# Incrementality and deck functions

## Simple protocols and efficient constructions in symmetric cryptography

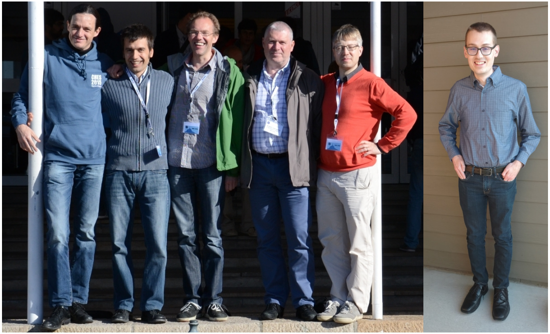
Gilles Van Assche[1]

[1]STMicroelectronics

FOSDEM 2020
Brussels, Belgium, February 1, 2020

# Joint work with the Keccak team

- Guido Bertoni
  Italy
- Joan Daemen
  The Netherlands
- Seth Hoffert
  USA
- Michaël Peeters
  Belgium
- Gilles Van Assche
  Belgium
- Ronny Van Keer
  Belgium



https://keccak.team/

# What we do: permutation-based crypto

Hashing

- Keccak, SHA-3, SHAKE, cSHAKE, ParallelHash, ...
- **KangarooTwelve** [ultra-fast, most third-party cryptanalysis to date]

Symmetric-key encryption and/or authentication

- Keyed duplex: Ketje, Keyak, Xoodyak
- Farfalle: Kravatte, **Xoofff** [fast from small to big platforms]

Re-thinking symmetric crypto

- Deck function interface
- Simpler authenticated encryption schemes

# What we do: permutation-based crypto

Hashing

- KECCAK, SHA-3, SHAKE, cSHAKE, ParallelHash, ...
- **KANGAROOTWELVE** [ultra-fast, most third-party cryptanalysis to date]

Symmetric-key encryption and/or authentication

- Keyed duplex: KETJE, KEYAK, XOODYAK
- Farfalle: KRAVATTE, **XOOFFF** [fast from small to big platforms]

Re-thinking symmetric crypto

- Deck function interface
- Simpler authenticated encryption schemes

# What we do: permutation-based crypto

Hashing

- Keccak, SHA-3, SHAKE, cSHAKE, ParallelHash, ...
- **KangarooTwelve** [ultra-fast, most third-party cryptanalysis to date]

Symmetric-key encryption and/or authentication

- Keyed duplex: Ketje, Keyak, Xoodyak
- Farfalle: Kravatte, **Xoofff** [fast from small to big platforms]

Re-thinking symmetric crypto

- Deck function interface
- Simpler authenticated encryption schemes

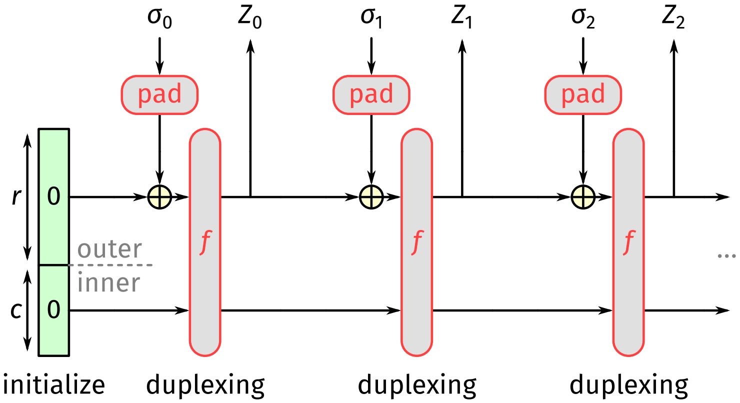
# Outline

1. Why is incrementality useful? (Example: Disco)
2. What are deck functions?
3. How to use a deck function?
4. How to build a fast deck function?

# Outline

# Duplex object

Duplex object = sponge function with incrementality



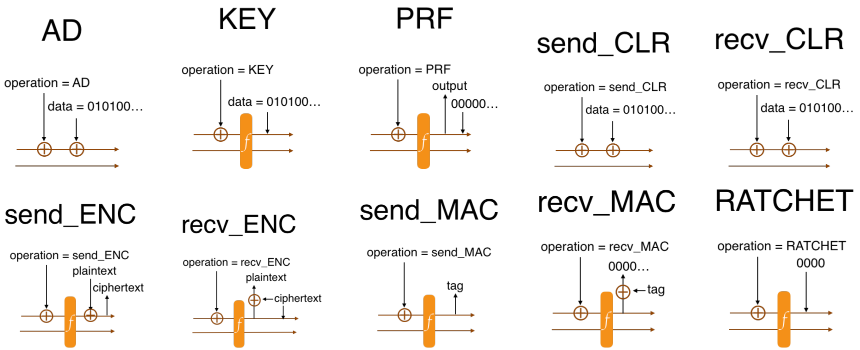

[Selected Areas in Cryptography 2011]

# STROBE

- Layer above a duplex object
  - compliant with **cSHAKE** [NIST SP 800-185]
- Safe and easy syntax, to achieve, e.g.,
  - secure channels
  - hashing of protocol transcripts
  - signatures over a complete session
- Very compact implementation

[Mike Hamburg, Real World Crypto 2017]

# STROBE functions




figures courtesy of David Wong

# Example: protocol

**KEY**(shared key *K*)

**AD**[nonce](seq. number *i*)

**AD**[auth-data]($IP_1 || IP_2$)

**send_ENC**("GET file")

**send_MAC**(128 bits)

**recv_ENC**(ciphertext buffer)

**recv_MAC**(128 bits)

# Example: protocol

**KEY**(shared key $K$) set key $K$, empty context $X$

**AD**[nonce](seq. number $i$)

**AD**[auth-data]($IP_1 || IP_2$)

**send_ENC**("GET file")

**send_MAC**(128 bits)

**recv_ENC**(ciphertext buffer)

**recv_MAC**(128 bits)

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($\mathrm{IP}_1 \| \| \mathrm{IP}_2$) | |
| **send_ENC**("GET file") | |
| **send_MAC**(128 bits) | |
| **recv_ENC**(ciphertext buffer) | |
| **recv_MAC**(128 bits) | |

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($IP_1 \| \| IP_2$) | $X \leftarrow (IP_1 \| \| IP_2) \circ$ "auth-data" $\circ X$ |
| **send_ENC**("GET file") | |
| **send_MAC**(128 bits) | |
| **recv_ENC**(ciphertext buffer) | |
| **recv_MAC**(128 bits) | |

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($IP_1 \vert\vert IP_2$) | $X \leftarrow (IP_1 \vert\vert IP_2) \circ$ "auth-data" $\circ X$ |
| **send_ENC**("GET file") | ciphertext $=$ $\mathrm{enc}_{K,X}$("GET file")<br>$X \leftarrow$ "GET file" $\circ X$ |
| **send_MAC**(128 bits) | |
| **recv_ENC**(ciphertext buffer) | |
| **recv_MAC**(128 bits) | |

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($IP_1 \| IP_2$) | $X \leftarrow (IP_1 \| IP_2) \circ$ "auth-data" $\circ X$ |
| **send_ENC**("GET file") | ciphertext $=$ $\mathrm{enc}_{K,X}$("GET file")<br>$X \leftarrow$ "GET file" $\circ X$ |
| **send_MAC**(128 bits) | $\mathrm{MAC}_K(X)$ |
| **recv_ENC**(ciphertext buffer) | |
| **recv_MAC**(128 bits) | |

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($IP_1 \| \| IP_2$) | $X \leftarrow (IP_1 \| \| IP_2) \circ$ "auth-data" $\circ X$ |
| **send_ENC**("GET file") | ciphertext $=$ $\mathrm{enc}_{K,X}$("GET file")<br>$X \leftarrow$ "GET file" $\circ X$ |
| **send_MAC**(128 bits) | $\mathrm{MAC}_K(X)$ |
| **recv_ENC**(ciphertext buffer) | plaintext $=$ $\mathrm{dec}_{K,X}$(ciphertext)<br>$X \leftarrow$ plaintext $\circ X$ |
| **recv_MAC**(128 bits) | |

# Example: protocol

| | |
|---|---|
| **KEY**(shared key $K$) | set key $K$, empty context $X$ |
| **AD**[nonce](seq. number $i$) | $X \leftarrow (i) \circ$ "nonce" $\circ X$ |
| **AD**[auth-data]($IP_1 \| \| IP_2$) | $X \leftarrow (IP_1 \| \| IP_2) \circ$ "auth-data" $\circ X$ |
| **send_ENC**("GET file") | ciphertext $=$ $\mathrm{enc}_{K,X}$("GET file")<br>$X \leftarrow$ "GET file" $\circ X$ |
| **send_MAC**(128 bits) | $\mathrm{MAC}_K(X)$ |
| **recv_ENC**(ciphertext buffer) | plaintext $=$ $\mathrm{dec}_{K,X}$(ciphertext)<br>$X \leftarrow$ plaintext $\circ X$ |
| **recv_MAC**(128 bits) | check that MAC $= \mathrm{MAC}_K(X)$ |

# The Noise protocol framework

Framework for crypto protocols based on Diffie-Hellman

- public-key handshake mechanism
- secret-key encryption and authentication
- used in WhatsApp, WireGuard, ...

[Trevor Perrin, Real World Crypto 2018]

# Inside Noise

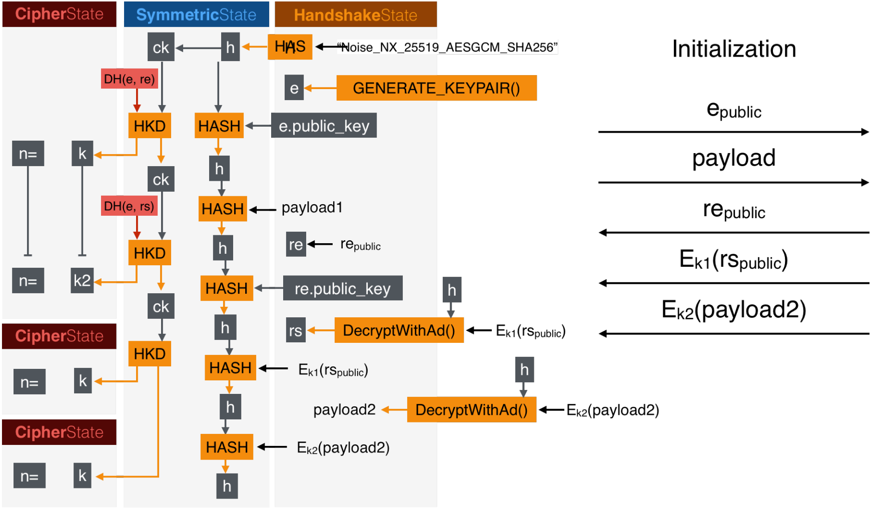



figure courtesy of David Wong

# Disco

Disco = Noise + STROBE

[David Wong, Black Hat Europe 2017]

# Inside Disco

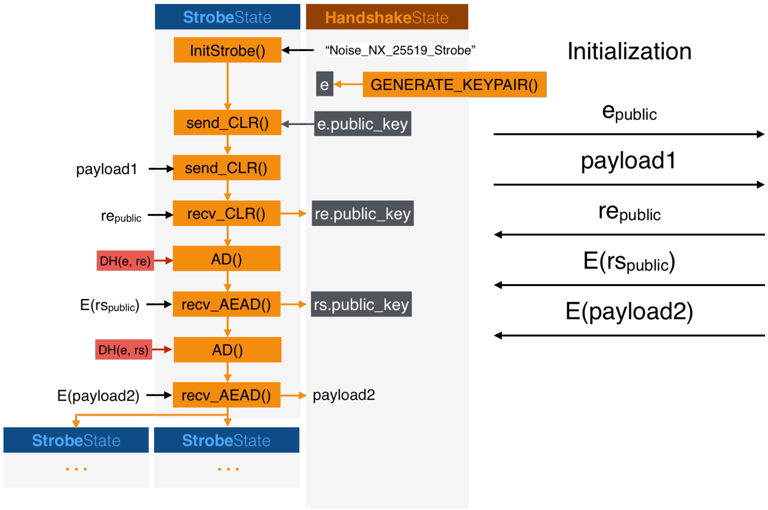



figure courtesy of David Wong

# Duplex object inside Disco

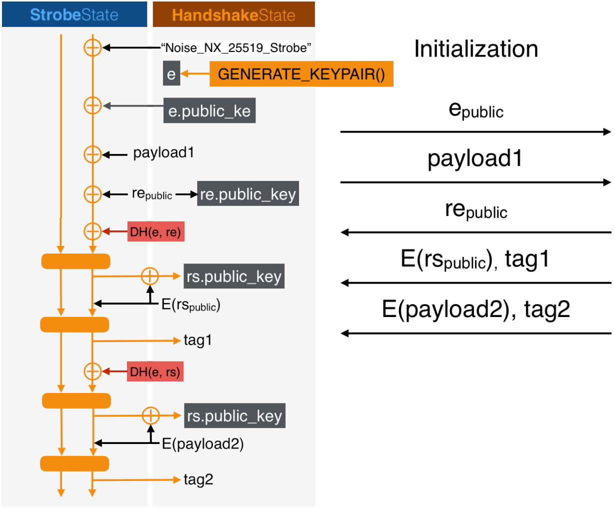



figure courtesy of David Wong

# Implementation complexity

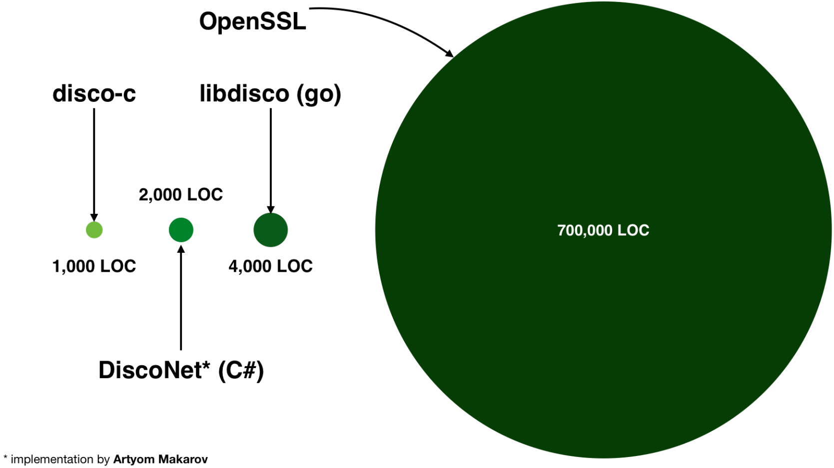



figure courtesy of David Wong

# Outline

# Definition of a deck function

## A deck function $F_K$

$$Z = 0^n + F_K\left(X^{(m)} \circ \cdots \circ X^{(1)}\right) \ll q$$

doubly extendable cryptographic keyed function

# Definition of a deck function

## A deck function $F_K$

$$Z = 0^n + F_K\left(X^{(m)} \circ \cdots \circ X^{(1)}\right) \ll q$$

- Input: sequence of strings $X^{(m)} \circ \cdots \circ X^{(1)}$

# Definition of a deck function

## A deck function $F_K$

$$Z = 0^n + F_K\left(X^{(m)} \circ \cdots \circ X^{(1)}\right) \ll q$$

- Input: sequence of strings $X^{(m)} \circ \cdots \circ X^{(1)}$
- Output: potentially infinite output
  - **pseudo-random function of the input**
  - taking $n$ bits starting from offset $q$

# Definition of a deck function

## A deck function $F_K$

$$Z = 0^n + F_K\left(X^{(m)} \circ \cdots \circ X^{(1)}\right) \ll q$$

## Efficient incrementality

- Extendable input
  1. Compute $F_K(X)$
  2. Compute $F_K(Y \circ X)$: cost independent of $X$

# Definition of a deck function

## A deck function $F_K$

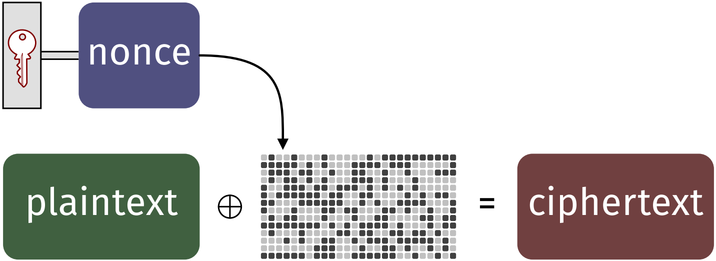
$$Z = 0^n + F_K\left(X^{(m)} \circ \cdots \circ X^{(1)}\right) \ll q$$

## Efficient incrementality

- Extendable input
  1. Compute $F_K(X)$
  2. Compute $F_K(Y \circ X)$: cost independent of $X$
- Extendable output
  1. Request $n_1$ bits from offset 0
  2. Request $n_2$ bits from offset $n_1$: cost independent of $n_1$

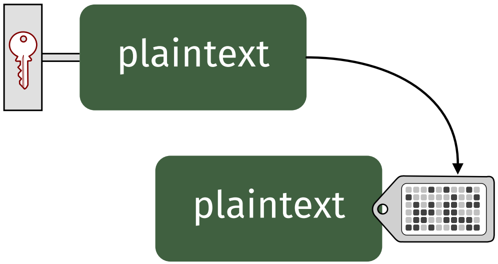
# Outline

# Stream cipher: short input, long output



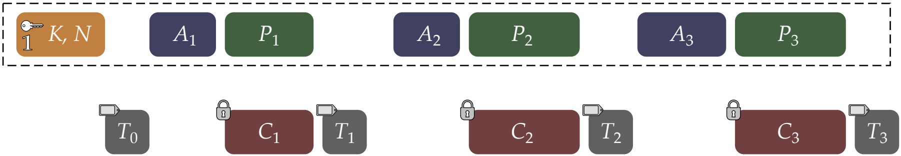

$$C \leftarrow P + F_K(N)$$

# MAC: long input, short output




$$T \leftarrow 0^t + F_K(P)$$

# Deck-SANE: session-supporting and nonce-based




[Xoodoo Cookbook, IACR ePrint 2018/767]

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$

$T_0 \leftarrow 0^t + F_K(N)$

**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)

$C_1 \leftarrow P_1 + F_K(N) \ll t$

$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$

**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)

$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$

$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$

**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Deck-SANE: session-supporting and nonce-based

**Initialize** session with nonce $N$
$T_0 \leftarrow 0^t + F_K(N)$
**return** startup tag $T_0$

**Encipher** message 1 (metadata $A_1$, plaintext $P_1$)
$C_1 \leftarrow P_1 + F_K(N) \ll t$
$T_1 \leftarrow 0^t + F_K(C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_1$, tag $T_1$)

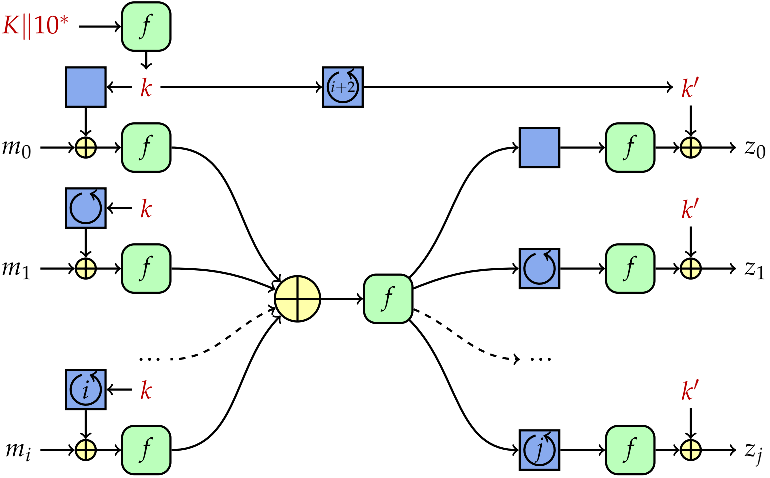
**Encipher** message 2 (metadata $A_2$, plaintext $P_2$)
$C_2 \leftarrow P_2 + F_K(C_1 \circ A_1 \circ N) \ll t$
$T_2 \leftarrow 0^t + F_K(C_2 \circ A_2 \circ C_1 \circ A_1 \circ N)$
**return** (ciphertext $C_2$, tag $T_2$)

# Other deck-based modes

- Deck-SANSE: synthetic nonce
- Deck-WBC: wide block cipher

# Outline

# Farfalle




[Fast Software Encryption 2018]

# KRAVATTE and XOOFFF

KRAVATTE [FSE 2018]

- $f = \text{KECCAK-}p[1600, n_r = 6]$
- Input mask rolling with LFSR, state rolling with NLFSR
- Target security: $\geq$ 128 bits (including post-quantum)

XOOFFF [FSE 2019]

- $f = \text{XOODOO}[6]$
  384-bit permutation $4 \times 3 \times 32$ bits
- Target security: $\geq$ 128 bits ($\geq$ 96 bits post-quantum)

# Kravatte and Xoofff

Kravatte [FSE 2018]

- $f = \text{Keccak-}p[1600, n_r = 6]$
- Input mask rolling with LFSR, state rolling with NLFSR
- Target security: $\geq$ 128 bits (including post-quantum)

Xoofff [FSE 2019]

- $f = \text{Xoodoo}[6]$
  384-bit permutation $4 \times 3 \times 32$ bits
- Target security: $\geq$ 128 bits ($\geq$ 96 bits post-quantum)

# KRAVATTE performance

| **KRAVATTE** | | |
|---|---|---|
| granularity | 200 | bytes |
| MAC computation use case: | | |
| long inputs | 0.64 | cycles/byte |
| Stream encryption use case: | | |
| long outputs | 0.63 | cycles/byte |
| AES-128 counter mode | 0.65 | cycles/byte |

Intel® Core™ i5-6500 (Skylake), single core, Turbo Boost disabled (256-bit SIMD)

`https://github.com/XKCP/XKCP`

# Xoofff performance

| Xoofff | | |
|---|---|---|
| granularity | 48 | bytes |
| MAC computation use case: | | |
| long inputs | 26.0 | cycles/byte |
| Stream encryption use case: | | |
| long outputs | 25.1 | cycles/byte |
| AES-128 counter mode | 121.4 | cycles/byte |

ARM® Cortex-M0

https://github.com/XKCP/XKCP

# XOOFFF performance

| | | |
|---|---|---|
| **XOOFFF** | | |
| granularity | 48 | bytes |
| MAC computation use case: | | |
| long inputs | 8.8 | cycles/byte |
| Stream encryption use case: | | |
| long outputs | 8.1 | cycles/byte |
| AES-128 counter mode | 33.2 | cycles/byte |

ARM® Cortex-M3

https://github.com/XKCP/XKCP

# XOOFFF performance

| XOOFFF | | |
|---|---|---|
| granularity | 48 | bytes |
| MAC computation use case: | | |
| long inputs | 0.90 | cycles/byte |
| Stream encryption use case: | | |
| long outputs | 0.94 | cycles/byte |
| AES-128 counter mode | 0.65 | cycles/byte |

Intel® Core™ i5-6500 (Skylake), single core, Turbo Boost disabled (256-bit SIMD)

https://github.com/XKCP/XKCP

# Xoofff performance

| **Xoofff** | | |
|---|---|---|
| granularity | 48 | bytes |
| MAC computation use case: | | |
| long inputs | 0.40 | cycles/byte |
| Stream encryption use case: | | |
| long outputs | 0.51 | cycles/byte |
| AES-128 counter mode | 0.65 | cycles/byte |

Intel® Core™ i7-7800X (SkylakeX), single core, Turbo Boost disabled (512-bit SIMD)

https://github.com/XKCP/XKCP

# Conclusions

Incrementality in symmetric crypto

- can simplify protocols (e.g., Disco)
- can make modes more natural (e.g., session-based AE)

The deck function interface

- is a way to define incrementality in keyed operations
- can be efficiently implemented with the Farfalle construction (e.g., Kravatte and Xoofff)

# Conclusions

Incrementality in symmetric crypto

- can simplify protocols (e.g., Disco)
- can make modes more natural (e.g., session-based AE)

The deck function interface

- is a way to define incrementality in keyed operations
- can be efficiently implemented with the Farfalle construction (e.g., KRAVATTE and XOOFFF)

# Any questions?

## Thanks for your attention!

For more permutation-based crypto, see you at
PBC 2020, co-located with Eurocrypt

https://permutationbasedcrypto.org/2020/