SPARKLE : une famille d'algorithmes scintillants

Christof Beierle[1] Alex Biryukov[2] Luan Cardoso dos Santos[2]
Johann Großschädl[2] Léo Perrin[3] Aleksei Udovenko[4]
Vesselin Velichkov[5] Qingju Wang[2]

[1]Ruhr University Bochum, Germany
[2]SnT and DCS, University of Luxembourg, Luxembourg
[3]Inria, France
[4]CryptoExperts, France
[5]University of Edinburgh, U.K.

sparklegrupp@googlegroups.com

FSE 2020

Outline

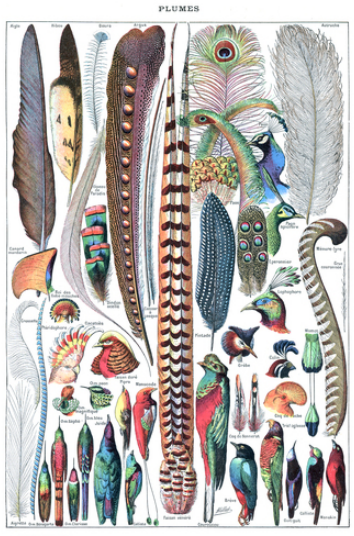
Plan of this Section

Au commencement, un débat philosophique

Être léger comme une plume...

0. Sources (images) : https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg, https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg

Au commencement, un débat philosophique

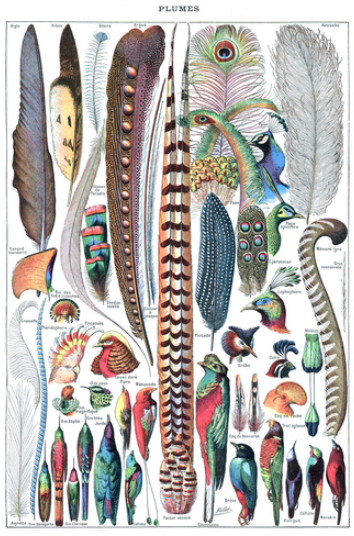


Être léger comme une plume…

mais quel genre de plume ?

0. Sources (images) : https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg, https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg

Au commencement, un débat philosophique



Être léger comme une plume…

mais quel genre de plume ?

Nos objectifs

1. Sécurité

0. Sources (images) : https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg, https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg

Au commencement, un débat philosophique

Être léger comme une plume...

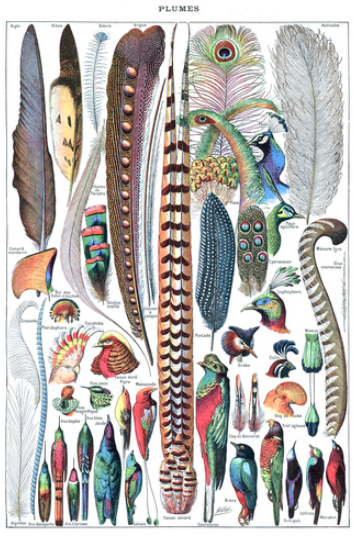
mais quel genre de plume ?

Nos objectifs

1. Sécurité
2. Implémentation sur des micro-controleurs (8-, 16-, 32-bit)

0. Sources (images) : https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg, https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg

Au commencement, un débat philosophique



Être léger comme une plume...

mais quel genre de plume ?

Nos objectifs

1. Sécurité
2. Implémentation sur des micro-controleurs (8-, 16-, 32-bit)
3. Code compact, vitesse élevée

0. Sources (images) : `https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg`, `https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg`

Au commencement, un débat philosophique

Être léger comme une plume...

mais quel genre de plume ?

Nos objectifs

1. Sécurité
2. Implémentation sur des micro-controleurs (8-, 16-, 32-bit)
3. Code compact, vitesse élevée
4. Permettre différents compromis (implémentation et sécurité)

---

0. Sources (images) : `https://commons.wikimedia.org/wiki/File:Buteo_buteo_primary_secondary.jpg`, `https://commons.wikimedia.org/wiki/File:Types_de_plumes._-_Larousse_pour_tous,_-1907-1910-.jpg`

Un premier cahier des charges

Utilisation de permutations : pour minimiser l'utilisation de mémoire et fournir AEAD et hachage

Un premier cahier des charges

Utilisation de permutations : pour minimiser l'utilisation de mémoire et fournir AEAD et hachage

ARX : diffusion rapide et un degré algébrique très élevé pour un coup très faible (en software)

Un premier cahier des charges

Utilisation de permutations : pour minimiser l'utilisation de mémoire et fournir AEAD et hachage

ARX : diffusion rapide et un degré algébrique très élevé pour un coup très faible (en software)

Structure du tour : quelque chose qui permette d'obtenir des arguments de sécurité forts contre les attaques différentielles et linéaires.

Un premier cahier des charges

Utilisation de permutations : pour minimiser l'utilisation de mémoire et fournir AEAD et hachage

ARX : diffusion rapide et un degré algébrique très élevé pour un coup très faible (en software)

Structure du tour : quelque chose qui permette d'obtenir des arguments de sécurité forts contre les attaques différentielles et linéaires. **Comment ?**

Un premier cahier des charges

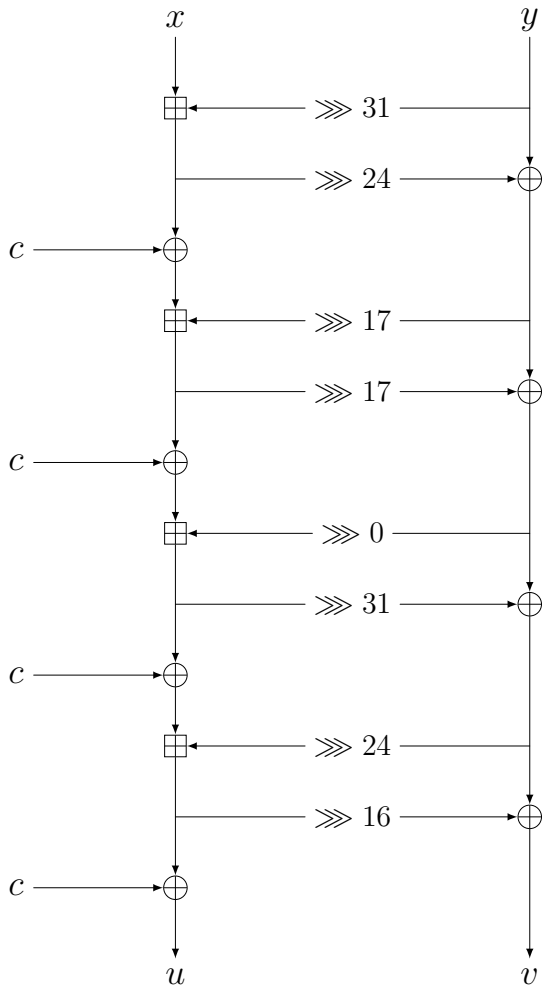
Utilisation de permutations : pour minimiser l'utilisation de mémoire et fournir AEAD et hachage

ARX : diffusion rapide et un degré algébrique très élevé pour un coup très faible (en software)

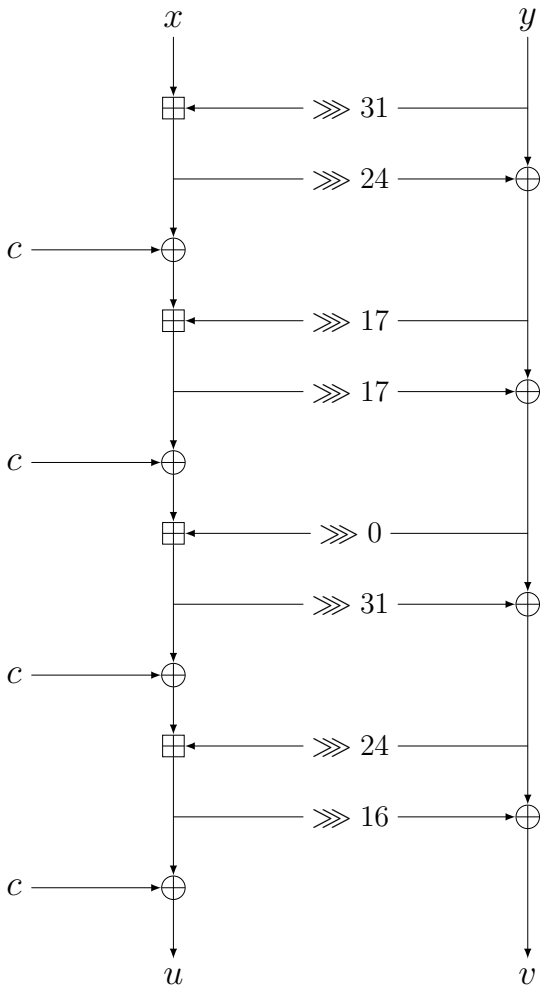
Structure du tour : quelque chose qui permette d'obtenir des arguments de sécurité forts contre les attaques différentielles et linéaires. **Comment ?**

Alzette + Stratégie des Longs Chemins

Au commencement était “Alzette”

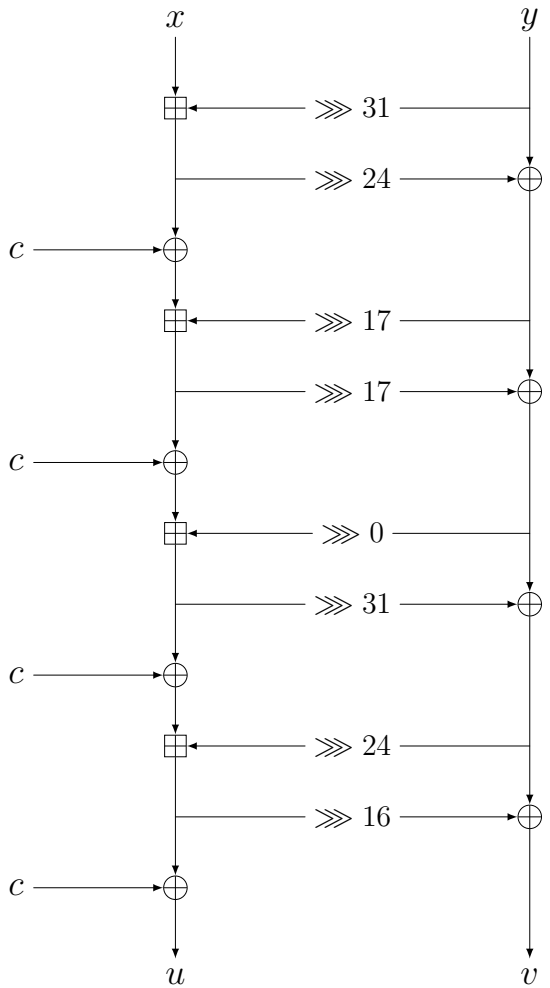


Au commencement était “Alzette”



	MEDCP	MELCC
Alzette	2^{-6}	2^{-2}
AES S-box layer	2^{-6}	2^{-3}
Double Alzette	$\leq 2^{-32}$	2^{-17}
AES super S-box layer	2^{-30}	2^{-15}

Au commencement était “Alzette”



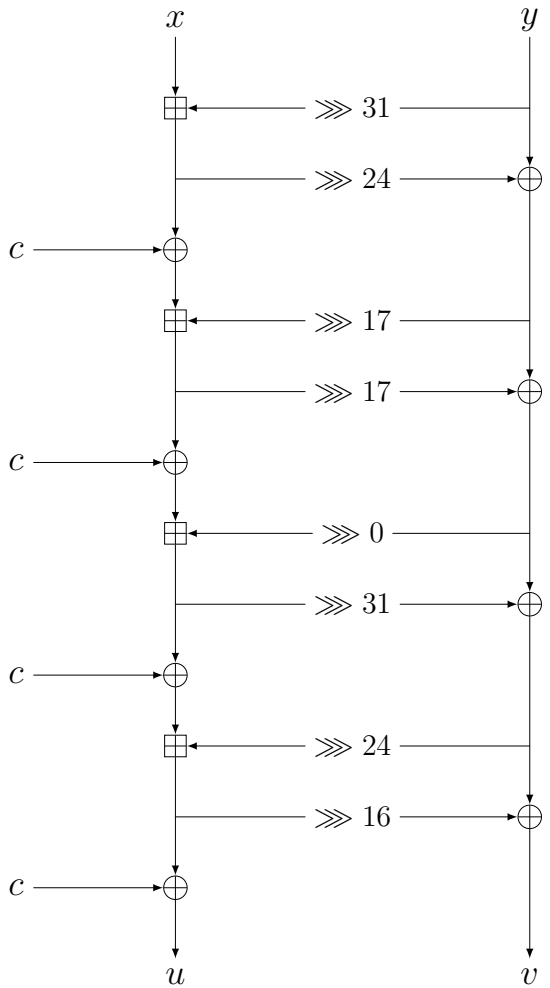
	MEDCP	MELCC
Alzette	2^{-6}	2^{-2}
AES S-box layer	2^{-6}	2^{-3}
Double Alzette	$\leq 2^{-32}$	2^{-17}
AES super S-box layer	2^{-30}	2^{-15}

Elle a aussi un très haut degré algébrique, une diffusion rapide, de bonnes propriétés intégrales...

Alzette : a 64-bit ARX-box (feat. CRAX and TRAX)

(CRYPTO'20, see also `eprint 2019/1378`)

Au commencement était “Alzette”



	MEDCP	MELCC
Alzette	2^{-6}	2^{-2}
AES S-box layer	2^{-6}	2^{-3}
Double Alzette	$\leq 2^{-32}$	2^{-17}
AES super S-box layer	2^{-30}	2^{-15}

Elle a aussi un très haut degré algébrique, une diffusion rapide, de bonnes propriétés intégrales...

Alzette : a 64-bit ARX-box (feat. CRAX and TRAX)

(CRYPTO'20, see also `eprint 2019/1378`)

Une itération ne donne pas beaucoup de protection... Deux en donnent beaucoup !

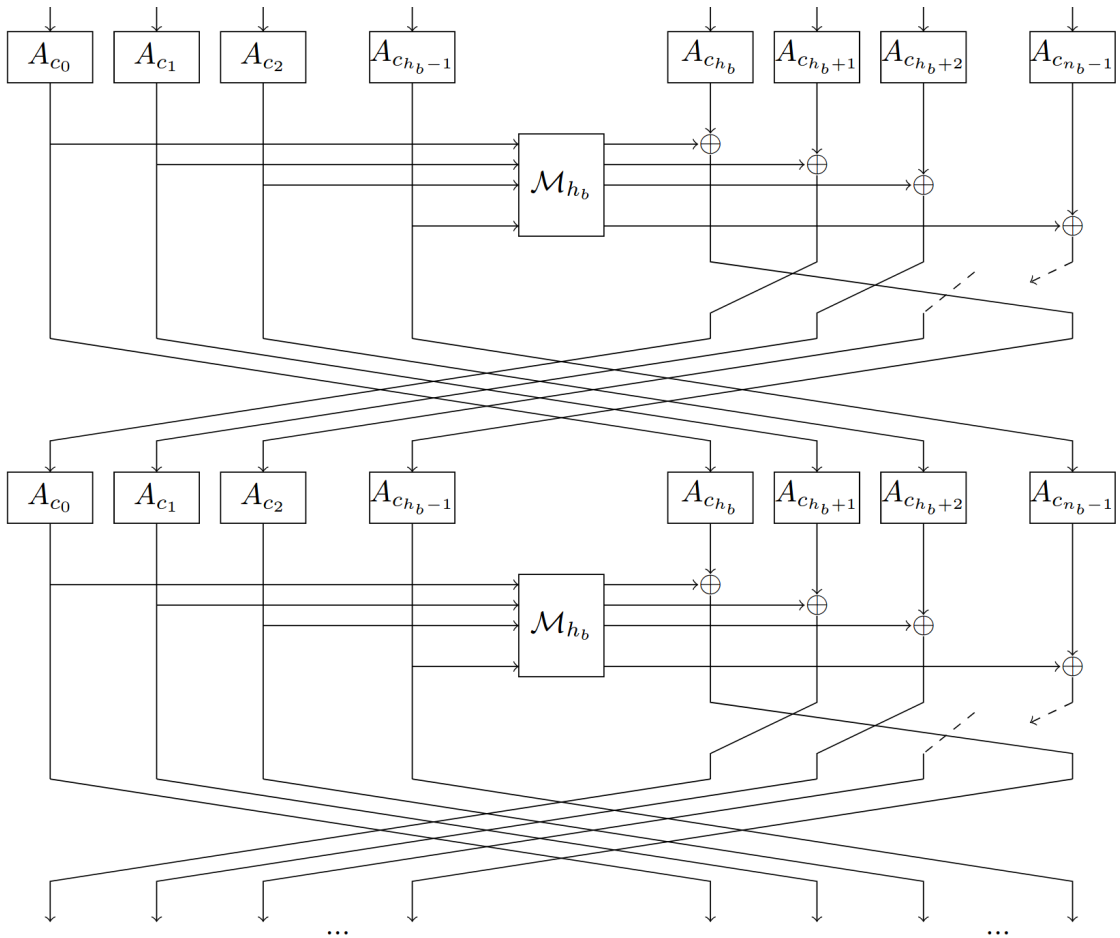
Plan of this Section

La stratégie des longs chemins

Comment construire une bonne fonction de tours sachant qu'Alzette a besoin de deux itérations ?

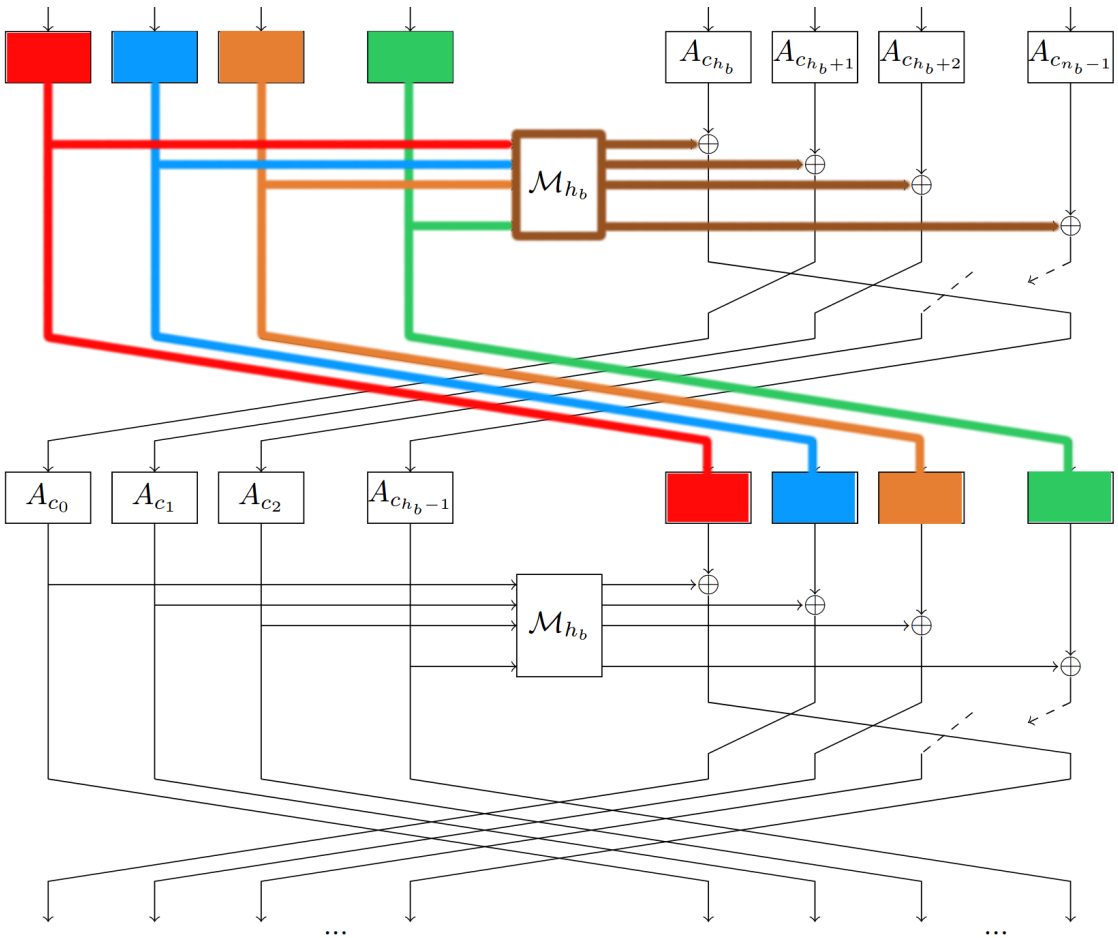
La stratégie des longs chemins

Comment construire une bonne fonction de tours sachant qu'Alzette a besoin de deux itérations ?



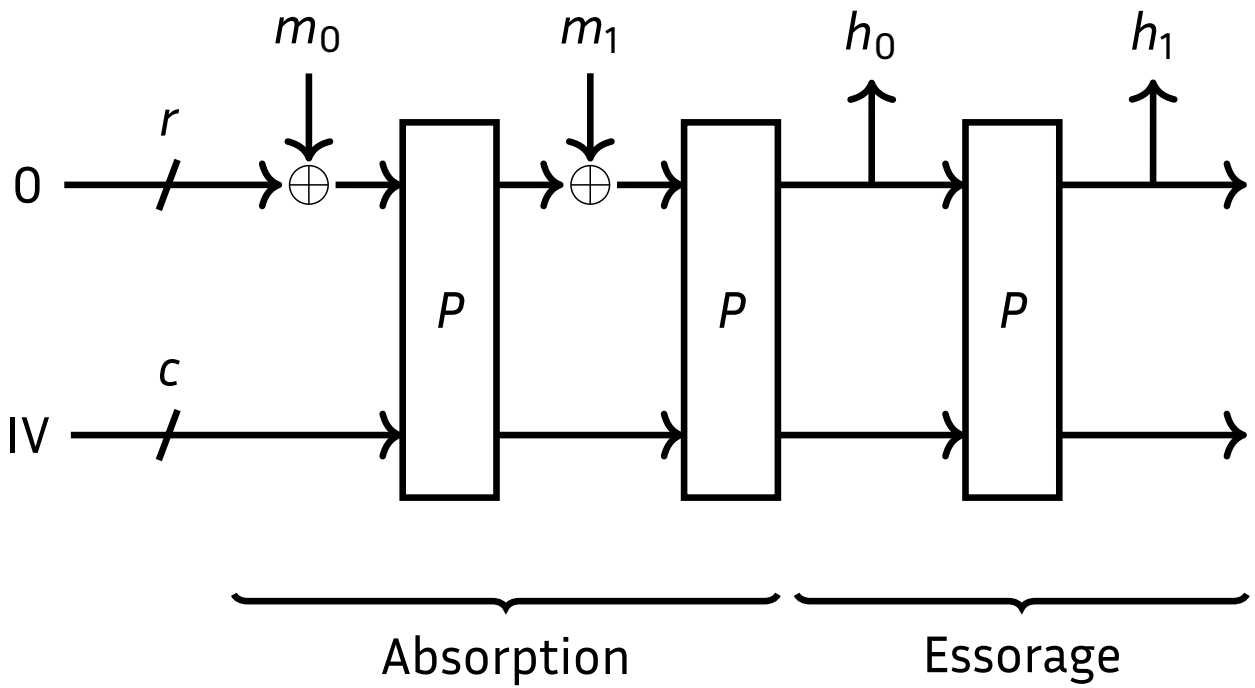
La stratégie des longs chemins

Comment construire une bonne fonction de tours sachant qu'Alzette a besoin de deux itérations ?



Mode pour permutation

Pour une primitive conçue en suivant la SLC, on peut borner les probabilités des chemins correspondants à des injections de message.



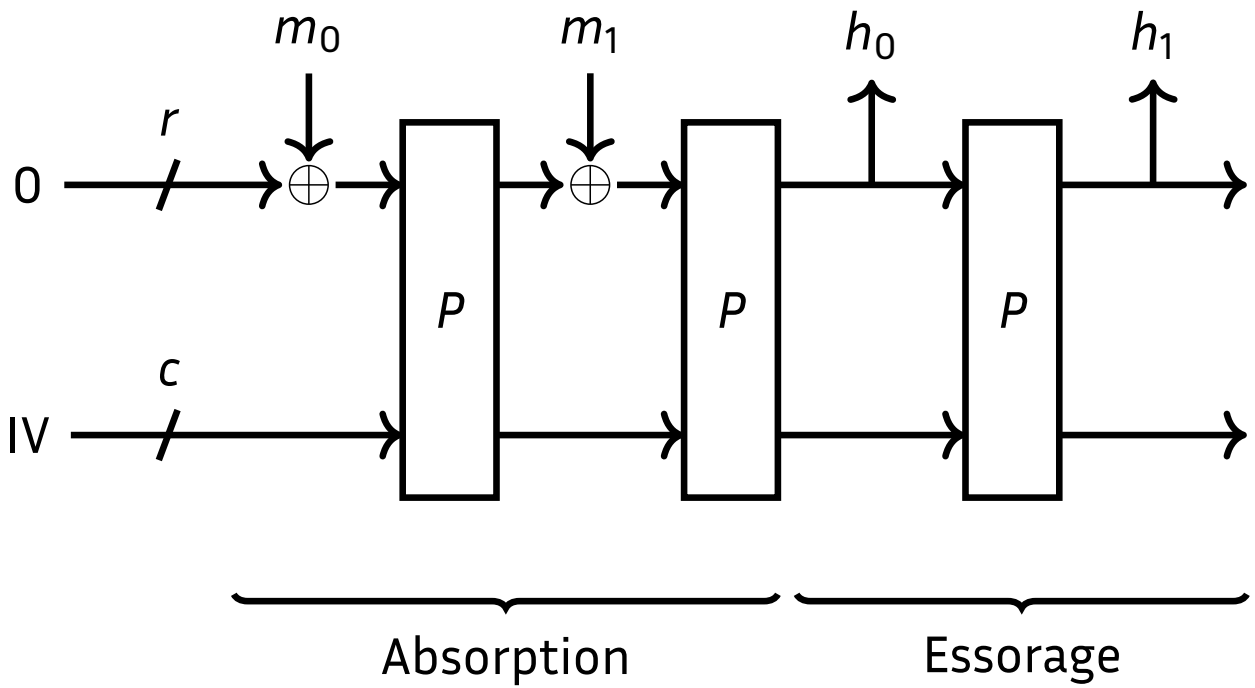
Mode pour permutation

Pour une primitive conçue en suivant la SLC, on peut borner les probabilités des chemins correspondants à des injections de message.

n	r	c (security)	Type	3	4	5	6	7	8
384			$n \to n$	70	100	178	200	230	260
	256	128	$r \to r$	108	148	180	200	268	296
			$r \to n$	70	140	178	200	230	296
	192	192	$r \to r$	128	160	256	256	288	320
			$r \to n$	128	148	178	210	276	306
	128	256	$r \to r$	128	160	256	256	320	320
			$r \to n$	128	160	180	210	276	306
	64	320	$r \to r$	$\emptyset$	160	256	256	320	320
			$r \to n$	128	160	180	210	276	306

Mode pour permutation

Pour une primitive conçue en suivant la SLC, on peut borner les probabilités des chemins correspondants à des injections de message.



Plan of this Section

Nos permutations

Nom	n	# $\|\|$ Alz.	# steps slim	# steps big
Sparkle256	256	4	7	10
Sparkle384	384	6	7	11
Sparkle512	512	8	8	12

Nos permutations

Nom	n	# $\|\|$ Alz.	# steps slim	# steps big
Sparkle256	256	4	7	10
Sparkle384	384	6	7	11
Sparkle512	512	8	8	12

Slim vs. Big

Nos permutations

Nom	n	# ‖ Alz.	# steps slim	# steps big
Sparkle256	256	4	7	10
Sparkle384	384	6	7	11
Sparkle512	512	8	8	12

Slim vs. Big

- Les instances “Big” peuvent être utilisées dans n’importe quel mode.
  Initialisation, finalisation

Nos permutations

Nom	n	# $\|\|$ Alz.	# steps slim	# steps big
Sparkle256	256	4	7	10
Sparkle384	384	6	7	11
Sparkle512	512	8	8	12

Slim vs. Big

- Les instances “Big” peuvent être utilisées dans n’importe quel mode. **Initialisation, finalisation**
- Les instances Slim ne peuvent être utilisées que dans des modes où le rate est là où nous l’avons spécifié.

Nos permutations

Nom	n	# ‖ Alz.	# steps slim	# steps big
Sparkle256	256	4	7	10
Sparkle384	384	6	7	11
Sparkle512	512	8	8	12

Slim vs. Big

- Les instances “Big” peuvent être utilisées dans n’importe quel mode.
  Initialisation, finalisation
- Les instances Slim ne peuvent être utilisées que dans des modes où le rate est là où nous l’avons spécifié. **Entre l’absorption des blocs**

L'implémentation des permutations

```c
#define MAX_BRANCHES 8
#define ROT(x, n) (((x) >> (n)) | ((x) << (32-(n))))
#define ELL(x) (ROT(((x) ^ ((x) << 16)), 16))

// Round constants
static const uint32_t RCON[MAX_BRANCHES] = {
    0xB7E15162, 0xBF715880, 0x38B4DA56, 0x324E7738,
    0xBB1185EB, 0x4F7C7B57, 0xCFBFA1C8, 0xC2B3293D};

void sparkle(uint32_t *state, int nb, int ns)
{
  int i, j;  // Step and branch counter
  uint32_t rc, tmpx, tmpy, x0, y0;
  for(i = 0; i < ns; i ++) {
    // Counter addition
    state[1] ^= RCON[i%MAX_BRANCHES]; state[3] ^= i;
    // ARXBox layer
    for(j = 0; j < 2*nb; j += 2) {
      rc = RCON[j>>1];
      state[j] += ROT(state[j+1], 31); state[j+1] ^= ROT(state[j], 24); state[j] ^= rc;
      state[j] += ROT(state[j+1], 17); state[j+1] ^= ROT(state[j], 17); state[j] ^= rc;
      state[j] += state[j+1];          state[j+1] ^= ROT(state[j], 31); state[j] ^= rc;
      state[j] += ROT(state[j+1], 24); state[j+1] ^= ROT(state[j], 16); state[j] ^= rc;
    }
    // Linear layer
    tmpx = x0 = state[0]; tmpy = y0 = state[1];
    for(j = 2; j < nb; j += 2) {
      tmpx ^= state[j]; tmpy ^= state[j+1];
    }
    tmpx = ELL(tmpx); tmpy = ELL(tmpy);
    for (j = 2; j < nb; j += 2) {
      state[j-2] = state[j+nb  ] ^ state[j  ] ^ tmpy; state[j+nb  ] = state[j];
      state[j-1] = state[j+nb+1] ^ state[j+1] ^ tmpx; state[j+nb+1] = state[j+1];
    }
    state[nb-2] = state[nb  ] ^ x0 ^ tmpy; state[nb+1] = y0;
    state[nb-1] = state[nb+1] ^ y0 ^ tmpx; state[nb  ] = x0;
  }
}
```

L'implémentation des permutations

```c
#define MAX_BRANCHES 8
#define ROT(x, n) (((x) >> (n)) | ((x) << (32-(n))))
#define ELL(x) (ROT(((x) ^ ((x) << 16)), 16))

// Round constants
static const uint32_t RCON[MAX_BRANCHES] = {
    0xB7E15162, 0xBF715880, 0x38B4DA56, 0x324E7738,
    0xBB1185EB, 0x4F7C7B57, 0xCFBFA1C8, 0xC2B3293D};

void sparkle(uint32_t *state, int nb, int ns)
{
  int i, j;  // Step and branch counter
  uint32_t rc, tmpx, tmpy, x0, y0;
  for(i = 0; i < ns; i ++) {
    // Counter addition
    state[1] ^= RCON[i%MAX_BRANCHES]; state[3] ^= i;
    // ARXBox layer
    for(j = 0; j < 2*nb; j += 2) {
      rc = RCON[j>>1];
      state[j] += ROT(state[j+1], 31); state[j+1] ^= ROT(state[j], 24); state[j] ^= rc;
      state[j] += ROT(state[j+1], 17); state[j+1] ^= ROT(state[j], 17); state[j] ^= rc;
      state[j] += state[j+1];          state[j+1] ^= ROT(state[j], 31); state[j] ^= rc;
      state[j] += ROT(state[j+1], 24); state[j+1] ^= ROT(state[j], 16); state[j] ^= rc;
    }
    // Linear layer
    tmpx = x0 = state[0]; tmpy = y0 = state[1];
    for(j = 2; j < nb; j += 2) {
      tmpx ^= state[j]; tmpy ^= state[j+1];
    }
    tmpx = ELL(tmpx); tmpy = ELL(tmpy);
    for (j = 2; j < nb; j += 2) {
      state[j-2] = state[j+nb  ] ^ state[j  ] ^ tmpy; state[j+nb  ] = state[j];
      state[j-1] = state[j+nb+1] ^ state[j+1] ^ tmpx; state[j+nb+1] = state[j+1];
    }
    state[nb-2] = state[nb  ] ^ x0 ^ tmpy; state[nb+1] = y0;
    state[nb-1] = state[nb+1] ^ y0 ^ tmpx; state[nb  ] = x0;
  }
}
```

1. Taille vs. vitesse : rouler/dérouler les boucles
2. Une implémentation pour toutes les variantes
3. Les 1ers tours de "Big" sont identiques à "Slim"

Nos Algorithmes

AEAD : Schwaemm-*r*-*c*

Name	*n*	*r*	*c*	*k*	*t*	*s*
Schwaemm-128-128	256	128	128	128	128	120
Schwaemm-256-128	384	256	128	128	128	120
Schwaemm-192-192	384	192	192	192	192	184
Schwaemm-256-256	512	256	256	256	256	248

Function de hachage : Esch *d*

Name	*n*	*r*	*c*	*d*
ESCH256	384	128	256	256
ESCH384	512	128	384	384

Mesures d'efficacité (hachage)

https://rweather.github.io/lightweight-crypto/index.html

Mesures d'efficacité (hachage)

`https://rweather.github.io/lightweight-crypto/index.html`

ESP-32 (32 bits)

Algorithm	Hash Bits	1024 bytes	128 bytes	16 bytes	Average
Xoodyak	256	0.35	0.33	0.73	0.47
Esch256 (SPARKLE)	256	0.38	0.34	0.64	0.45
GIMLI-24-HASH	256	0.35	0.29	0.50	0.38
SATURNIN-Hash	256	0.23	0.19	0.48	0.30
Esch384 (SPARKLE)	384	0.24	0.20	0.30	0.25
ASCON-HASH	256	0.19	0.16	0.25	0.20

Mesures d'efficacité (hachage)

https://rweather.github.io/lightweight-crypto/index.html

ARM-M3 (32 bits)

Algorithm	Hash Bits	1024 bytes	128 bytes	16 bytes	Average
Esch256 (SPARKLE) (*)	256	0.89	0.78	1.50	1.06
Xoodyak (*)	256	0.71	0.65	1.43	0.93
GIMLI-24-HASH (*)	256	0.54	0.47	0.86	0.62
ASCON-HASH (*)	256	0.51	0.41	0.63	0.52
DryGASCON128-HASH (*)	256	0.29	0.29	0.88	0.48
Esch384 (SPARKLE) (*)	384	0.45	0.37	1.50	0.47
SATURNIN-Hash	256	0.24	0.20	0.49	0.31

Mesures d'efficacité (hachage)

https://rweather.github.io/lightweight-crypto/index.html

AVR (8 bits)

Algorithm	Hash Bits	1024 bytes	128 bytes	16 bytes	Average
Esch256 (SPARKLE)	256	1.90	1.65	3.15	2.23
GIMLI-24-HASH	256	1.29	1.06	1.76	1.37
Esch384 (SPARKLE)	384	1.20	0.96	1.48	1.21
SATURNIN-Hash	256	0.94	0.77	1.86	1.19
Xoodyak	256	0.92	0.83	1.83	1.19

Mesures d'efficacité (hachage)

https://rweather.github.io/lightweight-crypto/index.html

AVR (8 bits)

Algorithm	Hash Bits	1024 bytes	128 bytes	16 bytes	Average
Esch256 (SPARKLE)	256	1.90	1.65	3.15	2.23
GIMLI-24-HASH	256	1.29	1.06	1.76	1.37
Esch384 (SPARKLE)	384	1.20	0.96	1.48	1.21
SATURNIN-Hash	256	0.94	0.77	1.86	1.19
Xoodyak	256	0.92	0.83	1.83	1.19

- Sur plate-forme 32-bit, Esch256 et Xoodyak sont en tête.
- Sur AVR 8-bit AVR, Esch256 est **loin devant!**
- Compromis sécurité/performance : Esch256 vs. Esch384

Plan of this Section

Conclusion

Merci !