Optimized Software Implementations for the Lightweight Encryption Scheme ForkAE

Arne Deprez[1], Elena Andreeva[2], Jose Maria Bermudo Mera[3], Angshuman Karmakar[3], and Antoon Purnal[3]

[1] arne.deprez1@gmail.com
[2] Alpen-Adria University, Austria elena.andreeva@aau.at
[3] imec-COSIC, KU Leuven. Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

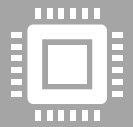
Introduction

ForkAE

AEAD: Authenticated Encryption with Associated Data

ForkSkinny primitive

2nd round NIST LWC candidate



Lightweight Cryptography

Constrained devices → current crypto too heavy

NIST LWC competition for new primitives/protocols

IoT, RFID, smart cards, automotive,…

Optimized Software Implementations

Cross platform & platform specific

Different devices

Resistance against timing attacks

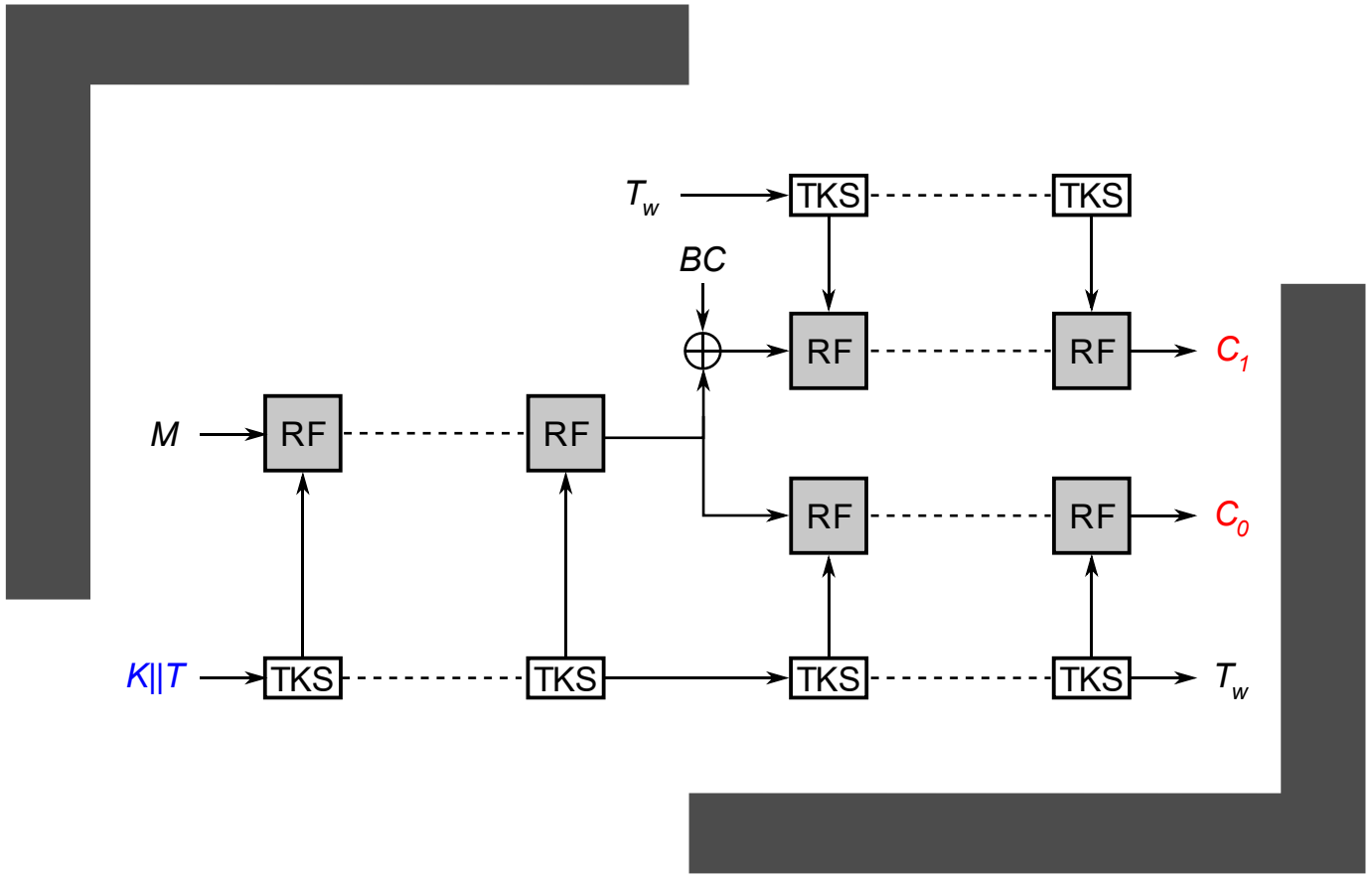
Contributions

- Analyze existing portable ForkAE implementations
- Optimize **decryption**
  - Reduce latency
  - Reduce code size
- Platform-specific optimizations:
  - Platforms where (cache-) timing attacks are not applicable ➔ **table-lookups**
  - Platforms with SIMD parallel hardware extensions ➔ exploit data-level **parallelism**
- **Benchmark** performance of implementations on two platforms
  - ARM Cortex-M0
  - ARM Cortex-A9
- **Compare** with other SKINNY-based schemes

Overview

- Introduction
- Contributions
- ForkAE
- Portable implementations
- Table-based implementations
- Parallel implementations
- Conclusion

ForkAE



- AEAD from new primitive: ForkSkinny
- Uses SKINNY round function but *forks* after certain amount of rounds
- Produces two independent permutations but with reduced computational cost
- Designed for encryption of small messages

ForkAE

- ForkAE uses ForkSkinny in PAEF/SAEF modes of operation
  - 1 ForkSkinny call with 1 output per associated data block
  - 1 ForkSkinny call with 2 outputs (#rounds x 1.6) per message block
- Standard block cipher modes of operation (e.g. GCM):
  - Fixed cost ➔ extra block function call(s) for processing nonce or generating tag
- Because of double output ➔ no fixed cost for ForkAE ➔ better performance for smallest messages

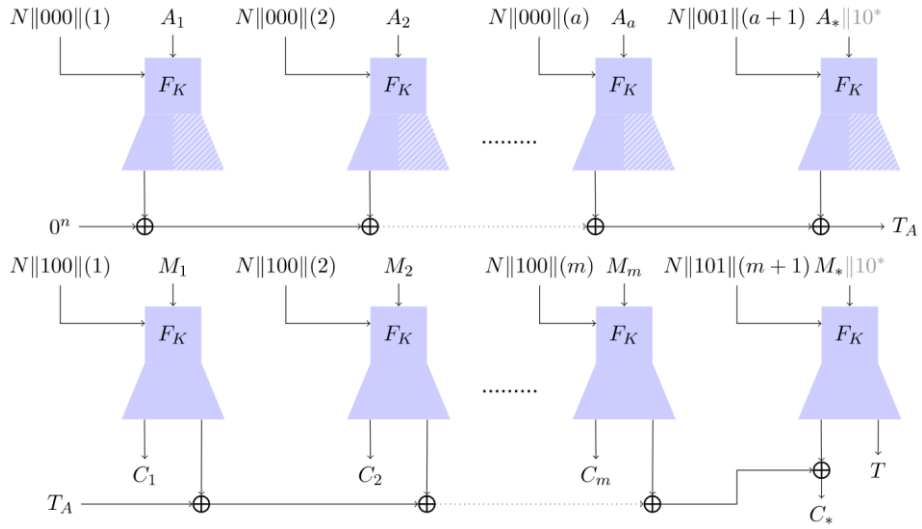


Image: [ForkAE website]

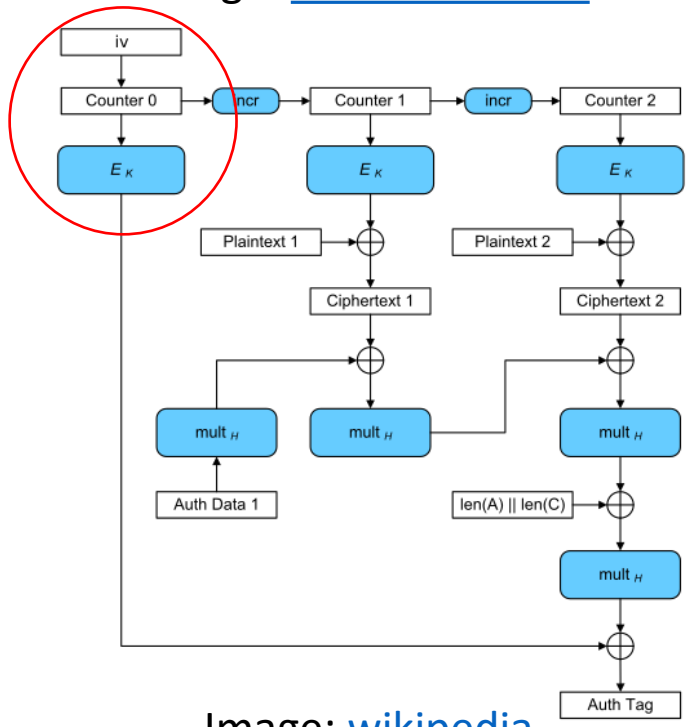


Image: [wikipedia]

ForkAE: PAEF (Parallel AEAD from a Forkcipher)

The PAEF mode achieves full n-bit security and processes associated data A and plaintext M with a nonce N as following:

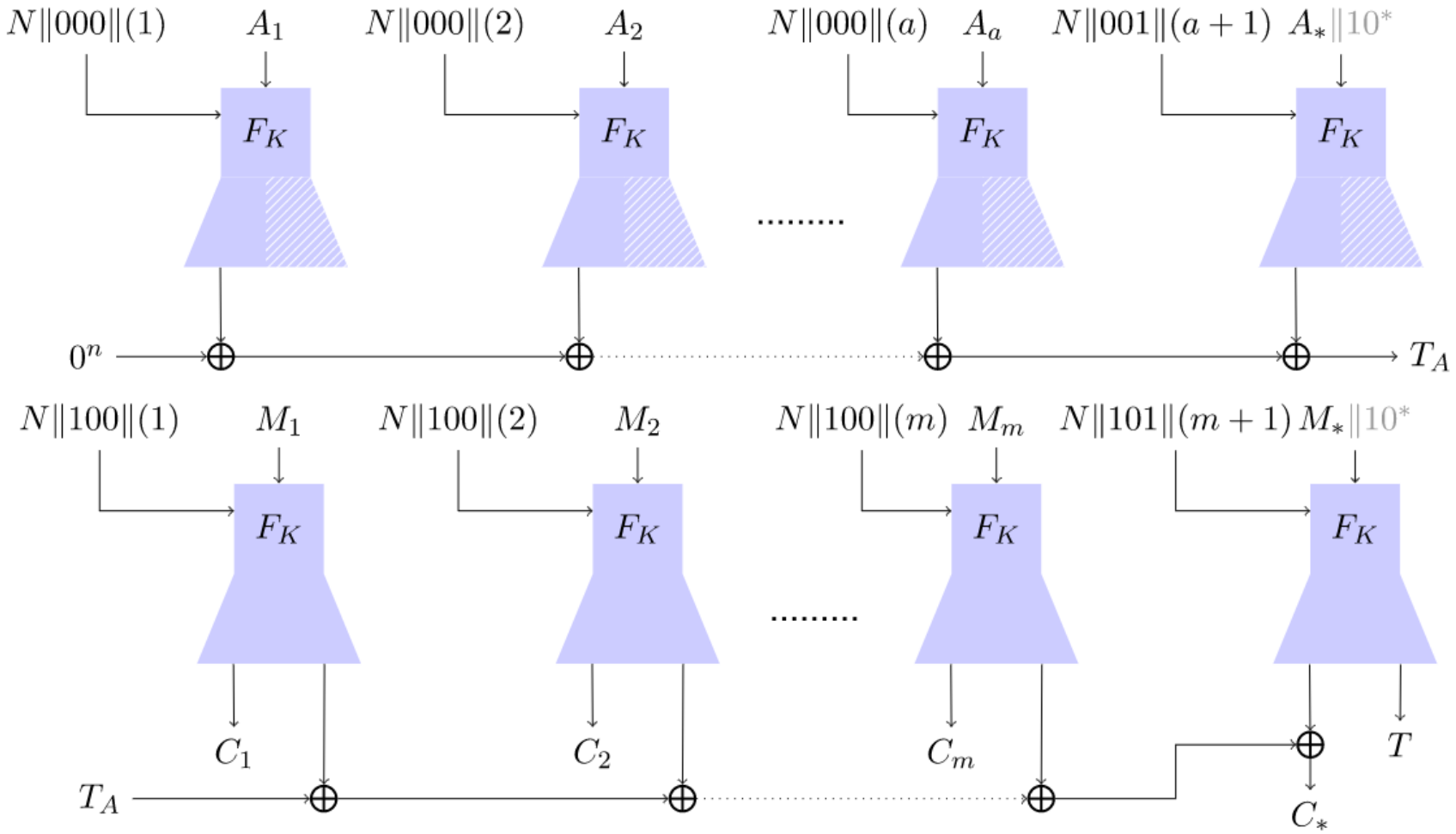


Image: [ForkAE website]

ForkAE: SAEF (Sequential AEAD from a Forkcipher)

The SAEFmode achieves n/2-bit security (with a smaller state than PAEF) and processes associated data A and plaintext M with a nonce N as following:

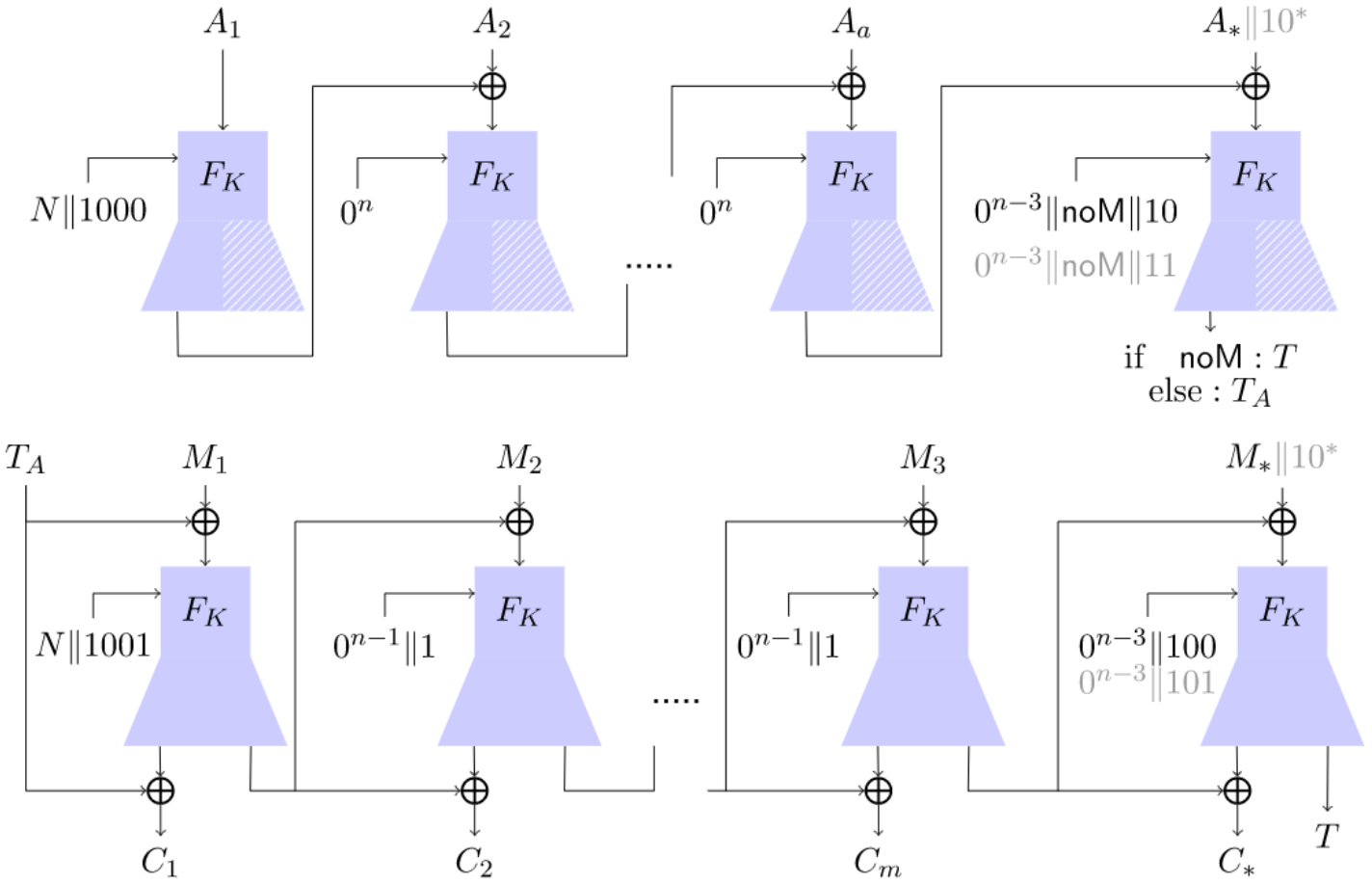


Image: [ForkAE website]

Overview

- Introduction
- Contributions
- ForkAE
- Portable implementations
- Table-based implementations
- Parallel implementations
- Conclusion

Portable implementations

- Low latency ➔ focus on primitive
- Optimized
  - Memory cost (ROM)
  - Memory usage (RAM)
  - Speed (clock cycles)
- Constant time
  - Resistance to (cache-)timing attacks
  - No secret-dependent table lookups
  - Verification using the dudect tool

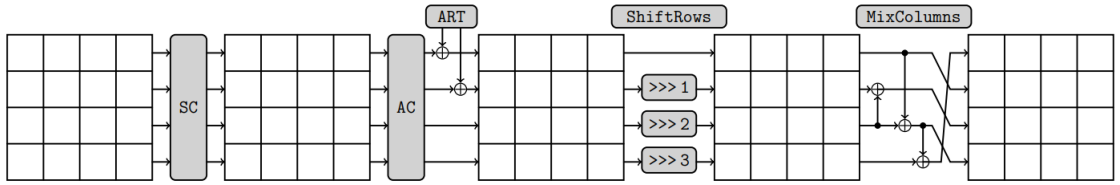


Figure 1: The SKINNY round function applies five different transformations: SubCells (SC), AddConstants (AC), AddRoundTweakey (ART), ShiftRows (SR) and MixColumns (MC).

Images: SKINNY specification

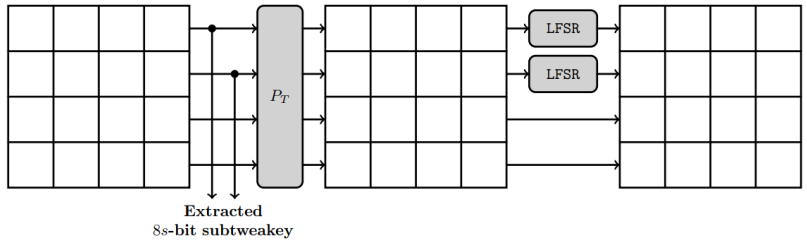


Figure 3: The tweakey schedule in SKINNY. Each tweakey word $TK1$, $TK2$ and $TK3$ (if any) follows a similar transformation update, except that no LFSR is applied to $TK1$.

Portable implementations

- Implementation by Rhys Weatherley https://rweather.github.io/lightweight-crypto/
- **32-bit** implementation:
  - State saved per 32-bit **row**
  - All steps calculated on entire row
- **S-box** calculated

  ➔ no more table-look ups

  ➔ resistance against timing attacks

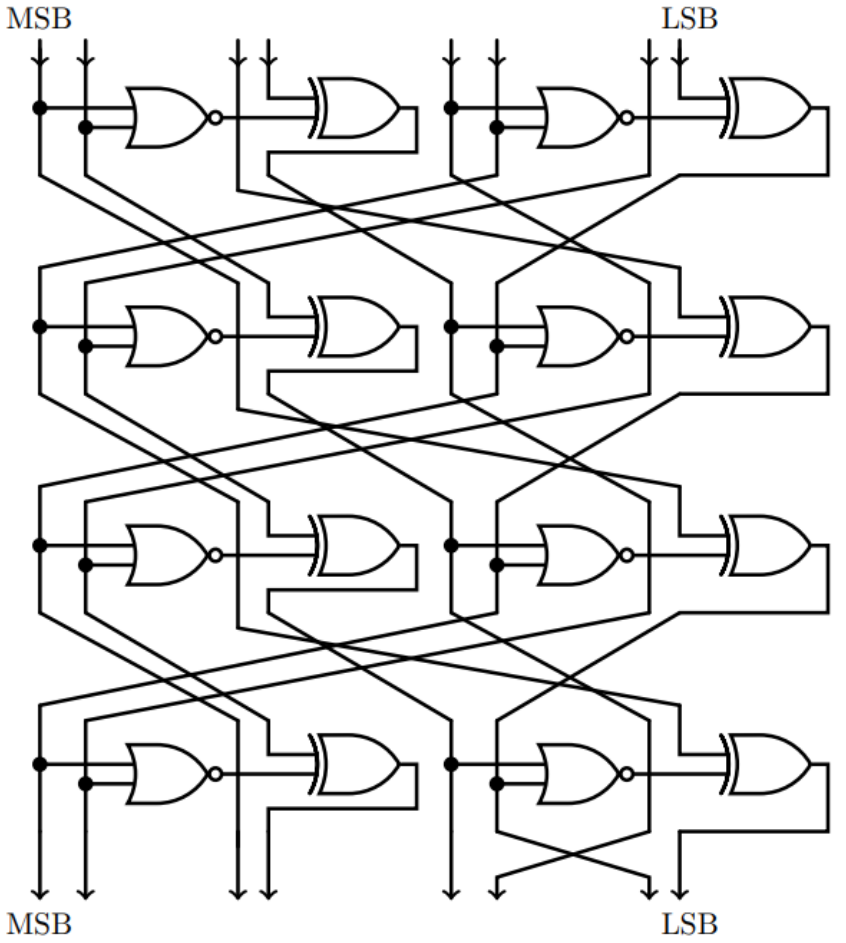


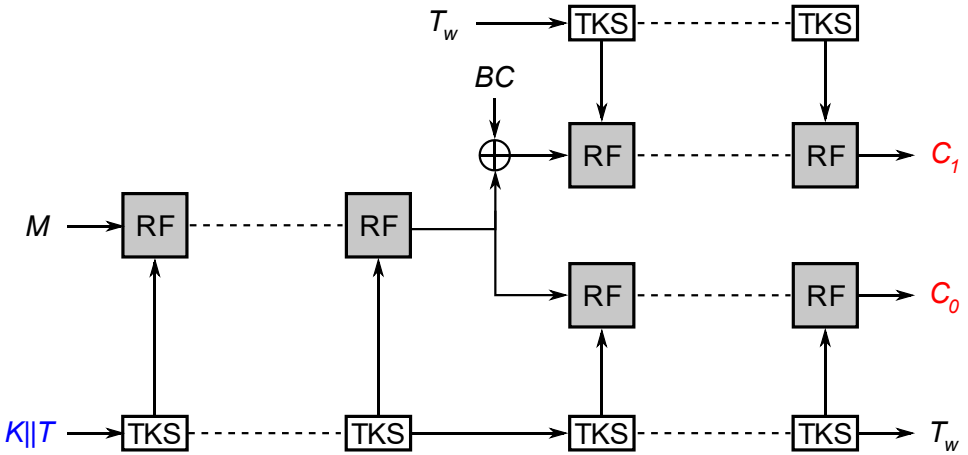
Figure 2: Construction of the Sbox $\mathcal{S}_8$.

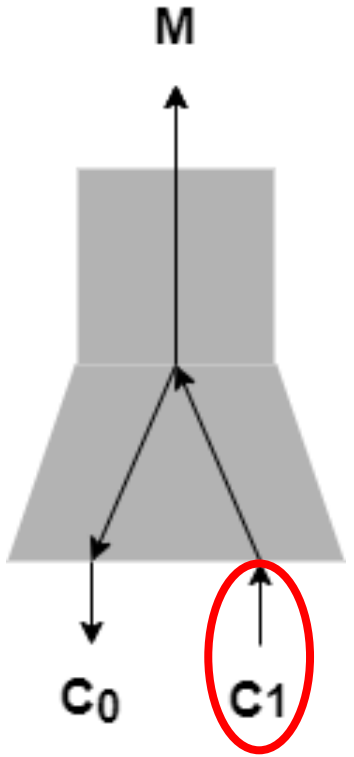
Image:
SKINNY specification

Portable implementations

Decryption

- In existing implementations
  ➔ fast-forward TKS + reverse
- Optimization
  ➔ preprocess TKS once + store
- Higher performance
  - 17 – 38 % speed-up
- Lower code size (ROM)
  - Reduction up to 1kB
- Higher memory usage (RAM)
  - 252-696 bytes





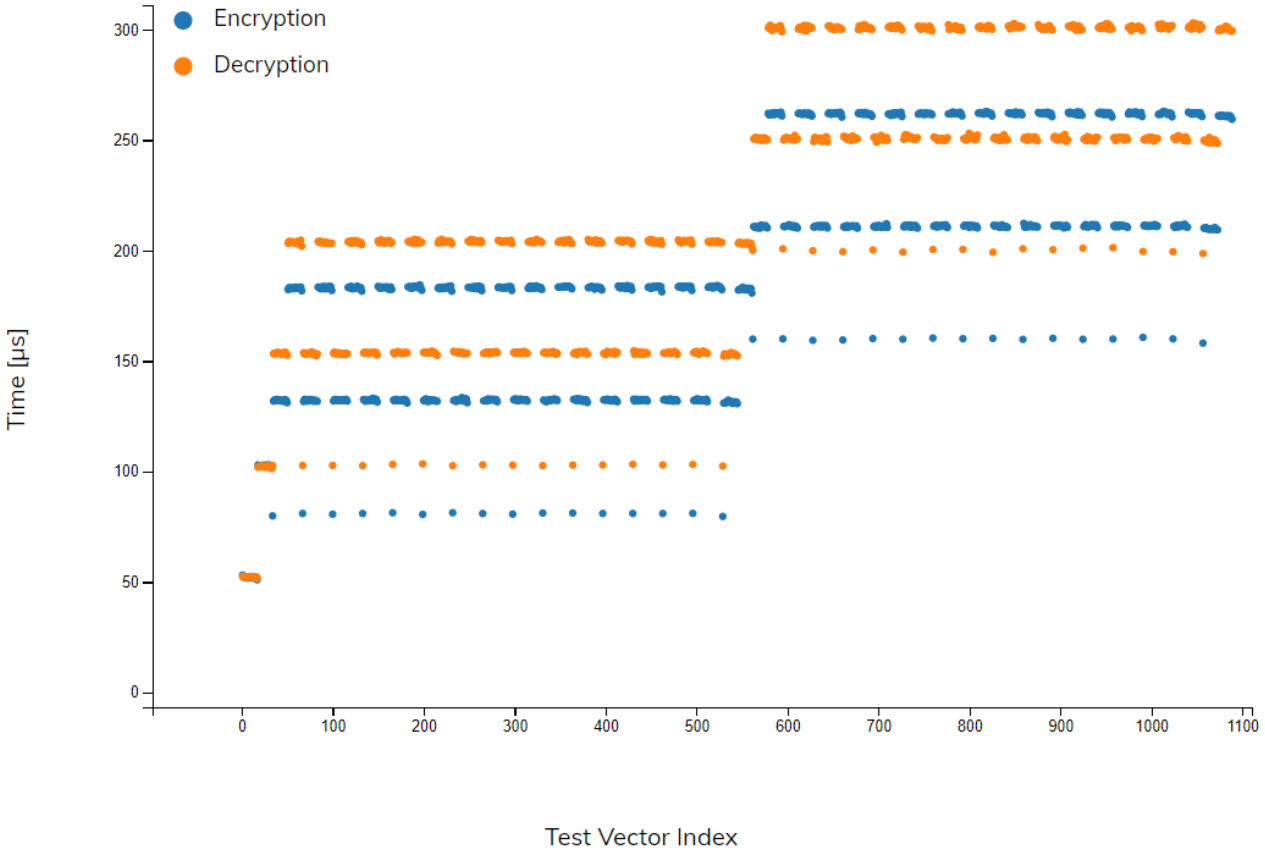
Encryption	Cortex-A9			Cortex-M0		
	c/B	ROM	RAM	c/B	ROM	RAM
PAEF-FS-64-192	1669	3067	107	4002	2067	107
PAEF-FS-128-192	1072	3187	161	2457	2251	161
PAEF-FS-128-256	1074	3219	169	2458	2247	169
PAEF-FS-128-288	1408	3483	189	3408	2541	189
SAEF-FS-128-19	1075	3015	161	2475	2187	161
SAEF-FS-128-256	1076	3043	169	2476	2173	169
Decryption						
PAEF-FS-64-192	2596	3999	140	6767	2819	140
PAEF-FS-128-192	1397	3735	210	3562	2715	210
PAEF-FS-128-256	1393	3767	218	3563	2707	218
PAEF-FS-128-288	2001	4399	254	5305	3243	254
SAEF-FS-128-192	1398	3599	210	3580	2771	210
SAEF-FS-128-256	1397	3603	218	3579	2757	218
Decryption (preprocessed tweakey schedule)						
PAEF-FS-64-192	1684	2927	392	4167	1955	392
PAEF-FS-128-192	1165	3131	810	2970	2303	810
PAEF-FS-128-256	1162	3163	818	2971	2295	818
PAEF-FS-128-288	1491	3363	950	4010	2571	950
SAEF-FS-128-192	1166	2995	810	2988	2359	810
SAEF-FS-128-256	1164	2999	818	2987	2345	818

Portable implementations

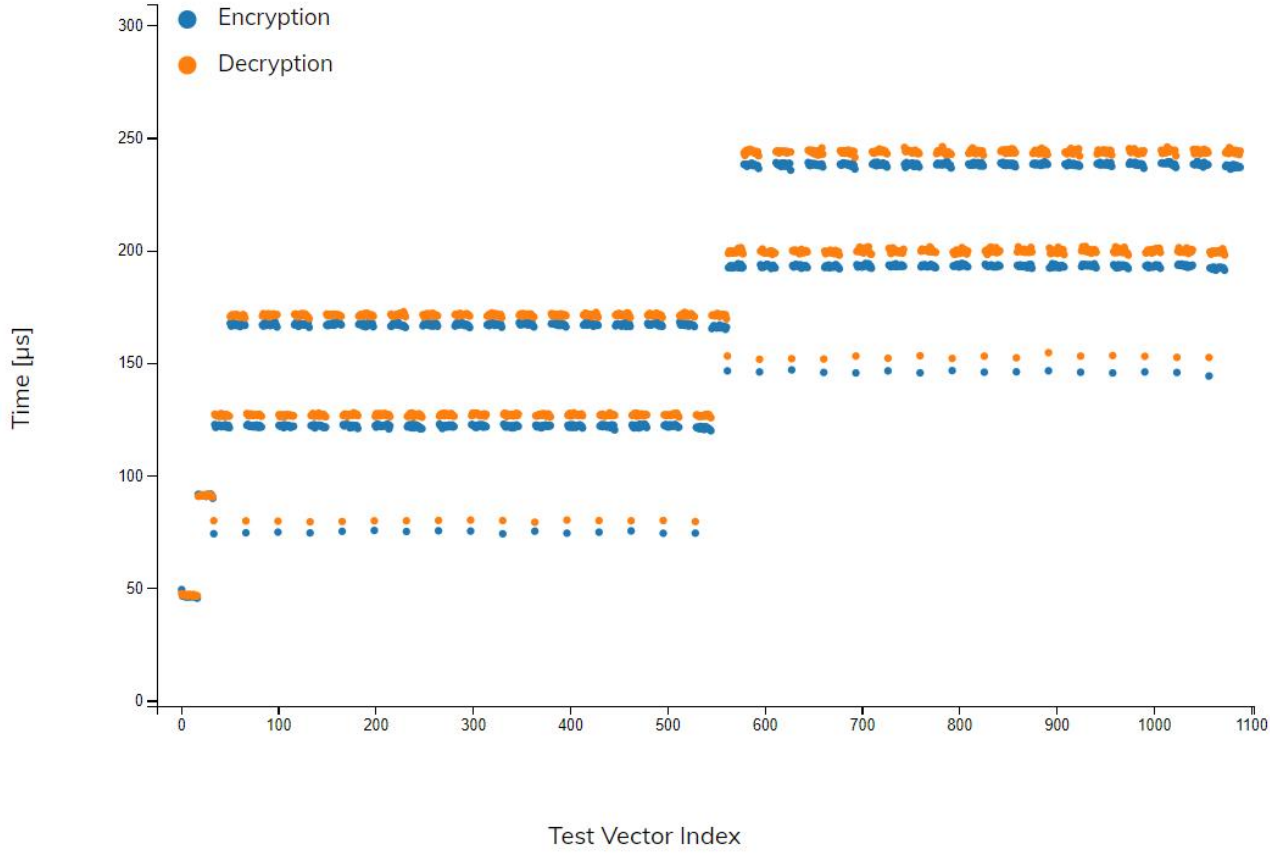
- Results on Cortex-A9 and Cortex-M0
  - Speed expressed in average cycles/byte
  - Code size (ROM) and memory usage (RAM) in bytes
- After optimization
  - Code size reduction
  - Speed-up
  - Higher RAM usage
- Difference between decryption and encryption reduced

Portable implementations

- **Without** preprocessed TKS



- **With** preprocessed TKS



- Results on STM32F7 from https://lwc.las3.de/

Overview

- Introduction
- Contributions
- ForkAE
- Portable implementations
- Table-based implementations
- Parallel implementations
- Conclusion

Table-based implementations

- Similar to AES table look-up implementation
- Calculate effect of round function on **column** (32-bit) of internal state
- Combine steps in table look-up
- Addition of key material before MixColumns
  ➔ More difficult than AES

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

SubCells	AddConstants	AddRoundTweakey
$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} S[a_{0j}] \\ S[a_{1j}] \\ S[a_{2j}] \\ S[a_{3j}] \end{pmatrix}$	$j{=}0 \begin{pmatrix} b_{00} \\ b_{10} \\ b_{20} \\ b_{30} \end{pmatrix} = \begin{pmatrix} b_{00} \\ b_{10} \\ b_{20} \\ b_{30} \end{pmatrix} \oplus \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ 0 \end{pmatrix}$ $j{=}2 \begin{pmatrix} b_{02} \\ b_{12} \\ b_{22} \\ b_{32} \end{pmatrix} = \begin{pmatrix} b_{02} \\ b_{12} \\ b_{22} \\ b_{32} \end{pmatrix} \oplus \begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} c_{0j} \\ c_{1j} \\ c_{2j} \\ c_{3j} \end{pmatrix} = \begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} \oplus \begin{pmatrix} TK_{0j} \\ TK_{1j} \\ 0 \\ 0 \end{pmatrix}$ $TK_{ij} = TK1_{ij} \oplus TK2_{ij} \oplus TK3_{ij}$

ShiftRows	MixColumns
$\begin{pmatrix} d_{0j} \\ d_{1j} \\ d_{2j} \\ d_{3j} \end{pmatrix} = \begin{pmatrix} c_{0j} \\ c_{1(j-1)} \\ c_{2(j-2)} \\ c_{3(j-3)} \end{pmatrix}$	$\begin{pmatrix} e_{0j} \\ e_{1j} \\ e_{2j} \\ e_{3j} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} d_{0j} \\ d_{1j} \\ d_{2j} \\ d_{3j} \end{pmatrix}$

$$X = \begin{pmatrix} c_0 & 0 & 2 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} e_{0j} \\ e_{1j} \\ e_{2j} \\ e_{3j} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \left(\begin{pmatrix} S[a_{0j}] \\ S[a_{1(j-1)}] \\ S[a_{2(j-2)}] \\ S[a_{3(j-3)}] \end{pmatrix} \oplus \begin{pmatrix} x_{0j} \\ x_{1(j-1)} \\ x_{2(j-2)} \\ x_{3(j-3)} \end{pmatrix} \oplus \begin{pmatrix} TK_{0j} \\ TK_{1(j-1)} \\ 0 \\ 0 \end{pmatrix} \right)$$

Table-based encryption

- **4 tables** of 1kB + 2 tables for AC
- 1 round = 18 lookups + 19 XOR
  (+ calc. of round-key columns)

- Alternative:
  - Use only **1 T-table** of 1kB
  - Minimal extra computation

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-1}] \oplus T_2[a_{2,j-2}] \oplus T_3[a_{3,j-3}] \oplus AC_j \oplus K_j$$

$$T_0[a] = \begin{pmatrix} S[a] \\ S[a] \\ 0 \\ S[a] \end{pmatrix} \quad T_1[a] = \begin{pmatrix} 0 \\ 0 \\ S[a] \\ 0 \end{pmatrix} \quad T_2[a] = \begin{pmatrix} S[a] \\ 0 \\ S[a] \\ S[a] \end{pmatrix} \quad T_3[a] = \begin{pmatrix} S[a] \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$AC = \begin{pmatrix} c_0 & 0 & 0 & 0 \\ c_0 & 0 & 2 & 0 \\ 0 & c_1 & 2 & 0 \\ c_0 & 0 & 0 & 0 \end{pmatrix} \qquad K_j = \begin{pmatrix} TK_{0j} \\ TK_{0j} \\ TK_{1(j-1)} \\ TK_{0j} \end{pmatrix}$$

Table-based decryption

- For table lookups to be possible:
  - SubCells (non-linear) 1st step
  - ShiftRows before MixColumns
- Re-order operations & define new rounds
- Addition of constants and tweakey at the end of the round
  ➔ more efficient than encryption

First round

MixColums_inv
AddRoundTweakey_shifted
AddConstants_shifted

SubCells_inv
ShiftRows_inv
MixColums_inv
AddRoundTweakey_shifted
AddConstants_shifted

⋮ (n-1) rounds

SubCells_inv
ShiftRows_inv
MixColums_inv
AddRoundTweakey_shifted
AddConstants_shifted

Final round

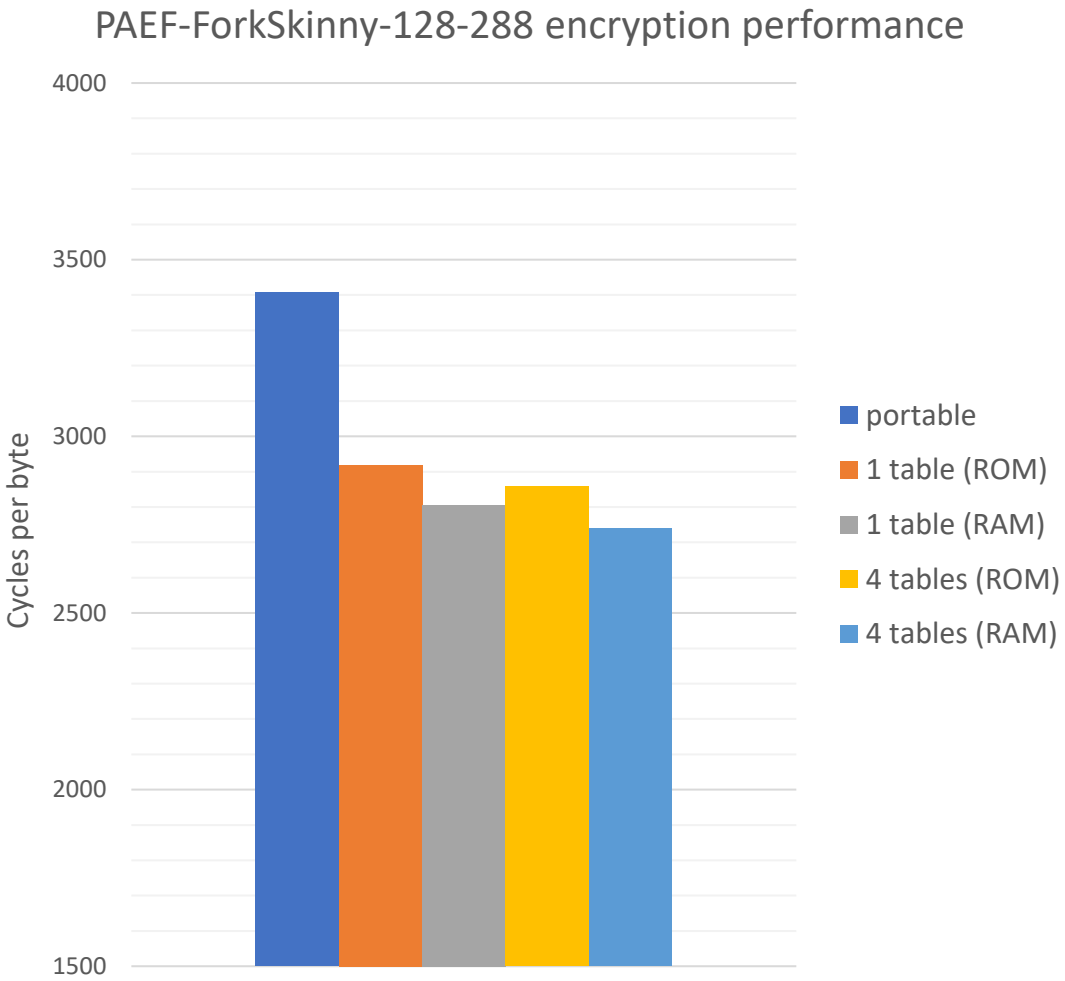
SubCells_inv

Results table-based implementations

Encryption	Tables in ROM			Tables in RAM		
4 lookup tables	c/B	ROM	RAM	c/B	ROM	RAM
PAEF-FS-128-192	2110	6752	192	2016	1960	4984
PAEF-FS-128-256	2111	6748	200	2017	1956	4992
PAEF-FS-128-288	2859	7034	220	2739	2242	5012
SAEF-FS-128-192	2128	6688	192	2035	1896	4984
SAEF-FS-128-256	2129	6674	200	2035	1882	4992
1 lookup table						
PAEF-FS-128-192	2138	3692	192	2030	1972	1912
PAEF-FS-128-256	2139	3688	200	2031	1968	1920
PAEF-FS-128-288	2919	3980	220	2805	2260	1940
SAEF-FS-128-192	2157	3628	192	2049	1908	1912
SAEF-FS-128-256	2157	3614	200	2049	1894	1920

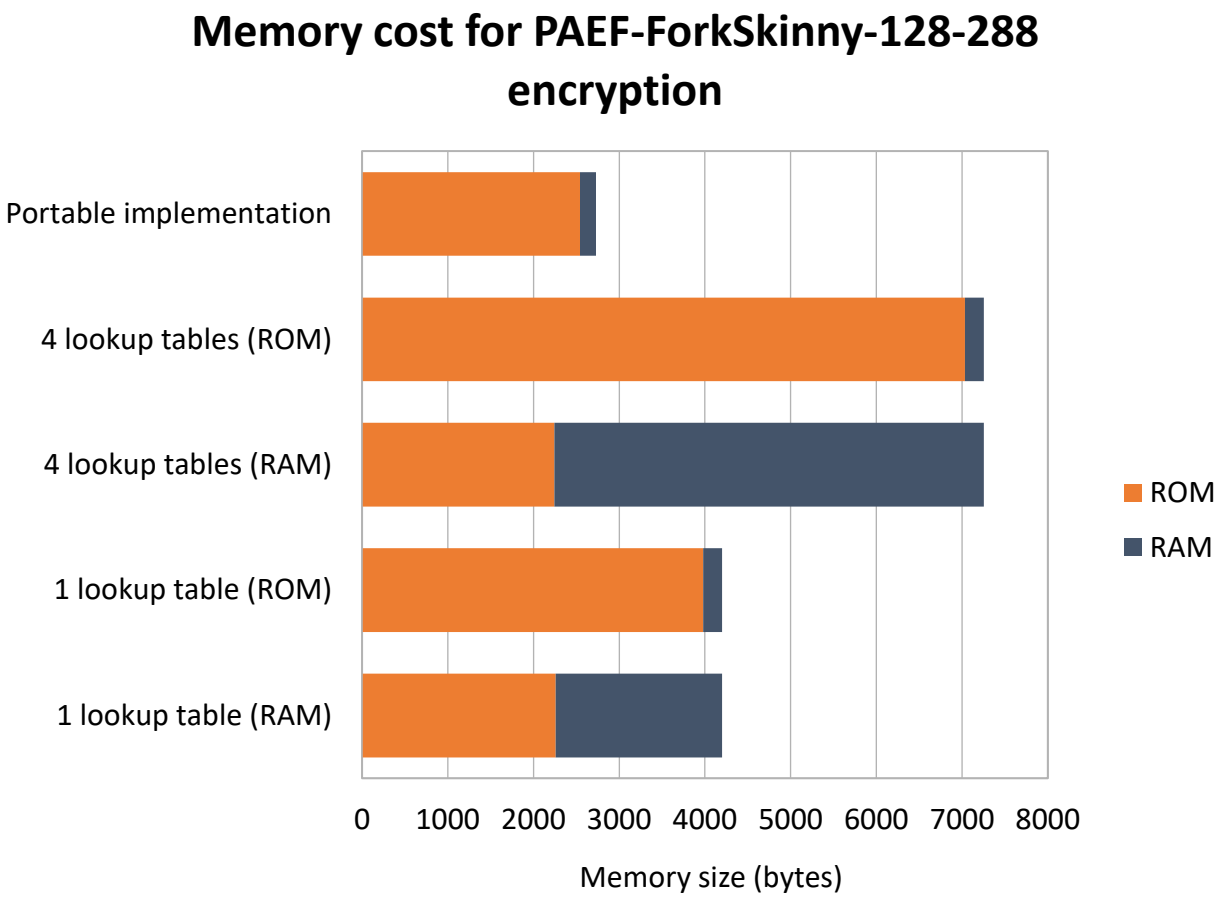
Decryption	c/B	ROM	RAM
PAEF-FS-128-192	2241	3261	818
PAEF-FS-128-256	2241	3253	826
PAEF-FS-128-288	3156	3529	958
SAEF-FS-128-192	2259	3317	818
SAEF-FS-128-256	2257	3303	826

Table-based implementations



- **Performance on Arm Cortex-M0**
  - **Encryption**
    - Speed-up of up to 20%
    - Fastest when tables stored in RAM
    - Small difference in performance for 4 tables vs. 1 table
  - **Decryption**
    - Speed-up of up to 25%
    - For implementation with
      - 1 lookup table
      - Stored in ROM

Table-based implementations



- **Memory cost on Arm Cortex-M0**
  - Reduced code size because of simpler round function
  - Impact on memory can be greatly reduced when using only 1 lookup table
  - RAM is limited resource
    ➔ carefully consider if speed-up is worth it

Overview

- Introduction
- Contributions
- ForkAE
- Portable implementations
- Table-based implementations
- Parallel implementations
- Conclusion

Parallel implementations

- Many optimized software implementations: **bitslicing**
- Works best for blockciphers with parallel mode of operation + enough data
  - e.g. AES bitsliced implementation: 8 blocks in parallel
    ➔ 128*8 = 1024-bit input data needed
- Bitslicing **not suitable for short messages**:
  - Not enough blocks for parallelization
  - Overhead (conversion to bitsliced representation) becomes dominant
- Throughput ↔ Latency

Parallel implementations

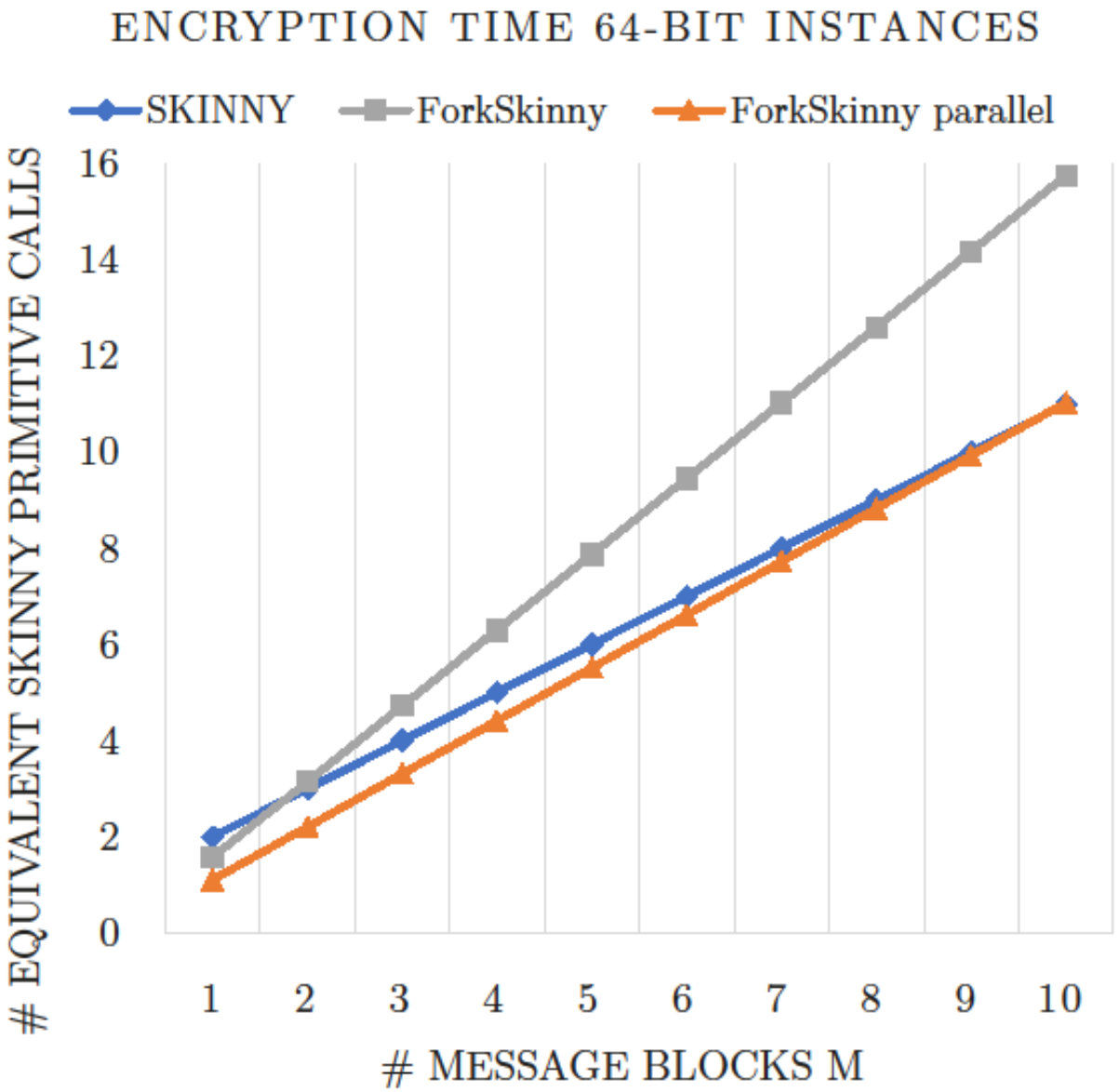
- Target ARM processors with NEON hardware extension
  - 128-bit SIMD (Single-Instruction Multiple-Data)
  - Arm Cortex-A9
- Exploit data-level parallelism in the ForkSkinny primitive
  - In the **round function**:
    - S-box calculated for **all cells in parallel**
  - Parallelism introduced by ***the fork***:
    - Calculate S-box for two branches in parallel
    - Only for 64-bit instance (with 256-bit SIMD also possible for other instances)
    - Needs preprocessed TKS
    - Only encryption

Parallel implementations

- **Neon assembly S-box implementations**
- **Results**
  - **128-bit instances**
    - 30% less clock cycles compared to portable implementations
    - 0,5 kB code size (ROM) reduction
    - RAM usage remains the same
  - **64-bit instance**
    - 29% speed-up for encryption, 17% for decryption
    - ROM size ± equal
    - RAM size increased for encryption (preprocessed TKS)

	Encryption			Decryption		
	c/B	ROM	RAM	c/B	ROM	RAM
PAEF-FS-64-192	1184	3235	331	1390	2653	392
PAEF-FS-128-192	736	2619	161	807	2551	810
PAEF-FS-128-256	737	2651	169	806	2583	818
PAEF-FS-128-288	1026	2863	189	1078	2783	950
SAEF-FS-128-192	743	2491	161	812	2415	810
SAEF-FS-128-256	743	2519	169	810	2419	818

Parallel implementations



ForkSkinny-64-192 on ARM Cortex-A9:

- 17 rounds before *fork*
  23 rounds after *fork*
- One round 64-bit SKINNY with NEON S-box:
  95 clock cycles
- Parallel calculation of 2 rounds after *fork:*
  112 clock cycles

$$\frac{\boldsymbol{ForkSkinny}}{\boldsymbol{SKINNY-AEAD}} = \frac{\mathbf{63 * 95}}{\mathbf{40 * 95}} = \mathbf{1.58}$$

$$\frac{\boldsymbol{ForkSkinny\ //}}{\boldsymbol{SKINNY-AEAD}} = \frac{\mathbf{17 * 95 + 23 * 112}}{\mathbf{40 * 95}} = \mathbf{1.10}$$

- Calls to primitive for M message blocks:
  - ForkAE M
  - Other SKINNY based ciphers M+1

Overview

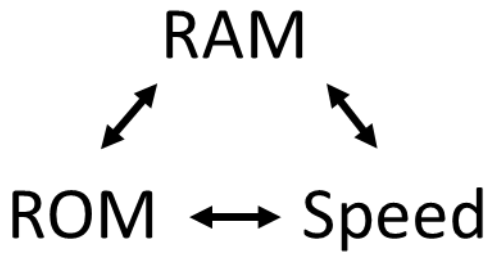
- Introduction
- Contributions
- ForkAE
- Portable implementations
- Table-based implementations
- Parallel implementations
- Conclusion

Conclusion

- Efficient and constant-time portable implementations of ForkAE
  - Trade memory usage for **faster decryption**
- Platform specific optimizations
  - **Table-based** implementations
    - Platforms without caches
    - Combine calculations in table-lookups
    - Reduce memory cost by using only 1 table
  - Neon SIMD **parallel** implementations
    - Data-level parallelism in ForkSkinny primitive
    - Parallel S-box calculations
    - Parallelism of *the fork*
- All implementations available at https://github.com/ArneDeprez1/ForkAE-SW

Takeaways

- Not “One implementation fits all”
- Always a trade-off



- Different platforms allow for different implementations
- Need a cipher that allows for different implementation strategies
  ➔ this is the case for ForkAE