Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.34, NO.4, Aug. 2024 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2024.34.4.629

연산 순서 변경에 따른 범용 프로세서에서 효율적인 CHAM-like 구조*

신 명 수,[1†] 김 선 규,[1] 신 한 범,[1] 김 인 성,[1] 김 선 엽,[1] 권 동 근,[1]
홍 득 조,[3] 성 재 철,[4] 홍 석 희[2‡]
[1,2]고려대학교 (대학원생, 교수), [3]전북대학교 (교수), [4]서울시립대학교 (교수)

Efficient CHAM-Like Structures on General-Purpose Processors with Changing Order of Operations*

Myoungsu Shin,[1†] Seonkyu Kim,[1] Hanbeom Shin,[1] Insung Kim,[1] Sunyeop Kim,[1]
Donggeun Kwon,[1] Deukjo Hong,[3] Jaechul Sung,[4] Seokhie Hong[2‡]
[1,2]Korea University (Graduate student, Professor),
[3]Jeonbuk National University (Professor), [4]University of Seoul (Professor)

요 약

CHAM은 ISO/IEC 표준 블록암호 운영 모드에서 암호화 함수가 복호화 함수보다 자주 사용되는 점을 고려하여 암호화 속도를 강조하여 설계되었다. 현대 범용 프로세서 구조의 슈퍼스칼라 아키텍처에서는 연산 구성이 동일하더라도 연산의 순서가 달라지면 처리 속도가 달라질 수 있다. 본 논문에서는 ARX 기반 블록암호인 CHAM의 연산 순서를 재배치한 구조 CHAM-like 구조들에 대해 범용 프로세서 환경에서 단일 블록 구현과 병렬 구현에 대한 구현 효율성과 안전성을 분석한다. 본 논문에서 제시한 구조는 암호화 속도 관점에서 최소 약 9.3%에서 최대 약 56.4% 효율적이다. 안전성 분석은 CHAM-like 구조들에 차분 공격과 선형 공격에 대한 저항성을 평가한다. 보안 마진 관점에서 차분 공격은 3.4%, 선형 공격은 6.8% 차이를 보여 효율성 차이에 비해 보안 강도는 비슷함을 보인다. 이러한 결과는 ARX 기반 블록암호 설계 관점에서 활용가능하다.

ABSTRACT

CHAM is designed with an emphasis on encryption speed, considering that in the ISO/IEC standard block cipher operation mode, encryption functions are used more often than decryption functions. In the superscalar architecture of modern general-purpose processors, different ordering of operations can lead to different processing speeds, even if the computation configuration is the same. In this paper, we analyze the implementation efficiency and security of CHAM-like structures, which rearrange the order of operations in the ARX-based block cipher CHAM, for single-block and parallel implementations in a general-purpose processor environment. The proposed structures are at least 9.3% and at most 56.4% efficient in terms of encryption speed. The security analysis evaluates the resistance of the CHAM-like structures to differential and linear attacks. In terms of security margin, the difference is 3.4% for differential attacks and 6.8% for linear attacks, indicating that the security strength is similar compared to the efficiency difference. These results can be utilized in the design of ARX-based block ciphers.

Keywords: ARX cipher, CHAM, superscalar architecture

Received(05. 02. 2024), Modified(07. 03. 2024), Accepted(07. 03. 2024)

* 본 연구는 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임. (No.2017-0-00520, (ICT 기초연구실) SCR-Friendly 대칭키 암호 및 응용모드 개발)

† 주저자, damper99@korea.ac.kr
‡ 교신저자, shhong@korea.ac.kr(Corresponding author)

I. 서 론

현대 암호시스템에서 블록암호는 안전한 통신과 데이터 보호에 중요한 역할을 한다. IoT 기기의 발전으로 저전력 기기에서의 암호 효율성이 요구되면서 ARX 기반 블록암호가 주목받기 시작했다. ARX 기반 블록암호는 비교적 간단한 연산인 법 덧셈(modular Addition), 비트 순환이동 연산(Rotation), 배타적 논리합(XOR)을 기반으로 구성된 라운드 함수가 반복적으로 동작하는 블록암호이다. 대표적인 ARX 구조 블록암호는 HIGHT[1], SPECK[2], LEA[3], CHAM[4]이 있다.

CHAM[4]은 국가보안기술연구소에서 제안한 ARX 기반 경량 블록암호로 ICISC'17 에서 발표되었으며, 추가적인 안전성 분석을 통해 라운드 수가 늘어난 revised version이 ICISC'19 에서 발표되었다[5]. CHAM은 LEA와 같이 ISO/IEC 표준 블록암호 운영 모드에서 암호화 함수가 복호화 함수보다 더 광범위하고 자주 사용되는 점을 고려하여, 암호화와 복호화 간의 속도 균형보다 암호화 속도를 강조하여 설계되었다.

현대 범용 프로세서에서 채택하고 있는 구조인 슈퍼스칼라 아키텍처에서는 명령어 수준 병렬화, 비순차 실행, 명령어별 실행 유닛 수 분배 등 여러 최적화 기법을 통해 1 클럭 사이클 동안 여러 명령어를 실행할 수 있다. 이러한 하드웨어 단에서의 최적화 기법 때문에 범용 프로세서 환경에서는 연산의 구성이 동일하더라도 순서를 바꾸는 것만으로도 암호화 속도가 크게 개선할 수 있다.

본 논문에서는 암호화 속도를 강조한 ARX 기반 블록암호 CHAM에 대해 연산 순서를 재배치한 6개의 변형 구조(CHAM-like 구조) 단일 블록 구현과 병렬 구현에 대한 구현 효율성을 평가한다. Table 1.은 범용 프로세서인 Intel i7-12700K, Xeon Gold 6230과 고성능 마이크로컨트롤러인 ARMv8 Cortex-A72에서 CHAM-64/128과 가장 빠른 구조의 CHAM-like-64/128을 구현하여 암호화 속도를 비교한다. x86_64와 AArch64에서는 단일블록 구현을, AVX2, AVX512, NEON에서는 병렬 구현에 대한 암호화 속도를 나타낸다. 가장 빠른 구조의 CHAM-like-64/128이 적게는 9.3%, 최대 56.4% 빠른 암호화 속도를 보여준다.

블록암호의 연산 순서 변경으로 인해 보안 강도에 변화가 있을 수 있다. 이에 따라 CHAM과 비슷한 보안 강도를 가졌는지 확인하기 위해 CHAM-like 구조들의 차분 공격 및 선형 공격 저항성을 평가한다. 6개의 CHAM-like 구조를 선형/차분 분석 관점에서 두 개의 class로 구분하며, 각 class 내의 구조들은 차분 공격 및 선형 공격 저항성이 동일하다. 두 개의 class 중 한 class는 CHAM과 동일한 최적 차분 경로를 가진다. 이에 따라 남은 한 class의 최적 차분 경로 및 차분 확률 가중치와 최적 선형 경로 및 선형 상관관계 가중치를 분석한다. 분석 결과, Table 1.에 제시된 가장 효율적인 CHAM-like-64/128이 속한 class는 CHAM-64/128에 비해 차분 관점에서 3라운드가 증가하여 보안 마진 관점에서 3.4%의 차이를 보인다. 선형 관점에서는 6라운드가 증가하여 보안 마진 관점에서 6.8% 차이를 보이며 차분 및 선형 관점에서 비슷한 보안 강도를 가짐을 확인하였다.

Table 1. Comparison of encryption speeds for CHAM and the most efficient CHAM-like-64/128 (cpb : cycles per byte)

env	CHAM-64/128	The most efficient CHAM-like-64/128
x86_64	23.05 cpb	14.73 cpb
AVX2	2.57 cpb	2.14 cpb
AVX512	1.48 cpb	1.24 cpb
AArch64	48.32 cpb	44.18 cpb
NEON	13.95 cpb	9.13 cpb

본 논문에서는 위 결과에 따라 CHAM보다 효율적이면서 비슷한 보안 강도를 갖는 CHAM-like 구조를 제시한다. 본 논문에서 제시한 구조 및 결과는 ARX 기반 블록암호의 설계 관점에서 활용가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 배경지식이 되는 CHAM의 명세, 슈퍼스칼라 아키텍처와 SMT 솔버를 활용한 ARX 암호 차분 분석 방법을 다룬다. 3장에서는 CHAM의 연산 순서를 재배치한 CHAM-like 구조를 제시하고 효율성 및 안전성 분석을 제시한다. 마지막으로 4장에서 본 논문에 대한 결론을 맺는다.

II. 배경 지식

2.1 표기법

$X_i[j]$: i-라운드 입력 블록의 j번째 워드

$x \| y$: 비트열 x와 y의 연접(Concatenation)

$x \boxplus y$: x와 y의 법 덧셈 (Addition modulo 2^w)

$x \oplus y$: x와 y의 배타적 논리합 (XOR)
$\mathrm{ROL}_n(x)$: x의 왼쪽으로 n만큼 비트 순환이동 연산 (Rotation)
ΔX_r: r-라운드 입력 차분
ΓX_r: r-라운드 입력 마스크

2.2 경량 블록암호 CHAM

CHAM[4]은 LEA[3]를 8-bit AVR 및 16-bit MSP 마이크로컨트롤러와 같은 자원 제한 환경에서 적합하도록 개선하려는 시도로 제안된 암호이다. CHAM 계열의 각 알고리즘은 n-bit 블록, k-bit 비밀키에 따라 CHAM-n/k로 표현되며 CHAM-64/128, CHAM-128/128, CHAM-128/256으로 이루어져 있다. Table 2.는 CHAM의 3가지 알고리즘과 각 알고리즘에 따른 파라미터를 나타낸다.

CHAM은 키의 상태를 업데이트하지 않는 stateless-on-the-fly 키스케줄을 사용하여 저장공간을 줄여 저사양 환경에서 높은 효율성을 보여준다. 또한 비트 순환이동 연산의 크기를 1과 8만을 사용하여 8-bit AVR 마이크로컨트롤러에서의 높은 구현 효율성을 갖는다.

4-branch GFN type-1 구조를 가진 CHAM-n/k는 n-bit 평문 P를 w-bit 4개 워드로 나눠 $P = X[0] \| X[1] \| X[2] \| X[3]$로 표현할 수 있다. Fig. 1.은 CHAM 계열의 암호화 라운드 함수를 나타낸 것이다. CHAM 계열의 라운드 함수는 짝수 라운드와 홀수 라운드에서 비트 순환이동 연산 크기를 다르게 사용해 짝수 라운드는

$$X_{i+1}[3] \leftarrow \mathrm{ROL}_8((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_1(X_i[1]) \oplus RK[i])),$$
$$X_{i+1}[j] \leftarrow X_i[j+1] \text{ for } 0 \le j < 3.$$

와 같이 계산되고, 홀수 라운드는

$$X_{i+1}[3] \leftarrow \mathrm{ROL}_1((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_8(X_i[1]) \oplus RK[i])),$$
$$X_{i+1}[j] \leftarrow X_i[j+1] \text{ for } 0 \le j < 3.$$

Table 2. List of revised version of CHAM ciphers and their parameters

cipher	n	k	r	w	k/w
CHAM-64/128	64	128	88	16	8
CHAM-128/128	128	128	112	32	4
CHAM-128/256	128	256	120	32	8

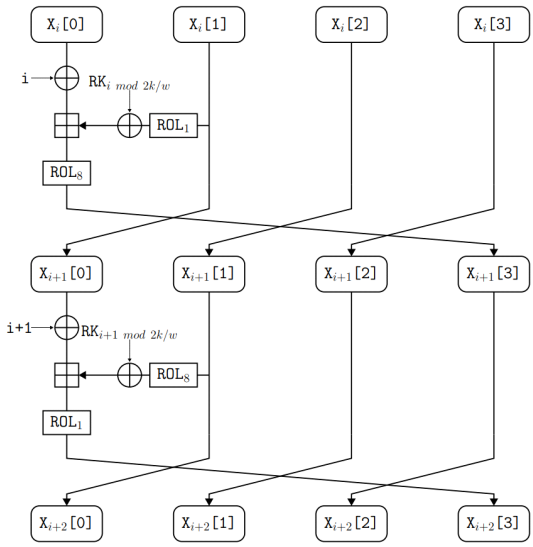

Fig. 1. 2-round encryption function of CHAM family.

2.3 슈퍼스칼라 아키텍처

슈퍼스칼라 아키텍처는 1 클록 사이클 동안 여러 개의 명령어를 가져오고 실행하도록 설계된 프로세서이다. 슈퍼스칼라 아키텍처는 비순차적 실행과 명령어 수준 병렬화를 활용하여 서로 의존성이 없는 명령어들을 먼저 또는 동시에 실행하도록 순서를 하드웨어 단에서 재배치한다. 또한, 복수의 실행 유닛을 가지고 있어 재배치된 명령어들을 병렬로 처리할 수 있다. 실행 유닛별로 처리할 수 있는 명령어 유형이 다르기 때문에 명령어의 순서가 성능에 영향을 미친다. 이에 따라, 명령어들의 순서를 재배치하기 위해 명령어 디코더와 스케줄러 등 제어 로직이 함께 동작한다. 명령어는 디코더에서 해석되고 이후 스케줄러에 의해 적절한 실행 유닛들에 할당된다. 실행 유닛들은 명령어를 병렬로 실행하고 결과를 레지스터에 저장한다. 슈퍼스칼라 아키텍처는 이와 같은 과정을 통해 하드웨어 단에서 최적화하여 작업을 수행한다.

본 논문에서는 암호의 연산 구성 요소는 같지만 연산 순서를 재배치하여 암호의 구조를 변경한다. 슈퍼스칼라 아키텍처에서 암호의 변경한 구조마다 최적화했을 때 속도가 달라질 수 있으므로 해당 구조들끼리 속도를 비교한다.

2.4 SMT 솔버를 활용한 ARX 암호 차분 경로 탐색

이진 충족성 문제(Boolean Satisfiability Problem)는 이진 변수들과 논리 연산자들을 사용하여 구성된 공식을 만족하는 값이 존재하는지를 결정하는 문제이다. 이진 충족성 문제를 해결할 수 있는 SAT 솔버는 다양한 휴리스틱과 최적화 기법을 사용하여 주어진 공식을 분석하고 만족시키는 변수들의 값을 찾아 문제를 해결하는 도구이다. SMT(Satisfiability Modulo Theories) 솔버는 이진 충족성 문제를 확장한 SMT 문제[6]를 해결할 수 있다. SMT 솔버는 암호 분석에 적합한 비트 벡터 등과 같은 다양한 데이터 타입과 이에 대한 제약을 처리할 수 있다. 또한 ARX 기반 암호는 워드 단위 연산이 포함되어 있기 때문에 SMT 솔버로 분석하기에 적합하다.

SMT 솔버를 활용하여 ARX 기반 암호의 차분 특성을 찾기 위한 프레임워크는 [7]에서 Mouha 등에 의해 제안되었고 Salsa20[10]에 적용하였다. 제안된 프레임워크는 SAT 솔버를 기반으로 구축된 SMT 솔버인 STP[8]를 사용하여 차분 확률 가중치 w를 가지는 차분 특성을 찾는다. 차분 특성을 찾기 위해 STP에 각 라운드별 워드 단위 차분 상태 변수, 덧셈 연산 후 차분 상태 변수, 차분 확률 가중치 변수와 ARX 암호의 XOR, 법 덧셈, 순환이동 연산에 대한 XOR 차분에 관한 방정식을 입력으로 넣는다. STP는 방정식을 CNF 형태로 변환하여 SAT 솔버에 입력으로 넣는다. SAT 솔버는 CNF 형태의 식의 해를 찾아 차분 특성과 해당 특성이 발생할 확률을 계산할 수 있다.

비트 순환이동 연산과 XOR 연산은 각 입력 차분(두 입력 평문에 대해 XOR 연산한 값)에 대한 유효한 출력 차분(두 출력 값에 대해 XOR 연산한 값)이 단 하나이기 때문에, 법 덧셈에 대한 XOR 차분과 확률만 고려하면 차분 경로의 확률을 계산할 수 있다. 법 덧셈에 대한 XOR 차분 확률에 관한 식은 Lipmaa와 Moriai가 [9]에서 제시한 정리를 이용한다.

정의 1. 법 덧셈에 대한 XOR 차분

$A, B, \alpha, \beta, \gamma \in \mathbb{F}^w$ 입력값 A, B와 입력 차분 α, β의 법 덧셈에 대한 출력 차분 γ는 다음과 같다.

$$\gamma = (A \boxplus B) \oplus ((A \oplus \alpha) \boxplus (B \oplus \beta)).$$

정의 2. 법 덧셈에 대한 XOR 차분 확률

$\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma)$는 입력 차분 α, β에 대한 출력 차분 γ 일 때 법 덧셈에 대한 XOR 차분 확률을 나타낸다.

$$\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma) = \frac{\#\{(x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta))\}}{2^{2w}}.$$

정리 1. 법 덧셈에 대한 XOR 차분 존재성[9]

입력 차분 α, β에 대한 출력 차분 γ인 법 덧셈에 대한 XOR 차분 $(\alpha, \beta \rightarrow \gamma)$의 유효한 필요 충분 조건은 아래와 같다.

$$\mathrm{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) = 0.$$

이때, $\mathrm{eq}(x, y, z) = (\neg x \oplus y) \wedge (\neg x \oplus z)$ 이다.

정리 2. 법 덧셈에 대한 차분 확률 가중치

모든 유효한 차분 $(\alpha, \beta \rightarrow \gamma)$에 대한 차분 확률 가중치 $weight(\alpha, \beta \rightarrow \gamma)$는 아래와 같이 정의된다.

$$weight(\alpha, \beta \rightarrow \gamma) = -\log_2(\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma)).$$

유효한 차분의 차분 확률 가중치는 다음 식으로 계산할 수

$$weight(\alpha, \beta \rightarrow \gamma) = h^*(\neg \mathrm{eq}(x, y, z)).$$

이때, $h^*(x)$는 x의 MSB를 제외한 나머지 비트 중 0이 아닌 비트의 수를 의미한다. Markov 가정[11]에 의해 유효한 차분 특성의 가중치를 각 법 덧셈 연산에서의 차분 확률 가중치 합으로 계산한다.

2.5 SMT 솔버를 활용한 ARX 암호 선형 경로 탐색

차분 경로 탐색과 마찬가지로, SAT/SMT 솔버를 활용한 ARX 기반 선형 경로 탐색에 관한 연구가 진행되었다. [12]에서 SAT 솔버를 활용하여 SPECK[2]과 chaskey[13]에 적용하였고 22라운드 SPECK32, 11라운드 SPECK48, 13라운드 SPECK64, 9라운드 SPECK96, 9라운드 SPECK128에 대한 최적 선형 경로를 탐색하였다.

STP를 사용하여 선형 경로를 찾기 위해 각 라운드별 워드 단위 선형 마스크 상태 변수, 선형 상관관

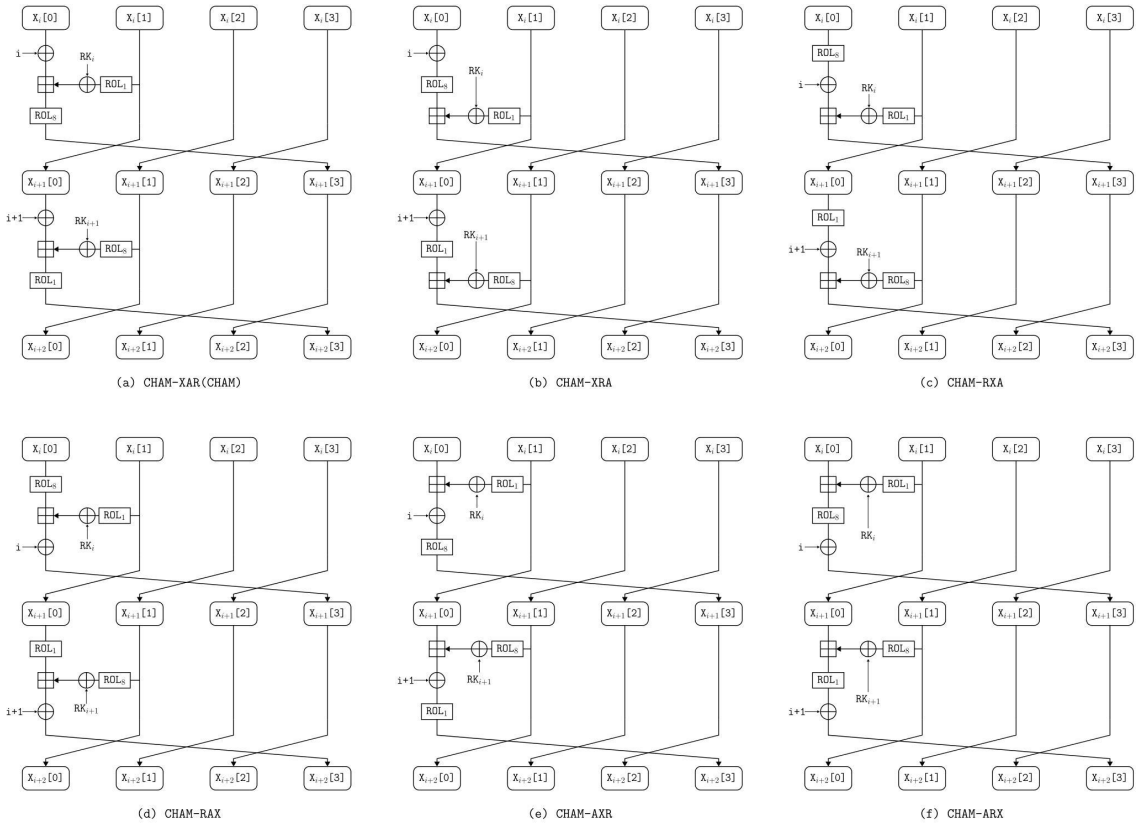

Fig. 2. 2-round encryption functions of CHAM-like structures

계 가중치 변수와 ARX 암호의 XOR, 덧셈, 비트 순환이동 연산에 대한 방정식을 입력으로 넣는다. 비트 순환이동 연산의 입력 마스크에 대한 유효한 출력 마스크는 입력 마스크에 비트 순환이동 연산을 한 것과 같다. 법 덧셈 연산에 대한 선형 근사의 상관관계를 계산하는 방법은 Schulte-Geers[14]가 제안한 정리를 이용한다.

정의 3. 선형 근사(linear approximation)
블록암호의 선형 공격은 선형 근사를 기반으로 한다. 함수 f가 두 입력 $(x_0,x_1)\in F_2^m\times F_2^m$을 가질 때, 입력 선형 마스크 $(\alpha,\beta)\in F_2^m\times F_2^m$와 출력 선형 마스크 $\gamma\in F_2^n$에 대해 f의 선형 근사$(\alpha,\beta)\rightarrow\gamma$의 상관관계는 다음과 같이 정의한다.

$$C(\alpha,\beta\rightarrow\gamma)=$$
$$2\cdot\frac{\#\{(x_0,x_1)\in F_2^m\times F_2^m|\alpha\cdot x_0\oplus\beta\cdot x_1\oplus\gamma\cdot f(x_0,x_1)=0\}}{2^{2m}}-1.$$

정리 3. 법 덧셈에 대한 선형 근사의 상관관계[14]
$z=(z_{n-1},z_{n-2},\cdots,z_0)$는
$z\oplus(z\gg 1)\oplus((\alpha\oplus\beta\oplus\gamma)\gg 1)=0,\ z_{n-1}=0$을 만족하는 n-bit 벡터이다. 이때 법 덧셈의 선형 근사 $(\alpha,\beta\rightarrow\gamma)$의 상관관계는 다음과 같다.

$$C(\alpha,\beta\rightarrow\gamma)=1_{(\gamma\oplus\alpha)\preceq z}1_{(\gamma\oplus\beta)\preceq z}(-1)^{(\alpha\oplus\gamma)\cdot(\beta\oplus\gamma)}2^{-h^*(z)}.$$

III. 효율적인 CHAM-like 구조

3.1 CHAM-like 구조

CHAM의 암호화 라운드 함수에서는 라운드 상수 XOR-Addition-Rotation(XAR) 연산 순서로 진행된다. 이 연산 구조의 순서를 변경하면 6가지 연산 구조 순열을 구성할 수 있다. Fig. 2.는 CHAM의 6가지 CHAM-like 구조들의 암호화 라운드 함수를 나타낸 것이다. 본 논문에서는 연산 구조를 식별하기 위해 각 연산의 앞 문자를 나열하여

(a)XAR, (b)XRA, (c)RXA, (d)RAX, (e)AXR, (f)ARX과 같이 표기한다.

CHAM의 라운드 함수에서 비트 순환이동 연산의 크기는 AVR 및 MSP 명령어 집합을 고려하여 선정하였다[4]. AVR 및 MSP 명령어 집합에서는 비트 순환이동 연산을 할 때 임의의 양만큼 순환이동 연산을 하는 단일 명령어가 존재하지 않아 1비트 순환이동을 여러 번 구현해야 한다. AVR 및 MSP 환경에서의 구현 효율성과 안전성을 고려하여 비트 순환이동 연산의 크기를 결정하였다. 본 논문에서는 CHAM의 제안 논문에서 제시한 비트 순환이동 연산의 크기를 CHAM-like 구조에 적용하여 안전성과 구현 효율성을 분석한다.

3.2 CHAM-like 구조 효율성 분석

범용 프로세서 환경에서 암호화 함수의 연산 순서를 재배치하여 수행하면 명령어 수준 병렬성, 명령어 간 의존성 및 스케줄링에 따라 암호화 속도가 달라질 수 있다. 이에 CHAM-like 구조들에 대해 효율성을 분석하여 CHAM보다 우수한 성능을 가진 구조를 탐색한다.

Table 3.은 x86_64, AVX2, AVX512, AArch64, NEON 명령어로 구현한 CHAM-like-64/128의 구현물에 대해 암호화 속도를 비교한 표이다. x86_64와 AVX2 구현물은 Intel i7-12700K(x64) 프로세서 환경에서 구현되었고, AVX512 구현물은 Intel Xeon Gold 6230 프로세서 환경에서 구현되었다. AArch64 구현물과 NEON 구현물은 ARMv8 cortex-A72 프로세서 환경에서 구현되었다. 모든 구현물은 키스케줄을 포함하지 않은 상태로 88라운드 암호화 함수의 속도만을 평가하며 컴파일 최적화 옵션은 -O3로 진행한다. CHAM-like-64/128에 대해 x86_64와 AArch64 구현은 1블록 암호화 함수 500,000회 수행의 바이트당 클럭 사이클 수를 나타내었고, NEON 구현은 8블록 병렬, AVX2 구현은 16블록 병렬, AVX512 구현은 32블록 병렬 암호화 함수 500,000회 수행의 바이트당 클럭 사이클 수를 나타낸다.

5가지 구현에서 가장 빠른 구조는 CHAM-RXA-64/128임을 확인할 수 있다. CHAM-RXA-64/128은 CHAM에 비해 x86_64 구현에서 54.87%, AVX2 구현에서 24.75%, AVX512 구현에서 19.35%, AArch64에서 9.36%, NEON에서 52.96% 향상된 암호화 속도를 확인할 수 있다.

3.3 CHAM-like 구조 차분 분석

본 절에서는 CHAM-like 구조들에 대해 SMT 솔버를 활용하여 최적 차분 경로를 탐색한 결과를 제시한다. 3.1절에서 언급한 바와 같이 CHAM-like 구조에서 비트 순환이동 연산 크기를 CHAM과 동일하게 구성한다.

차분 분석에서 XOR 연산은 입력 차분에 대한 출력 차분에 어떠한 영향도 주지 않으므로, 6가지 CHAM-like 구조에서 XOR 연산인 라운드 상수 XOR 연산과 라운드키 연산을 소거하여 구조를 간소화할 수 있다. 간소화된 구조는 CHAM-AR와 CHAM-RA 두 가지 class로 분류할 수 있다. CHAM-AR class는 CHAM(CHAM-XAR), CHAM-AXR, CHAM-ARX로 이루어져 있고,

Table 3. Comparison of encryption speeds for CHAM-like-64/128 (cpb : cycles per byte)

CHAM-like structure	x86_64	AVX2	AVX512	AArch64	NEON
CHAM-64/128	23.05 cpb	2.57 cpb	1.48 cpb	48.32 cpb	13.95 cpb
CHAM-XRA-64/128	18.25 cpb	2.35 cpb	1.52 cpb	48.31 cpb	11.80 cpb
CHAM-RXA-64/128	**14.73 cpb**	**2.14 cpb**	**1.24 cpb**	**44.18 cpb**	**9.13 cpb**
CHAM-RAX-64/128	20.50 cpb	2.14 cpb	1.26 cpb	45.87 cpb	9.88 cpb
CHAM-AXR-64/128	21.45 cpb	2.74 cpb	1.50 cpb	59.64 cpb	14.45 cpb
CHAM-ARX-64/128	27.97 cpb	2.71 cpb	1.50 cpb	56.29 cpb	14.53 cpb

x86_64 and AVX2	AVX512	AArch64 and NEON
⊙ processor : Intel i7-12700K ⊙ compiler : GNU gcc 8.1.0 ⊙ opt : -O3	⊙ processor : Intel Xeon Gold 6230 ⊙ compiler : GNU gcc 11.2.0 ⊙ opt : -O3	⊙ processor : ARMv8-A72 ⊙ compiler : GNU gcc 11.2.0 ⊙ opt : -O3

CHAM-RA class는 CHAM-XRA, CHAM-RXA, CHAM-RAX로 이루어져 있다. 각 class는 차분 분석 관점에서 모두 같은 최적 차분 경로를 갖는다. 따라서 CHAM-AXR과 CHAM-ARX는 차분 분석 관점에서 CHAM과 같은 안전성을 갖는다.

Table 4.는 SMT 솔버를 활용하여 탐색한 CHAM-AR-64/128과 CHAM-RA-64/128의 최적 차분 경로의 확률 가중치를 비교한 표이다. CHAM-AR-64/128의 라운드별 차분 확률 가중치는 [5]에서 제시한 CHAM-64/128의 라운드별 차분 확률 가중치와 동일하다. Table 4.에서 CHAM-RA-64/128이 CHAM-64/128에 비해 최적 차분 경로가 3라운드 더 긴 것을 확인할 수 있다. CHAM-RA-64/128이 revised CHAM-64/128과 같이 88라운드를 가진다고 했을 때, 보안 마진 관점에서 3.4% 차이를 가지므로 CHAM-64/128과 유사한 보안 강도를 가짐을 알 수 있다. 43라운드 CHAM-RA-64/128의 자세한 차분 경로와 가중치는 Appendix의 Table 6.에 나타낸다.

Table 4. The probability weights of the best differential trails for CHAM-64/128[5] and CHAM-RA-64/128

structure class	Round							
	36	37	38	39	40	41	42	43
CHAM[5]	56	58	60	63	>63	>63	>63	>63
CHAM-RA	51	53	54	57	60	61	63	64

3.4 CHAM-like 구조 선형 분석

본 절에서는 CHAM-like 구조들에 대해 SMT 솔버를 활용하여 최적 선형 경로를 탐색한 결과를 제시한다. 3.3절의 차분 분석과 마찬가지로 CHAM-like 구조에서 비트 순환이동 연산 크기를 CHAM과 동일하게 구성하며 2가지 class로 분류하여 최적 선형 경로를 탐색한다.

Table 5.는 CHAM과 CHAM-RA class의 최적 선형 경로의 라운드별 상관관계 가중치를 나타낸 표이다. CHAM-AR의 최적 선형 경로는 [15]에서 제시한 CHAM-64/128의 최적 선형 경로의 상관관계 가중치와 이론적으로 일치하므로 CHAM-RA의 최적 선형 경로를 탐색하여 비교한다. CHAM의 선형 분석 구별자의 최대 라운드는 34라운드이고,

Table 5. The correlation weights of the optimal linear trails for CHAM-64/128 [15] and CHAM-RA-64/128

structure class	Round							
	34	35	36	37	38	39	40	41
CHAM[15]	31	33	>32	>32	>32	>32	>32	>32
CHAM-RA	24	25	26	27	28	30	31	32

CHAM-RA class의 최대 라운드는 40라운드로 6라운드 더 긴 것을 알 수 있다. CHAM-RA-64/128이 revised CHAM-64/128과 같이 88라운드를 가진다고 했을 때, 보안 마진 관점에서 6.8% 차이를 가지므로 CHAM-64/128과 유사한 보안 강도를 가짐을 알 수 있다. CHAM-RA-64/128의 41라운드 최적 선형 경로와 상관관계 가중치는 Appendix의 Table 7.에 나타낸다.

IV. 결 론

본 논문에서는 ARX 기반 블록암호 CHAM의 연산 순서를 재배치하여 범용 프로세서에서 효율적인 CHAM-like 구조를 제시한다. 여러 범용 프로세서에서의 효율성 분석과 차분 분석 기반 안전성 분석 결과를 통해 6가지 CHAM-like 구조 중 암호화 속도 관점에서 가장 빠른 구조는 CHAM-RXA-64/128임을 알 수 있다. CHAM-RXA-64/128는 범용 프로세서 환경에서 암호화 속도가 CHAM-64/128보다 최대 약 55% 빠른 결과를 확인할 수 있고, 차분 분석 및 선형 분석 관점에서 CHAM-64/128과 유사한 보안 강도를 가짐을 3.3절과 3.4절에서 확인할 수 있다. 연산 구성은 동일하지만 순서를 바꾸는 것만으로도 범용 프로세서 환경에서 암호화 속도가 크게 달라질 수 있으므로 ARX 기반 블록암호 설계할 때 이를 고려할 수 있다.

향후 연구로 가능한 비트 순환이동 연산 크기에 대해 안전성 및 구현 효율성을 탐색하여 CHAM보다 안전하고 효율적인 CHAM variant 제시할 예정이다.

Appendix.

Table 6. 43-round optimal differential trail for CHAM-RA-64/128

43-round differential trail for CHAM-RA-64/128		
r	$\Delta X_r[0] \| \Delta X_r[1] \| \Delta X_r[2] \| \Delta X_r[3]$	$weight_r$
0	0x0020 0x1000 0x0020 0x0000	1
1	0x1000 0x0020 0x0000 0x0000	1
2	0x0020 0x0000 0x0000 0x0000	1
3	0x0000 0x0000 0x0000 0x2000	0
4	0x0000 0x0000 0x2000 0x0000	0
5	0x0000 0x2000 0x0000 0x0000	1
6	0x2000 0x0000 0x0000 0x0020	1
7	0x0000 0x0000 0x0020 0x0020	0
8	0x0000 0x0020 0x0020 0x0000	1
9	0x0020 0x0020 0x0000 0x0040	2
10	0x0020 0x0000 0x0040 0x2040	2
11	0x0000 0x0040 0x2040 0x6000	1
12	0x0040 0x2040 0x6000 0x4000	2
13	0x2040 0x6000 0x4000 0x0080	4
14	0x6000 0x4000 0x0080 0x4020	3
15	0x4000 0x0080 0x4020 0x80A0	0
16	0x0080 0x4020 0x80A0 0x0000	1
17	0x4020 0x80A0 0x0000 0x0040	3
18	0x80A0 0x0000 0x0040 0x2040	2
19	0x0000 0x0040 0x2040 0xA080	1
20	0x0040 0x2040 0xA080 0x4000	2
21	0x2040 0xA080 0x4000 0x0080	3
22	0xA080 0x4000 0x0080 0x4020	2
23	0x4000 0x0080 0x4020 0x00A0	0
24	0x0080 0x4020 0x00A0 0x0000	1
25	0x4020 0x00A0 0x0000 0x0040	2
26	0x00A0 0x0000 0x0040 0x2040	1
27	0x0000 0x0040 0x2040 0xA000	1
28	0x0040 0x2040 0xA000 0x4000	2
29	0x2040 0xA000 0x4000 0x0080	3
30	0xA000 0x4000 0x0080 0x4020	2
31	0x4000 0x0080 0x4020 0x80A0	0
32	0x0080 0x4020 0x80A0 0x0000	1
33	0x4020 0x80A0 0x0000 0x0040	3
34	0x80A0 0x0000 0x0040 0x2040	2
35	0x0000 0x0040 0x2040 0xA080	1
36	0x0040 0x2040 0xA080 0x4000	2
37	0x2040 0xA080 0x4000 0x0080	3
38	0xA080 0x4000 0x0080 0x4020	2
39	0x4000 0x0080 0x4020 0x00A0	0
40	0x0080 0x4020 0x00A0 0x0000	1
41	0x4020 0x00A0 0x0000 0x0040	2
42	0x00A0 0x0000 0x0040 0x2040	1
43	0x0000 0x0040 0x2040 0xA000	-

Table 7. 41-round optimal linear trail for CHAM-RA-64/128

41-round linear trail for CHAM-RA-64/128		
r	$\Gamma X_r[0] \| \Gamma X_r[1] \| \Gamma X_r[2] \| \Gamma X_r[3]$	Cw_r
0	0x0060 0x0000 0x8041 0x60B0	1
1	0x2000 0x8041 0x60B0 0x4000	1
2	0x8001 0x60B0 0x4000 0x6000	1
3	0x6030 0x4000 0x6000 0x0100	2
4	0x0080 0x6000 0x0100 0x8040	1
5	0x0000 0x0100 0x8040 0x8000	0
6	0x0100 0x8040 0x8000 0x0000	0
7	0x0040 0x8000 0x0000 0x0001	1
8	0x0000 0x0000 0x0001 0x00C0	0
9	0x0000 0x0001 0x00C0 0x0000	0
10	0x0001 0x00C0 0x0000 0x0000	1
11	0x0000 0x0000 0x0000 0x0100	0
12	0x0000 0x0000 0x0100 0x0000	0
13	0x0000 0x0100 0x0000 0x0000	0
14	0x0100 0x0000 0x0000 0x0000	0
15	0x8000 0x0000 0x0000 0x0001	0
16	0x0100 0x0000 0x0001 0x0001	0
17	0x8000 0x0001 0x0001 0x0001	0
18	0x0101 0x0001 0x0001 0x0001	1
19	0x80C1 0x0001 0x0001 0x0101	2
20	0x0300 0x0001 0x0101 0x0103	1
21	0x0000 0x0101 0x0103 0x0002	0
22	0x0101 0x0103 0x0002 0x0000	7
23	0x8180 0x0002 0x0000 0x0105	1
24	0x0100 0x0000 0x0105 0x0201	0
25	0x8000 0x0105 0x0201 0x0001	0
26	0x0005 0x0201 0x0001 0x0001	2
27	0x0081 0x0001 0x0001 0x0500	2
28	0x0300 0x0001 0x0500 0x0182	1
29	0x0000 0x0500 0x0182 0x0002	0
30	0x0500 0x0182 0x0002 0x0000	2
31	0x0180 0x0002 0x0000 0x0004	1
32	0x0000 0x0000 0x0004 0x0200	0
33	0x0000 0x0004 0x0200 0x0000	0
34	0x0004 0x0200 0x0000 0x0000	1
35	0x0000 0x0000 0x0000 0x0600	0
36	0x0000 0x0000 0x0600 0x0000	0
37	0x0000 0x0600 0x0000 0x0000	0
38	0x0600 0x0000 0x0000 0x0000	1
39	0x0003 0x0000 0x0000 0x0006	1
40	0x0600 0x0000 0x0006 0x0006	1
41	0x0002 0x0000 0x0006 0x0004	-

References

[1] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee, "HIGHT: A New Block Cipher Suitable for Low-Resource Device," Cryptographic Hardware and Embedded Systems (CHES 2006), LNCS 4249, pp. 46-59, Springer, 2006.

[2] Ray Beaulieu, Douglas Shors, Jason Smith. "The SIMON and SPECK lightweight block ciphers," Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE. IEEE, pp. 1-6, Jun. 2015.

[3] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, Dong-Geon Lee. "LEA: A 128-bit Block Cipher for Fast Encryption on Common Processors," International Workshop on Information Security Applications (WISA'13), LNCS 8267, pp. 3-27, Springer, 2014.

[4] Bonwook Koo, Dongyoung Roh, Hyeonjin Kim, Younghoon Jung, Dong-Geon Lee and Daesung Kwon, "CHAM: A family of lightweight block ciphers for resource-constrained devices," International Conference on Information Security and Cryptology (ICISC'17), LNCS 10779, pp. 3-25, Springer, 2018.

[5] Dongyoung Roh, Bonwook Koo, Younghoon Jung, Il Woong Jeong, Dong-Geon Lee, Daesung Kwon and Woo-Hwan Kim, "Revised version of block cipher CHAM," International Conference on Information Security and Cryptology (ICISC'19), LNCS 11975, pp. 1-19, Springer, 2020.

[6] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisability modulo theories. Handbook of satisability, pp. 825-885, IOS Press, Apr. 2009.

[7] Nicky Mouha and Bart Preneel, "Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20," IACR ePrint 2013-328, Nov. 2013.

[8] Vijay Ganesh and David L. Dill, "A Decision Procedure for Bit-Vectors and Arrays," Computer Aided Verification 2007, LNCS 4590, pp. 519-531, Springer, 2007.

[9] Helger Lipmaa and Shiho Moriai. "Efficient algorithms for computing differential properties of addition," FSE 2001, LNCS 2355, pp. 336 - 350, Springer, 2002.

[10] Daniel J. Bernstein, "The Salsa20 Family of Stream Ciphers," New Stream Cipher Designs, LNCS 4986, pp. 84-97, Springer, 2008.

[11] Xuejia Lai, James L. Massey and Sean Murphy, "Markov Ciphers and Differential Cryptanalysis," EUROCRYPT 1991, LNCS 547, pp. 17 - 38. Springer, 1991.

[12] Yunwen Liu, Qingju Wang and Vincent Rijmen, "Automatic Search of Linear Trails in ARX with Applications to SPECK and Chaskey," Applied Cryptography and Network Security - 14th international conference(ACNS 2016), LNCS 9696, pp. 485-499, Springer, 2016

[13] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel and Ingrid Verbauwhede "Chaskey: an efficient MAC algorithm for 32-bit microcontrollers," Selected Areas in Cryptography(SAC 2014), LNCS 8781, pp. 306 - 323, Springer,

2014.
〔14〕 Ernst Schulte-Geers, "On CCZ-equivalence of addition mod 2^n," Designs, Codes and Cryptography. vol. 66, pp. 111 - 127, May, 2012.
〔15〕 Huang, Mingjiang, and Liming Wang. "Automatic Search for the Linear (Hull) Characteristics of ARX Ciphers: Applied to SPECK, SPARX, Chaskey, and CHAM 64," IACR ePrint 2019-1319, Jan. 2020.

〈저자소개〉

신 명 수 (Myoungsu Shin) 학생회원
2023년 2월: 전북대학교 IT정보공학과 학사
2023년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

김 선 규 (Seonkyu Kim) 학생회원
2023년 2월: 고려대학교 수학과 학사
2023년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

신 한 범 (Hanbeom Shin) 학생회원
2022년 2월: 서울시립대 수학과 학사
2022년 3월~2024년 2월: 고려대학교 융합보안학과 석사
2024년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

김 인 성 (Insung Kim) 학생회원
2022년 2월: 서울시립대 수학과 학사
2022년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

김 선 엽 (Sunyeop Kim) 학생회원
2019년 8월: 고려대학교 수학과 학사
2019년 9월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

권 동 근 (Donggeun Kwon) 학생회원
2018년 2월: 고려대학교 수학과 학사
2018년 3월~2020년 2월: 고려대학교 정보보호학과 석사
2020년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 부채널 공격, 딥러닝, 머신러닝 기반 암호분석, 암호 알고리즘 설계 및 분석, 대칭키 암호

홍 득 조 (Deukjo Hong) 종신회원
1999년 8월: 고려대학교 수학과 학사
2001년 8월: 고려대학교 수학과 석사
2006년 2월: 고려대학교 정보보호대학원 박사
2006년 3월~2007년 12월: 고려대학교 정보보호기술연구센터 연구교수
2007년 12월~2015년 8월: 국가보안기술연구소 선임연구원
2015년 9월~현재: 전북대학교 컴퓨터인공지능학부 부교수
〈관심분야〉 암호 알고리즘 설계 및 분석, 네트워크 및 시스템

성 재 철 (Jaechul Sung) 종신회원
2002년 8월: 고려대학교 수학과 박사
2002년 8월~2004년 1월: 한국정보보호진흥원 선임연구원
2004년 2월~현재: 서울시립대학교 수학과 전임강사, 조교수, 부교수, 교수
〈관심분야〉 암호 알고리즘 설계 및 분석

홍 석 희 (Seokhie Hong) 종신회원
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀리지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털 포렌식