Generic attacks based on functional graphs

Rachelle Heim Boissier

Université de Versailles Saint-Quentin-en-Yvelines

Séminaire de Cryptographie de Rennes

Introduction

This talk is about:

- **Symmetric** cryptanalysis
- Using **random function** graphs statistics in **generic** attacks ...
- ... against a variety of **iterated** constructions:
  - Hash functions [Floyd]
  - Message authenticated codes (MAC) modes [LPW13]
  - Authenticated encryption (AE) modes [GHKR23]

Plan

1. Random function statistics
2. Memory-negligible collision search
3. State recovery attack against HMAC
4. Generic attack against AEAD modes

Random functions

Definition:

$\mathscr{F}_N$ is the set of functions which map a finite set of size $N \in \mathbb{N}^*$ to itself.

Our main focus: the **graph** of f (randomly drawn) in $\mathscr{F}_N$

Functional graph

The **graph of** f, denoted by $G(f)$, is a **directed graph** such that a vertex goes from node i to node j if and only if $f(i) = j$.

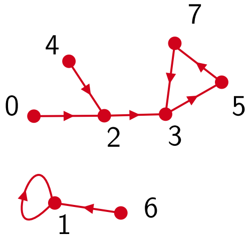
Functional graphs : an example

Functional graph

The **graph of** f, denoted by $G(f)$, is a **directed graph** such that a vertex goes from node i to node j if and only if $f(i) = j$.

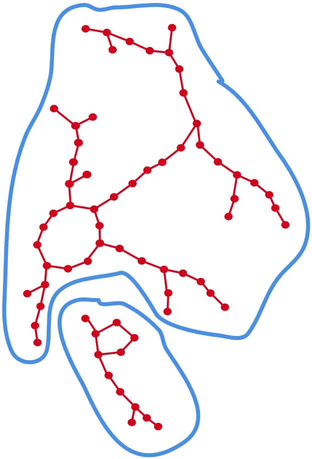
$$f : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$$

$$\begin{cases} 0 \longmapsto 2 \\ 1 \longmapsto 1 \\ 2 \longmapsto 3 \\ 3 \longmapsto 5 \\ 4 \longmapsto 2 \\ 5 \longmapsto 7 \\ 6 \longmapsto 1 \\ 7 \longmapsto 3 \end{cases}$$



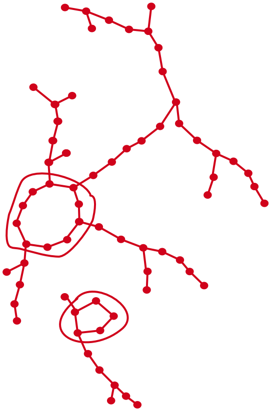
Functional graphs : important definitions (1)

- The graph of f can be seen as a set of **connected components**.



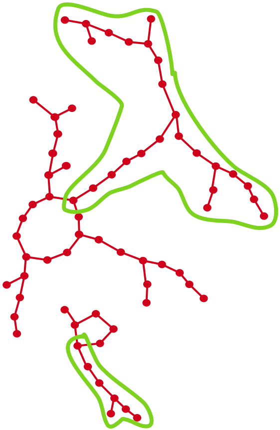
Functional graphs : important definitions (1)

- The graph of f can be seen as a set of **connected components**.
- Each connected component has a unique **cycle**.



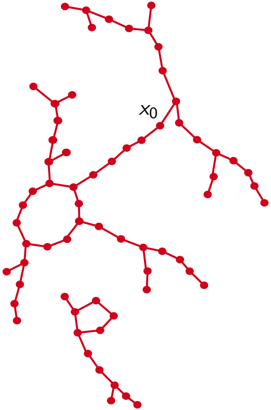
Functional graphs : important definitions (1)

- The graph of f can be seen as a set of **connected components**.
- Each connected component has a unique **cycle**.
- Each cyclic node is the root of a **tree**.



Functional graphs : important definitions (2)

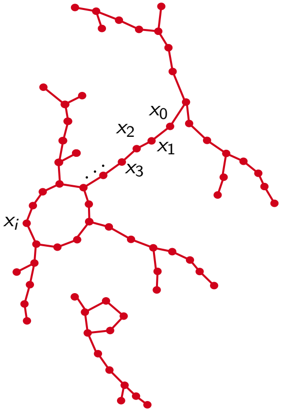
For any $x_0 \in G(f)$



Functional graphs : important definitions (2)

For any $x_0 \in G(f)$

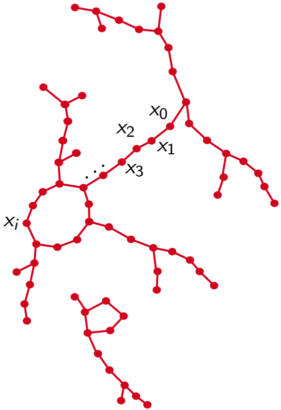
- $(x_i := f^i(x_0))_{i \in \mathbb{N}}$



Functional graphs : important definitions (2)

For any $x_0 \in G(f)$

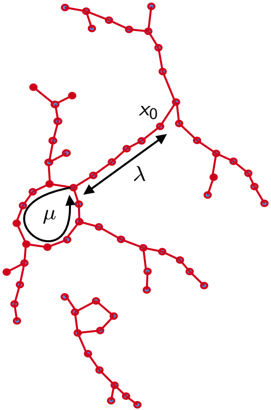
- $(x_i := f^i(x_0))_{i \in \mathbb{N}}$ is eventually periodic.
- $(x_i)_{i \in \mathbb{N}}$ graphically corresponds to a path linked to a **cycle**



Functional graphs : important definitions (2)

For any $x_0 \in G(f)$

- $(x_i := f^i(x_0))_{i \in \mathbb{N}}$ is eventually periodic.
- $(x_i)_{i \in \mathbb{N}}$ graphically corresponds to a path linked to a **cycle**



We define

- *Tail length.*

  $\lambda(x_0)$ is the smallest i s.t. x_i is in the cycle.
- *Cycle length.*

  $\mu(x_0)$ number of nodes in the cycle.

Random function graphs : important statistics

For f randomly drawn in $\mathfrak{F}_N$:

- Expected size of f's largest component : $0.76N$
- Expected size of f's largest tree : $0.5N$
- For x a random node:
  - Expected value of its tail length $\lambda(x)$: $\sqrt{\pi N/8}$
  - Expected value of its cycle length $\mu(x)$: $\sqrt{\pi N/8}$

Random function graphs : important statistics

For f randomly drawn in $\mathfrak{F}_N$:

- Expected size of f's largest component : $0.76N$
- Expected size of f's largest tree : $0.5N$
- For x a random node:
  - Expected value of its tail length $\lambda(x)$: $\sqrt{\pi N/8}$
  - Expected value of its cycle length $\mu(x)$: $\sqrt{\pi N/8}$

Many more statistics are known and used in generic attacks.

[DeLaurentis88] [FO89] [Harris60] ...

Plan

Cryptographic hash functions

Definition. A **cryptographic hash function** is a function $H : \mathbb{F}_2^* \to \mathbb{F}_2^n$ such that the following properties are verified

- *Preimage resistance.* Given $h \in \mathbb{F}_2^d$, it is difficult to find $m \in \mathbb{F}_2^*$ s.t. $H(m) = h$.
- *Second preimage resistance.* Given $m \in \mathbb{F}_2^*$, it is difficult to find $m' \neq m$ s.t. $H(m') = H(m)$.
- *Collision resistance.* It is difficult to find (m, m'), $m \neq m'$ such that $H(m) = H(m')$.

Cryptographic hash functions

Definition. A **cryptographic hash function** is a function $H : \mathbb{F}_2^* \to \mathbb{F}_2^n$ such that the following properties are verified

- *Preimage resistance.* Given $h \in \mathbb{F}_2^d$, it is difficult to find $m \in \mathbb{F}_2^*$ s.t. $H(m) = h$.
- *Second preimage resistance.* Given $m \in \mathbb{F}_2^*$, it is difficult to find $m' \neq m$ s.t. $H(m') = H(m)$.
- *Collision resistance.* It is difficult to find (m, m'), $m \neq m'$ such that $H(m) = H(m')$.

Generic collision attack: Compute $H(m)$ for $O(2^{n/2})$ messages m, store them in a list, sort it. WHP, you find a collision (Birthday paradox).

Cryptographic hash functions

Definition. A **cryptographic hash function** is a function $H : \mathbb{F}_2^* \to \mathbb{F}_2^n$ such that the following properties are verified

- *Preimage resistance.* Given $h \in \mathbb{F}_2^d$, it is difficult to find $m \in \mathbb{F}_2^*$ s.t. $H(m) = h$.
- *Second preimage resistance.* Given $m \in \mathbb{F}_2^*$, it is difficult to find $m' \neq m$ s.t. $H(m') = H(m)$.
- *Collision resistance.* It is difficult to find (m, m'), $m \neq m'$ such that $H(m) = H(m')$.

Generic collision attack: Compute $H(m)$ for $O(2^{n/2})$ messages m, store them in a list, sort it. WHP, you find a collision (Birthday paradox).

Problem: Memory complexity is high.

Cryptographic hash functions

Definition. A **cryptographic hash function** is a function $H : \mathbb{F}_2^* \to \mathbb{F}_2^n$ such that the following properties are verified

- *Preimage resistance.* Given $h \in \mathbb{F}_2^d$, it is difficult to find $m \in \mathbb{F}_2^*$ s.t. $H(m) = h$.
- *Second preimage resistance.* Given $m \in \mathbb{F}_2^*$, it is difficult to find $m' \neq m$ s.t. $H(m') = H(m)$.
- *Collision resistance.* It is difficult to find (m, m'), $m \neq m'$ such that $H(m) = H(m')$.

Generic collision attack: Compute $H(m)$ for $O(2^{n/2})$ messages m, store them in a list, sort it. WHP, you find a collision (Birthday paradox).

Problem: Memory complexity is high.

Solution: a generic memory-negligible collision attack using functional graphs.

A memory-negligible collision attack on H

Let $f \in \mathfrak{F}_{2^n}$ be defined as

$$f : \begin{array}{ccc} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ x & \longmapsto & H(x) \end{array}$$

Step 1. **Floyd's cycle finding algorithm** allows to recover a node x_c in a cycle of f's graph

- in **time** $O(2^{n/2})$;
- using a **negligible amount of memory**.

Step 2. Using x_c, it is easy to

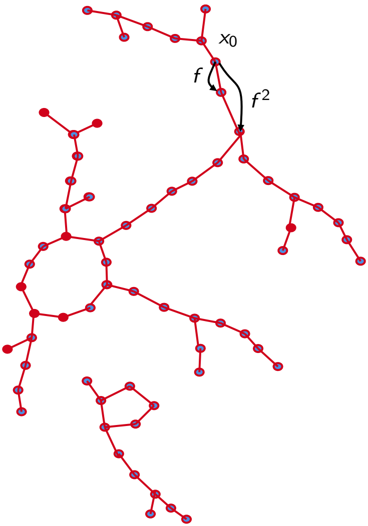
- recover the cycle's length μ
- find a collision on f, and thus on H,

in time $O(2^{n/2})$ and with negligible memory.

Floyd's cycle finding algorithm

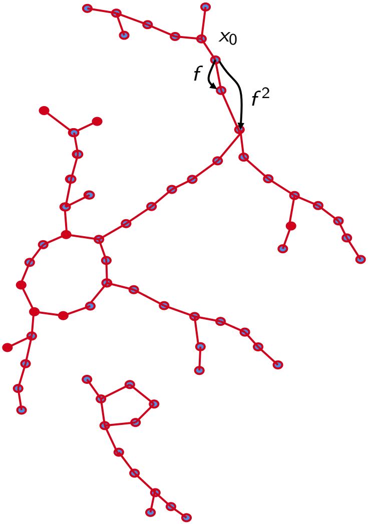
parameters : $f \in \mathfrak{F}_{2^n}$

$x_0 \leftarrow_R \mathbb{F}_2^n$
$turtle, hare \leftarrow x_0, x_0$
for $i = 1$ to $2^n - 1$ **do**
    $turtle \leftarrow f(turtle)$
    $hare \leftarrow f^2(hare)$
    if $turtle = hare$ **then**
        return $turtle$
    end if
end for



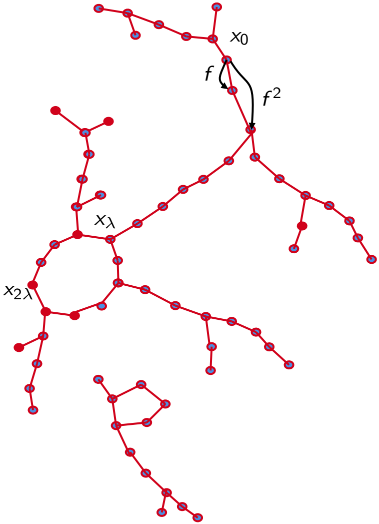
Floyd's cycle finding algorithm

λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.



Floyd's cycle finding algorithm

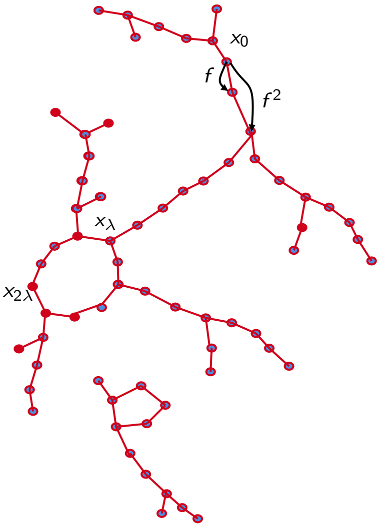
λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.



Floyd's cycle finding algorithm

λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.

Let $d_\lambda = dist(x_\lambda, x_{2\lambda})$.



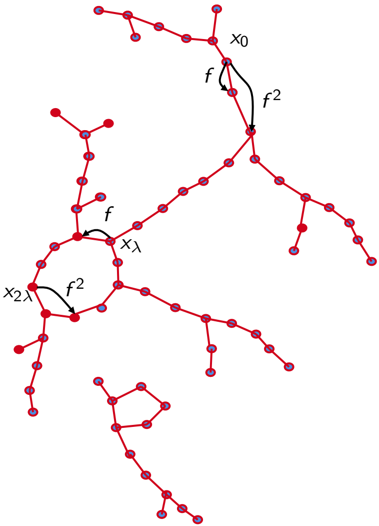
Floyd's cycle finding algorithm

λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.

Let $d_\lambda = dist(x_\lambda, x_{2\lambda})$.

Then

$$dist(f(x_\lambda), f^2(x_{2\lambda})) = d_\lambda + 1 \mod \mu \,.$$



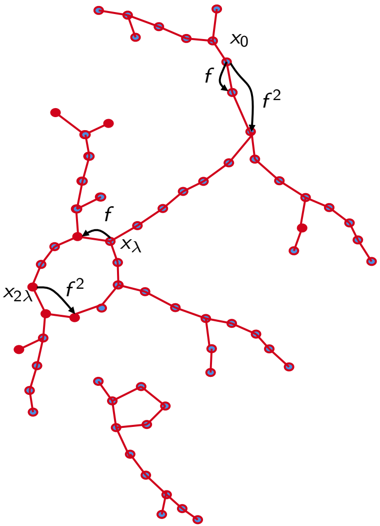
Floyd's cycle finding algorithm

λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.

Let $d_\lambda = dist(x_\lambda, x_{2\lambda})$.

Then

$$dist(x_{\lambda+k}, x_{2\lambda+2k}) = d_\lambda + k \mod \mu\,.$$



Floyd's cycle finding algorithm

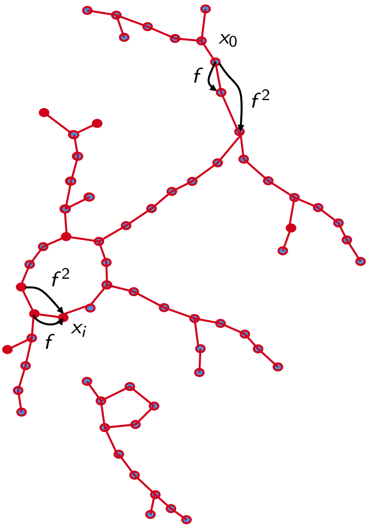
λ is the smallest integer j such that $x_j = f^j(x_0)$ is in the cycle.

Let $d_\lambda = dist(x_\lambda, x_{2\lambda})$.

Then

$$dist(x_{\lambda+k}, x_{2\lambda+2k}) = d_\lambda + k \mod \mu\,.$$

Thus, after at most μ tries, the algorithm finds i such that $x_i = x_{2i}$, and x_i is in the cycle.



Floyd's cycle finding algorithm

We need

- at most λ **for** loops to reach the cycle
- at most μ **for** loops to detect it

Functional graphs statistics.
f behaves like a RF, x_0 is randomly drawn. We thus expect

- $\lambda = O(2^{n/2})$
- $\mu = O(2^{n/2})$

---

parameters : $f \in \mathfrak{F}_{2^n}$

$x_0 \leftarrow_R \mathbb{F}_2^n$
turtle, *hare* $\leftarrow x_0, x_0$
for $i = 1$ to $2^n - 1$ **do**
    turtle $\leftarrow f(\textit{turtle})$
    hare $\leftarrow f^2(\textit{hare})$
    if *turtle* $=$ *hare* **then**
        return *turtle*
    end if
end for

Floyd's cycle finding algorithm

We need

- at most λ **for** loops to reach the cycle
- at most μ **for** loops to detect it

Functional graphs statistics.
f behaves like a RF, x_0 is randomly drawn. We thus expect

- $\lambda = O(2^{n/2})$
- $\mu = O(2^{n/2})$

parameters : $f \in \mathfrak{F}_{2^n}$

```
x_0 ←_R F_2^n
turtle, hare ← x_0, x_0
for i = 1 to 2^n − 1 do
    turtle ← f(turtle)
    hare ← f^2(hare)
    if turtle = hare then
        return turtle
    end if
end for
```

One can show that Floyd's time complexity is in $O(2^{n/2})$

Floyd's cycle finding algorithm

We need

- at most λ **for** loops to reach the cycle
- at most μ **for** loops to detect it

Functional graphs statistics.
f behaves like a RF, x_0 is randomly drawn. We thus expect

- $\lambda = O(2^{n/2})$
- $\mu = O(2^{n/2})$

```
parameters : f ∈ 𝔉_{2^n}
x_0 ←_R F_2^n
turtle, hare ← x_0, x_0
for i = 1 to 2^n − 1 do
    turtle ← f(turtle)
    hare ← f^2(hare)
    if turtle = hare then
        return turtle
    end if
end for
```

One can show that Floyd's time complexity is in $O(2^{n/2})$

... and it is straightforward that the memory complexity is negligible.

Plan

Message Authenticated Code (MAC) algorithms

Definition. A **Message Authenticated Code algorithm** is a **symmetric algorithm** that takes as input a **secret key** k and an arbitrary length message m to produce a fixed length **tag** that guarantees the **integrity** of the message.

Generation and verification procedure. Alice and Bob share a secret k.

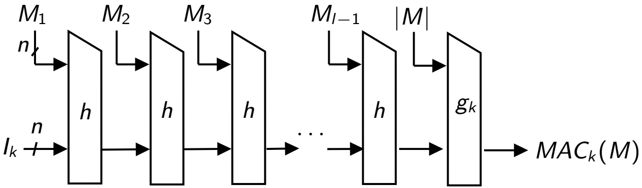
1. Alice (sender)
   - Using the secret and a MAC algorithm MAC, Alice computes a tag $T = MAC_k(M)$.
   - Alice sends (M, T) through an unsafe communication channel.
2. Bob (receiver)
   - Bob receives (M', T').
   - Bob computes $MAC_k(M')$. If it is equal to T', then he concludes that M' is the message sent by Alice. Otherwise, he discards (M', T').

Hash-based MACs

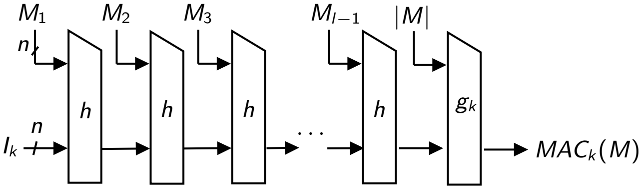
Hash functions can be used to build MACs.

- A good hash function behaves like a **random oracle**.
- It is easy to build a secure MAC with a RO.
- With a real hash function, it is essential to study **generic attacks**.
- There is a **great number of papers** which analyse the generic security of HMACs.

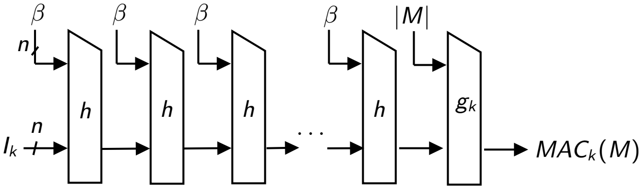
In this presentation. We present a 2013 **state recovery attack** by Leurent, Peyrin and Wang on the family of hash-based MACs with the following structure (e.g. HMAC [BCK96]).



State-recovery attack on HMAC [LPW13]

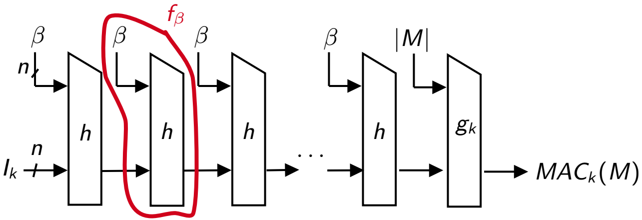


State-recovery attack on HMAC [LPW13]



- Let β be a random fixed block, and consider the message $M = \underbrace{\beta || \cdots || \beta}_{\ell}$.

State-recovery attack on HMAC [LPW13]

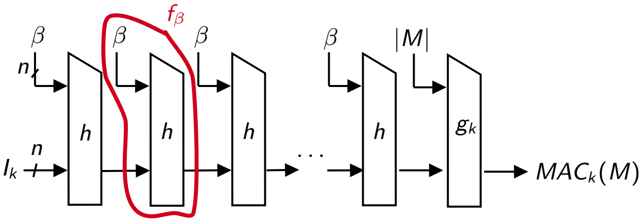


- Let β be a random fixed block, and consider the message $M = \underbrace{\beta || \cdots || \beta}_{\ell}$.
- We expect

$$\begin{array}{rcccl} f_\beta & : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ & & x & \longmapsto & h(\beta||x) \end{array}$$

to behave as a random function.

State-recovery attack on HMAC [LPW13]

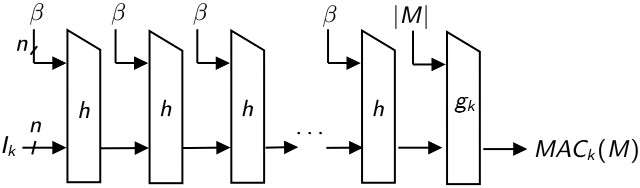


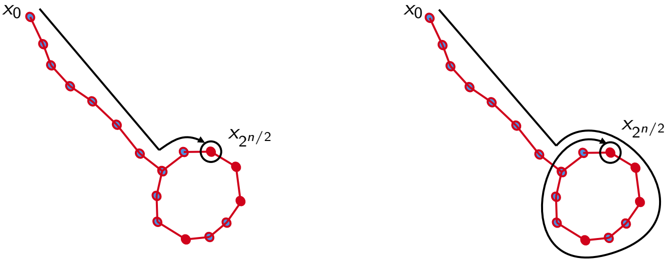
- Let β be a random fixed block, and consider the message $M = \underbrace{\beta || \cdots || \beta}_{\ell}$.
- We expect

$$\begin{array}{cccc} f_\beta & : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ & & x & \longmapsto & h(\beta || x) \end{array}$$

  to behave as a random function.
- Since the main component has size $0.76 \cdot 2^n$, $x_0 = I_k$ is in it WHP.

Idea 1: use two messages which reach the same state

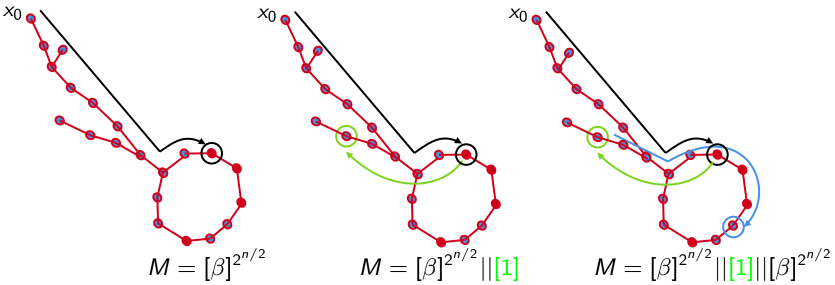




$M = [\beta]^{2^{n/2}}$ $\quad$ $M = [\beta]^{2^{n/2}+\mu}$

Two issues: Message size + the state is not recovered

Idea 2: reach the cycle twice

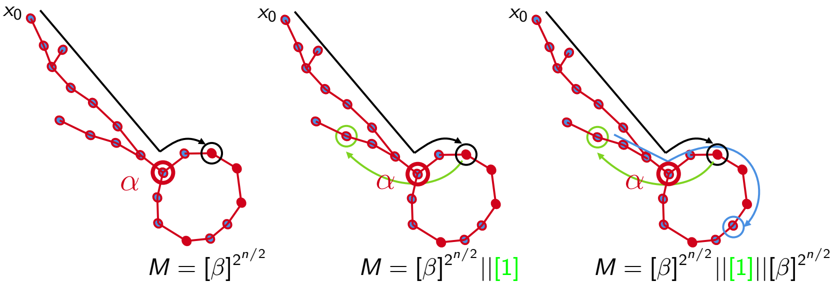


- $M_1 = [\beta]^{2^{n/2}+\mu} || [1] || [\beta]^{2^{n/2}}$
- $M_2 = [\beta]^{2^{n/2}} || [1] || [\beta]^{2^{n/2}+\mu}$

reach the same state with constant probability.

Still no state recovery.

Idea 2: reach the cycle twice



- $M_1 = [\beta]^{2^{n/2}+\mu} || [1] || [\beta]^{2^{n/2}}$
- $M_2 = [\beta]^{2^{n/2}} || [1] || [\beta]^{2^{n/2}+\mu}$

reach the same state with constant probability.

Still no state recovery. Solution: use the root of the main tree α.

- **Offline Step.**
  Find the cycle length μ of the main component of f_β and the root of the main tree α.
  Cost: $O(2^{n/2})$ applications of h.

- **Online Step.**
  Find the smallest z that yields a collision between
  - $MAC([\beta]^{z}||[1]||[\beta]^{2^{n/2}+\mu})$
  - $MAC([\beta]^{z+\mu}||[1]||[\beta]^{2^{n/2}})$.

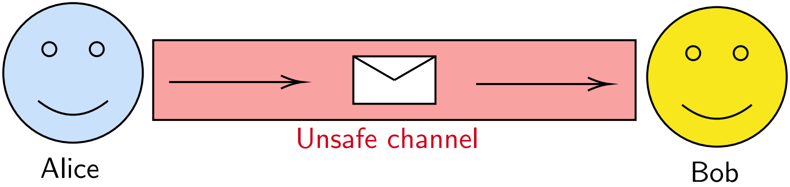
  using binary search.
  Cost: $O(2^{n/2} \cdot n)$ applications of h.

WCP, the state after $[\beta]^{z}$ is α.

Plan

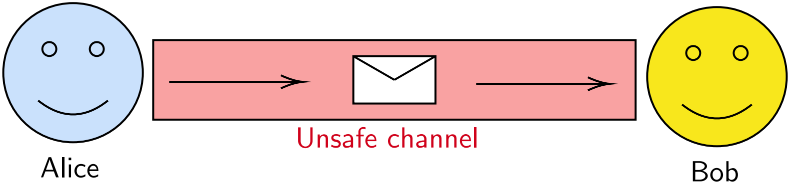
1. Random function statistics
2. Memory-negligible collision search
3. State recovery attack against HMAC
4. Generic attack against AEAD modes

Authenticated Encryption



A cryptographic scheme providing **Authenticated Encryption** ensures both the **privacy** and **integrity** of communications.

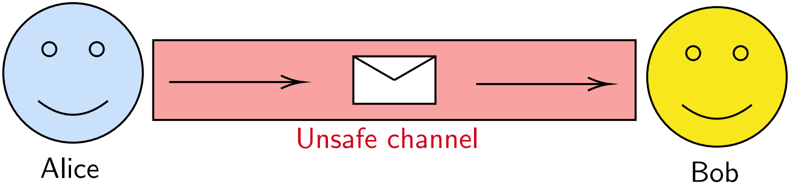
Authenticated Encryption



A cryptographic scheme providing **Authenticated Encryption** ensures both the **privacy** and **integrity** of communications.

- **Privacy**: The message can only be read by Alice and Bob.
- **Integrity**: If the message is modified in the unsafe communication channel, Bob will know.

Authenticated Encryption with Associated Data (AEAD)

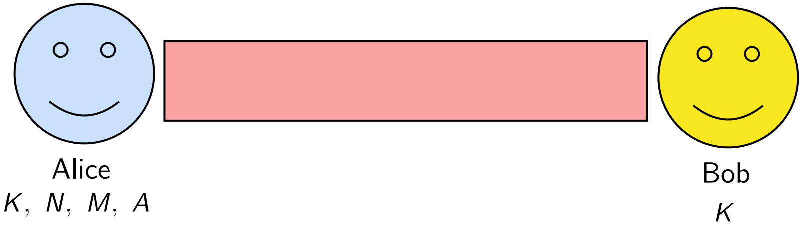


A cryptographic scheme providing **Authenticated Encryption** ensures both the **privacy** and **integrity** of communications.

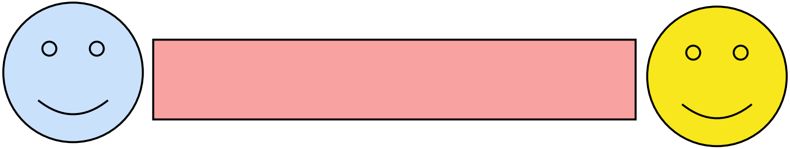
- **Privacy**: The message can only be read by Alice and Bob.
- **Integrity**: If the message is modified in the unsafe communication channel, Bob will know.

Associated data : *Public data* sent alongside the message and whose integrity is also guaranteed.

Authenticated Encryption with Associated Data (AEAD)



Authenticated Encryption with Associated Data (AEAD)



Alice

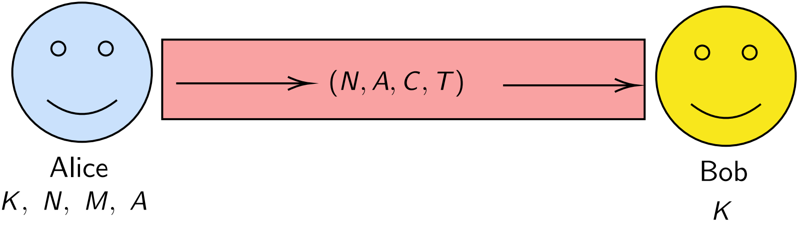
$K, \; N, \; M, \; A$

$(C, T) \; = \; Enc(K, N, M, A)$

Bob

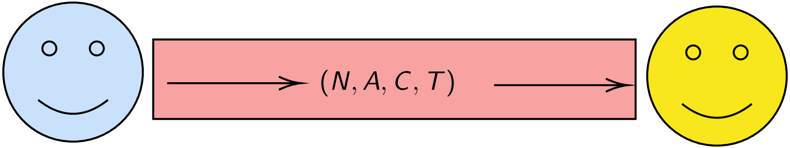
K

Authenticated Encryption with Associated Data (AEAD)



$(C, T) = Enc(K, N, M, A)$

Authenticated Encryption with Associated Data (AEAD)



Alice

$K,\ N,\ M,\ A$

$(C, T) = Enc(K, N, M, A)$

Bob

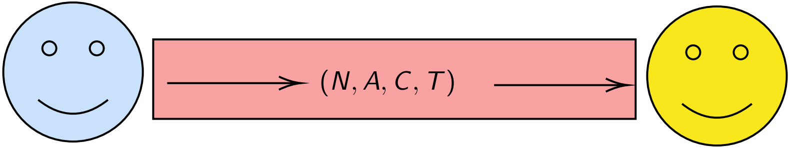
K

if $Verif(K, N, A, C, T)$

return $M = Dec(K, N, C, A)$

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds.

Authenticated Encryption with Associated Data (AEAD)



Alice

$K,\ N,\ M,\ A$

$(C, T) = Enc(K, N, M, A)$

Bob

K

if $Verif(K, N, A, C, T)$

return $M = Dec(K, N, C, A)$

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds.

It is assumed that:
- the adversary is **nonce-respecting**
- there is **no release of unverified plaintext**

Duplex-based AEAD modes

Authenticated Encryption with Associated Data

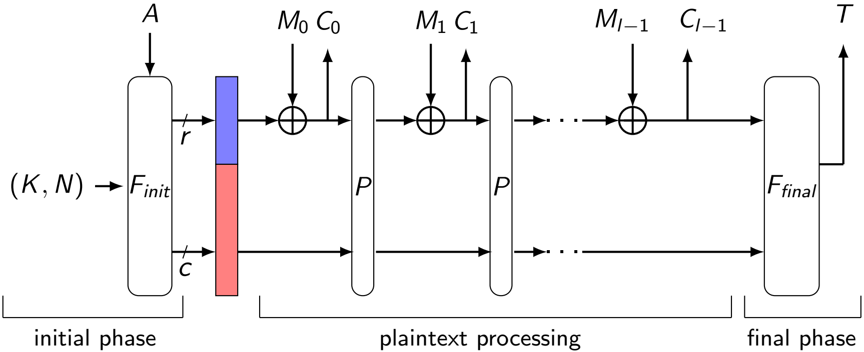
- Either **block-cipher based**: (tweakable) block cipher + mode
- Or **permutation-based**: public permutation + keyed mode
  Ex: XOODYAK = XOODOO[12] + Cyclist [DHPVAVK20]

Duplex-based modes of operation

- Permutation-based modes introduced by Bertoni, Daemen, Peeters, Van Assche [BDPVA11]
- An adaptation to the AEAD context of the **Sponge construction** [BDPVA07]

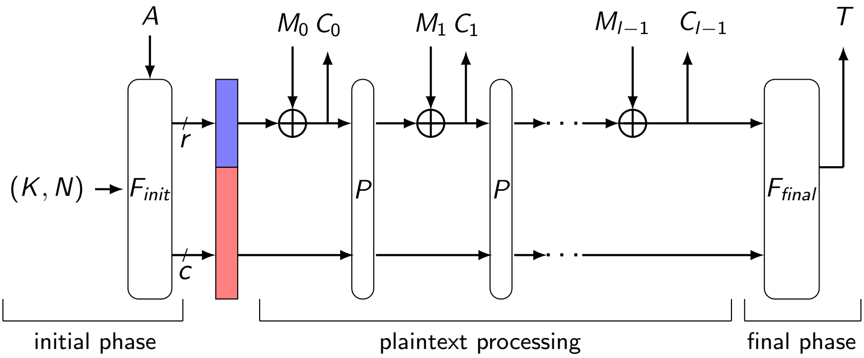
  Ex: SPONGEWRAP [BDPVA11], MonkeyWrap (KETJE) [BDPVAVK14], *etc.*

Duplex-based AEAD modes [BDPVA11]



- Permutation P operates on a state of length $b = r + c$ bits, where r is the **rate** and c the **capacity**. (Think of c as n!)
- First r bits : the **outer state**
- Next c bits : the **inner state**

Duplex-based AEAD modes [BDPVA11]



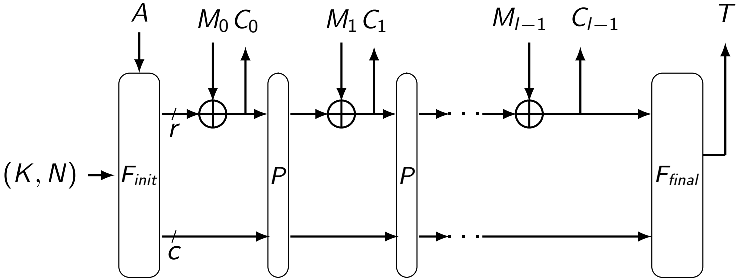
- Permutation P operates on a state of length $b = r + c$ bits, where r is the **rate** and c the **capacity**.
- First r bits : the **outer state**
- Next c bits : the **inner state**

Ex: XOODYAK
r = 192
c = 192

Forgery attack on duplex-based modes [GHKR23]

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds

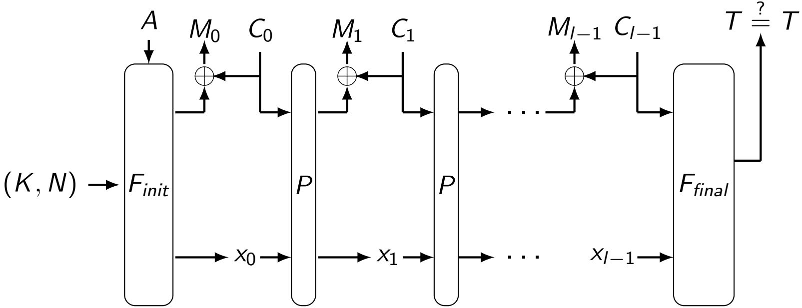
Encryption



Forgery attack on duplex-based modes [GHKR23]

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds

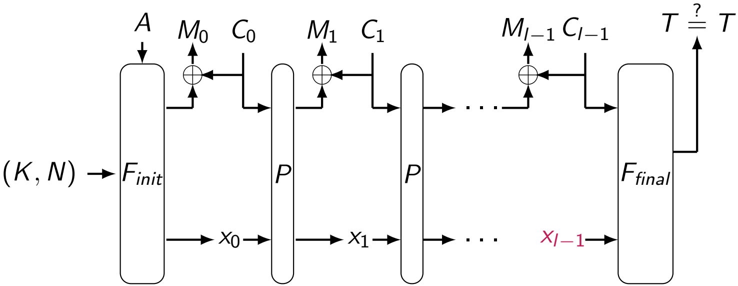
Decryption/verification



Forgery attack on duplex-based modes [GHKR23]

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds

Decryption/verification



Guessing x_{l-1} allows to build a forgery!

Forgery attack on duplex-based modes [GHKR23]

Forgery attack: find a decryption query (N, A, C, T) s.t. the tag verification succeeds

Total time complexity of an attack

$$\mathscr{T} = \sigma_e + \sigma_d + q_P + t_{extra-op}$$

where

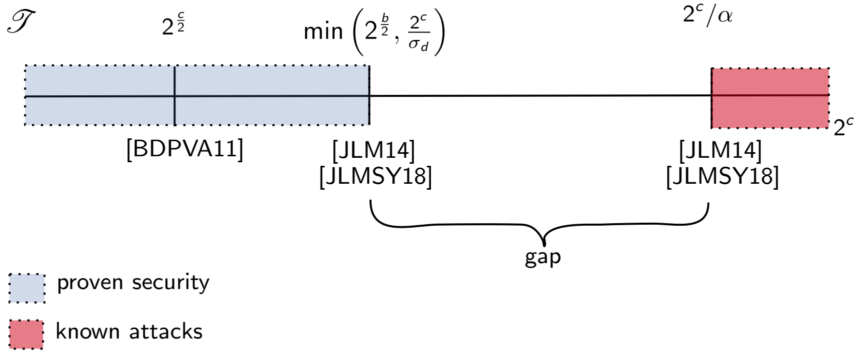
σ_e is the number of online calls to P caused by encryption queries

σ_d is the number of online calls to P caused by forgery attempts

q_P is the number of offline queries to P or P^{-1}

Disclaimer
this is (extremely) simplified

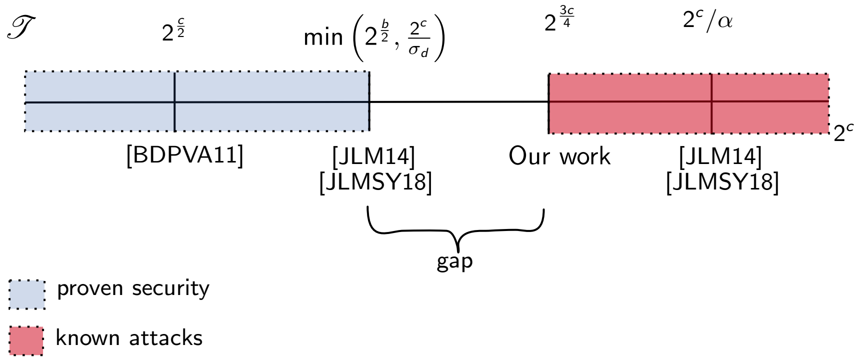
Assuming a sufficiently large key/tag length:



σ_d is the number of online calls to P caused by forgery attempts
α is a small constant

Disclaimer
this is (extremely) simplified

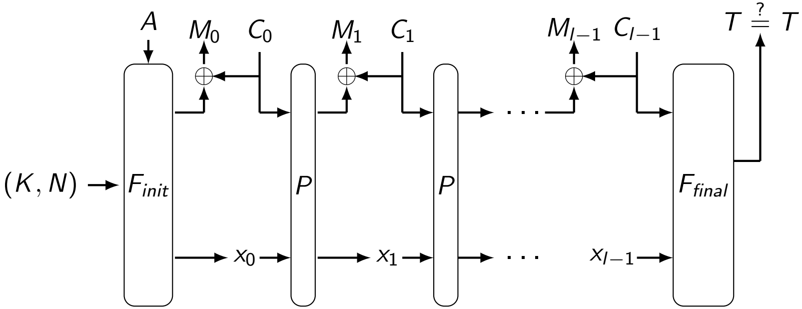
Assuming a sufficiently large key/tag length:



σ_d is the number of online calls to P caused by forgery attempts
α is a small constant

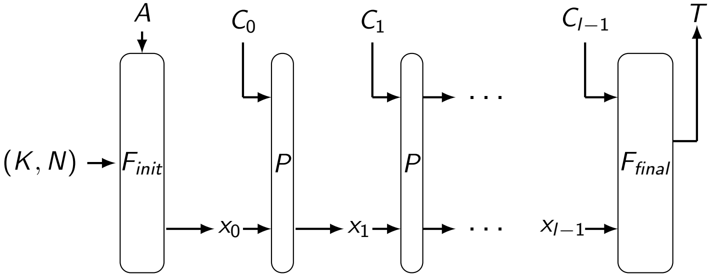
Main observation

Decrypting the ciphertext/tag pair $(C = C_0 \, || \, \cdots || C_{l-1}; T)$



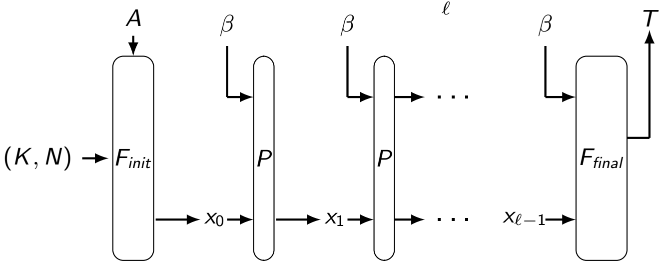
Main observation

Decrypting the ciphertext/tag pair $(C = C_0 \mid\mid \cdots \mid\mid C_{l-1}; T)$



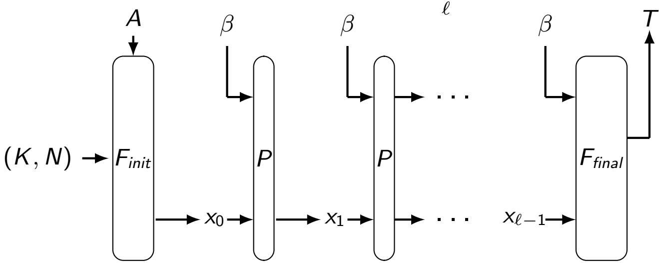
Main observation

Decrypting the long ciphertext/tag pair $(\beta_\ell = \underbrace{\beta || \cdots || \beta}_{\ell}; T)$

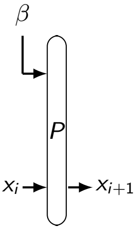


Main observation

Decrypting the long ciphertext/tag pair $(\beta_\ell = \underbrace{\beta || \cdots || \beta}_{\ell}; T)$

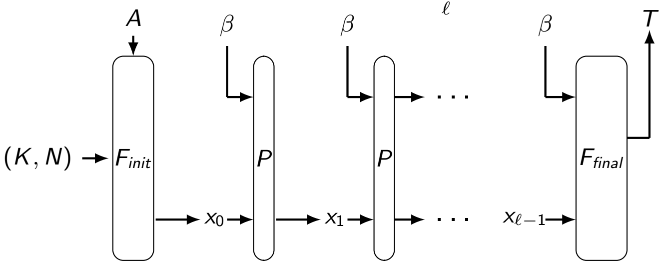


The tag verification iterates the function $f_\beta : \mathbb{F}_2^c \to \mathbb{F}_2^c$

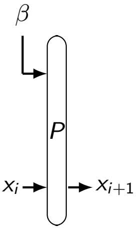


Main observation

Decrypting the long ciphertext/tag pair $(\beta_\ell = \underbrace{\beta || \cdots || \beta}_{\ell}; T)$

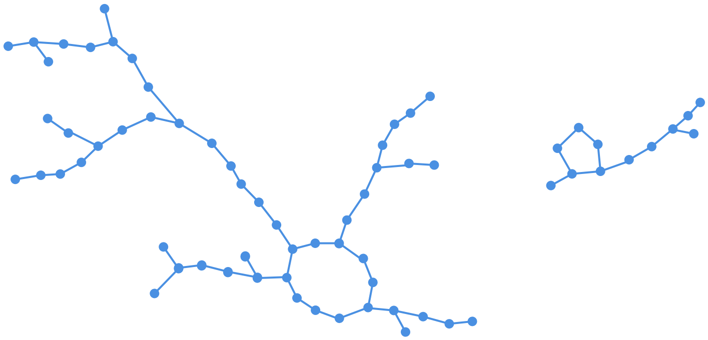


The tag verification iterates the function $f_\beta : \mathbb{F}_2^c \to \mathbb{F}_2^c$



- For a random β, we expect f_β to behave as a **random function** drawn in $\mathfrak{F}_{2^c}$.
- For each nonce, we expect x_0 to behave as a **random point** drawn in the graph of f_β.

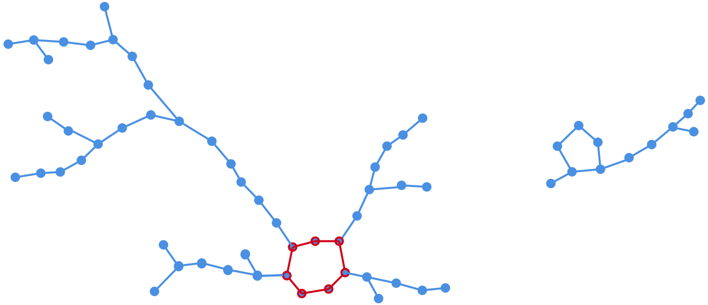
Reminder: graph of a random function f in $\mathfrak{F}_{2^c}$



Average. . .

- Size of the largest component: $2^c \times 0.76$. [FO89]
- Cycle/tail length of a random point: $2^{\frac{c}{2}}\sqrt{\pi/8}$

Reminder: graph of a random function f in $\mathfrak{F}_{2^c}$



The probability that a random function has a component

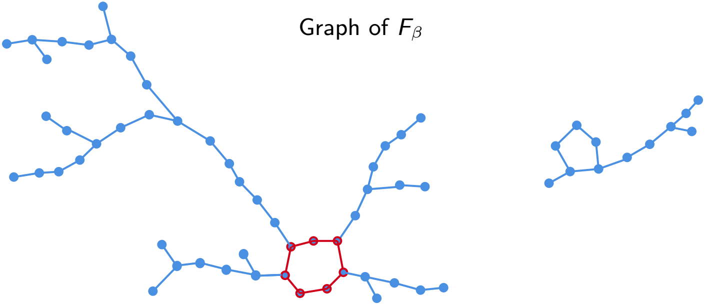
- of cycle length at most $\leq 2^{\frac{c}{2}-\nu}$ $\rightarrow$ its cycle is **exceptionally small**:
- of size at least $\geq 2^c \times s$ $\rightarrow$ this component is **reasonably large**;

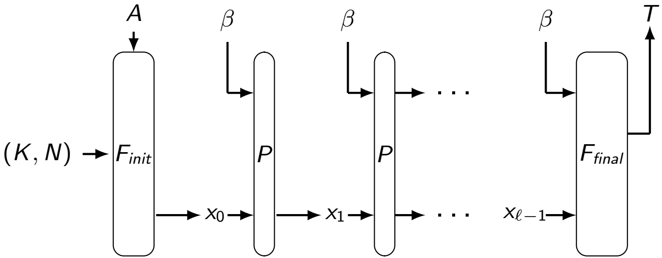
$$p_{s,\nu} \approx \sqrt{\frac{2(1-s)}{\pi s}} 2^{-\nu}$$

[DeLaurentis87]

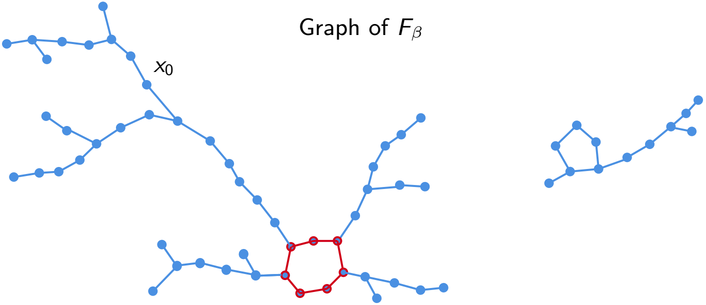
Ex: proba for $s = 65\%$ and $\nu = \frac{c}{4}$ (cycle of length $\leq 2^{\frac{c}{4}}$): $0.6 \times 2^{-\frac{c}{4}}$

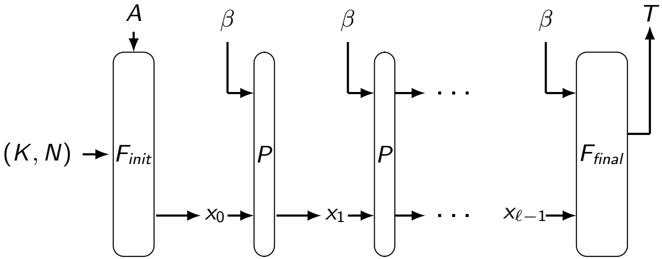
Core idea of our forgery attack



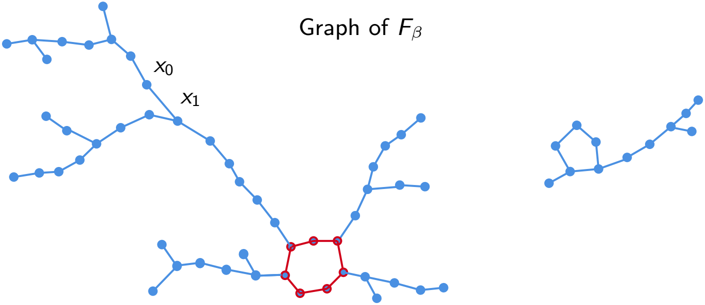


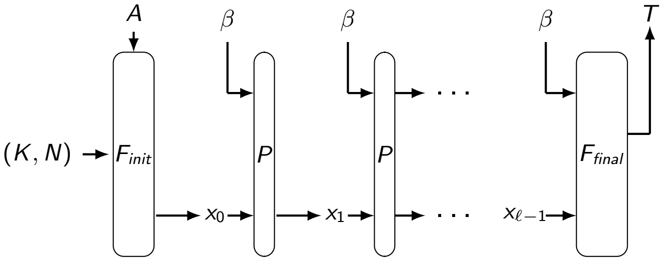
Core idea of our forgery attack



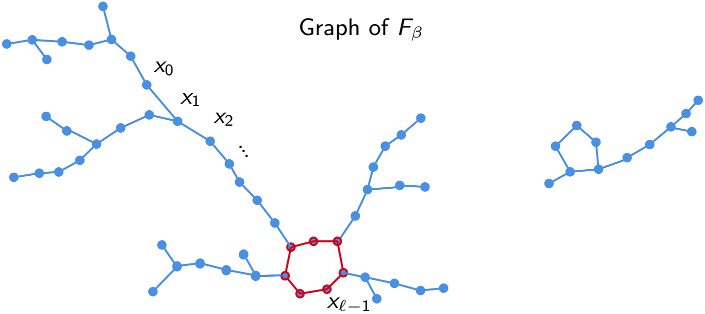


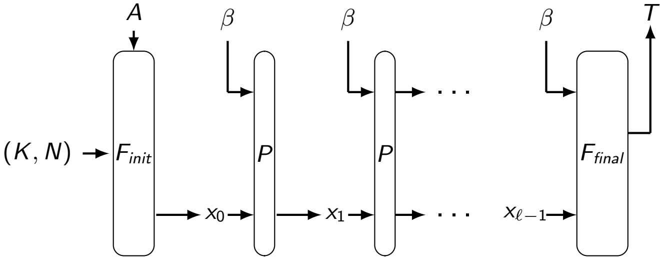
Core idea of our forgery attack



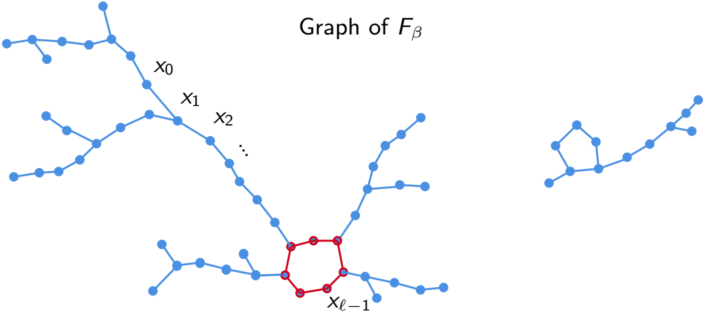


Core idea of our forgery attack



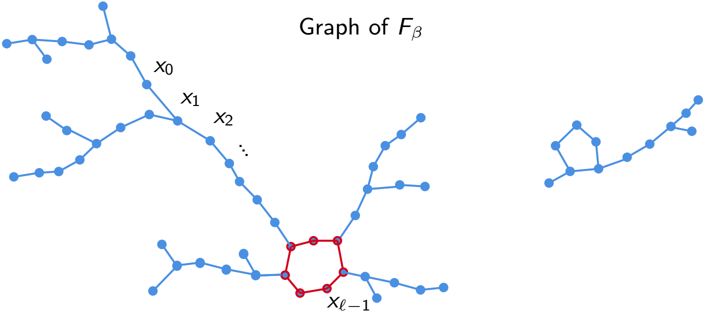


Core idea of our forgery attack



If one finds β s.t. f_β has a reasonably **large component** (say $\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** (say $\leq 2^{\frac{c}{4}}$)...

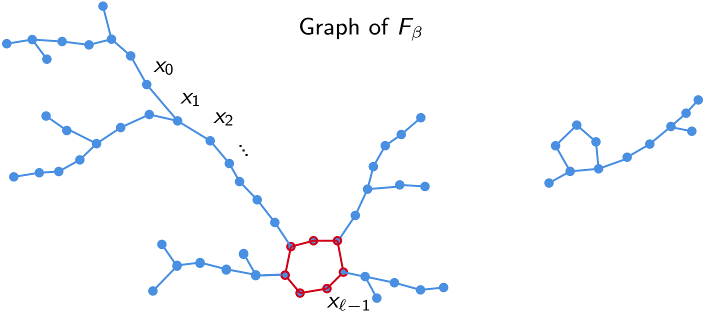
Core idea of our forgery attack



If one finds β s.t. f_β has a reasonably **large component** (say $\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** (say $\leq 2^{\frac{c}{4}}$)...

$\rightarrow$ Since the component is large, x_0 belongs to it with good probability (≈ 0.65)

Core idea of our forgery attack

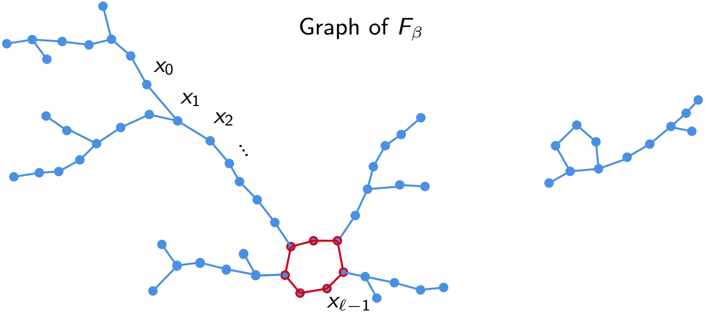


If one finds β s.t. f_β has a reasonably **large component** (say $\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** (say $\leq 2^{\frac{c}{4}}$)...

$\rightarrow$ Since the component is large, x_0 belongs to it with good probability (≈ 0.65)

$\rightarrow$ If so, if ℓ is 'large enough' (say $\ell \approx 2^{\frac{c}{2}}$), $x_{\ell-1}$ is in the cycle with good probability

Core idea of our forgery attack



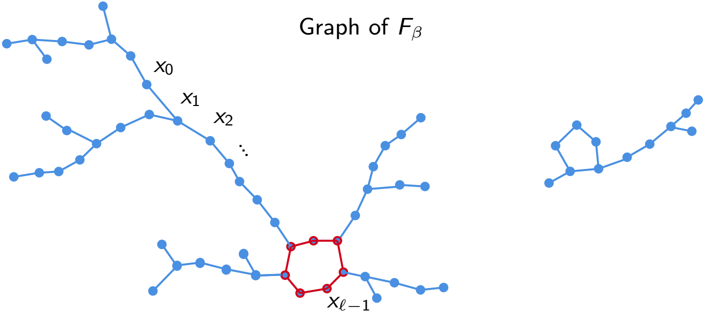
If one finds β s.t. f_β has a reasonably **large component** (say $\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** (say $\leq 2^{\frac{c}{4}}$)...

$\rightarrow$ Since the component is large, x_0 belongs to it with good probability (≈ 0.65)

$\rightarrow$ If so, if ℓ is 'large enough' (say $\ell \approx 2^{\frac{c}{2}}$), $x_{\ell-1}$ is in the cycle with good probability

$\rightarrow$ If so, there are at most $2^{\frac{c}{4}}$ possible values for $x_{\ell-1}$ *i.e.* **at most $2^{\frac{c}{4}}$ possible tags**

Core idea of our forgery attack



If one finds β s.t. f_β has a reasonably **large component** (say $\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** (say $\leq 2^{\frac{c}{4}}$)...

$\rightarrow$ Since the component is large, x_0 belongs to it with good probability (≈ 0.65)

$\rightarrow$ If so, if ℓ is 'large enough' (say $\ell \approx 2^{\frac{c}{2}}$), $x_{\ell-1}$ is in the cycle with good probability

$\rightarrow$ If so, there are at most $2^{\frac{c}{4}}$ possible values for $x_{\ell-1}$ *i.e.* **at most $2^{\frac{c}{4}}$ possible tags**

Resulting forgery attack: try the $\leq 2^{\frac{c}{4}}$ possible values for T.

Core idea of our forgery attack

Precomputation phase
Find β s.t. f_β has a **large component** ($\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** ($\leq 2^{\frac{c}{4}}$), recover this cycle.

} **key independant**

Online phase
Submit $(N, A, C = \underbrace{\beta || \cdots || \beta}_{\ell}, T)$ queries to the decryption oracle where:

- N is randomly sampled
- A is set to the empty string
- ℓ is 'big enough' ($\approx 2^{\frac{c}{2}}$)
- $T = P_{final}(\beta || x)$, for x in the small cycle

Simplified complexity analysis (precomputation phase)

Precomputation phase: Find β s.t. f_β has a **large component** ($\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** ($\leq 2^{\frac{c}{4}}$), recover this cycle.

Complexity analysis:

- Drawing about $1/p_{s,\nu} \approx 2^{\frac{c}{4}}$ random β's
- For each β, investigating F_β costs $\approx 2^{\frac{c}{2}}$ per β thanks to Floyd's algorithm.

The total complexity is $\approx 2^{\frac{3c}{4}}$ **applications of** P.

Simplified complexity analysis (precomputation phase)

Precomputation phase: Find β s.t. f_β has a **large component** ($\geq 0.65 \times 2^c$) with an exceptionnally **small cycle** ($\leq 2^{\frac{c}{4}}$), recover this cycle.

Complexity analysis:

- Drawing about $1/p_{s,\nu} \approx 2^{\frac{c}{4}}$ random β's
- For each β, investigating F_β costs $\approx 2^{\frac{c}{2}}$ per β thanks to Floyd's algorithm.

The total complexity is $\approx 2^{\frac{3c}{4}}$ **applications of** P.

Note: the algorithm includes a test that the component is likely to be large enough.

Simplified complexity analysis (online phase)

Online phase. Submit $(N, A, C = \underbrace{\beta || \cdots || \beta}_{\ell}, T)$ queries to the decryption oracle where $T = F_{final}(\beta || x)$, x in the cycle.

Simplified complexity analysis (online phase)

Online phase. Submit $(N, A, C = \underbrace{\beta||\cdots||\beta}_{\ell}, T)$ queries to the decryption oracle where $T = F_{final}(\beta||x)$, x in the cycle.

Complexity analysis:

- x_0 belongs to the desired component with probability $s = 65\%$
- For $x_{\ell-1}$ to belong to the cycle with good probability, we set $\ell = 3 \times 2^{\frac{c}{2}}$
- We try at most $2^{\frac{c}{4}}$ values for T (at most the length of the cycle).

The total complexity is $\approx 2^{\frac{3c}{4}}$ **applications of** P.

Simplified complexity analysis (online phase)

Online phase. Submit $(N, A, C = \underbrace{\beta||\cdots||\beta}_{\ell}, T)$ queries to the decryption oracle where $T = F_{final}(\beta||x)$, x in the cycle.

Complexity analysis:

- x_0 belongs to the desired component with probability $s = 65\%$
- For $x_{\ell-1}$ to belong to the cycle with good probability, we set $\ell = 3 \times 2^{\frac{c}{2}}$
- We try at most $2^{\frac{c}{4}}$ values for T (at most the length of the cycle).

The total complexity is $\approx 2^{\frac{3c}{4}}$ **applications of** P.

Note: At the cost of a more expensive prec. phase, the complexity of this step can be brought close(r) to $2^{\frac{c}{2}}$.

Small scale experiments

- Our attack is somewhat heuristic based.

→ Ex: corroborate that the f_β behave as **random functions** in practice.

- We implemented experiments with XOODOO[12] as P.

- All our practical results match our heuristic-based results.

→ Ex: the average tail length for a random f_β matches the average tail length for a random permutation.

- We also implemented the **precomputation algorithm**.

→ We found some **valid β values** for c up to 40.

Summary of our results

Our attack

- has **total time complexity** $\leq 21 \times 2^{\frac{3c}{4}}$;
- a **probability of success** $\geq 95\%$;
- can be transformed into a **key recovery** at a negligible extra cost if P_{init} is reversible (**how**: using the plaintext);
- is applicable to the modes of Norx v2, Ketje, KNOT and Keyak;
- breaks the 184-bit security claim made by the designers of Xoodyak with an attack of complexity 2^{148};
- $\neq$ attack on HMAC that has complexity $\approx 2^{n/2}$: for a given C, we cannot ask an oracle to provide a valid T.

Preventing the attack

Two main features frustrate our cryptanalysis:

- **Key-dependent final phase.** (ASCON, NORX v3)

$\rightarrow$ a correct guess on $x_{\ell-1}$ cannot be transformed into a forgery (still a state recovery)

- **No outer state overwriting.** (Beetle, SPARKLE)

$\rightarrow$ the decryption of $\underbrace{\beta || \cdots || \beta}_{\ell}$ does not correspond to the iteration of a function

Thank you for your attention :)

Any questions?