POSTER: Tortoise: An Authenticated Encryption Scheme

Kenneth Odoh

kenneth.odoh@gmail.com
https://kenluck2001.github.io

Abstract. Given the open nature of the Internet, authentication schemes are designed to address inherent trust issues. We present Tortoise, an experimental nonce-based authenticated encryption scheme modeled on the Synthetic Counter-in-Tweak. This paper demonstrates a generalizable plug-and-play framework for converting block cipher into Authenticated Encryption with Associated Data. As part of this work, we utilized an XOR procedure for constructing a generic tweakable cipher. Finally, we support two modes: nonce-respecting and nonce-misuse-resistant [1].

1 Introduction

The Internet provides an open platform for several forms of interaction between users. Given the open nature of the web, there are trust issues due to bad actors taking advantage of unsuspecting users. Cryptography-based systems can provide the right tool for achieving trust. There are two broad categories of cryptosystems: Symmetric-Key encryption and Asymmetric-Key encryption [5]. The encryption and decryption of messages in Symmetric-Key encryption use the same key and a pair of keys (private and public key) for encryption and decryption in Asymmetric-Key encryption. Authenticated encryption (AE) allows multiple parties to exchange messages with secrecy and integrity. Users can verify the real identity of the message creator and prevent message forgeries that can erode trust.

Recent CAESER competitions [2] have shown the need for authenticated encryption schemes. Our work introduced an Authenticated Encryption with Associated Data (AEAD) scheme based on well-known cipher primitives with well-studied security characteristics. Tortoise is a novel cryptographic algorithm based on a tweakable variant of AES [1]. Our proposed scheme provides a generalizable framework capable of converting any block cipher (slightly modified to accommodate different block sizes) into an authenticated encryption scheme. Tortoise is a symmetric cryptographic scheme that allows parties to communicate by encrypting and authenticating their messages. AEAD provides confidentiality, integrity, and authentication. Tortoise utilizes 128-bit message blocks, determined by the underlying cipher's block size (in this case, AES). It emphasizes practicality, with an emphasis on efficient hardware and software implementations. A nonce-resistant variant of Tortoise achieves full misuse-resistance (MRAE) security.

[1] **Source**: https://github.com/kenluck2001/cipherResearch/tree/main/src/tortoise

2 Related Work

Synthetic Counter-in-Tweak (SCT) [13] is a combination of Encrypted Parallel Wagman-Carter (EPWC) and Counter-in-Tweaks (CTRT). As a result, it provides security for nonce-respecting mode beyond birthday bounds. On the other hand, in the nonce-misuse resistance, security is provided until the birthday-bound period [7]. The Tweakey framework [9] is used in Deoxys [8] and limits the flexibility of the encryption round. This setup is undesirable because of tight coupling with the underlying encryption routine. On the contrary, the XOR scheme does not modify the encryption round. Hence, we can use it as a foundation to create a generic framework for tweaking any block cipher.

EAX [4] is an authenticated scheme that introduced the use of associated data with provable security in AE. This is also utilized in Deoxys [8], Ascon [6]. Nonce reuse schemes are frequent in many authentication schemes. However, HBS and BTM provide nonce-misuse resistance [7] as supported in Deoxys [8]. Our authentication scheme follows authenticate-then-encrypt, which was proven secure using a tweakable cipher or CBC mode [11]. We have adopted a conservative approach to cipher design, leveraging primitives with established security foundations. Our construction involves building a novel authenticated encryption (AE) scheme. This scheme utilizes the robust SCT [13] primitive as its core, followed by a tweakable cipher based on the AES cipher. The resulting AEAD supports arbitrary message lengths through a padding scheme for multiples of the message block size, n, employing the PKCS7 strategy [10].

Algorithm 1 Encryption Algorithm, $E_A(P, K, A, N)$

```
Require: $P$: plainText, $K$: key, $A$: associated data, and $N$: nonce where $l_a$: length of associated data, $A$, of multiple of blocks of size, $n$ i.e $|A_i| = n$, $l_p$: length of plainText, $P$, of multiple of blocks of size, $n$ i.e $|P_i| = n$
Ensure: $C$: cipherText i.e $|C_i| = n$, $tag$
  $auth = 0$
                                   ▷ // Processing associated data
  for all $i \in l_a$ do
      $auth = auth \oplus E_n(K, 0010|i, A_i)$
  end for
                                   ▷ // Processing plaintext data
  $checksum = 0^n$
  for all $j \in l_p$ do
      $checksum = checksum \oplus P_j$
      $C_j = E_n(K, 0000|N|j, P_j)$
  end for
  $fTag = E_n(K, 0001|N|l_p, checksum)$
                                   ▷ // Tag generation
  $tag = fTag \oplus auth$
```

Algorithm 2 Decryption Algorithm, $D_A(C, K, A, tag, N)$

```
Require: $C$: cipherText, $K$: key, $A$: associated data, tag, and $N$: nonce where $l_a$: length of associated data, $A$, of multiple of blocks of size, $n$ i.e $|A_i| = n$, $l_c$: length of cipherText, $C$, of multiple of blocks of size, $n$ i.e $|C_i| = n$
Ensure: $P$: plainText i.e $|P_i| = n$, $\hat{tag}$
  $auth = 0$
                                   ▷ // Processing associated data
  for all $i \in l_a$ do
      $auth = auth \oplus E_n(K, 0010|i, A_i)$
  end for
                                   ▷ // Processing ciphertext data
  $checksum = 0^n$
  for all $j \in l_c$ do
      $P_j = D_n(K, 0000|N|j, C_j)$
      $checksum = checksum \oplus P_j$
  end for
  $fTag = E_n(K, 0001|N|l_c, checksum)$
                                   ▷ // Tag generation
  $\hat{tag} = fTag \oplus auth$
  if $\hat{tag} == tag$ then return plaintext, $P$
  else
                                   ▷ // Do nothing
  end if
```

3 Design Rationale and Implementation

Algorithm 3 Encryption Algorithm, $E_A(P, K, A, N)$ as shown in Definition 1

```
Require: P: plainText, K: key, A: associated data, and N: nonce where $l_a$: length of associated data, A, of multiple of blocks of size, n i.e $|A_i| = n$, $l_p$: length of plainText, P, of multiple of blocks of size, n i.e $|P_i| = n$
Ensure: C: cipherText i.e $|C_i| = n$, tag
                                        ▷ // Processing associated data
  $auth = 0$
  for all $i \in l_a$ do
      $auth = auth \oplus E_n(K, 0010|i, A_i)$
  end for
                                        ▷ // Processing plaintext data
  $tag = auth$
  for all $j \in l_p$ do
      $tag = tag \oplus E_n(K, 0000|N|j, P_j)$
  end for
                                        ▷ // Tag generation
  $tag = E_n(K, 0001|0^4|N, tag)$
                                        ▷ // Message encryption
  for all $j \in l_p$ do
      $C_j = P_j \oplus E_n(K, tag \oplus j, 0^8|N)$
  end for
```

Algorithm 4 Decryption Algorithm, $D_A(C, K, A, tag, N)$

```
Require: C: cipherText, K: key, A: associated data, tag, and N: nonce where $l_a$: length of associated data, A, of multiple of blocks of size, n i.e $|A_i| = n$, $l_c$: length of cipherText, C, of multiple of blocks of size, n i.e $|C_i| = n$
Ensure: P: plainText i.e $|P_i| = n$, $\hat{tag}$
                                        ▷ // Message encryption
  for all $j \in l_c$ do
      $P_j = C_j \oplus E_n(K, tag \oplus j, 0^8|N)$
  end for
                                        ▷ // Processing associated data
  $auth = 0$
  for all $i \in l_a$ do
      $auth = auth \oplus E_n(K, 0010|i, A_i)$
  end for
                                        ▷ // Processing plaintext data
  $\hat{tag} = auth$
  for all $j \in l_p$ do
      $\hat{tag} = \hat{tag} \oplus E_n(K, 0000|N|j, P_j)$
  end for
                                        ▷ // Tag generation
  $\hat{tag} = E_n(K, 0001|0^4|N, \hat{tag})$
                                        ▷ // Tag verification
  if $\hat{tag} == tag$ then return plaintext, P
  else
                                        ▷ // Do nothing
  end if
```

Our scheme defines encryption as $C, tag = E_A(P, K, A, N)$ and decryption as $P = D_A(C, K, A, tag, N)$, where successful decryption requires a valid tag match, ensuring authenticity and P: plaintext, C: ciphertext, K: key, A: associated data, and N: nonce respectively as seen in Algorithms 1, 2, 3 and 4. Unlike Deoxys [8], this scheme mandates that the tweak block size matches the message block size. Instead of the Tweakey framework used in Deoxys [8], this scheme employs tweaks using XOR with a 2-universal hashing function [12] for greater cipher flexibility, potentially enabling support for ciphers beyond the AES family. This approach offers a more versatile AEAD scheme with broader applicability.

Algorithm 1 encrypts plaintext, authenticates associated data, and generates a tag. It iteratively encrypts associated data tokens with tweaks derived from their indices, accumulating the results. Plaintext tokens are encrypted with tweaks derived from the nonce and token indices, and a checksum is calculated. Finally, the checksum is encrypted with a tweak derived from the nonce and the number of plaintext tokens, and the result is XORed with the accumulated authentication value to produce the tag. Similarly, Algorithm 2 decrypts ciphertext, authenticates associated data, and verifies the integrity of the data. It decrypts ciphertext tokens with tweaks derived from the nonce and token indices, accumulating the results into a checksum. Associated data is authenticated similarly to the encryption algorithm. A tag is computed, and its comparison with the provided tag determines decryption success.

Nonce reuse, while potentially unavoidable in certain scenarios, can compromise the security of traditional encryption schemes. These algorithms incorporate mechanisms to mitigate the impact of nonce reuse. Algorithm 3 encrypts plaintext, authenticates associated data, and generates a tag. This routine achieves nonce misuse resistance. It authenticates associated data, generates a preliminary tag by XORing encryptions of plaintext tokens, and derives the final tag by encrypting the accumulated tag. Ciphertext is created by XORing plaintext with the encryption of custom data using a tweak derived from the final tag and the token index. Similarly, Algorithm 4 decrypts ciphertext, authenticates associated data, and verifies the integrity of the data. Plaintext is recovered by XORing ciphertext with the encryption of custom data using a tweak derived from the provided tag and the token index. Associated data is authenticated, and a tag is computed. The computed tag is compared to the provided tag, determining decryption success.

4 Security Analysis

We have adopted a conservative approach to our cipher design, basing the foundation on well-tested and verified security primitives and benefiting from their composability in our unique construction. As a result, we can have minimal doubt that we can demonstrate the security of our construction without a very elaborate security analysis.

SCT [13] has proven its security properties by serving as the foundation for ASCON [6] and Deoxys [8]. The underlying cipher is AES (immaterial to our authentication routine). The cipher choice in this work is arbitrary, and we focus on building a plug-and-play framework for converting any block cipher into an authenticated encryption scheme, provided that the tweak size is equal to the message block size. The proof of security properties of our tweaking scheme used in our formulation in Theorem 2 of paper [12]. We have provided evidence of a strong tweakable block cipher.

The paper specifically used AES, but any block cipher would suffice. Hence, we can revisit the security of AES [1] with a track record of subjection to rigorous real-world security analysis. Our framework is influenced by the choice of underlying cipher. For example, if the base cipher is AES, then we support security levels of 2^{128} for AES 128. Our underlying tweakable AES [1] is reasonably resilient to differential and linear cryptanalysis. However, it may be vulnerable to the meet-in-the-middle attack [3]. More investigations are needed to evaluate resilience to related-key attacks.

5 Conclusions and Future Work

We have released Tortoise as a general-purpose scheme for converting any block into AEAD. As a result of this work, we have created a universal procedure for constructing a generic tweakable cipher which makes it ideal to employ multiple esoteric cipher that do not have a MAC for authentication. Furthermore, future work could entail relaxing the notion of MRAE in the form of Online AE (OAE), which requires only a single pass for nonce-misuse resistant ciphers [2] [3].

References

1. Advanced Encryption Standard (AES). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (2001), date accessed: July 28, 2023
2. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar.html (2001), date accessed: September 26, 2023
3. Bar-On, Achiya and Dunkelman, Orr and Keller, Nathan and Ronen, Eyal and Shamir, Adi: Improved Key Recovery Attacks On Reduced-Round AES with Practical Data and Memory Complexities. In: Proceedings of the 38th Annual International Cryptology Conference on Advances in Cryptology. pp. 185–212 (2018)
4. Bellare, M., Rogaway, P., Wagner, D.A.: Eax: A conventional authenticated-encryption mode. IACR Cryptology ePrint Archive **2003**, 69 (2003)
5. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (2006)
6. Dobraunig, C., Eichlseder, M., Mendel, F., Schlaffer, M.: Ascon v1.2: Submission to the CAESAR Competition. https://competitions.cr.yp.to/round3/asconv12.pdf (2016), date accessed: July 28, 2023
7. Fleischmann, E., Forler, C., Lucks, S.: Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In: Workshop on Fast Software Encryption. pp. 196–215 (2012)
8. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41: Submission to the CAESAR Competition. https://competitions.cr.yp.to/round3/deoxysv141.pdf (2016), date accessed: July 28, 2023
9. Jean, Jérémy and Nikolić, Ivica and Peyrin, Thomas: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: Proceedings of the 34th Annual International Cryptology Conference on Advances in Cryptology. pp. 274–288 (2014)
10. Kaliski, B.: PKCS #7: Cryptographic Message Syntax Version 1.5. https://datatracker.ietf.org/doc/html/rfc2315 (1998), date accessed: July 28, 2023
11. Krawczyk, H.: The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In: Proceedings of the 38th Annual International Cryptology Conference on Advances in Cryptology. pp. 310–331 (2001)
12. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology. pp. 31–46 (2002)
13. Peyrin, T., Seurin, Y.: Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In: Proceedings of the 36th Annual International Cryptology Conference on Advances in Cryptology. pp. 33–63 (2016)

[2] Detailed version available: https://arxiv.org/pdf/2309.05769.pdf

[3] This work arose from independent research done in this blog post: https://kenluck2001.github.io/blog_post/probing_real-world_cryptosystems.html