2019. URL: <https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ascon-spec.pdf>.

DEMS19b. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  fer. Preliminary analysis of Ascon-XOF and Ascon-Hash. IACR Cryptology ePrint Archive, Paper 2024/908, 2019. URL: <https://eprint.iacr.org/2024/908>.

DEMS21. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  fer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9.

DEMS24. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  fer. Ascon MAC, PRF, and short-input PRF – lightweight, fast, and efficient pseudorandom functions. In *CT-RSA 2024*, volume 14643 of *LNCS*, pages 381–403. Springer, 2024. doi:10.1007/978-3-031-58868-6_15.

DGL⁺24. Xiaoyang Dong, Jian Guo, Shun Li, Phuong Pham, and Tianyu Zhang. Improved meet-in-the-middle Nostradamus attacks on AES-like hashing. *IACR Transactions on Symmetric Cryptology*, 2024(1):158–187, 2024. doi:10.46586/tosc.v2024.i1.158-187.

DZQ⁺24. Xiaoyang Dong, Boxin Zhao, Lingyue Qin, Qingliang Hou, Shun Zhang, and Xiaoyun Wang. Generic mitm attack frameworks on sponge constructions. In *CRYPTO 2024*, volume 14923 of *LNCS*, pages 3–37. Springer, 2024. doi:10.1007/978-3-031-68385-5_1.

FLYS23. Qinggan Fu, Ye Luo, Qianqian Yang, and Ling Song. Preimage and collision attacks on reduced Ascon using algebraic strategies. IACR Cryptology ePrint Archive, Paper 2023/1453, 2023. URL: <https://eprint.iacr.org/2023/1453>.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO ’86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986. doi:10.1007/3-540-47721-7_12.

GLS16. Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 249–274, 2016. doi:10.1007/978-3-662-53887-6_9.

HM96. Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO ’96*, volume 1109 of *LNCS*, pages 201–215. Springer, 1996. doi:10.1007/3-540-68697-5_16.

- JL17. Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, 2017. doi:10.17487/rfc8032.
- KK06. John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 183–200. Springer, 2006. doi:10.1007/11761679_12.
- LHC⁺23. Huina Li, Le He, Shiyao Chen, Jian Guo, and Weidong Qiu. Automatic preimage attack framework on Ascon using a linearize-and-guess approach. *IACR Transactions on Symmetric Cryptology*, 2023(3):74–100, 2023. doi:10.46586/tosc.v2023.i3.74–100.
- LM22. Charlotte Lefevre and Bart Mennink. Tight preimage resistance of the sponge construction. In *CRYPTO 2022*, volume 13510 of *LNCS*, pages 185–204. Springer, 2022. doi:10.1007/978-3-031-15985-5_7.
- MSP⁺23. John Preuß Mattsson, Göran Selander, Santeri Paavolainen, Ferhat Karakoç, Marco Tiloca, and Robert Moskowitz. Proposals for standardization of the Ascon family. NIST Lightweight Cryptography Workshop 2023, 2023.
- Nat23. National Institute of Standards and Technology. Digital signature standard (DSS). Technical Report Federal Information Processing Standards Publications (FIPS) 186-5, 2023. doi:10.6028/nist.fips.186-5.
- Nat24. National Institute of Standards and Technology. Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions. Technical Report NIST Special Publication (SP) 800-232 (Initial Public Draft), 2024. doi:10.6028/nist.sp.800-232.ipd.
- NHS⁺24. Zhongfeng Niu, Kai Hu, Siwei Sun, Zhiyu Zhang, and Meiqin Wang. Speeding up preimage and key-recovery attacks with highly biased differential-linear approximations. In *CRYPTO 2024*, volume 14923 of *LNCS*, pages 73–104. Springer, 2024. doi:10.1007/978-3-031-68385-5_3.
- NSW09. Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *J. Math. Cryptol.*, 3(1):69–87, 2009. doi:10.1515/jmc.2009.004.
- QHD⁺23. Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-middle preimage attacks on sponge-based hashing. In *EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 158–188. Springer, 2023. doi:10.1007/978-3-031-30634-1_6.
- Sch89. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO ’89*, volume 435 of *LNCS*, pages 239–252. Springer, 1989. doi:10.1007/0-387-34805-0_22.
- Sch25. Lorenz Schmid. Preimages for Ascon-Xof in EdDSA. Master’s thesis, Graz University of Technology, 2025. doi:10.3217/6jv2t-pgn22.
- ZSWH23. Zhiyu Zhang, Siwei Sun, Caibing Wang, and Lei Hu. Classical and quantum meet-in-the-middle Nostradamus attacks on AES-like hashing. *IACR Transactions on Symmetric Cryptology*, 2023(2):224–252, 2023. doi:10.46586/tosc.v2023.i2.224–252.

A Detailed Experimental Results

Table 3: Exemplary random-prefix-preimage for 1-round Ascon-Hash256, breaking the binding property of the commitment scheme. Each row states a message block and the internal state after XORING the message and applying the Ascon permutation.

	M_i	x_0	x_1	x_2	x_3	x_4	Comment
M_0	685903260457ea53	9b1e5494e934d681	4bc3a01e333751d2	ae65396c6b34b81a	3c7fd4a4d56a4db3	1a5c464906c5976d	Ascon-Hash256 IV
M_1	af1cb14c5d612ec1	7970a54f7a956205	e9a17ea5020eaf64	d414b335e550a038	ce2b77bea3c33fae	6c4b2bc95f2b8612	randomize
M_2	18caa93c20b674b7	916deea6ca0c72b3	0a6160777409cca1	70870fd1dfdfe47	085e826fa9975606	f358a8241564bdab	randomize
M_3^0	02034e8424004000	387d44f426400f5d	0000000000000000	20cd0294856ce9d9	e661109dca462fe5	a5ae5f097f76c9e7	$x_1 = 0$
M_4^0	01f669c9a1d2ac91	7b5fd54cc3442713	79b3c813ba6959a9	8e981997454cb201	eb022c6a4ca098c2	8d1249789d7b6069	
M_3^1	00002c0313ccfde0	f8a9dabce0271af2	0000000000000000	5e954963f707cbff	93940c511e30a8b1	b0a4a03ea5cf574c	$\text{HW}(x_3 \oplus x_4) = 44$
M_4^1	00002c0313ccfde0	08e2144b56261f9d	ce0ae2d036a6b480	5fea35b137080f08	ade02e27b0350c3a	eeab32c89a199854	
M_3^2	15a3f9744b803c32	00e92efb764370dc	0000000000000000	daae749cd5a45915	c38ee7010cc0932f	e458bcddef33f6cd0	$\text{HW}(x_3 \oplus x_4) = 53$
M_4^2	00844284fede47b6	39714d7a7ada4901	ba0e93d81e0db0a2	45855c2dc2b33c60	867ea1897720a565	36660897bf93eb2d	
M_3^3	f0ecba1838f63fe3	ae7c216a92878d36	0000000000000000	d08ff4f28c946d24	67e6360c2c81f91f	ea19c9f3d37e06e0	$\text{HW}(x_3 \oplus x_4) = 60$
M_4^3	84651c261793f7ac	f0b342b5edf85168	0b62ef4c276c5728	3640848a0a9f0819	1227e8b7b5b1f1c1	8d1bfffffc6ffff	
M_3^4	40cfd2b5adc1516c	3ad693c7c7a2c56b	0000000000000000	051854c4920ed35c	fbdc2a1e3a528196	4423d5e1c5ad7e69	$\text{HW}(x_3 \oplus x_4) = 63$
M_4^4	12ea247b8325fbbc	76c4d6d49a7b27a3	0bad11bcb3f78c5c	065d504c96d30a8b	16bd4fa276b7e65b	bf7fffffffdfdf	
M_3^5	d44b76d580c036c3	e52f8615a999649f	0000000000000000	eb86bd4233149d27	b7aa2d4e2d66f7f0	4c55d2b1d299080f	$\text{HW}(x_3 \oplus x_4) = 63$
M_4^5	701b354e4cb8b2d3	0bb9f7786ebb76c5	58a0a5c730f2b3ff	034a6a4a21329678	01f5d6d1c2c6be3e	fbf7fffffffdfdf	
M_3^6	abb1f71862b2a4f7	e12feb45172b0c5b	0000000000000000	b864c997c9c9cc64	d5174e2ae7415ebf	aae8b1d518bea140	$\text{HW}(x_3 \oplus x_4) = 63$
M_4^6	68cf246ff07f4737	e2614bcbe9faa9bb	ada27d29b6866f6f	1a7f8a7d4d76f992	aa62f368370fa547	7efffffffbffff	
M_3^7	2b434b61edbc88fa	7c76c3ee03ffe1f7	0000000000000000	56927a3259d256c6	4f1fd664b03eb777	70e0299b4fc14888	$\text{HW}(x_3 \oplus x_4) = 62$
M_4^7	1a442a5a23dc2146	07643ea2da87265e	b008f46aaa6d1b88	3f2d4cccc6bd2280	a15eca1a17a2f10e	3e7fffffff9ffff	
M_3^8	a66c7fabd8c72266	40f931a2dc9d06e2	0000000000000000	ec3d2628b747674a	455dd67eb2e95fa1	baa229814d16a05e	$\text{HW}(x_3 \oplus x_4) = 64$
M_4^8							
M_5	fb9a0e8fd133565	512431250c059e61	e0b3d4c3f1117122	7f750bbc5f6f9c12	6039208051811284	ffffffffffff	
M_6	47a0bf2fae0db670	45d9c05dee430635	d6ffe8ad49ad9387	ffffffffffff	0000000000000000	d6ffe8ad49ad93cc	matching point x^M
M_7	932628f0a7ee95b1	01a0b14024b94fd2	ffffffffffff	0000000000000000	ffffffffffff	960000002580004b	$x_4 = \Sigma_4(RC)$
M_8	01a0b14024b94fd2	ce097679458004f9	d9b4ffffef753fffb	ac0000000000006f	9625cb00258972c0	0000000000000000	cancel x_0
M_{pad}	0000000000000001	19970aba0d463889	333c724c2fe75a29	4d47ac0188d8b030	a6e0330b337df6e8	43d58d477b36f471	state after padding