Forkciphers: New and Exciting Symmetric Primitives

20th Central European Conference on Cryptology CECC 2020

Elena Andreeva
elean@dtu.dk
Invited talk

Technical University of Denmark, DK

Symmetric Primitives

Main Primitives

1. (Tweakable) block cipher: AES, Skinny, etc.

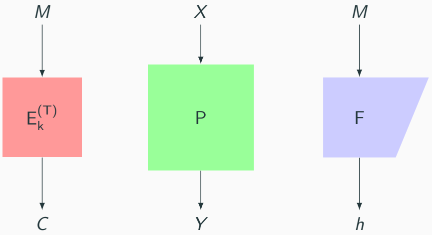
(T)PRP: Indistnguishabile from a random (tweakable) permutation

2. Permutation: Keccak, PRIMATEs permutations, etc.

IP: Ideal permutation

3. Compression function: SHA2

Collision, second preimage and preimage resistance, PRF when keyed or ideal compression function



Applications

Use symmetric primitives in some composition to build provably secure cryptographic schemes processing arbitrary long inputs:

- **Encryption schemes:** CTR, CBC, etc.
- **Message authentication codes:** CBC-MAC, HMAC, PMAC, etc.
- **Authenticated Encryption schemes:** GCM, CCM, OCB, COLM, PRIMATEs, etc.

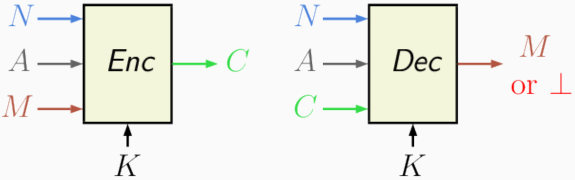
Question: Are we always using the right primitives for the right applications?

Authenticated Encryption

Authenticated Encryption

Data confidentiality and authentication

nonce-based AE(AD) syntax: a triplet $\Pi = (\mathcal{K}, Enc, Dec)$



inherent data expansion $|C| = |M| + \tau$

Authenticated Encryption of Short Messages

Important AE design goal for lightweight applications

Efficiency for short messages

- **ECRYPT-CSA Report, 2017**: *"The **performance target is wrong** . . . Another increasingly common scenario is that an authenticated cipher is applied to many **small messages** . . . The challenge here is to minimize overhead."*
- **NIST Requirements and Evaluation Criteria for LW Cryptography, May 24, 2018**: AEAD submissions is that they shall be "*optimized to be efficient for **short messages** (e.g., as short as **8 bytes**)*".

Numerous LW AEAD applications with short messages

- ✓ **Automotive industry**, e.g. CAN-FD automotive protocol
  payload$\leq$ 64 bytes (4 blocks)
- ✓ **5G and LW communication protocols**: Bluetooth, SigFox, LoraWan, and ZigBee protocols
  small status updates (one to few blocks)
- ✓ **Narrowband IoT** (NB-IoT) applications: smart sensors, traffic lights, smart parking, and smart anything
  16 bits $\leq$ transport block size $\leq$ 680 bits
- ✓ **Health applications**
- ✓ **Industrial control systems**

Yet, most AE schemes optimized for long messages!

Minimize cost for short messages

$|A| = a$ and $|M| = m$ blocks

How many **extra** primitive calls to $(a + m)$ for an AEAD?

	GCM	CCM	OCB3	CLOC	TAE
Enc	$m + 1$	$m + 1$	$m + 1$	m	m^*
Auth	$a{+}m{+}1^{\#}$	$a{+}m{+}1$	$a + 1$	$a{+}m{+}1$	$a^* + 1^*$
Extra	$1, m + 1^{\#}$	$m + 2$	2	$m + 1$	1^*

Nr of (BC, TBC*, GF mul$^{\#}$) calls with $m = |M|$ and $a = |A|$.

Minimize cost for short messages

$|A| = a$ and $|M| = m$ blocks

How many **extra** primitive calls to $(a + m)$ for an AEAD?

	GCM	CCM	OCB3	CLOC	TAE
Enc	2	2	2	1	1^*
Auth	$2^\#$	2	1	2	1^*
Extra	$1, 2^\#$	3	2	2	1^*

Nr of (BC, TBC*, GF mul$^\#$) calls with $m = 1$ and $a = 0$.

Minimize cost for short messages

$|A| = a$ and $|M| = m$ blocks

How many **extra** primitive calls to $(a + m)$ for an AEAD?

	GCM	CCM	OCB3	CLOC	TAE
Enc	2	2	2	1	1^*
Auth	$2^\#$	2	1	2	1^*
Extra	$1, 2^\#$	3	2	2	1^*

Nr of (BC, TBC*, GF mul$^\#$) calls with $m = 1$ and $a = 0$.

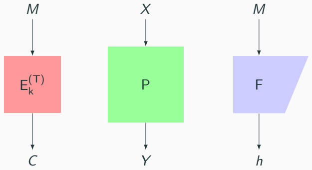
Goal: achieve purely rate-1 AE scheme

A rate-1 AE makes $(a + m)$ primitive calls to authenticate and encrypt (A, M).

The overhead problem

At least 2 extra BC or 1 extra TBC primitive calls

No expanding (inherent to AE) primitives



Question: Are we always using the right primitives for the right applications?

Existing primitives have no inherent AE security and structure

Forkcipher

Results in [a] and [b]

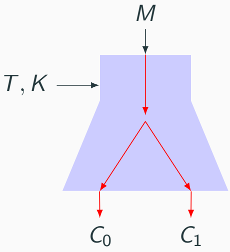
[a] Andreeva et al. "ForkAE", second round candidate in the NIST LW Standardization Process, 2019

[b] Andreeva et al. "Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages", ASIACRYPT 2019

Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

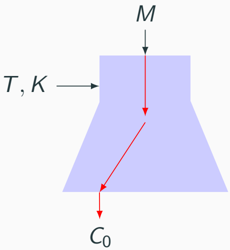
Forward execution



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

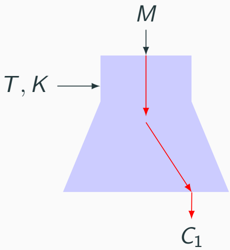
Output selection C_0



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

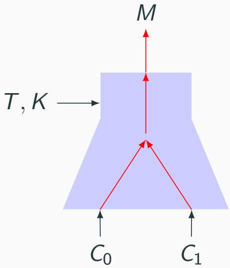
Output selection C_1



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

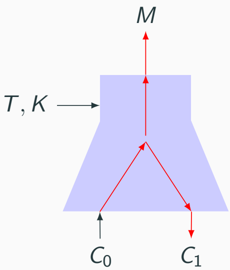
Inversion from either one or both C_0 and C_1



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

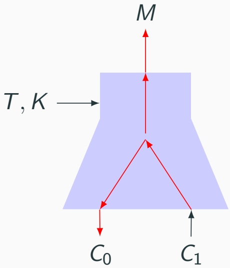
Reconstruction from C_0



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

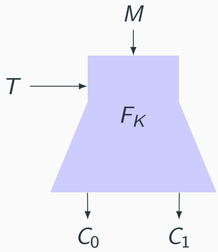
Reconstruction from C_1



Forkcipher: a new primitive

$F : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \mathcal{C}_0 \times \mathcal{C}_1$ with $|\mathcal{M}| = |\mathcal{C}_0| = |\mathcal{C}_1|$

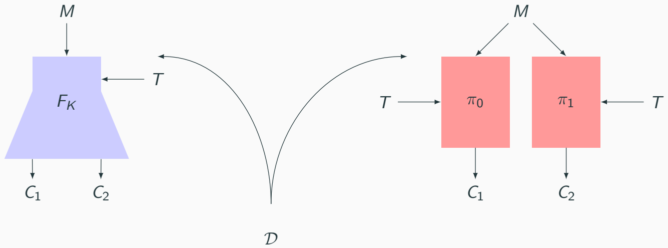
Forkcipher



Minimizes overhead

Under some *appropriate definition* forkcipher securely authenticates and encrypts M.

Forkcipher security



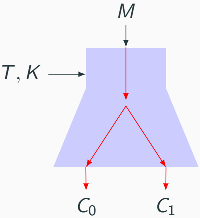
Pseudorandom forked permutation PRFP

Indistinguishability from a pair of random permutations under chosen ciphertext attack

$$Adv^{prfp}(\mathcal{D}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{D}^{F_K} \Rightarrow 1] - \Pr[\mathcal{D}^{\pi_0,\pi_1} \Rightarrow 1].$$

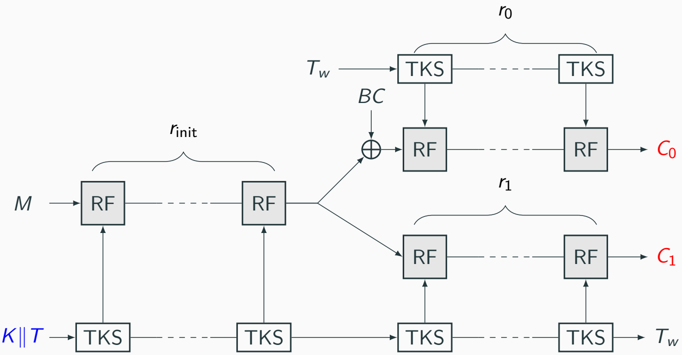
Forkcipher instance: ForkSkinny

iterate-fork-iterate (IFI) generic approach: allows reuse of iterative (T)BC structures



Primitive F	n	t	$t + \|K\|$
ForkSkinny-64-192	64	64	192
ForkSkinny-128-192	128	64	192
ForkSkinny-128-256	128	128	256
ForkSkinny-128-288	128	128	288

ForkSkinny



RF: round function; TKS: tweakey schedule; BC: branch constant; r_{init}, $r_0 = r_1$: nr rounds before and after fork.

Primitive	block	tweak	tweakey	r_{init}	r_0	r_1
ForkSkinny-64-192	64	64	192	17	23	23
ForkSkinny-128-192	128	64	192	21	27	27
ForkSkinny-128-256	128	128	256	21	27	27
ForkSkinny-128-288	128	128	288	25	31	31

ForkSkinny cryptanalysis

- Inherits many of the SKINNY results
- Our cryptanalysis:
  - ✓ truncated and impossible differential
  - ✓ boomerang
  - ✓ meet-in-the-middle
  - ✓ integral and algebraic
- Forkcipher-specific:
  - ✓ reconstruction attacks
  - ✓ branch constant and forking point
- **Third party cryptanalysis** by A. Bariant et al. **at ToSC 2020**: the best attacks on Skinny can be extended to 1 more round for most ForkSkinny variants, and at most 3 more rounds for ForkSkinny-128-256.

ForkSkinny in modes of operation

Question

Can we construct secure AEAD modes of rate-1?

ForkSkinny in modes of operation

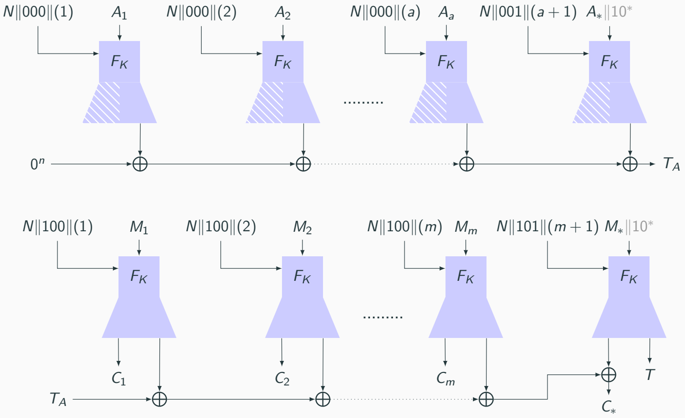
Question

Can we construct secure AEAD modes of rate-1?

Yes

we design parallel **PAEF** and **rPAEF** and sequential **SAEF** rate-1 modes with ForkSkinny.

Parallel AE from a Forkcipher: PAEF

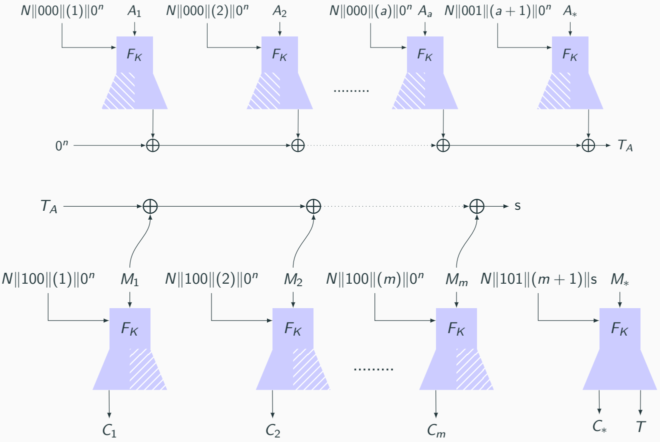


n-bit AE security

$$Adv^{privacy}_{PAEF}(\mathcal{A}) \leq Adv^{PRFP}_{F}(\mathcal{D})$$

$$Adv^{auth}_{PAEF}(\mathcal{A}) \leq Adv^{PRFP}_{F}(\mathcal{D}) + \frac{q_v \cdot 2^n}{(2^n - 1)^2}$$

Reduced Parallel AE from a Forkcipher: rPAEF

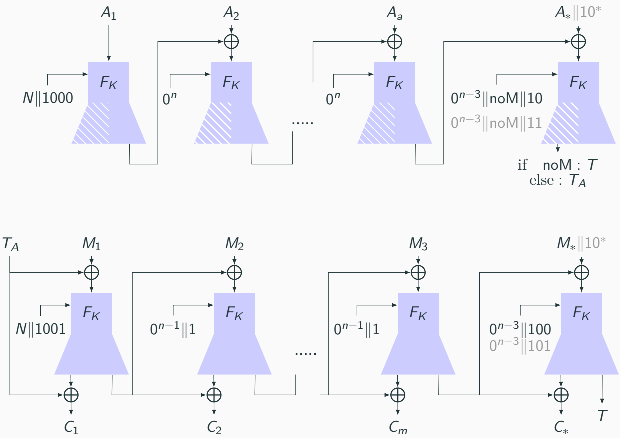


n-bit AE security

$$Adv_{rPAEF}^{privacy}(\mathcal{A}) \leq Adv_{F}^{PRFP}(\mathcal{D})$$

$$Adv_{rPAEF}^{auth}(\mathcal{A}) \leq Adv_{F}^{PRFP}(\mathcal{D}) + \frac{q_v \cdot 2^n}{(2^n - 1)^2}$$

Sequential AE from a Forkcipher: SAEF



$n/2$-bit AE security

$$Adv_{SAEF}^{privacy}(\mathcal{A}) \leq Adv_{F}^{PRFP}(\mathcal{D}) + 2\frac{(\sigma - q)^2}{2^n}$$

$$Adv_{SAEF}^{auth}(\mathcal{A}) \leq Adv_{F}^{PRFP}(\mathcal{D}) + \frac{2(\sigma - q + 1)^2}{2^n} + \frac{\sigma(\sigma - q)}{2^n} + \frac{q_v(q+2)}{2^n}$$

Software implementation

Results in [a] and [b]

[a] https://github.com/byt3bit/forkae

[b] A. Deprez Master Thesis 2020, Optimized software implementations for ForkAE

Portable SW implementations

- Efficient and constant-time ForkAE SW implementations at https://github.com/rweather/lightweight-crypto

	Arm Cortex-A9			Arm Cortex-M0		
	cycles/B	ROM (B)	RAM (B)	cycles/B	ROM (B)	RAM (B)
PAEF-ForkSkinny-64-192	1669	3067	107	4002	2067	107
PAEF-ForkSkinny-128-192	1072	3187	161	2457	2251	161
PAEF-ForkSkinny-128-256	1074	3219	169	2458	2247	169
PAEF-ForkSkinny-128-288	1408	3483	189	3408	2541	189
SAEF-ForkSkinny-128-192	1075	3015	161	2475	2187	161
SAEF-ForkSkinny-128-256	1076	3043	169	2476	2173	169

- Decryption can further improved with preprocessed TKS https://github.com/ArneDeprez1/ForkAE-SW
  - ✓ 38% less clock cycles
  - ✓ 1kB smaller ROM size
  - ✓ 252-696 bytes higher RAM usage

Table-based SW implementations

- Suitable for platforms without a cache, e.g. Cortex-M0
- Efficient implementations by combining different steps of the round function in XOR of table-lookups.
  1 round = 18 lookups + 19 XOR
- SW performance on Arm Cortex-M0 compared to portable implementations:
  - ✓ Encryption: up to 20% faster
  - ✓ Decryption: up to 25% faster
  - ✓ Increased memory cost for storing 4 tables of 1kB each
  - ✓ Memory impact can be reduced by using only 1 table of 1kB without significant loss of performance
- https://github.com/ArneDeprez1/ForkAE-SW

Neon SIMD SW implementations

- Platforms with SIMD hardware extensions can exploit data-level parallelism in ForkSkinny primitive
  - ✓ RF-parallelism: S-box in parallel for every cell
  - ✓ Fork parallelism: compute 2 branches parallel
- Implementation for Neon SIMD on Arm Cortex-A9 https://github.com/ArneDeprez1/ForkAE-SW
- 128-bit instances (S-box parallelism) :
  - ✓ 30% less clock cycles
  - ✓ 0.5 kB reduction in ROM size
  - ✓ RAM size equal
- 64-bit instance (S-box + fork parallelism):
  - ✓ 29 % less clock cycles
  - ✓ ROM size approx. equal
  - ✓ RAM size increased

Hardware implementation

Results in [a] and [b]

[a] T. Purnal et al. “What the Fork: Implementation Aspects of Forkcipher”, NIST LW Workshop 2019

[b] J. Pittevils Master Thesis 2020, “Low-area Optimized Hardware Implementations for ForkAE”

HW comparison [1]

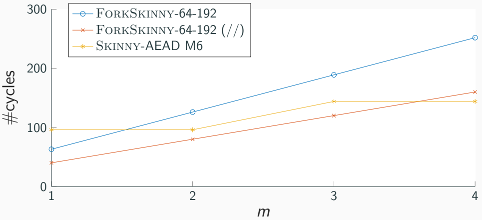
Implementation (round-based)	Area [GE] E-only	Area [GE] EncDec	Number of cycles for encrypting $(a+m)$ **64-bit** blocks						
			$a=0$			$a=1$			General
			$m=1$	$m=2$	$m=3$	$m=0$	$m=1$	$m=2$	
Sk-AEAD M6	8095	9458	96	96	144	48	96	**96**	$48(\lceil \frac{a}{2} \rceil + \lceil \frac{m}{2} \rceil + 1)$
PAEF-64-192	5034	6704	**63**	126	189	**40**	103	166	$40(a+1.575m)$
PAEF-64-192 (//)	5500	7422	**40**	**80**	**120**	**40**	**80**	120	$40(a+m)$

Implementation (round-based)	Area [GE] E-only	Area [GE] EncDec	Number of cycles for encrypting $(a+m)$ **128-bit** blocks						
			$a=0$			$a=1$			General
			$m=1$	$m=2$	$m=3$	$m=0$	$m=1$	$m=2$	
Romulus-N3	6288	6406	96	144	192	**48**	**96**	**144**	$48(\lceil \frac{a-1}{1.75} \rceil + m + 1)$
SAEF-128-192	7197	9203	**75**	150	225	**48**	123	198	$48(a+1.562m)$
SAEF-128-256	7740	9999	**75**	150	225	**48**	123	198	$48(a+1.562m)$
SAEF-128-192 (//)	7713	10804	**48**	**96**	**144**	**48**	**96**	**144**	$48(a+m)$
SAEF-128-256 (//)	8288	11646	**48**	**96**	**144**	**48**	**96**	**144**	$48(a+m)$
Sk-AEAD M5	8746	10109	96	144	192	96	144	192	$48(a+m+1)$
PAEF-128-192 (//)	8020	11112	**48**	**96**	**144**	**48**	**96**	**144**	$\mathbf{48(a+m)}$
PAEF-128-256 (//)	8745	12103	**48**	**96**	**144**	**48**	**96**	**144**	$\mathbf{48(a+m)}$
rPAEF (aggr.) (//)	8203	na	**87**	**135**	**183**	**48**	**135**	**183**	$\mathbf{48(a+m)+39}$
Romulus-N1	7018	7136	112	168	224	**56**	**112**	**168**	$56(\lceil \frac{a-1}{2} \rceil + m + 1)$
Sk-AEAD M1-2	9966	12363	112	168	224	112	168	224	$56(a+m+1)$
PAEF-128-288	9274	11705	**87**	174	261	**56**	**143**	230	$56(a+1.553m)$
PAEF-128-288 (//)	10141	13697	**56**	**112**	**168**	**56**	**112**	**168**	$56(a+m)$
rPAEF (const.) (//)	8178	na	**87**	**143**	**199**	**56**	**143**	**199**	$56(a+m)+31$

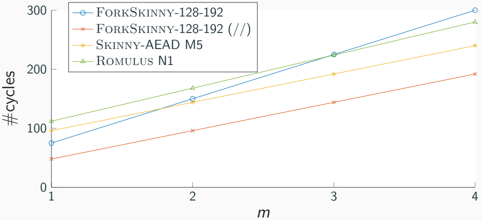
(//) ForkSkinny-level parallelism; (SFF/MUX/XOR/NAND with 7.67/2.33/2/1 GE)

[1] T. Purnal et al. "What the Fork: Implementation Aspects of Forkcipher", NIST LW Workshop 2019

Message sizes

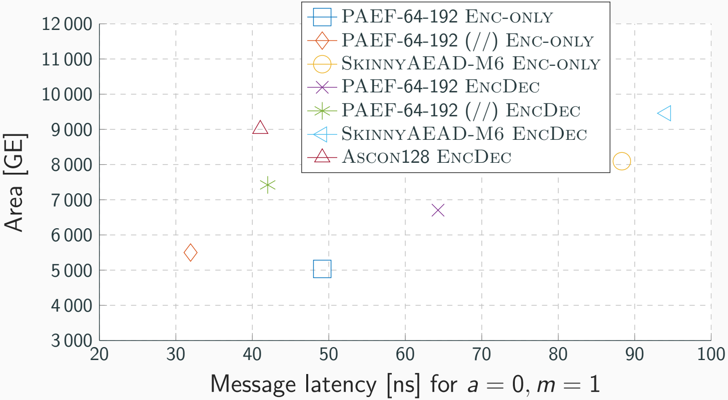


64-bit blocks



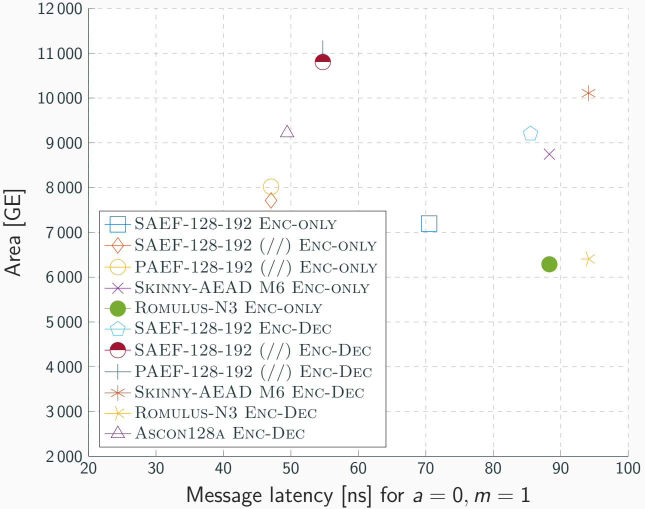
128-bit blocks

Speed-area exploration (64 bits)



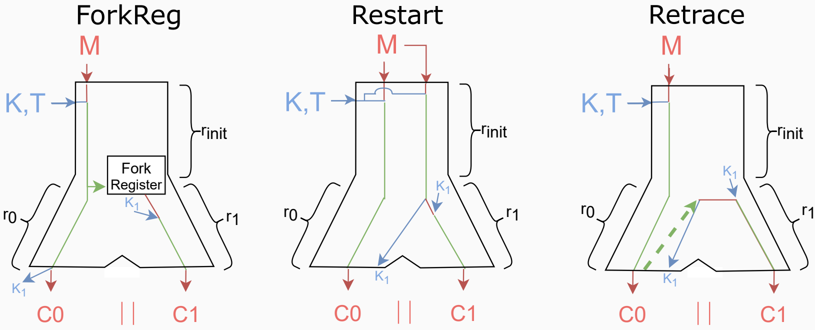
64-bit blocks $(a = 0,\ m = 1)$

Speed-area exploration (128 bits)



128-bit blocks $(a = 0,\ m = 1)$

Low-area ForkSkinny HW architectures[2]



[2] "Low-area Optimized Hardware Implementations for ForkAE" Master Thesis 2020 by Jowan Pittevils

Word-based architectures results

Algorithm	Architecture	Area [GE]	Area [GE]	Delay [ns]	Cycles	T'put enc [kbit/s]	T'put dec [kbit/s]
		E-only	EncDec			E-only	EncDec
Skinny-64-192		2448	3754	1.04	872	7.33	3.7
Skinny-128-256		3525	5499	0.94	1040	12.29	6.2
Skinny-128-384		4680	7157	0.93	1208	10.58	5.4
ForkSkinny-64-192	Restart	**2718**	na	1.15	2218	2.8-5.7	na
ForkSkinny-128-256	Restart	**3917**	na	1.19	2638	4.8-9.7	na
ForkSkinny-128-288	Restart	**4567**	na	1.39	3058	4.2-8.4	na
ForkSkinny-64-192	Retrace	3867	**3894**	1.67	2328	2.7-5.5	2.3-**4.6**
ForkSkinny-128-256	Retrace	5648	**5685**	1.73	2748	4.7-9.3	3.9-**7.8**
ForkSkinny-128-288	Retrace	6645	**6650**	1.92	3168	4.0-8.1	3.3-**6.7**
ForkSkinny-64-192	ForkReg	3243	4470	0.93	1362	4.7-**9.3**	2.3-**4.6**
ForkSkinny-128-256	ForkReg	4977	6787	1.4	1614	7.9-**15.9**	3.9-**7.8**
ForkSkinny-128-288	ForkReg	5629	7795	1.23	1866	6.8-**13.7**	3.3-**6.7**

(SFF/MUX/XOR/NAND with 7.67/2.33/2/1 GE)

- ForkSkinny area very close to Skinny for enc.only (Restart)
- ForkSkinny area-usage very close to Skinny for enc./dec. (Retrace)
- Serial ForkSkinny better throughput than serial Skinny for short messages, both for enc.only (Forkreg) and enc./dec. (Retrace)

Our ForkAE Design

Secure

- ✓ Well-analysed: based on SKINNY
- ✓ Provably secure: PAEF, rPAEF, SAEF

Efficient

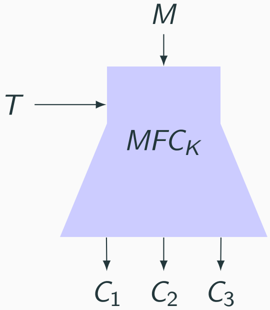
- ✓ **Excellent performance for small messages**
- ✓ Excellent throughput per area in HW
- ✓ Inherits LW implementation features of SKINNY
- ✓ Multiple trade-offs in speed-resource design space

Flexible

key size: 128 bits and variable block, nonce, tag sizes

Generalization: Multi-Forkcipher

Multi-Forkcipher with $s = 3$



Multi-Forkcipher

Forward algorithm

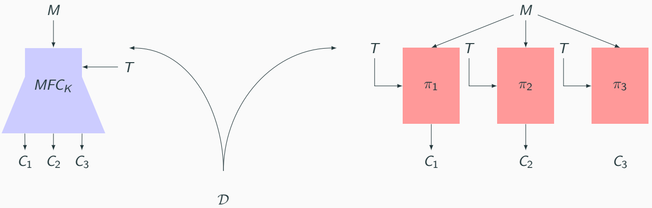
$$MFC_s : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \times 2^{\{1,2,\ldots,s\}} \rightarrow \bigcup_{e=1}^{s} \{0,1\}^{en}$$

and the backward (or the inversion) algorithm

$$MFC_s^{-1} : \mathcal{K} \times \mathcal{T} \times \mathcal{C} \times \{1,2,\ldots,s\} \times 2^{\{\mathrm{i},1,2,\ldots,s\}} \rightarrow \bigcup_{e=1}^{s} \{0,1\}^{en}$$

- When $s = 1$, then MFC = TBC
- When $s = 2$, then MFC = FC

MFC security



Pseudorandom multi-fork permutation

Indistinguishability from s-tuple of independent random permutations

$$Adv^{prtmfp}(\mathcal{D}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{D}^{MFC_K^s} \Rightarrow 1] - \Pr[\mathcal{D}^{\pi_1,\ldots,\pi_s} \Rightarrow 1].$$

MFC Advantages and Uses

- MFC with authenticated encryption MFC-AE brings in benefits for even longer messages, i.e. depending on the input message, adjustments possible to number of branches needed.
- Replace a TBC with MFC = allows for larger pseudorandom string output generation with strong security benefits.

Question: where can we effectively replace TBC with an MFC and gain in efficiency?

Multi-forkciphers for Encryption [a]

[a]joint work with A. Singh Bhati, B. Preneel, and D. Vizár

Goal

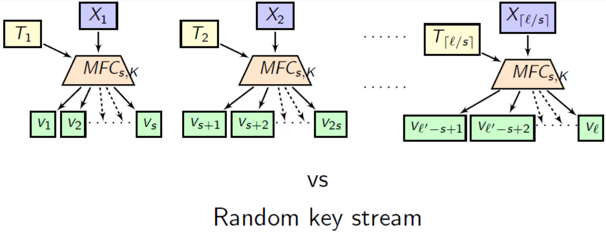
Study the security and efficiency of MFCs in a CTR-style mode motivated by:

- MFC uses in forward-only direction
- Possibility to obtain BBB security and/or graceful security degradation with nonce repetitions (like CTRT: CounTeR in Tweak [Peyrin and Seurin'15])
- Systematic investigation of tweakable CTR variants with random IV and/or nonce N
- Provide security/efficiency analysis of CTR-style mode variants with MFC

Tweakable CTR framework

Tweakable CTR framework

- Tweakable CTR (TCTR) takes a *sequence* of **tweak-input** pairs, and generates key stream by applying a MFC to each pair.

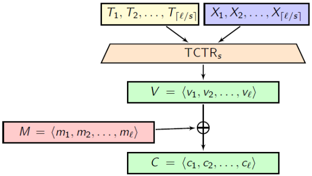


- X_i-s and T_i-s generated with N and/or a random IV and/or a counter.

Generic CTR Encryption Mode

Generic CTR Encryption Mode

GCTR is defined via TCTR_s



GCTR_s Encryption: $\mathcal{E}_K(N, R, M)$

for $j \leftarrow 1$ **to** $\lceil \ell/s \rceil$ **do**
$\quad X_j = f_X(N, R, j)$
$\quad T_j = f_T(N, R, j)$
$T, X \leftarrow \langle T_1, \ldots, T_{\lceil \ell/s \rceil} \rangle, \langle X_1, \ldots, X_{\lceil \ell/s \rceil} \rangle$
$V \leftarrow \mathrm{TCTR}_{s,K}(T, X)$
$C \leftarrow M \oplus V$
return C

- $f_X(N, R, j)$ and $f_T(N, R, j)$ are input-tweak (X_j, T_j) generating functions
- exhaustive study of all f_X and f_T with $\{\|, \oplus, \mathrm{copy}\}$
- varying security (BB, BBB, NMR, NAE)
- usage constraints (tradeoff of the size of the parameters)
- example: counter in tweak CTRT has $X_j = N$ and $T_j = R \oplus j$

Security Model: Nonce and IV-based Encryption (nivE)

- An nivE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{R} \times \mathcal{M} \to \{0,1\}^*$ and $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{R} \times \{0,1\}^* \to \mathcal{M}$
- Let $\mathcal{E}^\$: \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \mathcal{R} \times \{0,1\}^*$ denote the randomized encryption algorithm, which internally samples an $R \xleftarrow{\$} \mathcal{R}$, computes $C \leftarrow \mathcal{E}(K, N, R, M)$ and returns R, C. We further let $\mathcal{E}^\$_K(N, M) = \mathcal{E}^\$(K, N, M)$.

nivE indistinguishability of ciphertexts from random strings in a chosen plaintext attack for a nonce respecting adversary A is defined as:

$$Adv_\Pi^{nivE} = \left| \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}^\$_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{\mathrm{Rand}^\$(\cdot,\cdot)} \Rightarrow 1\right] \right|$$

Efficiency

The cost to compute 1 keystream n-bit block:

with ForkSkinny GCTR: **0.8** of cost with SKINNY GCTR

Efficiency

The cost to compute 1 keystream n-bit block:

with ForkSkinny GCTR: **0.8** of cost with SKINNY GCTR

ForkSkinny GCTR becomes **twice** faster than SKINNY GCTR
using primitive level parallelism [NIST workshop'19]

Conclusions and open question

This work:

- New ForkSkinny forkcipher primitive
- New (multi-)forkcipher formalism (M)FTPRP
- ForkSkinny cryptanalysis
- New (multi-)forkcipher modes and proofs

Future work:

- New multi-forkcipher instantiations
- New multi-forkcipher paradigms
- Multi-forkcipher applications beyond AE and encryption
- Multi-forkcipher side-channel, quantum attacks, etc. resistance

ForkAE

ForkAE with ForkSkinny is a NIST second round candidate

Full version: https://eprint.iacr.org/2019/1004.pdf

Optimized implementations

https://www.esat.kuleuven.be/cosic/forkae/

https://github.com/ArneDeprez1/ForkAE-SW

https://github.com/byt3bit/forkae/

ForkAE

ForkAE with ForkSkinny is a NIST second round candidate

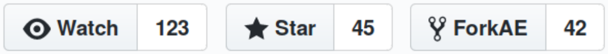
Full version: https://eprint.iacr.org/2019/1004.pdf

Optimized implementations

https://www.esat.kuleuven.be/cosic/forkae/

https://github.com/ArneDeprez1/ForkAE-SW

https://github.com/byt3bit/forkae/



Thank you!

elean@dtu.dk