A Short Universal Hash Function from Bit Rotation, and Applications to Blockcipher Modes

Kazuhiko Minematsu

NEC Corporation, 1753 Shimonumabe, Nakahara-Ku, Kawasaki, Japan
k-minematsu@ah.jp.nec.com

Abstract. In this paper we propose a new universal hash function based on bit rotation. The proposed scheme, called Circulant hash, is a variant of the classical random matrix-based hash of Carter and Wegman, called H_3, and Toeplitz hash by Krawczyk. However, Circulant hash has a smaller key space and the proved differential probability is not implied by the previous analyses on these functions.

Since Circulant hash is an almost XOR-universal hash function for balanced input/output, it may not be a perfect substitute for H_3 and Toeplitz hash. However, we show that Circulant hash is a useful tool for blockcipher modes, specifically as an alternative to Galois field constant multiplications. We provide some illustrative examples of the constructions of tweakable blockcipher and vector-input pseudorandom function using Circulant hash. Our schemes are as efficient as previous ones using GF constant multiplications, and provide some unique features.

Keywords: Bit rotation, Toeplitz hash, Blockcipher Mode.

1 Introduction

Bit rotation is one of the most basic operations appearing in numerous fields of computer science. In case of cryptography, bit rotation mainly serves as a basic tool for building cryptographic primitives [4,6,14,31,40]. This paper shows that bit rotation also offers a powerful tool in the field of provable security. We propose a simple bit rotation-based function, called Circulant hash, and show that it is ϵ-almost XOR universal (ϵ-AXU) hash function if the length of input vector satisfies certain conditions. As the name suggests, it is basically a matrix-vector product of a random circulant matrix over GF(2) and the input vector. Circulant hash can be seen as a variant of classical random matrix-based hash of Carter and Wegman [10], called H_3, or Toeplitz hash by Krawczyk [19], with a restriction to square matrix. One can also take it as an extension of Data-dependent rotation (DDR) by Rivest [29]. However, Circulant hash has a smaller key space than H_3 and Toeplitz hash, while much larger input space than DDR of the same key length.

Despite the simple look, proving the differential probability (i.e. the AXU bias) of Circulant hash is non-trivial. We prove that, for Circulant hash using κ-bit key

W. Susilo and R. Reyhanitabar (Eds.): ProvSec 2013, LNCS 8209, pp. 221–238, 2013.

and $(\kappa-1)$-bit input and κ-bit output, the differential probability is at most $2/2^\kappa$, if κ is a special prime (see Definition 1 and Lemma 1). This result is not implied by the previous analyses on H_3, Toeplitz hash, and DDR [10, 11, 19, 30, 37, 38]. In fact, our finding is based on an old paper by Daykin [12] discussing how to derive the rank of a matrix over a finite field, which has been overlooked by the cryptography community, to the best of our knowledge.

Circulant hash realizes an ϵ-AXU hash function having almost balanced input and output. When compared with square Toeplitz hash, Circulant hash has a reduced key length and hardware complexity, hence is a better substitute when (almost) square Toeplitz hash has been used, such as [9,22,23]. In contrast, even though we can basically extend the input length via tree hashing [10], it may not be appropriate for very long inputs.

We then show that Circulant hash provides a powerful tweaking tool for blockcipher modes. In the field of blockcipher modes, the constant multiplication over a Galois Field (GF) has been widely used as a tweaking tool [13, 16, 28, 33, 35]. We provide some illustrative examples showing that, Circulant hash can be an alternative to GF constant multiplication, or even more useful in some cases. We choose two illustrative applications. The first application is tweakable blockcipher (TBC) [21] based on a blockcipher. A previous TBC scheme called XEX [33] utilizes constant GF multiplications for efficient sequential tweak update. We build TBC using Circulant hash instead of constant GF multiplication. It allows efficient sequential tweak update as well, and also effectively handles certain non-sequential tweak update without using a precomputation, which may be useful in the real-world applications of TBC.

The second application is vector-input pseudorandom function (PRF). Rogaway and Shrimpton [35] proposed a concrete instantiation of vector-input PRF, called S2V, using a string-input PRF with a post-processing based on constant GF multiplications. In S2V, the computations of string-input PRFs are parallelizable, however the post-processing is logically serial. We show that, the post-processing can be replaced with (an decomposed form of) Circulant hash, which is essentially bit rotations and is fully parallelizable. Our proposal keeps the most features of S2V while achieves a faster parallel computation. Moreover, it enables powerful incremental update using the previous output (i.e., it is an incremental message authentication code (MAC) [5]), which is impossible with S2V. One can also use our result to build a fast, parallelizable short-input PRF.

These two examples imply that we can build a Circulant hash-based counterparts for the most of previous blockcipher modes utilizing GF constant multiplications, and the each of the resulting scheme exhibits some unique advantages.

2 Preliminaries

Let $\{0,1\}^n$ be the space of n-bit strings, and let $\{0,1\}^*$ be the space of all binary strings, including the empty string, ε. A bit length of a binary string X is written as $|X|$. We define $|X|_n \stackrel{\text{def}}{=} \lceil |X|/n \rceil$. Here $|\varepsilon| = 0$. The first (last) c bits of X is denoted by $\mathsf{msb}_c(X)$ ($\mathsf{lsb}_c(X)$). We write $\mathbb{N}_c$ to denote

$\{1, 2, \ldots, c\}$. A concatenation of two strings, X and Y, is written as $X\|Y$ or simply XY. A sequence of i zeros is written as 0^i. An i-bit left rotation of n-bit string $X = (X[1]\|X[2]\|\ldots\|X[n])$ is written as $X \lll i = (X[i+1]\|\ldots\|X[n]\|X[1]\|\ldots\|X[i])$. For $X, Y \in \{0,1\}^*$ with $|X| \leq |Y|$, let $X \oplus_{\mathrm{end}} Y$ be the XOR of X into the end of Y, i.e. $X \oplus_{\mathrm{end}} Y = (0^{|Y|-|X|}\|X) \oplus Y$. For $X \in \{0,1\}^*$, let $X[1]\|X[2]\|\ldots\|X[m] \xleftarrow{n} X$ denote the n-bit block partitioning of X, i.e., $X[1]\|X[2]\|\ldots\|X[m] = X$ where $m = |X|_n$, and $|X[i]| = n$ for $i < m$ and $|X[m]| \leq n$.

If X is uniformly distributed over set $\mathcal{X}$, we write $X \xleftarrow{\$} \mathcal{X}$. The set of all functions having n-bit inputs and m-bit outputs is denoted by $\mathrm{Func}(n,m)$ and the set of all n-bit permutations is denoted by $\mathrm{Perm}(n)$. A keyed function F with key $K \in \mathcal{K}$, input domain $\mathcal{X}$, and output domain $\mathcal{Y}$ is written as $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. We may write $F_K : \mathcal{X} \to \mathcal{Y}$ if the existence of key is obvious. A pair of two keyed functions, $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and $G : \mathcal{K}' \times \mathcal{X} \to \mathcal{Y}$, are said to be compatible (the key spaces are not necessarily the same).

We define the uniform random function (URF) $\mathsf{R} : \{0,1\}^n \to \{0,1\}^m$ as the keyed function with a key being uniform over $\mathrm{Func}(n,m)$. The n-bit uniform random permutation (URP), $\mathsf{P} : \{0,1\}^n \to \{0,1\}^n$ is a keyed permutation with a key being uniform over $\mathrm{Perm}(n)$. Note that the notion of URF can be extended to the case that input domain is an infinite set, say, $\{0,1\}^*$, by using the lazy sampling. The inverse of keyed permutation E_K (P) is written as E_K^{-1} (P^{-1}).

We define the two classes of universal hash function.

Definition 1. *For $H_K : \mathcal{X} \to \mathcal{Y}$, if $\Pr[H_K(x) = H_K(x')] \leq \epsilon$ for any distinct $x, x' \in \mathcal{X}$, H_K is ϵ-almost universal (ϵ-AU). If $\mathcal{Y} = \{0,1\}^n$ and $\Pr[H_K(x) \oplus H_K(x') = c] \leq \epsilon$ for any distinct $x, x' \in \mathcal{X}$ and $c \in \{0,1\}^n$, H_K is ϵ-almost XOR universal (ϵ-AXU).*

Note that if H_K is ϵ-AXU it is also ϵ-AU.

Pseudorandom Function. For a pair of compatible keyed function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and $G : \mathcal{K}' \times \mathcal{X} \to \mathcal{Y}$ and an adversary A who performs (possibly adaptive) chosen-plaintext queries and makes a binary output, we write

$$\mathtt{Adv}^{\mathtt{cpa}}_{F,G}(\mathsf{A}) \stackrel{\mathrm{def}}{=} \Pr[K \xleftarrow{\$} \mathcal{K} : \mathsf{A}^{F_K} \Rightarrow 1] - \Pr[K' \xleftarrow{\$} \mathcal{K}' : \mathsf{A}^{G_{K'}} \Rightarrow 1]$$

where $K \xleftarrow{\$} \mathcal{K} : A^{F_K} \Rightarrow 1$ denotes the event that A outputs 1 by querying F_K, when $K \xleftarrow{\$} \mathcal{K}$ is the underlying key sampling. Using URF compatible to $F_K : \{0,1\}^n \to \{0,1\}^m$, R, we write $\mathtt{Adv}^{\mathtt{prf}}_{F_K}(\mathsf{A})$ to denote $\mathtt{Adv}^{\mathtt{cpa}}_{F_K,\mathsf{R}}(\mathsf{A})$, which means $\Pr[K \xleftarrow{\$} \mathcal{K} : \mathsf{A}^{F_K} \Rightarrow 1] - \Pr[\mathsf{R} \xleftarrow{\$} \mathrm{Func}(n,m) : \mathsf{A}^{\mathsf{R}} \Rightarrow 1]$.

The definition of $\mathtt{Adv}^{\mathtt{prf}}_{F_K}(\mathsf{A})$ may be extended when F_K takes variable-length input in $\{0,1\}^*$. In this case the underlying R is replaced with \$ oracle that outputs independent and random value for any new input; for colliding inputs, the outputs are the same.

3 Universal Hash Function from Bit Rotation

3.1 Constructions Based on Matrix-Vector Product

In [10], Carter and Wegman introduced the idea of universal hash function and provided several examples. Among them a function called H_3 is particularly relevant to our proposal. Suppose we need a universal hash function of η-bit input and κ-bit output. The key of H_3 is a binary $\kappa \times \eta$ matrix, $\mathbb{M}$, whose elements are independent and random. Hence the key length is $\eta \cdot \kappa$ bits. For input vector $x \in \{0,1\}^\eta$, the output of H_3 is a matrix-vector product over GF(2), written as $\mathbb{M} \cdot x^T$, where x^T denotes the column vector of x.

Clearly H_3 provides $1/2^\kappa$-AXU[1] for any positive κ and η. Krawczyk [19] showed a variant of H_3 with reduced key bits, called Toeplitz hash. In Toeplitz hash the key is randomly sampled to specify the $\kappa \times \eta$ Toeplitz matrix over GF(2), $\mathbb{M}_T^{(\kappa,\eta)}$. For input $x \in \{0,1\}^\eta$ the κ-bit output is computed as the matrix-vector product over GF(2), i.e. $y = \mathbb{M}_T^{(\kappa,\eta)} \cdot x^T$. As $\mathbb{M}_T^{(\kappa,\eta)}$ has $(\kappa + \eta - 1)$ independent bits to be specified (i.e. the first column and row vectors), the key length is reduced to $(\kappa+\eta-1)$ bits. This keyed function has η-bit input and κ-bit output, and is $1/2^\kappa$-AXU [19].

In this paper, we present a new variant of H_3 having even reduced key space from Toeplitz, applicable when η is close to κ. The idea is to use a random circulant matrix, which requires only key of κ bits.

Definition 2. *Let κ be a positive integer. The Circulant hash (CLH for short) is a keyed function $: \{0,1\}^\kappa \times \{0,1\}^{\kappa-1} \to \{0,1\}^\kappa$ defined as*

$$\mathrm{CLH}_\kappa(K, x) = \bigoplus_{1 \le i \le \kappa-1:\ x[i]=1} (K \lll (i-1)),$$

where $x = (x[\kappa-1] \| \ldots \| x[1])$ and $x[i] \in \{0,1\}$.

For example, $\mathrm{CLH}_\kappa(K, 0^{\kappa-1}) = 0^\kappa$, and $\mathrm{CLH}_\kappa(K, 0^{\kappa-4}\|101) = K \oplus (K \lll 2)$. It is easy to see that $\mathrm{CLH}_\kappa(K, x)$ is equivalent to a matrix-vector product over GF(2), represented as $\mathbb{M}_C^{(\kappa,\kappa-1)} \cdot \overline{x}^T$, where $\mathbb{M}_C^{(\kappa,\kappa-1)}$ denotes the first $\kappa - 1$ columns of circulant matrix of order κ whose first column vector is the key K and $\overline{x}^T \in \{0,1\}^{\kappa-1}$ is the transposed input of $\overline{x} = (x[1] \| \ldots \| x[\kappa-1])$.

Despite the simple look, proving ϵ-AXU for CLH_κ turns out to be quite non-trivial. The fact that random matrix works fine with H_3 does not necessarily mean the goodness of reduced-key variants. For example, when $\kappa = 5$, we can see (by an exhaustive search) that $\mathrm{CLH}_5(K, x) \oplus \mathrm{CLH}_5(K, x')$ for any $x \neq x$ contains at least 4 independent bits of K, resulting in $1/2^4$-AXU, which is close to the theoretical minimum, $1/2^5$. However, when $\kappa = 8$, $\mathrm{CLH}_8(K, x) \oplus \mathrm{CLH}_8(K, x')$ has only 2 independent bits when $x \oplus x' = (1,0,1,0,1,0,1)$, thus the probability is $1/2^2$. When $\kappa = 7$, $\mathrm{CLH}_7(K, x) \oplus \mathrm{CLH}_7(K, x')$ with $x \oplus x' = (1,1,1,0,1,1,0)$ has 3 independent bits. This arises a natural question on the condition of κ that assures ϵ-AXU for a small ϵ. The following lemma shows the answer.

[1] [10] only proved that it is $1/2^\kappa$-AU, but it is easily extended to AXU.

Lemma 1. *Let* $K \xleftarrow{\$} \{0,1\}^\kappa$. *We have*

$$\max_{\substack{c\in\{0,1\}^\kappa,\\ x,x'\in\{0,1\}^{\kappa-1},\ x\neq x'}} \Pr_K[\mathrm{CLH}_\kappa(K,x)\oplus\mathrm{CLH}_\kappa(K,x')=c] \le \frac{2}{2^\kappa},\ \textit{and}$$

$$\max_{c\in\{0,1\}^\kappa,\ x\in\{0,1\}^{\kappa-1}\setminus\{0^{\kappa-1}\}} \Pr_K[\mathrm{CLH}_\kappa(K,x)=c] \le \frac{2}{2^\kappa},$$

when κ *is prime and* 2 *is the primitive root modulo* κ, *which we call* p*-prime.*

Proof. We first observe that $\mathrm{CLH}_\kappa(K,x)\oplus\mathrm{CLH}_\kappa(K,x') = \mathrm{CLH}_\kappa(K,x\oplus x')$, hence the first claim is proved by showing the maximum of probability $\Pr[\mathrm{CLH}_\kappa(K,x)=c]$ for all $c\in\{0,1\}^\kappa$ and $x\in\{0,1\}^{\kappa-1}\setminus\{0^{\kappa-1}\}$, i.e. proving the second claim also proves the first. Now, let $\mathbb{R}$ be $\kappa\times\kappa$ GF(2)-matrix defined as

$$\mathbb{R} = \begin{bmatrix} 0\,1\,0\ldots\,0 \\ 0\,0\,1\ldots\,0 \\ \vdots\,\vdots\,\vdots\ \ \vdots\ \ \vdots \\ 0\,0\,0\ldots\,1 \\ 1\,0\,0\ldots\,0 \end{bmatrix}. \tag{1}$$

Then we have $(K\lll i)^T = \mathbb{R}^i\cdot K^T$, where $\cdot$ is the matrix-vector multiplication over GF(2), and $\mathbb{R}^i$ denotes the matrix exponentiation over GF(2) (i.e. $\mathbb{R}^3 = \mathbb{R}\times\mathbb{R}\times\mathbb{R}$ with matrix multiplication $\times$). Here we define $\mathbb{R}^0$ as the identity matrix, thus $\mathbb{R}^0\cdot K^T$ means $(K\lll 0)=K$.

From the theory of linear systems, we have

$$\begin{aligned}
&\Pr[\mathrm{CLH}_\kappa(K,x)=c] \\
&= \Pr_K\left[\sum_{1\le i\le\kappa-1:x[i]=1}\mathbb{R}^i\cdot K = c\right] = \Pr_K\left[\left(\sum_{1\le i\le\kappa-1:x[i]=1}\mathbb{R}^i\right)\cdot K = c\right] \\
&= \frac{\left|k\in\{0,1\}^\kappa : \left(\sum_{i:x[i]=1}\mathbb{R}^i\right)\cdot k = c\right|}{2^\kappa} \qquad (2)\\
&\le \frac{2^{\kappa-\mathrm{rank}\left(\sum_{i:x[i]=1}\mathbb{R}^i\right)}}{2^\kappa} = \frac{1}{2^{\mathrm{rank}\left(\sum_{i:x[i]=1}\mathbb{R}^i\right)}}, \qquad (3)
\end{aligned}$$

where K is uniform over $\{0,1\}^\kappa$ and the matrix sums are over GF(2), and $\mathrm{rank}(\mathbb{M})$ denotes the rank of matrix $\mathbb{M}$ over GF(2). Hence, we have to prove that $\mathrm{rank}(\sum_{i\in\mathcal{I}}\mathbb{R}^i)\le\kappa-1$ for any nonempty $\mathcal{I}\subseteq\{0,\ldots,\kappa-2\}$.

For a finite field $\mathscr{F}$, let $\mathbb{M}$ be the matrix over $\mathscr{F}$. Let $\mathscr{F}[\mathbb{M}]$ be the set of all (nonempty) univariate polynomials for $\mathbb{M}$ with coefficients in $\mathscr{F}$. For instance $\mathscr{F}[\mathbb{M}]$ contains $a\cdot\mathbb{M}^2 + b\cdot\mathbb{M}^1 + c\cdot\mathbb{M}^0$, where $a,b,c\in\mathscr{F}$ and addition and multiplication are defined over $\mathscr{F}$. The corresponding $\mathscr{F}$-polynomial is

$f(x) = ax^2 + bx + c$. Specifically, we let $\mathscr{F} = \mathrm{GF}(2)$ and $\mathbb{M} = \mathbb{R}$, then $\sum_{i \in \mathcal{I}} \mathbb{R}^i$ is a member of $\mathrm{GF}(2)[\mathbb{R}]$. We then apply a useful formula of Daykin [12] which provides the $\mathscr{F}$-rank of a square matrix in $\mathscr{F}[\mathbb{M}]$ for any field $\mathscr{F}$ and matrix $\mathbb{M}$. Using Theorem 1 and Section 5 of [12], for any $f[\mathbb{R}] \in \mathrm{GF}(2)[\mathbb{R}]$ we have

$$\mathrm{rank}(f(\mathbb{R})) = \kappa - \mathrm{DegL}(x^\kappa - 1, f(x)) \tag{4}$$

where $\mathrm{DegL}(g(x), g'(x))$ denotes the degree of largest common factor of GF(2)-polynomials, g and g'. Here, $x^\kappa - 1$ is factored into $(x-1)(x^{\kappa-1}+x^{\kappa-2}+\cdots+x+1)$, where addition and subtraction are XOR, for any n. The latter factor is called the all-one polynomial (AOP). Because the degree of $f(\mathbb{R})$ we consider is at most $\kappa - 2$, if AOP of degree $\kappa - 1$ is irreducible over GF(2), $\mathrm{rank}(f(\mathbb{R}))$ is at least $\kappa - 1$. Here, Wah et al. [39] proved that over GF(2)-AOP of degree m is irreducible if and only if $m+1$ is prime and 2 is the primitive root modulo $m+1$. This proves the second claim, and thus concludes the proof. □

Lemma 1 shows that if κ satisfies the conditions, CLH_κ with $K \xleftarrow{\$} \{0,1\}^\kappa$ is $2/2^\kappa$-AXU, and that $\mathrm{CLH}^+_\kappa : \{0,1\}^\kappa \times (\{0,1\}^{\kappa-1} \times \{0,1\}^\kappa) \to \{0,1\}^\kappa$ defined as $\mathrm{CLH}^+_\kappa(K, (x_1, x_2)) = \mathrm{CLH}_\kappa(K, x_1) \oplus x_2$ is $2/2^\kappa$-AU. As a slight extension of the lemma, if K is not uniform but $\max_k \Pr[K = k] \le 1/2^p$ holds for some p, then the resulting CLH is $2/2^p$-AXU.

For example, $3, 5, 11, 13$, and 19 are p-primes. Larger values can be easily derived (e.g.) from the table [2] or by using software. Table 1 shows some examples, where $\kappa_{<2^i}$ ($\kappa_{<2^i}$) denotes the largest (smallest) κ being p-prime smaller (larger) than 2^i. It is worth noting that for many cases there exists a p-prime close to a power of two.

Table 1. Examples of p-primes

$\kappa_{<2^5}$	$\kappa_{>2^5}$	$\kappa_{<2^6}$	$\kappa_{>2^6}$	$\kappa_{<2^7}$	$\kappa_{>2^7}$	$\kappa_{<2^8}$	$\kappa_{>2^8}$	$\kappa_{<2^9}$	$\kappa_{>2^9}$	$\kappa_{<2^{10}}$	$\kappa_{>2^{10}}$	$\kappa_{<2^{11}}$	$\kappa_{>2^{11}}$
29	37	61	67	107	131	227	269	509	523	1019	1061	2029	2053

3.2 Useful Variants

The output and key lengths of CLH are prime, however we frequently need a function of n-bit output with n-bit key, for n being a power of two. For this purpose, we define two variants of CLH.

Definition 3. *Let $\kappa \le n \le \lambda$. Let $f^1_{n,\kappa} : \{0,1\}^n \times \mathbb{N}_{\kappa-1} \to \{0,1\}^n$ and $f^2_{n,\lambda} : \{0,1\}^n \times \mathbb{N}_{\lambda-1} \to \{0,1\}^n$, where*

$$f^1_{n,\kappa}(K, i) = (\mathsf{msb}_\kappa(K) \lll i) \| 0^{n-\kappa},$$
$$f^2_{n,\lambda}(K, i) = \mathsf{msb}_n(K \| 0^{\lambda-n} \lll i),$$

and we define $\mathrm{CLH}'_{n,\kappa} : \{0,1\}^n \times \{0,1\}^{\kappa-1} \to \{0,1\}^n$ *and* $\mathrm{CLH}''_{n,\kappa} : \{0,1\}^n \times \{0,1\}^{\lambda-1} \to \{0,1\}^n$ *as*

$$\mathrm{CLH}'_{n,\kappa}(K,x) \stackrel{\mathrm{def}}{=} \bigoplus_{1\le i\le \kappa-1:\ x[i]=1} f^1_{n,\kappa}(K,i), \quad \textit{for } x = (x[\kappa-1]\|\dots\|x[1]),$$

and

$$\mathrm{CLH}''_{n,\lambda}(K,x) \stackrel{\mathrm{def}}{=} \bigoplus_{1\le i\le \lambda-1:\ x[i]=1} f^2_{n,\lambda}(K,i), \quad \textit{for } x = (x[\lambda-1]\|\dots\|x[1]).$$

Note that $\mathrm{CLH}'_{n,\kappa}(K,x)$ and $\mathrm{CLH}''_{n,\lambda}(K,x)$ are respectively equivalent to $\mathrm{CLH}_\kappa(\mathsf{msb}_\kappa(K),x)\|0^{n-\kappa}$ and $\mathsf{msb}_n(\mathrm{CLH}_\lambda(K\|0^{\lambda-n},x))$, and when $\kappa = n = \lambda$, they are the same as the original CLH_κ. Both $f^1_{n,\kappa}(K,i)$ and $f^2_{n,\lambda}(K,i)$ can be computed with two shifts and one logic operation.

We have the following lemma.

Lemma 2. *Let K be uniform over $\{0,1\}^n$. For $\kappa \le n \le \lambda$ we have*

$$\max_{\substack{c\in\{0,1\}^n,\\ x,x'\in\{0,1\}^{\kappa-1},\ x\ne x'}} \Pr_K[\mathrm{CLH}'_{n,\kappa}(K,x)\oplus\mathrm{CLH}'_{n,\kappa}(K,x')=c] \le \frac{2}{2^\kappa}, \textit{ and}$$

$$\max_{\substack{c\in\{0,1\}^n,\\ x\in\{0,1\}^{\kappa-1}\setminus\{0^{\kappa-1}\}}} \Pr_K[\mathrm{CLH}'_{n,\kappa}(K,x)=c] \le \frac{2}{2^\kappa},$$

$$\max_{\substack{c\in\{0,1\}^n,\\ x,x'\in\{0,1\}^{\lambda-1},\ x\ne x'}} \Pr_K[\mathrm{CLH}''_{n,\lambda}(K,x)\oplus\mathrm{CLH}''_{n,\lambda}(K,x')=c] \le \frac{2}{2^{2n-\lambda}}, \textit{ and}$$

$$\max_{\substack{c\in\{0,1\}^n,\\ x\in\{0,1\}^{\lambda-1}\setminus\{0^{\lambda-1}\}}} \Pr_K[\mathrm{CLH}''_{n,\lambda}(K,x)=c] \le \frac{2}{2^{2n-\lambda}},$$

when κ and λ are p*-primes.*

The proof of Lemma 2 is a simple extension of the proof of Lemma 1 (the bound of the last two claims are obtained as $(2/2^n)\cdot 2^{\lambda-n} = 2/2^{2n-\lambda}$), hence omitted. For example, $\mathrm{CLH}'_{64,61}(K,x)$ and $\mathrm{CLH}''_{64,71}(K,x)$ implement about 64-bit input/output space with differential probability $1/2^{60}$, and $\mathrm{CLH}''_{128,131}(K,x)$ implements 130-bit input, 128-bit output space with differential probability $1/2^{124}$.

3.3 Notes

Relation to DDR. The keyed DDR, defined as $\mathrm{DDR}(K,x) \stackrel{\mathrm{def}}{=} (K \lll x)$ for $x \in \{0,\dots,\kappa-1\}$ with $|K| = \kappa$, is $2/2^\kappa$-AXU if κ is prime [11]. However, the $\log|K|$ input space is too small for most practical applications. With CLH, we can extend the input space from $\log|K|$ to $|K|/2$.

Toeplitz Hash with LFSR. In generation of $\kappa \times \eta$ Toeplitz matrix, Krawczyk [19] also suggested to use the $(\kappa+\eta-1)$-bit output of κ-bit linear feedback shift register (LFSR). If the initial seed of LFSR is uniformly chosen from $\{0,1\}^\kappa$ and the feedback polynomial is uniformly chosen from the set of *all irreducible polynomials*, the resulting Toeplitz hash is $2\eta/2^\kappa$-AXU [3, 19]. In this case the key can be represented as a pair of κ-bit strings, K_1 and K_2, where K_1 specifies the coefficients of feedback polynomial and K_2 specifies the initial seed of LFSR. The hardware implementation requires an κ-bit accumulator register and an κ-bit LFSR [19]. As pointed out by [26, 36] the K_1's distribution is not uniform over $\{0,1\}^\kappa$ and is hard to determine if κ is large, say, 80.

Compared with $\kappa \times \kappa$ square Toeplitz hash, CLH can roughly halve the key bits. Table 2 provides a comparison of Toeplitz and Circulant hashs for the accumulator-based hardware implementation. It shows that, as an AXU hash function of balanced I/O, CLH provides a smaller footprint while keeping the small differential probability (DP). This will be useful for some applications, e.g. [9, 22, 23].

Extending Input Length. When we want to extend input length, we can use Tree hashing [10] with CLH^+_κ of Section 3.1, in a similar manner to Badger [8]. At the cost of logarithmic key increase, we can process a long input with small circulant matrices. Effectiveness of such implementation is an interesting future topic.

Table 2. Comparison of Toeplitz and Circulant hashs for accumulator-based hardware implementation. DP $= \epsilon$ means that the function is ϵ-AXU. For Circulant hash we require κ to be a p-prime.

Function	I/O (bit)	Key (bit)	ShReg (bit)	ShReg Feedback	DP
Toeplitz (LFSR) [19]	κ/κ	2κ	κ	Key-dep. IRPoly	$2\kappa/2^\kappa$
Toeplitz (Naive) [19]	κ/κ	$2\kappa-1$	$2\kappa-1$	Nothing	$1/2^\kappa$
Circulant (This paper)	$\kappa-1/\kappa$	κ	κ	Rotation	$2/2^\kappa$

4 Tweakable Blockcipher

We describe how to use CLH for blockcipher modes of operations. Our first target is tweakable blockcipher (TBC), proposed by Liskov et al. [21].

4.1 Definition of Tweakable Blockcipher

TBC is a keyed permutation with auxiliary input called tweak. Formally, a ciphertext of a TBC, $\widetilde{E}_K : \mathcal{M} \times \mathcal{T} \to \mathcal{M}$, is $C = \widetilde{E}_K(M,T)$ for plaintext $M \in \mathcal{M}$ and tweak $T \in \mathcal{T}$. The encryption, $\widetilde{E}_K$, must be a keyed permutation over $\mathcal{M}$ for every $T \in \mathcal{T}$, and the decryption is defined as $\widetilde{E}_K^{-1}(C,T) = M$ with $\widetilde{E}_K^{-1} : \mathcal{M} \times \mathcal{T} \to \mathcal{M}$. We here assume $\mathcal{M} = \{0,1\}^n$ for some fixed n and $\mathcal{T}$ is a certain finite set. TBC works as a building-block of blockcipher modes for various purposes [13, 15, 21, 27, 33].

To define the security, let $\mathrm{Perm}(\mathcal{T}, n)$ be the set of all mappings from $\mathcal{T}$ to n-bit permutations. The size of $\mathrm{Perm}(\mathcal{T}, n)$ is $|\mathrm{Perm}(n)|^{|\mathcal{T}|}$. The sampling $\widetilde{\mathsf{P}} \xleftarrow{\$} \mathrm{Perm}(\mathcal{T}, n)$ implements a set of independent n-bit URPs indexed by $T \in \mathcal{T}$, where $\widetilde{\mathsf{P}}$ and $\widetilde{\mathsf{P}}^{-1}$ have the same interfaces as $\widetilde{E}_K$ and $\widetilde{E}_K^{-1}$. The security notion for $\widetilde{E}_K$ is the indistinguishability from $\widetilde{\mathsf{P}}$ under a chosen-ciphertext attack (CCA), that is,

$$\begin{aligned}&\mathbf{Adv}^{\mathbf{tsprp}}_{\widetilde{E}}(\mathsf{A})\\ &\stackrel{\mathrm{def}}{=} \Pr[K \xleftarrow{\$} \mathcal{K} : \mathsf{A}^{\widetilde{E}_K, \widetilde{E}_K^{-1}} \Rightarrow 1] - \Pr[\widetilde{\mathsf{P}} \xleftarrow{\$} \mathrm{Perm}(\mathcal{T}, n) : \mathsf{A}^{\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1}} \Rightarrow 1], \qquad (5)\end{aligned}$$

where A^{O_1, O_2} denotes the adversary A querying two oracles, O_1 and O_2, in an arbitrary order.

4.2 Previous Constructions

Liskov et al. [21] showed how to build a secure TBC in the sense of Eq. (5), using $E_K : \mathcal{M} \to \mathcal{M}$ and an ϵ-AXU hash, $H_L : \mathcal{T} \to \mathcal{M}$, for independent keys, K and L. Extending the idea of [21], Rogaway proposed a one-key variant called XEX [33] using $\mathrm{GF}(2^n)$ constant multiplications. Let $\alpha_1, \ldots, \alpha_k$ be the distinct non-zero elements of $\mathrm{GF}(2^n)$ called bases. For each α_i we define the set of allowed indices, $\mathbb{I}_i \subseteq \mathbb{Z}$, which is an integer interval (e.g. $\mathbb{I}_i = [0 \ldots 10]$). The tweak space of (basic) XEX is $\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2$ with $\mathcal{T}_1 = \mathbb{I}_1 \times \cdots \times \mathbb{I}_k$, $\mathcal{T}_2 = \{0,1\}^n$, and it is defined as

$$\mathrm{XEX}[E_K](M, T) = E_K(M \oplus \Gamma \cdot E_K(T_2)) \oplus \Gamma \cdot E_K(T_2), \qquad (6)$$

where tweak is $T = (T_1, T_2)$ with $T_1 = (i_1, \ldots, i_k)$ and $\Gamma = \alpha_1^{i_1} \cdot \alpha_2^{i_2} \cdot \cdots \cdot \alpha_k^{i_k}$, and the multiplications are over $\mathrm{GF}(2^n)$. The multiplication $\Gamma \cdot E_K(T_2)$ is also over $\mathrm{GF}(2^n)$ by seeing $E_K(T_2)$ as a coefficient vector of a polynomial in $\mathrm{GF}(2^n)$. The security in terms of Eq. (5) is proved when bases and $\mathcal{T}_1$ satisfy some conditions (see [33]). The point of such construction is that the sequential update of a component index of T_1, i.e., $i_j \to i_j + 1$ for some j, can be quite efficient if we cache the previous value of Γ, because it is essentially the multiplication of the cached Γ by α_j. Typically we set $\alpha_1 = \mathbf{2}$ (the primitive element) since the multiplication by $\mathbf{2}$ is particularly simple. In this case the update procedure is called "doubling".

Alternatively, based on [21], we could simply use GF multiplication. Assuming n-bit tweak T and n-bit second key[2] L, we take a multiplication of L and T, denoted by $L \cdot T$, and encrypt as $E_K(M \oplus L \cdot T) \oplus L \cdot T$. By precomputing all powers of L, i.e. $L, \mathbf{2}L, \ldots, \mathbf{2}^{n-1}L$, $L \cdot T$ is computed as $L \cdot T = \oplus_{i:T[i]=1} \mathbf{2}^{i-1}L$, where $T[i]$ denotes the i-th bit of T for $i = 1, \ldots, n$. Thanks to the precomputed powers of L, this scheme enables an efficient incrementation of T using Gray code (see Section 4.3).

[2] With a slight modification one can generate L from K in a similar manner to XEX, see [25] (Section 5).

4.3 Tweakable Blockcipher Using CLH

We present a single-key TBC based on CLH, in a similar manner to XEX.

Definition 4. *Let $\kappa \le n \le \lambda$. Let E_K be an n-bit blockcipher. The single-key TBCs,* XEX-R1$[E_K]$ *and* XEX-R2$[E_K]$*, are defined as*

$$\text{XEX-R1}[E_K](M,T) \stackrel{\text{def}}{=} E_K(M \oplus \Gamma_1) \oplus \Gamma_1, \text{ and },$$
$$\text{XEX-R2}[E_K](M,T) \stackrel{\text{def}}{=} E_K(M \oplus \Gamma_2) \oplus \Gamma_2,$$

where $\Gamma_1 = \text{CLH}'_{n,\kappa}(E_K(T_2), T_1)$ and $\Gamma_2 = \text{CLH}''_{n,\lambda}(E_K(T_2), T_1)$.

Here, a tweak is $T = (T_1, T_2) \in \mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2$ with $\mathcal{T}_2 = \{0,1\}^n$. XEX-R1 *has $\mathcal{T}_1 = \{0,1\}^{\kappa-1} \setminus \{0^{\kappa-1}, 0^{\kappa-2}1\}$.* XEX-R2 *has $\mathcal{T}_1 = \{0,1\}^{\lambda-1} \setminus \{0^{\lambda-1}, 0^{\lambda-2}1\}$.*

When the underlying blockcipher is a URP, the security of our schemes are proved as follows. The computational counterparts are trivial.

Theorem 1. *Suppose κ and λ are* p*-primes. Let* P *be an n-bit URP, and let* A *be an adversary against TBC, using q CCA-queries. Then we have*

$$\mathtt{Adv}^{\mathtt{tsprp}}_{\text{XEX-R1[P]}}(\mathsf{A}) \le \left(\frac{6}{2^\kappa} + \frac{4}{2^n}\right) q^2, \quad \mathtt{Adv}^{\mathtt{tsprp}}_{\text{XEX-R2[P]}}(\mathsf{A}) \le \left(\frac{8}{2^{2n-\lambda}} + \frac{2}{2^n}\right) q^2.$$

Proof. See Appendix A.

When $n = 128$, XEX-R1 with $\kappa = 107$ provides about 52-bit security with 106-bit tweak, and XEX-R2 with $\lambda = 131$ provides about 61-bit security with 130-bit tweak.

Properties. Our proposals enable efficient sequential updates of T_1. For simplicity, let us assume $\kappa = n$. Then, since CLH_κ is XOR-linear, the computation of $\text{CLH}_\kappa(L, T_1')$ using $\text{CLH}_\kappa(L, T_1)$ (for $L = E_K(T_2)$) is easy if the hamming weight of $T_1 \oplus T_1'$ is small. To fully utilize this property we can use Gray code in a similar manner to the previous works [20, 34], which is as follows. We first take $T_1 \in \mathcal{T}_1$ as a positive integer, thus $2 \le T_1 \le 2^{\kappa-1} - 1$, and we modify Definition 4 so that the input to CLH is Gray code of T_1, $\text{gc}(T_1)$. This causes no security degradation since Gray code is a permutation and $\text{gc}(0^{\kappa-1}) = 0^{\kappa-1}$ and $\text{gc}(0^{\kappa-2}1) = 0^{\kappa-2}1$. Let $Z = \text{CLH}_\kappa(L, \text{gc}(T_1 - 1))$ and $Z' = \text{CLH}_\kappa(L, \text{gc}(T_1))$. We want to compute Z' using Z. From the property of Gray code we have

$$\begin{aligned} Z' &= Z \oplus \text{CLH}_\kappa(L, \text{gc}(T_1) \oplus \text{gc}(T_1 - 1)) \\ &= Z \oplus \text{CLH}_\kappa(L, (0 \ldots 01 \ll \mathtt{ntz}(T_1))) = Z \oplus (L \lll (\mathtt{ntz}(T_1) + 1)), \end{aligned}$$

where $\mathtt{ntz}(v)$ denotes the number of trailing zero for v (e.g. $\mathtt{ntz}(0100) = 2$). This can be quite efficient; most CPUs natively support an `ntz` instruction and there exist fast generic methods [1]. Moreover, this does not require any precomputation on L or additional blockcipher calls. In general, the computation of $\text{CLH}_\kappa(L, \text{gc}(T_1'))$ from $\text{CLH}_\kappa(L, \text{gc}(T_1))$ is fast as long as the weight of

$\mathrm{gc}(T_1) \oplus \mathrm{gc}(T_1')$ is small. That is, we can easily "jump" to such T_1'. Though conceptually a similar operation is possible with XEX using multiple bases, ours seems to have more flexibility. The above method can be easily extended to the case $\kappa < n$ or $n < \lambda$, using $\mathrm{CLH}'_{n,\kappa}$ or $\mathrm{CLH}''_{n,\lambda}$, where the latter needs to keep κ-bit output before truncation. We remark that jump operation with Gray code trick is also possible with a TBC construction described in the last of Section 4.2, that is, mask is generated by $\mathrm{GF}(2^n)$ multiplication based on the precomputed powers, $\{\mathbf{2}^i L\}_{i=0,\ldots,n-1}$.

In summary, our CLH enables incremental tweak update and certain non-incremental (jump) update without precomputation, while the basic form of doubling enables only incremental update. GF multiplication using precomputed powers enables both incremental and non-incremental updates, though the cost of precomputation and memory can be problematic, in particular for constrained devices.

If our TBCs replace blockcipher modes where internal tweak update is mostly sequential (e.g. OCB, PMAC [33], and XTS [13]), ours enable additional functionalities, such as selective decryption, without harming the efficiency of normal operation. If we built an online cipher using TBC [27], internal TBC has random tweaks. In [27], using GF multiplication is suggested, however using CLH' or CLH'' may be another option.

Software Results. According to our experiments, even random input to CLH is manageable. We implement $\mathrm{CLH}'_{64,61}$ on Intel Xeon E5620 (2.4GHz) and 64-bit Windows OS, using C with `ntz` instruction, called `BitScanForward`. It processes random inputs using 22 cycles per byte (cpb). For random inputs with weight 16 it runs at about 7.5 cpb, and for sequential update with Gray code, it runs at below 0.5 cpb. The same performance can be obtained for parallel computing of two $\mathrm{CLH}'_{64,61}$ functions by using XMM registers and SSE intrinsics. For reference, a naive C implementation of doubling function, $\mathrm{dbl}_L(i) = \mathbf{2}^i L$ for $L \in \mathrm{GF}(2^{64})$, runs at 1.38 cpb for $i = 1$, 18.6 cpb for $i = 10$, and 52.5 cpb for $i = 30$ on the same platform.

5 Vector-Input PRF

5.1 Construction of S2V-R

For string $X[i] \in \{0,1\}^*$ with $i = 1, \ldots, \ell$, we call $X = (X[1], \ldots, X[\ell])$ a *vector*. Let $\{0,1\}^{**} \stackrel{\mathrm{def}}{=} \bigcup_{\ell=0,1,2,\ldots} \{(X[1], \ldots, X[\ell]) : X[i] \in \{0,1\}^*\}$, i.e. the set of all vectors. Note that $\{0,1\}^{**}$ includes the empty vector (which contains no string) which is denoted by ε_v. Rogaway and Shrimpton [35] called a PRF of input domain $\{0,1\}^{**}$ a vector-input PRF (vPRF). They showed how to build vPRF: $\{0,1\}^{**} \to \{0,1\}^n$ from a string-input PRF, sPRF: $\{0,1\}^* \to \{0,1\}^n$ such as CMAC [16]. Their construction, called S2V, is used as a component of a deterministic AE (DAE) called SIV. S2V uses GF constant multiplications in a different way from XEX of Section 4. For reference it is presented in Appendix

Algorithm S2V-R$[f, F_K](X[1], \ldots, X[\ell])$, $0 \le \ell \le t-1$

1. $S \leftarrow 0^n$, $L \leftarrow F_K(0^n)$
2. **if** $\ell = 0$ **then return** $F_K(f(L,t))$
3. **for** $i \leftarrow 1$ **to** $\ell - 1$ **do** $S \leftarrow S \oplus f(F_K(X[i]), i)$
4. **if** $|X[\ell]| \ge n$ **then** $V \leftarrow (S \oplus f(L, t-1)) \oplus_{\mathrm{end}} X[\ell]$
5. **else** $V \leftarrow S \oplus f(L,t) \oplus X[\ell]\|10^*$
6. **return** $F_K(V)$

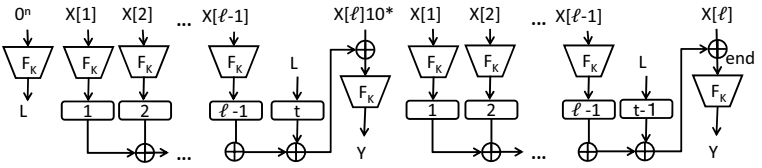

Fig. 1. Vector-input PRF using $F_K : \{0,1\}^* \to \{0,1\}^n$ and post-processing $f : \{0,1\}^n \times \mathbb{N}_t \to \{0,1\}^n$. In the lower figure, the box with $i = 1, 2, \ldots$ denotes the post-processing $f(*, i)$.

B. Building a vPRF from an sPRF is basically possible by first applying an invertible function (encoding) $g : \{0,1\}^{**} \to \{0,1\}^*$ to the input vector then applying the sPRF to the encoded string. However, as explained by [35], S2V has a number of practical advantages over this naive construction.

This section shows a new S2V-like vPRF. Our vPRF, which we call S2V-R, can be based on any sPRF, $F_K : \{0,1\}^* \to \{0,1\}^n$. The pseudo-code and the figure are given in Fig. 1, where $X[i]\|10^*$ denotes the padding, $X[i]\|10^{n-1-|X[i]|}$ for $0 \le |X[i]| \le n-1$. The key component of our proposal is the post-processing function, $f : \{0,1\}^n \times \mathbb{N}_t \to \{0,1\}^n$, applied to the outputs of underlying sPRF. Here, t denotes the maximum post-processing variations and each vector can contain at most $t-1$ strings. We show that, f can be a (variant of) unit computation of CLH_κ, i.e., a bit rotation of the input.

Let $\mathsf{R}_{**} : \{0,1\}^{**} \to \{0,1\}^n$ be the vector-input URF. For security notion of a vector-input keyed function, $F_K : \{0,1\}^{**} \to \{0,1\}^n$, we write $\mathtt{Adv}^{\mathtt{prf}}_{F_K}(\mathsf{A})$ to mean the indistinguishability of F_K from R_{**} under a CPA-adversary $\mathcal{A}$. The security bound of our proposal is as follows.

Theorem 2. *Let $f : \{0,1\}^n \times \mathbb{N}_t \to \{0,1\}^n$ be a post-processing function satisfying*

$$\max_{\mathcal{I} \subseteq \mathbb{N}_t, \mathcal{I} \neq \emptyset, c \in \{0,1\}^n} \Pr\left[U \xleftarrow{\$} \{0,1\}^n : \bigoplus_{i \in \mathcal{I}} f(U,i) = c\right] \le p_f$$

for $1/2^n \le p_f \le 1$. Let $\mathsf{R} : \{0,1\}^* \to \{0,1\}^n$ be URF, and let S2V-R$[f, \mathsf{R}]$ *be* S2V-R *using f and R. Let A be an adversary querying* S2V-R$[f, \mathsf{R}]$ *with q*

chosen-plaintext queries and the total number of component strings among q queries being σ_s*. Then we have*

$$\mathtt{Adv}^{\mathtt{prf}}_{\text{S2V-R}[f,\mathsf{R}]}(\mathsf{A}) \leq (2q\sigma_s + q^2)p_f.$$

Corollary 1. *For* $n = 128$*, we define* S2V-R1$[F_K]$ *and* S2V-R2$[F_K]$ *as* S2V-R$[f^1_{128,107}, F_K]$ *and* S2V-R$[f^2_{128,131}, F_K]$ *using* $f^1_{128,107}$ *and* $f^2_{128,131}$ *of Definition 3. Then,* S2V-R1 *can accept a vector of* 105 *strings, with security bound* $(4q\sigma_s + 2q^2)/2^{107}$*, and* S2V-R2 *can accept a vector of* 129 *strings, with security bound* $(2q\sigma_s + q^2)/2^{124}$*.*

The proof of Theorem 2 will be given in the full version. The proof of Corollary 1 is obtained by Theorem 2 and Lemma 2.

5.2 Properties of S2V-R

Basic Points. We could implement S2V-R with F_K being (e.g.) CMAC-AES or HMAC-SHA2. If $L = F_K(0^n)$ is precomputed S2V-R$[f, F_K]$ requires one F_K invocation to process one string. These features are shared with the original S2V. The acceptable number of component strings in a vector is largely the same as S2V, which accepts at most $n-1$ strings. One can build a DAE using S2V-R in the same manner as SIV.

In sequential computation, the computation cost of S2V-R is basically the same as S2V. A difference arises in parallel computation. As well as S2V, the computations of $F_K(X[i])$ in S2V-R are parallelizable. Moreover, S2V-R allows the parallel computation of the post-processing after $F_K(X[i])$, namely bit rotations, while those of S2V is sequential constant multiplications (See Appendix B). This implies that our proposal enables a faster parallel computation. We remark that a variant using a powering-based post-processing, e.g., $f(x, i) = \mathbf{2}^i x$, is also possible. This has the same parallelizability as S2V-R1 or S2V-R2, however the computation cost is much higher.

Short-Input PRF. When we implement F_K by an n-bit blockcipher, E_K, the resulting S2V-R$[f, E_K]$ is a PRF accepting short inputs, i.e. at most $n(t-1)$ bits. For instance, S2V-R2 of Corollary 1 accepts $16 \cdot 128 = 2$Kbyte inputs, which is enough for most of the packet communications[3]. In case of the parallel processing, S2V-R2 with blockcipher is advantageous compared to PMAC, as PMAC needs serial mask computation of $\mathbf{2}^i E_K(0^n)$ for $i = 1, \ldots, 128$, or, needs 2Kbyte memory to store the precomputed masks. In hardware (parallel) implementation, the post-processing of S2V-R1 and S2V-R2 are just wires, hence quite fast and small.

Incremental Update. One unique feature of S2V is that it efficiently handles static (invariant) strings. More generally, once we have computed the

[3] For example IPSec authenticates packets of 43 to $1.5K$ Bytes.

output for an input vector $(X[1], \ldots, X[\ell])$ and cached the outputs of F, $\{F_K(X[1]), \ldots, F_K(X[\ell-1])\}$, the output computation for the next input, $(X'[1], \ldots, X'[\ell'])$, requires only the computations of $F_K(X'[i])$ for all $X'[i] \notin \{X[1], \ldots, X[\ell-1]\}$. That is, a restricted form of incremental update. An incremental update for vPRF is particularly valuable when component strings can be long. Our S2V-R shares this feature. Moreover, if the post-processing is commutative (i.e. $f(f(x,i),j) = f(f(x,j),i) = f(x,i+j)$), as with S2V-R1, we can say much more about the incremental operation. Suppose the last string is at most n bits and F_K is invertible for n-bit inputs, which is satisfied with (e.g.) CMAC. Then, S2V-R allows the incremental update from previous outputs, without caching the internal F_K outputs. As well as PMAC [7], this update is secure under the basic security notion for incremental update defined by [5]. For example, suppose we have $Y = \text{S2V-R}[f, F_K](X)$ for $X = (X[1], \ldots, X[\ell])$ with $X[i] \in \{0,1\}^*$ for $i \leq \ell - 1$ and $|X[\ell]| \leq n$. Let us write $X_{<i} = X[1] \| \cdots \| X[i-1]$ and $X_{>i} = X[i+1] \| \cdots \| X[\ell]$. Then, the output computation for a new vector, $(X_{<i} \| X'[i] \| X_{>i})$ for some $X'[i] \neq X[i]$, can be done as

1. $V' \leftarrow F_K^{-1}(Y)$
2. $V' \leftarrow V' \oplus f(F_K(X[i]), i) \oplus f(F_K(X'[i]), i)$
3. $Y' \leftarrow F_K(V')$,

where F_K^{-1} denotes the inversion for n bits. Namely, we can handle the `replace` operation written as $X \rightarrow (X_{<i} \| X'[i] \| X_{>i})$. Similarly, `truncate`, $X \rightarrow X_{<\ell}$, and `append`, $X \rightarrow X \| X'[\ell+1]$, are efficiently handled. We remark that the same (block-wise) update operations are also supported by PMAC [7,33].

Thanks to the nature of rotation, we can do even more. When $X|[\ell]| = n$, `insert` operation, $X \rightarrow X'[1] \| X$, is also possible as

1. $V' \leftarrow F_K^{-1}(Y) \oplus f(L, t-1) \oplus X[\ell]$
2. $V' \leftarrow f(V', 1) \oplus f(F_K(X[1]), 1) \oplus f(L, t-1) \oplus X[\ell]$
3. $Y' \leftarrow F_K(V')$,

where $L = F_K(0^n)$. Generally, if we insert a string $X'[i]$ before $X[i]$, the update requires $\min\{i, \ell - i\}$ F_K calls with few additional F_K and F_K^{-1} calls, thus we can save *at least* the half of F_K calls. One more example, `merge` operation, which means the output computation for $X \| X'$ using $Y_1 = \text{S2V-R}[f, F_K](X)$ and $Y_2 = \text{S2V-R}[f, F_K](X')$, also possible with few F_K calls. There should be more examples of practical, application-specific incremental operations that can be handled by S2V-R, and the set of these update operations can offer a very powerful incremental vPRF beyond the ability to handle static strings.

6 Conclusion

This paper has presented Circulant hash, a simple keyed hash function consisting of bit rotations and XORs. We showed that it is ϵ-AXU for ϵ close to the minimum if the length of rotated vectors satisfies certain conditions. Circulant hash can be a good alternative to the famous Toeplitz hash in case we need an

ϵ-AXU hash of balanced I/O lengths. We also showed that Circulant hash works as a powerful tweaking tool for blockcipher modes, and presented two illustrative examples for tweakable blockcipher and vector-input PRF.

Acknowledgments. The author would like to thank Norifumi Kamiya for the discussion on the work of Daykin. The author also would like to thank Mohammad Reza Reyhanitabar for constructive suggestions, and the anonymous reviewers for many useful comments, in particular for pointing out H_3 function of Carter and Wegman.

References

1. Chess Programming Wiki, `http://chessprogramming.wikispaces.com/`
2. The On-Line Encyclopedia of Integer Sequences: A046145 Smallest primitive root of n, or 0 if no root exists, `http://oeis.org/A046145/`
3. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple Constructions of Almost k-Wise Independent Random Variables. In: FOCS, pp. 544–553. IEEE Computer Society (1990)
4. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE, Round 2 (2009)
5. Bellare, M., Goldreich, O., Goldwasser, S.: Incremental cryptography and application to virus protection. In: Leighton, F.T., Borodin, A. (eds.) STOC, pp. 45–56. ACM (1995)
6. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, Billet (eds.) [32], pp. 84–97
7. Black, J., Rogaway, P.: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In: Knudsen (ed.) [18], pp. 384–397
8. Boesgaard, M., Christensen, T., Zenner, E.: Badger – A Fast and Provably Secure MAC. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 176–191. Springer, Heidelberg (2005)
9. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient Helper Data Key Extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
10. Carter, L., Wegman, M.N.: Universal Classes of Hash Functions. J. Comput. Syst. Sci. 18(2), 143–154 (1979)
11. Contini, S., Yin, Y.L.: On differential properties of data-dependent rotations and their use in MARS and RC6 (Extended Abstract). In: Proceedings of the Second AES Candidate Conference, pp. 230–239 (2000)
12. Daykin, D.E.: On the Rank of the Matrix f(A) and the Enumeration of Certain Matrices over a Finite Field. Journal of the London Mathematical Society s1-35(1), 36–42 (1960)
13. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. Special Publication 800-38E pp. 175–182 (2010)
14. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to the NIST SHA-3 Competition, Round 2 (2009)

15. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
16. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
17. Jetchev, D., Özen, O., Stam, M.: Understanding Adaptivity: Random Systems Revisited. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 313–330. Springer, Heidelberg (2012)
18. Knudsen, L.R. (ed.): EUROCRYPT 2002. LNCS, vol. 2332. Springer, Heidelberg (2002)
19. Krawczyk, H.: LFSR-based Hashing and Authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
20. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
21. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
22. Ma, X., Xu, F., Xu, H., Tan, X., Qi, B., Lo, H.K.: Postprocessing for quantum random number generators: entropy evaluation and randomness extraction (2012), http://arxiv.org/abs/1207.1473
23. Maes, R., Tuyls, P., Verbauwhede, I.: Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 332–347. Springer, Heidelberg (2009)
24. Maurer, U.M.: Indistinguishability of Random Systems. In: Knudsen (ed.) [18], pp. 110–132
25. Minematsu, K.: Improved Security Analysis of XEX and LRW Modes. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 96–113. Springer, Heidelberg (2007)
26. Nguyen, L.H., Roscoe, A.W.: Simple construction of epsilon-biased distribution. Cryptology ePrint Archive, Report 2012/429 (2012), http://eprint.iacr.org/
27. Rogaway, P., Zhang, H.: Online Ciphers from Tweakable Blockciphers. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 237–249. Springer, Heidelberg (2011)
28. Ristenpart, T., Rogaway, P.: How to Enrich the Message Space of a Cipher. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 101–118. Springer, Heidelberg (2007)
29. Rivest, R.L.: The RC5 Encryption Algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 86–96. Springer, Heidelberg (1995)
30. Rivest, R.L.: The invertibility of the xor of rotations of a binary word. Int. J. Comput. Math. 88(2), 281–284 (2011)
31. Rivest, R.L., Robshaw, M.J.B., Yin, Y.L.: Rc6 as the aes. In: AES Candidate Conference, pp. 337–342 (2000)
32. Robshaw, M., Billet, O. (eds.): New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008)
33. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
34. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) ACM Conference on Computer and Communications Security, pp. 196–205. ACM (2001)

35. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
36. Sarkar, P.: A new multi-linear universal hash family. Designs, Codes and Cryptography pp. 1–17, http://dx.doi.org/10.1007/s10623-012-9672-8, 10.1007/s10623-012-9672-8
37. Stankovski, P., Hell, M., Johansson, T.: Analysis of Xorrotation with Application to an HC-128 Variant. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 419–425. Springer, Heidelberg (2012)
38. Thomsen, S.S.: Cryptographic hash functions. PhD thesis, Technical University of Denmark (2008)
39. Wah, P., Wang, M.Z.: Realization and application of the Massey-Omura lock. Digital Commnucation, International Zurich Seminar, 175–182 (1984)
40. Wu, H.: The Stream Cipher HC-128. In: Robshaw, Billet (eds.) [32], pp. 39–47

A Proof of Theorem 1

We use a result of Minematsu [25][4]. We first show the proof for XEX-R1. Let us write $h(V, t_1) = \mathrm{CLH}'_{n,\kappa}(V, t_1)$ for $V \in \{0,1\}^n$ and $t_1 \in \mathcal{T}_1$. Then the tweakable encryption of Theorem 1 can be written as $\widetilde{E}_K(M, T) = E_K(M \oplus h(E_K(T_2), T_1)) \oplus h(E_K(T_2), T_1)$, which fits into the model discussed by [25]. First, we have to bound

$$\gamma \stackrel{\mathrm{def}}{=} \max_{t_1 \in \mathcal{T}_1, c \in \{0,1\}^n} \Pr[h(V, t_1) = c],$$

$$\epsilon \stackrel{\mathrm{def}}{=} \max_{t_1 \neq t'_1 \in \mathcal{T}_1, c \in \{0,1\}^n} \Pr[h(V, t_1) \oplus h(V, t'_1) = c], \text{and}$$

$$\rho \stackrel{\mathrm{def}}{=} \max_{t_1 \in \mathcal{T}_1, c \in \{0,1\}^n} \Pr[h(V, t_1) \oplus V = c],$$

where probabilities are defined over $V \stackrel{\$}{\leftarrow} \{0,1\}^n$ and $\mathcal{T}_1 = \{0,1\}^{\kappa-1} \setminus \{0^{\kappa-1}, 0^{\kappa-2}1\}$. For γ, the probability is at most the maximum point probability of $\mathrm{CLH}'_{n,\kappa}(V, t_1)$ for $t_1 \neq 0^{\kappa-1}$. As V is uniform, we have $\gamma = 2/2^\kappa$ from Lemma 1. Then, ϵ is equivalent to $\Pr[\mathsf{msb}_\kappa(h(V, t_1) \oplus h(V, t'_1)) = \mathsf{msb}_\kappa(c)]$, which is at most $2/2^\kappa$ from Lemma 1. For ρ, let $V = V_l \| V_r$ and $c = c_l \| c_r$ with $|V_l| = |c_l| = \kappa$ and $|V_r| = |c_r| = n - \kappa$. Then we have

$$\Pr[h(V, t_1) \oplus V = c] = \Pr[\mathrm{CLH}_\kappa(V_l, t_1) \oplus \mathrm{CLH}_\kappa(V_l, 0^{\kappa-2}\|1) = c_l, V_r = c_r].$$

Since $\mathcal{T}_1$ does not contain $0^{\kappa-2}\|1$ and that V_l and V_r are independent and random, the probability of the right hand side is at most $2/2^\kappa \cdot 1/2^{n-\kappa} = 2/2^n$ from Lemma 2. Combining Lemma 2 and Theorem 4 of [25] with the result $(\gamma, \epsilon, \rho) = (2/2^\kappa, 2/2^\kappa, 2/2^n)$, we obtain the bound of TSPRP-advantage being $(2\epsilon + \gamma + \rho + 2/2^n)q^2 = (6/2^\kappa + 2.5/2^n)q^2$.

For proving the bound for XEX-R2, we similarly have $\gamma, \epsilon \leq 2/2^{2n-\lambda}$ from Lemma 2. For ρ, since $V = \mathrm{CLH}''_{n,\lambda}(V, 0^{\lambda-2}\|1)$ and $t_1 = 0^{\lambda-2}\|1$ is excluded, we obtain $\rho \leq 2/2^{2n-\lambda}$.

[4] This result is obtained by using Maurer's random system method [24], and the result does not suffer from a flaw of a theorem of [24] recently found by Jetchev et al. [17].

B String-to-Vector (S2V) PRF

Fig. 2 shows the String-to-Vector (S2V) PRF [35]. Here $\mathbf{2}S$ denotes the GF doubling over GF(2^n).

Algorithm S2V$[F_K](X[1], \ldots, X[\ell])$

1. **if** $\ell = 0$ **then return** $F_K(0^{n-1}1)$
2. $S \leftarrow F_K(0^n)$
3. **for** $i \leftarrow 1$ **to** $\ell - 1$ **do** $S \leftarrow \mathbf{2}S \oplus F_K(X[i])$
4. **if** $|X[\ell]| \geq n$ **then** $V \leftarrow S \oplus_{\mathrm{end}} X[\ell]$ **else** $V \leftarrow \mathbf{2}S \oplus X[\ell]\|10^*$
5. **return** $F_K(V)$

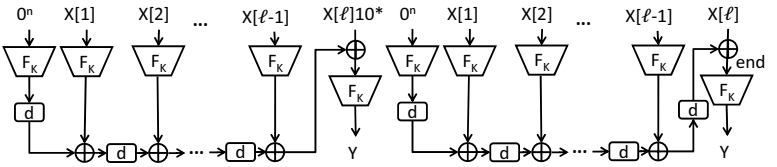

Fig. 2. S2V vector-input PRF using $F_K : \{0,1\}^* \rightarrow \{0,1\}^n$. The box with "d" in the lower figure denotes the GF doubling.