

Tink Outside the Deck

Presenters: Dor Alter

Radboud University, Nijmegen, the Netherlands

End-of-ESCADA Workshop

Date: August 27, 2024

Authenticated Encryption Introduction

Google Tink AEAD and its Security Notion

Jammin cipher

Deck-Tink

Xoofff-Tink

Xoofff-Tink benchmarks

iHub collaboration

Conclusions

Authenticated Encryption 101

- Authenticated encryption is achieved when one not only encrypts the message but also provides means to authenticate its content.
- Combining confidentiality and data integrity.
- It consists of two calls: wrap (encryption) and unwrap (decryption).

AE wrap

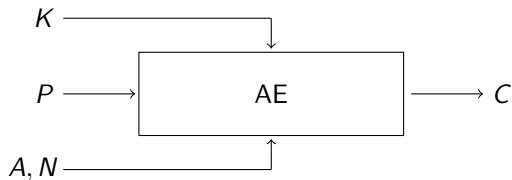


Figure: Illustration of wrap operation of a nonce-based authenticated encryption scheme.

Wrap maps plaintext P to ciphertext C under parameters:

- Associated data A : not encrypted but authenticated.
- Nonce N : not encrypted but authenticated.
- Secret key K .

AE unwrap

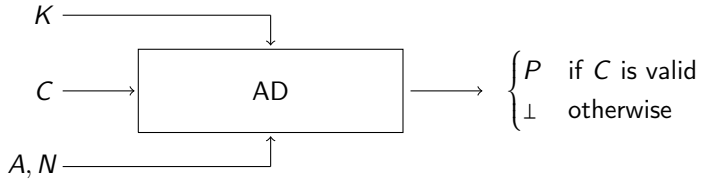


Figure: Illustration of unwrap operation for a nonce-based authenticated encryption scheme.

Unwrap of ciphertext C to plaintext P if C is valid ($\exists(A, N, P)$ that maps to C under K), otherwise, returns an error $\perp$.

Authenticated Encryption Terms

- Authentication sessions: authenticating a message in the context of previously sent ones within the sequence. In such cases, the cryptogram C_3 authenticates the sequence $M_1; M_2; M_3$ (where $M_i = P_i; A_i$).
- Communication sessions: sequences of messages from when a party initialized their communication until they finalized it. If they later decide to continue their communication, a new communication session should be started.
- Online: allows un/wrap operation to be implemented with a constant memory and a single one-direction pass over the plaintext (ciphertext), writing out the result during that pass.

Google Tink

- Tink is an open-source cryptography library implemented and managed by Google.
- The goal is to make the use of cryptographic libraries and schemes "simple to use, secure, and compatible" with different implementations [Goo23].
- One of their tasks is authenticated encryption, for which they provided their Google Tink Authenticated Encryption with Associated Data (AEAD).

Google Tink AEAD

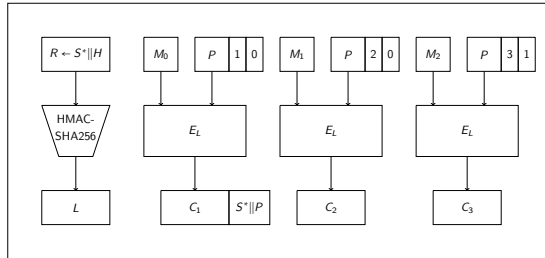


Figure: Tink procedure where P is a 7 byte random nonce prefix, H is the header of the messages, S^* is a 16 bytes random salt, M_i is the i message in the sequence and E is the encryption procedure of GCM using AES.

Google Tink AEAD

- Supports out-of-order messages and missing messages.
- To prove its security, the authors of [HS20] extended an existing notion called nOAE to nOAE2, which supports the additional features of Google Tink AEAD and multi-target security.

nOAE2 history

- The first notion OAE in paper [FFL12].
- In [HRRV15] the OAE notion was corrected but still had some issues so the authors created the OAE2 notion.
- Also, in [HRRV15], the authors introduce the nOAE notion, which is nonce respecting.
- In order to prove Google Tink AEAD notion a new notion was created called nOAE2 in [HS20].
- Most of those notions are referential but not operational.

Jammin Cipher

- Jammin Cipher is a AE notion that originated from the idea that the OAE2 definition is not operational [BDH⁺22].
- It can serve as a security reference for both nonce-enforcing and nonce-misuse-resistant schemes. For OAE2, different ideal-world schemes are required such as nOAE or dOAE.
- It supports unwrap and wrap calls in any order, including bi-directional communication.
- The jammin cipher is specified in an object-oriented manner, making it easier to compare to written code or implement code out of the specification of a scheme.
- We used this notion for our security bound proof.

Deck-Tink

- We design a deck function mode for nonce-based AE called Deck-Tink.
- It only wraps the segment sent and not the whole session.
- Unlike other deck function modes like Deck-Plain, it supports out-of-order messages and missing messages.

Deck-Tink

Parameters: deck function F , expansion length t

Constructor: $init(K, H)$ taking key K and absorbing header H

return instance $Q.F_k(H)$ and finish $\leftarrow \text{False}$

Instance cloner: $inst.clone()$

return new instance $inst'$ with all attributes (K, H) copied from $inst$

Bit String get current squeeze: $inst.getCurrentValue()$

return the squeeze value of the $inst$ without absorbing any input

Deck-Tink

Interface: $Q.TinkWrap(P, \text{counter}, a)$ returning C

if finish **then**

return ERROR: finished instance

$Q' \leftarrow Q.clone$

if $|P| > 0$ **then**

$X \leftarrow P + Q'.F_k(\text{counter}||0)$

$T \leftarrow 0^t + Q'.F_k(X||1)$

$C \leftarrow X||T$

else

$T \leftarrow 0^t + Q'.F_k(\text{counter}||1)$

$C \leftarrow T$

if $a > 0$ **then**

 finish $\leftarrow \text{True}$

return $C, \text{counter}$

```

Interface:  $Q.TinkUnwrap(C, \text{counter}, a)$  returning  $P$  or  $\perp$ 
if finish then
    return ERROR: finished instance
if  $|C| < t$  then return  $\perp$ 
 $Q' \leftarrow Q.clone$ 
if  $|C| > t$  then
    Parse  $C$  in  $X$  and  $T$ 
     $Q' \leftarrow Q'.F_k(\text{counter}||0)$ 
     $Q'' \leftarrow Q'.clone$ 
     $T' \leftarrow 0^t + Q'.F_k(X||1)$ 
else
     $T' \leftarrow 0^t + Q'.F_k(\text{counter}||1)$ 
if  $T' \neq T$  then return  $\perp$ 
if  $a > 0$  then finish  $\leftarrow \text{True}$ 
if  $|C| > t$  then
     $P \leftarrow X + Q''.getCurrentValue()$ 
    return  $P$ 
return  $\emptyset$ 

```

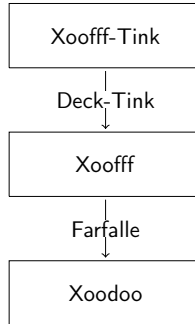
Security

Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish $\text{Deck-Tink}(F, t, l)$ from $\mathcal{J}'$, an ideal cipher based on the jammin cipher with $\text{WrapExpand}(p) = p + t$ in the multi-target settings where there are μ users. Let $N \in \mathbb{N}$ and $N > 0$ represent the amount of queries an adversary performs and let $k \in \mathbb{N}$ be the length of a key. If in the queries of $\mathcal{D}$ the header is a nonce, there exists an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that

$$\Delta_{\mathcal{D}}(\text{Deck-Tink}(F, t, \ell); \mathcal{J}') \leq \frac{q_{\text{unwrap}}}{2^t} + \text{Adv}_F^{\text{prf}}(\mathcal{D}') + \frac{\mu(\mu-1)}{2^{k+1}} + \frac{N\mu}{2^k},$$

$\text{Adv}_F^{\text{prf}} = |\mathbb{P}[K \xleftarrow{\$} \mathcal{K}^u : \mathcal{D}^{F_{K_1}, \dots, F_{K_u}} = 1] - \mathbb{P}[\mathcal{D}^{\mathcal{RO}_1, \dots, \mathcal{RO}_u} = 1]|$ where $\mathcal{RO}$ is a random oracle that takes as input a string sequence.

Xoofff-Tink



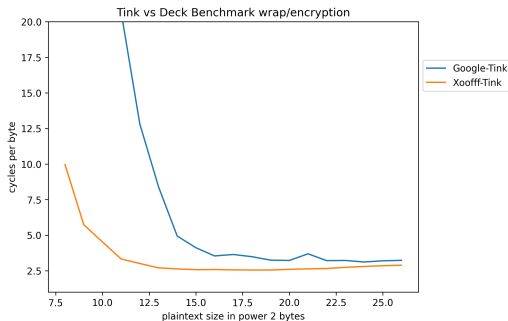
Xooff-Tink implementations

- We implemented Xooff-Tink both in C and Rust.
- The C implementation is built on top of XKCP, therefore, all optimizations are available.
- The Rust implementation is a combination of available primitives plus AVX optimization we implemented.

Xoofff-Tink benchmarks

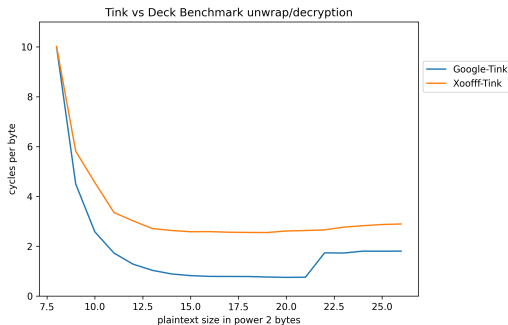
- We compare Xoofff-Tink in C against Google Tink in C++.
- We compare Xoofff-Tink in Rust against AES-GCM, AES-GCM-SIV, chacha20poly1305 and Ascon.

Xoofff-Tink benchmark C wrap



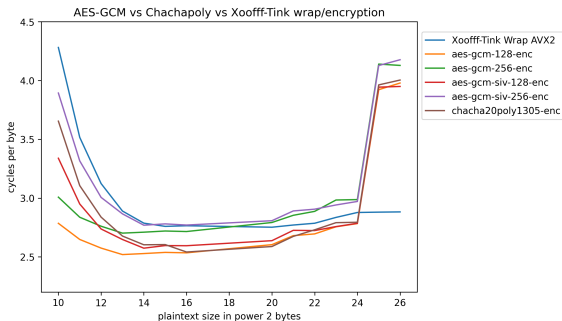
Xoofff-Tink is at least twice as fast for small plaintext (less than 128 Kib) and between 1.10 and 1.42 faster for bigger plaintexts.

Xoofff-Tink benchmark C unwrap



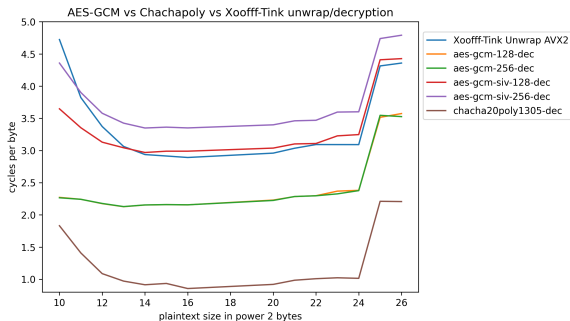
Xoofff-Tink is around 2 to 3 times slower for medium-size plaintext (more than 1 Kib and less than 4 MiB) and between 1.003 and 1.76 slower for other plaintext sizes, this can be explain as Google Tink AEAD uses AES which is implemented in hardware.

Xooff-Tink Rust benchmark vs AES-GCM & chachapoly wrap



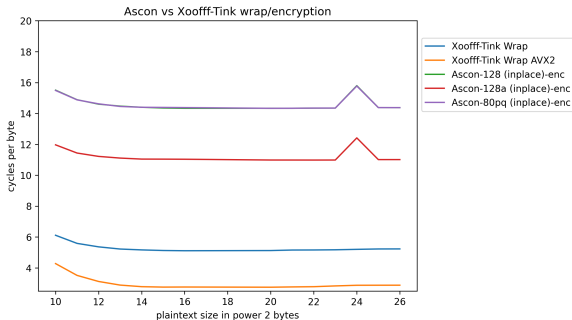
For short plaintexts (less than 1 MiB) Xooff-Tink is around 1.2-1.5 times slower, and for medium, they are around the same time, and for bigger ones (more than 16 MiB) Xooff-Tink is faster.

Xoofff-Tink Rust benchmark vs AES-GCM & chachapoly unwrap



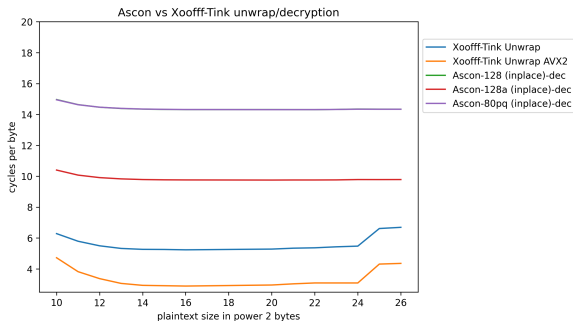
Xoofff-Tink is around 1.04-2.4 slower for messages less than 8 MiB and around 1-1.2 slower for bigger messages, with some cases in which Xoofff-Tink is faster, for example, against AES-GCM-SIV and Xoofff-Tink around 1.2 times faster for messages is more than 16 MiB.

Xooff-Tink Rust benchmark vs Ascon wrap



Xooff-Tink is around 2-5 times faster for the AVX2 implementation and 2-3 times faster for the non-parallelized version.

Xooff-Tink Rust benchmark vs Ascon unwrap



Xooff-Tink is around 2-5 times faster for the AVX2 implementation and 2-3 times faster for the non-parallelized version.

PostGuard

- iHub is a Radboud interdisciplinary research hub on digitalization and society.
- One of their current projects is PostGuard.
- The project aims to create an easy-to-use plugin on top of our traditional email application, which will provide encryption and authentication for the users [BBJ⁺23].
- Their novelty lies in combining their cryptography approach, identity-based encryption with an identity wallet, used by each individual to prove their identity.
- When a sender sends an email, they should specify who can decrypt it by providing the recipient attributes (such as name, email address, or some other attribute).
- On the recipient side, the user can use their attributes from the wallet to decrypt the email.

Integrating Xoofff-Tink in PostGuard

- Currently, PostGuard uses AES-GCM as its AE scheme.
- Our task was to replace AES-GCM with Xoofff-Tink.
- With the help of Leon Botros, we could do that by modifying a few lines in the code, highlighting the ease of use of Xoofff-Tink.
- Although implementation-wise, replacing the AE scheme of PostGuard is easy, it causes a challenging software engineering problem.

Conclusions

- We created a new deck function authenticated encryption mode, Deck-Tink and proved security bound for it.
- We initialized the mode using Xoofff to create Xoofff-Tink and implemented it both in C and Rust.
- We benchmarked our implementation against various AE schemes to provide an ideal of our implementation speed.
- We integrated Xoofff-Tink into the PostGuard project showcasing its ease of use.

Thank you for your attention.

Questions ?

Bibliography I

-  Leon Botros, Merel Brandon, Bart Jacobs, Daniel Ostkamp, Hanna Schraffenberger, and Marloes Venema.
Postguard: Towards easy and secure email communication.
In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023. Association for Computing Machinery.
<https://doi.org/10.1145/3544549.3585622>.
-  Norica Băcuieți, Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer.
Jammin' on the deck.
Cryptology ePrint Archive, Paper 2022/531, 2022.
<https://eprint.iacr.org/2022/531>.

Bibliography II

-  Ewan Fleischmann, Christian Forler, and Stefan Lucks.
Mcoe: A family of almost foolproof on-line authenticated encryption schemes.
In *Fast Software Encryption Workshop*, 2012.
<https://www.iacr.org/archive/fse2012/75490200/75490200.pdf>.
-  Google.
Tink library (code), 2023.
<https://github.com/google/tink>.
-  Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár.
Online authenticated-encryption and its nonce-reuse misuse-resistance.
In *Advances in Cryptology – CRYPTO 2015*, page 493–517, Berlin, Heidelberg, 2015.
Springer-Verlag.
<https://eprint.iacr.org/2015/189.pdf>.

Bibliography III

-  Viet Tung Hoang and Yaobin Shen.
Security of streaming encryption in google's tink library.
In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 243–262, New York, NY, USA, 2020. Association for Computing Machinery.
<https://eprint.iacr.org/2020/1019.pdf>.