The design and implementation of XForró14-Poly1305: a new Authenticated Encryption Scheme

Murilo Coutinho [1,2], **Iago Passos** , **Fábio Borges** [1]

[1] Laboratório Nacional de Computação Científica (LNCC)
Petrópolis, Brasil

[2]Nubank, São Paulo, Brasil

Abstract. *At Asiacrypt 2022 and its extended version at Journal of Cryptology 2023, Coutinho et al. proposed Forró, a novel ARX-based stream cipher with a design reminiscent of Salsa and ChaCha ciphers. The authors demonstrated that Forró provides a higher security margin using fewer operations, thereby reducing the total number of rounds while preserving the security level. This results in a faster cipher across various platforms, particularly on constrained devices. However, Forró's primary limitation is its exclusive encryption capability, with no authentication support. To address this issue, in this paper, we introduce the XForró14 cipher and combine it with Poly1305 to create an Authenticated Encryption with Associated Data (AEAD) scheme. Furthermore, to facilitate the practical implementation of this cipher, we have developed a new fork of the libsodium project (`https://doc.libsodium.org/`), incorporating XForró14-Poly1305 as a fresh AEAD alternative. Our project can be accessed at `https://github.com/murcoutinho/libsodium`.*

Resumo. *No Asiacrypt 2022 e em sua versão estendida no Journal of Cryptology 2023, Coutinho et al. propõe o algoritmo Forró, uma nova cifra de fluxo baseada em ARX com um design similar às cifras Salsa e ChaCha. Os autores demonstraram que o Forró oferece uma margem de segurança maior usando menos operações, reduzindo assim o número total de rodadas enquanto preserva o nível de segurança. Isso resulta em uma cifra mais rápida em várias plataformas, particularmente em dispositivos restritos. No entanto, a principal limitação do Forró é sua capacidade exclusiva de encriptação, sem suporte para autenticação. Para resolver esse problema, neste artigo apresentamos a cifra XForró14 e a combinamos com a Poly1305 para criar um esquema de Encriptação Autenticada com Dados Associados (AEAD). Além disso, para facilitar a implementação prática desta cifra, desenvolvemos um novo fork do projeto libsodium (`https://doc.libsodium.org/`), incorporando XForró14-Poly1305 como uma nova alternativa de AEAD. Nosso projeto pode ser acessado em `https://github.com/murcoutinho/libsodium`.*

1. Introduction

In recent years, the demand for secure and efficient cryptographic schemes has increased significantly, driven by the rapid growth of communication and data storage technologies. Stream ciphers, in particular, have become indispensable for providing confiden-

tiality in various applications such as secure communications, digital signatures, and secure storage. Among these, the Salsa and ChaCha ciphers, designed by D.J. Bernstein [Bernstein 2008a, Bernstein 2008b], have gained widespread popularity due to their security and performance characteristics. Salsa and ChaCha are examples of ARX ciphers. An ARX cipher refers to cryptographic algorithms that utilize three primary operations: Addition, Rotation, and XOR. These operations are chosen for their simplicity, efficiency on various platforms, and scalability to different word sizes.

Salsa and ChaCha are heavily used. For example, ChaCha is one of the cipher suites of the new TLS 1.3 [Langley et al. 2016], which has been used by Google on both Chrome and Android. Not only has ChaCha been used in TLS but also in many other protocols such as SSH, Noise, and S/MIME 4.0. In addition, the RFC 7634 [Nir 2015] proposes the use of ChaCha in IKE and IPsec. ChaCha has been used not only for encryption, but also as a pseudo-random number generator in any operating system running Linux kernel 4.8 or newer. Additionally, ChaCha has been used in several applications such as WireGuard (VPN) (`https://www.wireguard.com`) (see [IANIX 2020] for a huge list of applications, protocols, and libraries using ChaCha).

Nevertheless, there is an ongoing quest for more efficient ciphers that can provide enhanced security, especially in resource-constrained environments [McKay et al. 2016]. At Asiacrypt 2022 and Journal of Cryptology 2023, Coutinho et al. [Coutinho et al. 2022, Coutinho et al. 2023] introduced Forró, a new ARX-based stream cipher designed to offer higher security margins using fewer operations. The cipher's design is similar to Salsa and ChaCha, but it reduces the total number of rounds while maintaining the security level, resulting in a faster cipher across various platforms, with a particular focus on constrained devices. In particular, recent cryptanalyses [Beierle et al. 2020, Coutinho and Neto 2021, Dey et al. 2022] show attacks up to 8 rounds of Salsa (time and data complexities of $2^{217.14}$ and $2^{113.14}$), 7 rounds of ChaCha (time and data complexities of $2^{221.95}$ and $2^{48.83}$), but only 5 rounds of Forró (time and data complexities of 2^{158} and 2^{57}) with the same number of operations. This means that 6 rounds of Forró is safer than 7 or 8 rounds of ChaCha and Salsa.

Despite its superior performance compared to Salsa and ChaCha [Coutinho et al. 2023], Forró has a significant limitation: it only supports encryption and lacks authentication capabilities. Authenticated Encryption with Associated Data (AEAD) schemes are essential components in modern cryptography, as they ensure both confidentiality and authenticity of data [Bellare and Namprempre 2000, Jimale et al. 2022]. The importance of authentication cannot be overstated, as it verifies the integrity of encrypted data and confirms the legitimacy of the communicating parties, preventing unauthorized access, tampering, or forgery. In the context of secure communications, this dual functionality significantly enhances the robustness of cryptographic protocols and helps thwart various types of attacks.

Given the need for an AEAD scheme that harnesses the efficiency and security advantages of the Forró cipher, this paper presents the XForró14 cipher, a novel extension to the original design. By combining XForró14 with the well-established Poly1305 authentication mechanism, we create a comprehensive AEAD scheme that offers a high level of security and performance. This enhanced versatility makes XForró14-Poly1305

Notation	Description
X	a 4×4 state matrix.
$X^{(m)}$	state matrix after application of m rounds.
$X^{[s]}$	state matrix after application of s subrounds.
Z	output of Forró, i.e., $Z = X + X^{(R)}$.
$x_i^{(m)}$	i^{th} word of the state matrix $X^{(m)}$.
$x_{i,j}^{(m)}$	j^{th} bit of i^{th} word of the state matrix $X^{(m)}$.
k_i	a 32-bit word representing a subkey.
c_i	a 32-bit word representing a constant.
v_i	a 32-bit word representing a part of the nonce.
t_i	a 32-bit word representing a part of the counter.
$x + y$	addition of x and y modulo 2^{32}.
$x \oplus y$	bitwise XOR of x and y.
$x \lll n$	rotation of x by n bits to the left.
pad(.)	a padding function.
len(.)	a length function.

Table 1. Notations.

an attractive option for developers and users seeking a secure and efficient cryptographic solution.

To facilitate the practical implementation of the XForró14-Poly1305 AEAD scheme, we have developed a new fork of the popular libsodium cryptographic library [Denis] (`https://doc.libsodium.org/`). This fork incorporates the XForró14-Poly1305 AEAD option, providing developers with an easy-to-use and efficient cryptographic tool. The source code for our project is available at `https://github.com/murcoutinho/libsodium`, encouraging community involvement and further development.

This paper is organized as follows: Section 2 provides a brief overview of the Forró cipher and its design principles, and also provides a description of the Poly1305 authenticator. Section 3 introduces the XForró14 cipher including HForró, a key derivation function using in the design of XForró14. Section 4 gives a mathematical proof for the security of XForró14. In Section 5, we propose the new AEAD scheme XForró14-Poly1305 by combining XForró14 with Poly1305. Section 6 presents a implementation of XForró14-Poly1305 as a fork of libsodium, including benchmarks to compare performance of XForró14-Poly1305 against XChaCha20-Poly1305 (`https://datatracker.ietf.org/doc/draft-irtf-cfrg-xchacha/`). Finally, 7 presents the conclusion and discusses future research directions.

2. Previous work

In this section, we revise previous work, in particular the stream cipher Forró and the authenticator Poly1305. To improve readability, Table 1 defines some of the notation of this work.

2.1. The Forró stream cipher

Although they have a very similar structure, the literature suggests that ChaCha is safer than Salsa, due to the fact that both have the same number of rounds but attacks reach more rounds against Salsa [Aumasson et al. 2008, Choudhuri and Maitra 2016]. At recent work [Coutinho et al. 2022], Coutinho et al. proposed a new cipher called Forró that achieves higher security when compared to Salsa and ChaCha, being faster in constrained devices and suited to IoT applications [Costa et al. 2022, Coelho et al. 2022].

Forró consists of a series of ARX (addition, rotation, and XOR) operations on 32-bit words, and is highly efficient in software and hardware. Each round of Forró has a total of 12 bitwise XOR, 24 additions modulo 2^{32}, and 12 constant-distance rotations. Forró operates on a state of 64 bytes, organized as a 4×4 matrix with 32-bit integers, initialized with a 256-bit key $k_0, k_1, \ldots, k_7$, a 64-bit nonce v_0, v_1 and a 64-bit counter t_0, t_1, and 4 constants $c_0 = \text{0x746C6F76}, c_1 = \text{0x61616461}, c_2 = \text{0x72626173}, c_3 = \text{0x61636E61}$. These constants correspond to the ASCII string "*voltadaasabranca*", little-endian encoded. For Forró, we have the following initial state matrix:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} k_0 & k_1 & k_2 & k_3 \\ t_0 & t_1 & c_0 & c_1 \\ k_4 & k_5 & k_6 & k_7 \\ v_0 & v_1 & c_2 & c_3 \end{pmatrix}. \tag{1}$$

The state matrix is modified in each round by a *Subround* function (SR), denoted by $X^{[m]} = SR_{forro}(a,\ b, c, d,\ e, X^{[m-1]})$, which receives and updates 5 integers (see Figure 1 for a visual description of the circuit). This is one parameter more than Salsa and ChaCha. The reason is that Forró uses a technique called Pollination to bring non-linearity, diffusion, and confusion faster than other ciphers.

Forró subround function $SR_{forro}(a, b, c, d, e)$

$$\begin{aligned}
&x_d'^{(m-1)} = x_d^{(m-1)} + x_e^{(m-1)};\ \ x_c'^{(m-1)} = x_c^{(m-1)} \oplus x_d'^{(m-1)};\\
&x_b'^{(m-1)} = \left(x_b^{(m-1)} + x_c'^{(m-1)}\right) \lll 10;\\
&x_a'^{(m-1)} = x_a^{(m-1)} + x_b'^{(m-1)};\ \ x_e^{(m)} = x_e^{(m-1)} \oplus x_a'^{(m-1)};\\
&x_d^{(m)} = \left(x_d'^{(m-1)} + x_e^{(m)}\right) \lll 27;\\
&x_c^{(m)} = x_c'^{(m-1)} + x_d^{(m)};\ \ x_b^{(m)} = x_b'^{(m-1)} \oplus x_c^{(m)};\\
&x_a^{(m)} = \left(x_a'^{(m-1)} + x_b^{(m)}\right) \lll 8;
\end{aligned} \tag{2}$$

In Forró, the last element updated in a subround application (the element $x_a^{(r)}$) is introduced as pollen element ($x_e^{(r)}$) in the subsequent subround application, being a source of confusion and diffusion.

Each round of Forró can be described in terms of its subrounds. More precisely, each round has 4 subrounds, thus, we have $X^{(r)} = X^{[4r]}$. Therefore, in an odd round, when $r \in \{1, 3, 5, 7, \ldots\}$, $X^{(r)}$ is defined from $X^{(r-1)}$ in the following manner

$$\begin{aligned}
X^{[4r-3]} &= SR(0, 4, 8, 12, 3, X^{[4r-4]}); & X^{[4r-2]} &= SR(1, 5, 9, 13, 0, X^{[4r-3]});\\
X^{[4r-1]} &= SR(2, 6, 10, 14, 1, X^{[4r-2]}); & X^{[4r]} &= SR(3, 7, 11, 15, 2, X^{[4r-1]});
\end{aligned}$$

and for even rounds $r \in \{2, 4, 6, 8, \ldots\}$ from

$$X^{[4r-3]} = SR(0, 5, 10, 15, 3, X^{[4r-4]}); \qquad X^{[4r-2]} = SR(1, 6, 11, 12, 0, X^{[4r-3]});$$
$$X^{[4r-1]} = SR(2, 7, 8, 13, 1, X^{[4r-2]}); \qquad X^{[4r]} = SR(3, 4, 9, 14, 2, X^{[4r-1]});$$

The output of Forró14 is then defined as the sum of the initial state with the state after 14 rounds $Z = X^{(0)} + X^{(14)}$.

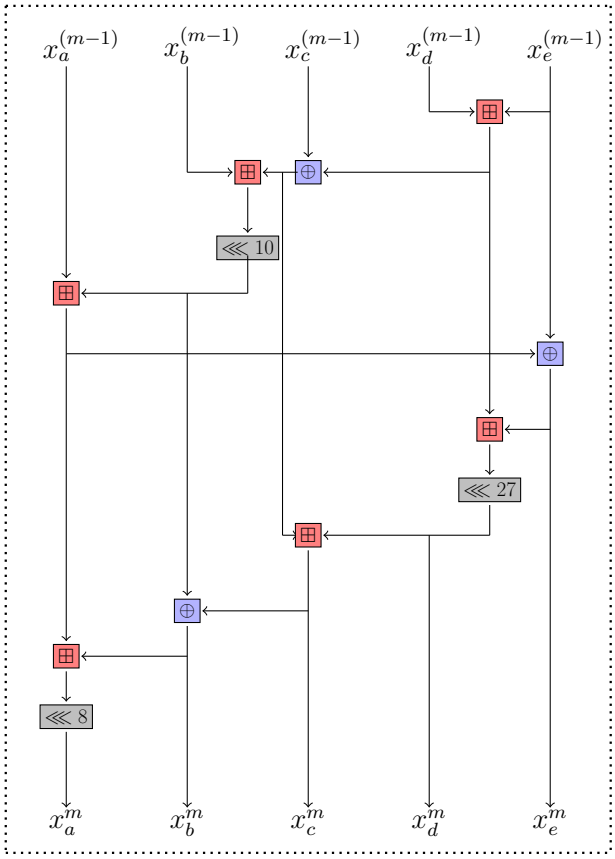

Figure 1. One SR_{forro}

2.2. Poly1305

Poly1305, designed by D.J. Bernstein [Bernstein 2005], is a widely used message authentication code (MAC) algorithm that provides cryptographic authentication for data. It has become popular due to its efficiency, security, and ease of implementation, making it a suitable choice for various applications, including secure communications and digital signatures.

Poly1305 takes a 32-byte one-time key and a message and produces a 16-byte tag. This tag is used to authenticate the message. Poly1305 operates on a 130-bit prime field (thus the name "Poly1305") and computes the MAC using a polynomial evaluation.

The key, regardless of its generation method, is divided into two segments, denoted as r and s. Each invocation should have a unique and unpredictable pair of (r, s). Here, denote r_i for $i = (0, 1, ..., 15)$ as bytes, then r can be represented as a 16-octet little-endian number and the modifications are as follows:

- The top four bits of r_3, r_7, r_{11}, and r_{15} must be zero, meaning that each byte should be less than 16.
- The bottom two bits of r_4, r_8, and r_{12} must be zero, meaning they should be divisible by 4.

From the input, the message is divided into 128-bit blocks with a pad to the last block if necessary. Each block is interpreted as a little-endian integer, integrated into an accumulator, multiplied by the key, and reduce it modulo $2^{130} - 5$. At the end, s is added to the accumulator to obtain the 128-bit authentication tag.

3. The proposal of XForró14

In this section, we propose XForró14, a new family of stream ciphers, including HForró14, an intermediate step towards XForró14.

3.1. Definition of HForró14

HForró14 is a Key Derivation Function (KDF) that produces a 256-bit output key starting from an input 256-bit key k and a 128-bit nonce v. As in Forró, we write the key and nonce as 32-bit words $(k_0, k_1, ..., k_7)$ and (v_0, v_1, v_2, v_3). We write:

$$(k_0^*, k_1^*, ..., k_7^*) = HForro14((k_0, k_1, ..., k_7), (v_0, v_1, v_2, v_3)). \tag{3}$$

The initial state of HForró14 is then defined as Eq. (1) replacing (t_0, t_1, v_1, v_2) by (v_0, v_1, v_2, v_3). From this initial state, HForró14 proceeds just like Forró14, by using the subround function to compute the state $X^{(14)}$. However, instead of summing it to the initial state, HForró14 just outputs $(x_6^{(14)}, x_7^{(14)}, x_{14}^{(14)}, x_{15}^{(14)}, x_4^{(14)}, x_5^{(14)}, x_{12}^{(14)}, x_{13}^{(14)})$. The indices 6, 7, 14, 15, 4, 5, 12, 13 here are important for the security proof later in this paper.

3.2. Definition of XForró14

XForró14 is a stream cipher that receives as input a 256-bit key $k = (k_0, k_1, ..., k_7)$, a 192-bit nonce $n = (v_0, v_1, ..., v_5)$, and a counter (t_0, t_1) and output a 512-bit stream block Z. XForró14 is divided in two steps:

1. Compute $k^* = HForro14((k_0, k_1, ..., k_7), (v_0, v_1, v_2, v_3))$.
2. Compute the stream block $Z = Forro14(k^*, (v_4, v_5), (t_0, t_1))$.

From an implementation perspective, when encrypting a long stream with the same key and nonce (and incremental counters), step 1 may be computed only once. Then, the resulting k^* together with (v_4, v_5) used to encrypt with Forró14 for all counter values.

4. Security proof

This section proves that HForró14 and XForró14 are secure if Forró14 is secure. The proof relies on the standard security notion that a cipher is secure if the cipher outputs for a uniform random secret key are indistinguishable from independent uniform random strings.

4.1. The basics

To get to the proof, we first need to establish some base definitions and theorems.

Definition 1. *Let $A(\mathcal{C})$ be an algorithm that receives as input a component $\mathcal{C}$. We define the A-distance δ from $\mathcal{C}$ to uniform as*

$$\delta = |\Pr[A(\mathcal{C}) = 1] - \Pr[A(U) = 1]|,$$

where U denote the uniform distribution.

The proof of security of XForró14 is based on a general proof for generalized cascades. Precisely, a generalized cascade is a type of encryption algorithm that involves combining multiple rounds of different encryption functions in a cascading fashion: each round of encryption involves applying a different encryption function to the output of the previous round. The output of the final round is the ciphertext. Previous works already establishes the security of this construction, and we use results from [Bellare et al. 1996, Bernstein 2011], to derive our proof.

Here, let K_1, I_1, K_2, I_2, L be sets, with K_1, K_2, L finite. Let H be a function from $K_1 \times I_1$ to K_2. Let F be a function from $K_2 \times I_2$ to L. Define X as a generalized cascade of the form $(k_1, i_1, i_2) \mapsto F(H(k_1, i_1), i_2)$ from $K_1 \times I_1 \times I_2$ to L. The general proof for generalized cascades of the form $X = ((k_1, i_1, i_2) \mapsto F(H(k_1, i_1), i_2))$ is based on an attack with the goal of distinguish an oracle for $X(k_1)$, where k_1 is a uniform random element of K_1, from an oracle for a uniform random function from $I_1 \times I_2$ to L. Here $X(k_1)$ means $(i_1, i_2) \mapsto X(k_1, i_1, i_2)$; i.e., $(i_1, i_2) \mapsto F(H(k_1, i_1), i_2)$. Starting from a fast successful attack against $X(k_1)$, the security proof constructs fast attacks against $H(k_1)$ and $S(k_2)$, where k_2 is a uniform random element of K_2, and shows that at least one of these attacks must be successful. In other words, if H and F are both secure, then X must also be secure.

Next, we provide additional definitions that are relevant for describing the main result on generalized cascades at Theorem 1.

Definition 2. *Given an algorithm A, we define algorithms $A_0, A_1, A_2, \ldots$ as follows:*

- *The algorithm A_0, given an oracle $O : I_1 \to K_2$, runs the algorithm A with the oracle $(i_1, i_2) \mapsto F(O(i_1), i_2)$.*
- *For $j \geq 1$: The algorithm A_j, given an oracle $O : I_2 \to L$, generates a uniform random function U from $I_1 \times I_2$ to L and, independently of U, a uniform random function V from I_1 to K_2. It runs the algorithm A with the following oracle: respond to (i_1, i_2) with $U(i_1, i_2)$ for the first $j - 1$ distinct query prefixes i_1 that appear, with $O(i_2)$ for the jth distinct query prefix, and with $F(V(i_1), i_2)$ for all other query prefixes.*

Theorem 1. *(Theorem 3.1 of [Bernstein 2011]) Let K_1, I_1, K_2, I_2, L be sets, with K_1, K_2, L finite. Let H be a function from $K_1 \times I_1$ to K_2. Let F be a function from $K_2 \times I_2$ to L. Define X as the function $(k_1, i_1, i_2) \mapsto F(H(k_1, i_1), i_2)$ from $K_1 \times I_1 \times I_2$ to L. Let A be an algorithm that makes at most q oracle queries. Define $A_0, A_1, \ldots, A_q$*

as in Definition 2. Define δ_0 (Definition 1 as the A_0-distance from $H(k_1)$ to uniform, where k_1 is a uniform random element of K_1. Define δ_j, for $j \in \{1, \ldots, q\}$, as the A_j-distance from $F(k_2)$ to uniform, where k_2 is a uniform random element of K_2. Hence, the A-distance from $X(k_1)$ to uniform, where k_1 is a uniform random element of K_1, is at most $\delta_0 + \delta_1 + \cdots + \delta_q$.

Proof. See [Bernstein 2011]. □

4.2. The security of XForró14

For the proof, we rewrite Forró as a generalized cascade allowing us to use previous results, simplifying the proof. To do so, define $\mathrm{HForro}_k(i)$, where k is a 256-bit string and i is a 128-bit string, as the 256-bit HForró14 output block for key k and nonce i. Recall that this output is $(x_6^{(14)}, x_7^{(14)}, x_{14}^{(14)}, x_{15}^{(14)}, x_4^{(14)}, x_5^{(14)}, x_{12}^{(14)}, x_{13}^{(14)})$ where indexes 6,7,14,15 represent the constant position and 4,5,12,13 the nonce.

Define $\mathrm{Forro}_k(i)$, where k is a 256-bit string and i is a 128-bit string, as the 512-bit Forró14 output block for key k, nonce equal to the first half of i, and block counter equal to the second half of i. Recall that this output is $\left(x_0^{(0)} + x_0^{(14)}, x_1^{(0)} + x_1^{(14)}, \ldots, x_{15}^{(0)} + x_{15}^{(14)}\right)$ where $\left(x_6^{(0)}, x_7^{(0)}, x_{14}^{(0)}, x_{15}^{(0)}\right)$ is the Forró14 constant, $\left(x_0^{(0)}, x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, x_8^{(0)}, x_9^{(0)}, x_{10}^{(0)}, x_{11}^{(0)}\right)$ is the key k, $\left(x_4^{(0)}, x_5^{(0)}, x_{12}^{(0)}, x_{13}^{(0)}\right)$ is the input i. Consequently XForró

$$\mathrm{XForro}_k(i) = \mathrm{Forro}_{\mathrm{HForro}}(i_1)(i_2)$$

where i_1 is the first half of the input i and i_2 is the second half of the input i.

Now, we can define Theorem 2 that states that XForró14 is secure if Forró14 and HForró14 are secure.

Theorem 2. *Let A be an algorithm that makes at most q oracle queries. Define $A_0, A_1, \ldots, A_q$ as in Definition 2, where $K_1 = K_2 = \{0,1\}^{256}, I_1 = I_2 = \{0,1\}^{128}$, and $L = \{0,1\}^{512}$. Let k be a uniform random element of $\{0,1\}^{256}$. Define δ_0 as the A_0-distance from $\mathrm{HForro14}_k$ to uniform. Define δ_j, for $j \in \{1, \ldots, q\}$, as the A_j-distance from $\mathrm{Forro14}_k$ to uniform. Then the A-distance from $\mathrm{XForro14}_k$ to uniform is at most $\delta_0 + \delta_1 + \cdots + \delta_q$.*

Proof. Define $H : K_1 \times I_1 \to K_2$ as $(k, i) \mapsto \mathrm{HForro}_k(i)$. Define $F : K_2 \times I_2 \to L$ as $(k, i) \mapsto \mathrm{Forro}_k(i)$. Define $X : K_1 \times I_1 \times I_2 \to L$ as $(k, i_1, i_2) \mapsto \mathrm{XForro}_k(i_1, i_2)$. Then $X(k, i_1, i_2) = F(H(k, i_1), i_2)$. The A-distance from $X(k)$ to uniform is at most $\delta_0 + \delta_1 + \cdots + \delta_q$ by Theorem 1. □

Now, note that Theorem 2 still depends on the security of HForró14, which we did not establish. To solve this, Theorem 3 states that HForró14/r is secure if Forró14/r is secure. The theorem applies to any distribution of keys, and in particular to the uniform distribution considered in Theorem 2. Combining Theorem 3 with Theorem 2 shows that XForró14/r is secure if Forró14/r is secure.

Theorem 3. *Let k be a random element of $\{0,1\}^{256}$. Define $s_0, s_1, ..., s_{15}$ as the output of Forró14. Let A be an algorithm. Define $Q : \{0,1\}^{128} \times \{0,1\}^{512} \to \{0,1\}^{256}$ by $Q(x_4, x_5, x_{12}, x_{13}, s_0, s_1, \ldots, s_{15}) = (s_6 - c_0, s_7 - c_1, s_{14} - c_2, s_{15} - c_3, s_4 - x_4, s_5 - x_5, s_{12} - x_{12}, s_{13} - x_{13})$ where (c_0, c_1, c_2, c_3) is the Forró14 constant. Define B as the algorithm that, given an oracle $O : \{0,1\}^{128} \to \{0,1\}^{512}$, runs A with the oracle $i \mapsto Q(i, O(i))$. Then the A-distance from HForro_k to uniform is the same as the B-distance from Forro_k to uniform.*

Proof. Compare the definitions of Forró14 and HForró14 to see that if $i = (x_4, x_5, x_{12}, x_{13})$ and $\mathrm{Forro}_k(i) = (s_0, s_1, \ldots, s_{15})$ then

$$\mathrm{HForro}_k(i) = (s_6 - c_0, s_7 - c_1, s_{14} - c_2, s_{15} - c_3, s_4 - x_4, s_5 - x_5, s_{12} - x_{12}, s_{13} - x_{13})$$

and then

$$\mathrm{HForro}_k(i) = Q\left(i, \mathrm{Forro}_k(i)\right).$$

Hence, $B\left(\mathrm{Forro}_k\right) = A\left(i \mapsto Q\left(i\ , \mathrm{Forro}_{\ k}(i)\right)\right) = A\left(\mathrm{HForro}_k(i)\right)$.

Let U be a uniform random function from $\{0,1\}^{128}$ to $\{0,1\}^{512}$. Define $V(i) = Q(i, U(i))$. Thus, V is a uniform random function from $\{0,1\}^{128}$ to $\{0,1\}^{256}$. Furthermore, $B(U) = A(V)$. The B-distance from Forro_k to U is therefore the same as the A-distance from $\mathrm{HForro}_k(i)$ to V. □

As a corollary, if Forró14 has insecurity $\leq \epsilon$ against any algorithm as fast as B, then HForró14 has insecurity $\leq \epsilon$ against any algorithm as fast as A. Note that B has almost the same speed as A.

5. A novel AEAD scheme: XForró14-Poly1305

To create the XForró14-Poly1305 AEAD scheme, we integrate the Poly1305 authentication mechanism with the XForró14 cipher. XForró14-Poly1305 is similar with ChaCha20-Poly1305 defined in RFC 8439 `https://datatracker.ietf.org/doc/html/rfc8439`. XForró14-Poly1305 receives as input a message m, the associated data d, a 256-bit key k, and a 192-bit long nonce v. Then, the algorithm proceeds as follows:

1. Compute $k^* = HForro14((k_0, k_1, ..., k_7), (v_0, v_1, v_2, v_3))$.
2. Compute the stream block $Z = Forro14(k^*, (v_4, v_5), (0, 0))$.
3. Define r as bits $0, 1, ..., 127$ of Z, and s as bits $128, 129, ..., 255$ of Z.
4. Next, the Forró14 encryption function is called to encrypt the plaintext, using the same key k^* and nonce (v_4, v_5), and with the initial counter set to 1. The resulting ciphertext is denoted by c.
5. The Poly1305 function is called to compute a tag τ. The Poly1305 function receives as input the key pair r and s calculated above, and a message constructed as follows:

   $$m^* = d|\mathrm{pad}(d)|c|\mathrm{pad}(c)|\mathrm{len}(d)|\mathrm{len}(c),$$

   where len(.) computes the length of the additional data in octets, and pad(.) is a padding up to 15 zero bytes, and it brings the total length so far to an integral multiple of 16.

6. Finally, the output of the AEAD is the concatenation of $c|\tau$.

We summarize the encryption scheme in Figure 2.

The decryption process shares similarities with encryption, but with a few key distinctions:

- The roles of ciphertext and plaintext are swapped. In this case, the Forró14 encryption function is applied to the ciphertext, which results in the plaintext.
- The Poly1305 function is executed on the associated data and the ciphertext, not on the plaintext.
- The computed tag is then compared bit by bit with the received tag. The message is considered authenticated only if the tags are identical.

The amount of encrypted data possible in a single invocation is $2^{64} - 1$ blocks of 64 bytes each, because of the size of the block counter field in the Forró14 block function. This gives a total of 1,180,591,620,717,411,303,360 bytes, or over a million petabytes. This should be enough for traffic protocols such as IPsec and TLS, and also for file and/or disk encryption.

The integration of Poly1305 with the XForró14 cipher not only provides authentication capabilities but also enhances the security of the overall AEAD scheme. The combined XForró14-Poly1305 AEAD scheme offers robust protection against various attacks, including tampering and forgery, while maintaining the efficiency and performance advantages of the Forró cipher.

6. Implementation

To demonstrate XForró14-Poly1305, libsodium was forked, adding our proposed AEAD scheme to it. libsodium is a modern, easy-to-use software library for encryption, decryption, signatures, password hashing, and more. It provides a high-level and comprehensive set of cryptographic primitives, aiming to make it easier for developers to build secure applications. Our implementation can be accessed at `https://github.com/murcoutinho/libsodium`. This approach maintains compatibility with existing libsodium-dependent systems and benefits from its well-tested Poly1305 implementation.

To implement XForró-Poly1305 into libsodium, HForró14 was incorporated into the core functionality of libsodium, while Forró14 was included as part of its stream cipher suite. This incorporation facilitated the utilization of these components in the construction of the desired scheme. Forró14 offers three distinct implementations: AVX2 and SSSE3, both of which are derived from chacha's dolbeau implementation, along with a reference implementation. The compilation flags for libsodium remain unchanged, building with `gcc` using:

```
-Ofast -pthread -fvisibility=hidden -fPIC -fPIE
-fno-strict-aliasing -fno-strict-overflow
-fstack-protector -Wno-deprecated-declarations
-Wno-unknown-pragmas -ftls-model=local-dynamic
-D_FORTIFY_SOURCE=2
```

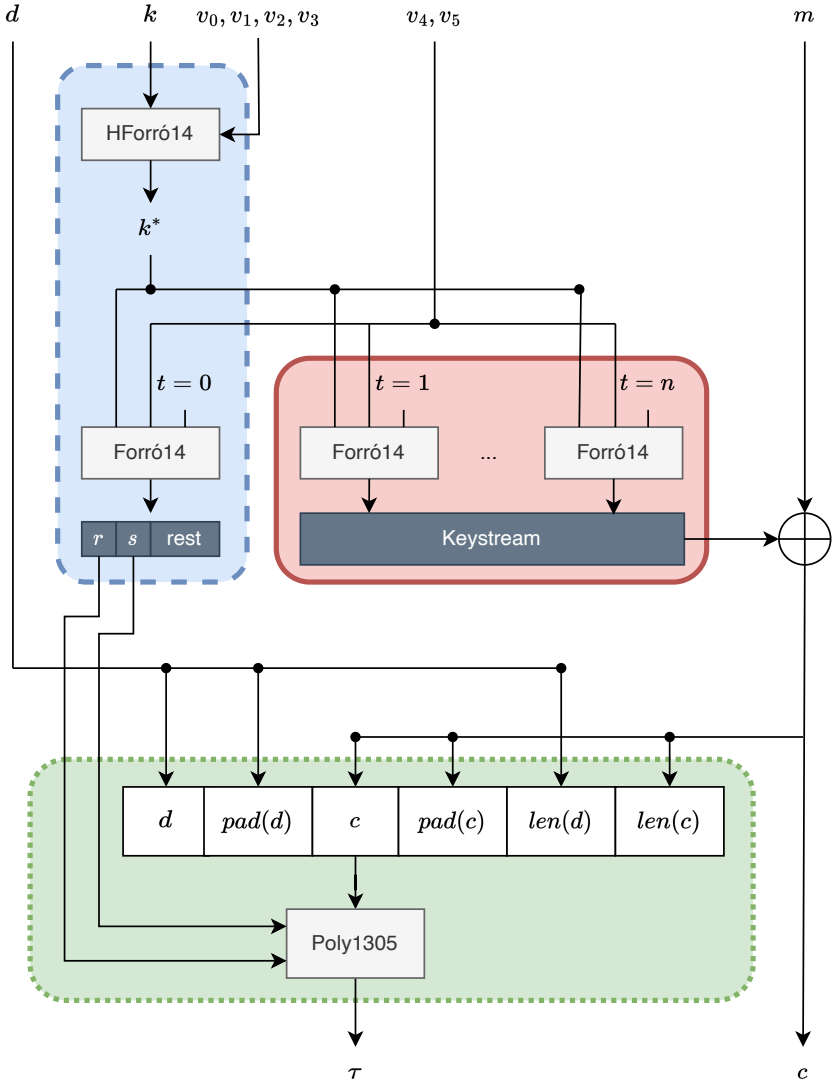

Figure 2. A graphical representation of XForró14-Poly1305. The blue area (between dashed lines) represents the XForró14 step. The red area (between a solid line) represents the encryption step with Forró14, using a total of n blocks, starting from counter $t = 1$. The green area (between dotted lines) represents the authentication with Poly1305.

What implementation is being compiled dictates the inclusion or exclusion of either `-mavx2` or `-mssse3`.

Following the integration of the schemes, it becomes possible to assess their performance in comparison to other available AEAD schemes. The performance evaluation encompasses various metrics, which are presented in Table 2. Based on these findings, XForró14-Poly1305 demonstrates superior performance on resource-constrained devices, particularly on resource-limited devices like the ARMv7l found in the tested Xilinx Pynq-Z1, which features a Cortex-A9 processor. Furthermore, even on the ARMv8 platform, specifically the Cortex-A72 processor in a Raspberry Pi 4B, XForró14-Poly1305 outperforms XChaCha20-Poly1305, through the use of the Xote technique discovered in [Coutinho et al. 2022, Coutinho et al. 2023].

Table 2. Performance Comparison of AEAD Schemes on ARMv7l and ARMv8 64-bit, including AEGIS [Wu and Preneel 2013] which is available in libsodium.

AEAD Scheme	ARMv7l (cycles/byte)	ARMv8 64-bit (cycles/byte)
XForró14-Poly1305	23.33	11.88
XChaCha20-Poly1305	26.75	12.89
ChaCha20Poly1305	26.73	12.88
AEGIS256	425.66	228.27
AEGIS128L	286.73	152.45

Another interesting comparison can be drawn against lightweight cryptographic alternatives, such as ASCON, a winner of the NIST competition [Dobraunig et al. 2016]. Unfortunately, libsodium does not support ASCON, and its implementation was beyond the scope of this paper. Nevertheless, the authors report performance metrics of 33.3 and 10.5 cycles/byte for ARMv7 (Cortex-A9) and ARMv8 (Cortex-A72), respectively, demonstrating that XForró14-Poly1305 stands as a competitive alternative. It's worth noting that while ASCON may perform approximately 1 cycle/byte faster on the ARMv8 architecture, it only provides 128 bits of security. In contrast, XForró14-Poly1305 offers a higher level of security at 256 bits, making it an important consideration in assessing the relative merits of these algorithms. These findings suggest that XForró14-Poly1305 offers a promising solution for efficient cryptographic operations on constrained devices.

7. Conclusion

In this paper, we address the authentication limitation of the Forró cipher by introducing the XForró14 cipher and combining it with the Poly1305 authentication mechanism to create a comprehensive Authenticated Encryption with Associated Data (AEAD) scheme. The resulting XForró14-Poly1305 AEAD scheme leverages the efficiency and security advantages of the original Forró cipher while providing the critical authentication functionality necessary for a wide range of use cases.

To ensure the practical implementation of the XForró14-Poly1305 AEAD scheme, we developed a new fork of the libsodium cryptographic library, which incorporates our proposed solution as a readily available AEAD option. This allows developers to utilize a secure and efficient cryptographic tool in their projects with ease, contributing to the broader adoption of the XForró14-Poly1305 AEAD scheme.

Our work demonstrates that it is possible to enhance the capabilities of an existing cipher, such as Forró, by extending it with authentication features, resulting in a more versatile and secure cryptographic solution. Future research could explore further optimizations, performance improvements, and potential applications for the XForró14-Poly1305 AEAD scheme. Moreover, the cryptographic community could benefit from a thorough security analysis of the proposed scheme, assessing its resilience against various attack vectors and confirming its suitability for deployment in real-world applications.

References

Aumasson, J., Fischer, S., Khazaei, S., Meier, W., and Rechberger, C. (2008). New features of latin dances: Analysis of Salsa, ChaCha, and Rumba. In Nyberg, K., editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer.

Beierle, C., Leander, G., and Todo, Y. (2020). Improved differential-linear attacks with applications to ARX ciphers. In Micciancio, D. and Ristenpart, T., editors, *Advances in Cryptology - CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 329–358. Springer.

Bellare, M., Canetti, R., and Krawczyk, H. (1996). Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 514–523. IEEE Computer Society.

Bellare, M. and Namprempre, C. (2000). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto, T., editor, *Advances in Cryptology - ASIACRYPT 2000, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer.

Bernstein, D. J. (2005). The poly1305-aes message-authentication code. In Gilbert, H. and Handschuh, H., editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer.

Bernstein, D. J. (2008a). ChaCha, a variant of Salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5.

Bernstein, D. J. (2008b). The Salsa20 family of stream ciphers. In Robshaw, M. J. B. and Billet, O., editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer.

Bernstein, D. J. (2011). Extending the Salsa20 nonce. In *Workshop record of Symmetric Key Encryption Workshop*, volume 2011.

Choudhuri, A. R. and Maitra, S. (2016). Significantly improved multi-bit differentials for reduced round Salsa and ChaCha. *IACR Trans. Symmetric Cryptol.*, 2016(2):261–287.

Coelho, K. K., Nogueira, M., Marim, M. C., Silva, E. F., Vieira, A. B., and Nacif, J. A. M. (2022). Lorena: Low memory symmetric-key generation method for based on

group cryptography protocol applied to the internet of healthcare things. *IEEE Access*, 10:12564–12579.

Costa, V. L. R. D., Camponogara, A., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

Coutinho, M. and Neto, T. C. S. (2021). Improved linear approximations to ARX ciphers and attacks against ChaCha. In Canteaut, A. and Standaert, F., editors, *Advances in Cryptology - EUROCRYPT 2021 - Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 711–740. Springer.

Coutinho, M., Passos, I., Vásquez, J. C. G., de Mendonça, F. L. L., de Sousa, R. T., and Borges, F. (2022). Latin dances reloaded: Improved cryptanalysis against Salsa and ChaCha, and the proposal of Forró. In Agrawal, S. and Lin, D., editors, *Advances in Cryptology - ASIACRYPT 2022 - Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 256–286. Springer.

Coutinho, M., Passos, I., Vásquez, J. C. G., de Mendonça, F. L. L., Sarkar, S., de Sousa, R. T., and Borges, F. (2023). Latin dances reloaded: Improved cryptanalysis against Salsa and ChaCha, and the proposal of Forró (extended version). *J. Cryptol.*, 36(18).

Denis, F. libsodium.

Dey, S., Garai, H. K., Sarkar, S., and Sharma, N. K. (2022). Revamped differential-linear cryptanalysis on reduced round ChaCha. In *Advances in Cryptology–EUROCRYPT 2022, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*, pages 86–114. Springer.

Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. (2016). Ascon v1. 2. *Submission to the CAESAR Competition*, 5(6):7.

IANIX (2020). ChaCha usage & deployment. `https://ianix.com/pub/chacha-deployment.html`. Accessed: 2020-01-13.

Jimale, M. A., Z'aba, M. R., Kiah, M. L. M., Idris, M. Y. I. B., Jamil, N., Mohamad, M. S., and Rohmad, M. S. (2022). Authenticated encryption schemes: A systematic review. *IEEE Access*, 10:14739–14766.

Langley, A., Chang, W., Mavrogiannopoulos, N., Strömbergson, J., and Josefsson, S. (2016). ChaCha20-Poly1305 cipher suites for transport layer security (TLS). *RFC*, 7905:1–8.

McKay, K., Bassham, L., Sönmez Turan, M., and Mouha, N. (2016). Report on lightweight cryptography. Technical report, National Institute of Standards and Technology.

Nir, Y. (2015). Rfc 7634: Chacha20, poly1305, and their use in the internet key exchange protocol (ike) and ipsec.

Wu, H. and Preneel, B. (2013). AEGIS: A fast authenticated encryption algorithm. In Lange, T., Lauter, K. E., and Lisonek, P., editors, *Selected Areas in Cryptography - SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer.