FAST: Disk Encryption and Beyond

Debrup Chakraborty[1], Sebati Ghosh[1], Cuauhtemoc Mancillas López[2], and Palash Sarkar[1]

[1] Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: {debrup, sebati_r, palash}@isical.ac.in
[2] Computer Science Department, CINVESTAV-IPN
Mexico, D.F., 07360, Mexico
email: cuauhtemoc.mancillas83@gmail.com

Abstract. This work introduces FAST which is a new family of tweakable enciphering schemes. Several instantiations of FAST are described. These are targeted towards two goals, the specific task of disk encryption and a more general scheme suitable for a wide variety of practical applications. A major contribution of this work is to present detailed and careful software implementations of all of these instantiations. For disk encryption, the results from the implementations show that FAST compares very favourably to the IEEE disk encryption standards XCB and EME2 as well as the more recent proposal AEZ. FAST is built using a fixed input length pseudo-random function and an appropriate hash function. It uses a single-block key, is parallelisable and can be instantiated using only the encryption function of a block cipher. The hash function can be instantiated using either the Horner's rule based usual polynomial hashing or hashing based on the more efficient Bernstein-Rabin-Winograd polynomials. Security of FAST has been rigorously analysed using the standard provable security approach and concrete security bounds have been derived. Based on our implementation results, we put forward FAST as a serious candidate for standardisation and deployment.
Keywords: disk encryption, tweakable enciphering schemes, pseudo-random function, Horner, BRW.

1 Introduction

There is a huge amount of data residing on various kinds of storage devices. For example, the Indian national repository of biometric data called Aadhaar runs into several petabytes[3]. In today's world, much of the data at rest are sensitive and require encryption to be protected from unwanted access or tampering. The solution is to use full disk encryption where the storage device holds the encryption of the data under a secret key. Reading from the disk requires decrypting the relevant portion of the disk, while writing to the disk requires encrypting the data and then storing it at an appropriate location on the disk. The tasks of encryption and decryption are performed using a disk encryption algorithm. To be useful in practice a disk encryption algorithm needs to be both secure and efficient. The goal of security is to ensure that unwanted access or tampering is indeed not feasible while the goal of efficiency is to ensure that there is no noticeable slowdown in the process of reading from or writing to the disk.

A logical level view of a hard disk and most other storage devices is as a collection of sectors where each sector can store a fixed number of bytes. For example, present day hard disks have 4096-byte sectors while some of the older disks had 512-byte sectors[4]. Each sector has a unique address. A read or write operation on a disk works at the granularity of sectors. A read operation will specify a bunch of sector addresses and the complete contents of those sectors will be returned.

[3] `https://www.cse.iitb.ac.in/~comad/2010/pdf/Industry Sessions/UID_Pramod_Varma.pdf`
[4] `https://en.wikipedia.org/wiki/Disk_sector`

Similarly, a write operation will specify the data and a bunch of sector addresses and the contents of the corresponding sectors will be overwritten with the new data.

A disk encryption algorithm proceeds sector by sector. The content of a sector is encrypted using the secret key and stored in-place, i.e., the content of the sector is overwritten using the encrypted content. The original unencrypted content is not stored anywhere. Using only encryption is not sufficient for security as can be seen from the following simple attack. Suppose that the contents of two successive sectors s_1 and s_2 are C_1 and C_2 corresponding to plaintexts P_1 and P_2 respectively. An adversary may simply swap C_1 and C_2. Subsequent decryption will show s_1 containing P_2 and s_2 containing P_1 whereas decryption before the swap would have shown s_1 containing P_1 and s_2 containing P_2. If it turns out that s_1 containing P_2 and s_2 containing P_1 is meaningful data, then by a simple swap operation, the adversary has been able to alter the content of the disk to a meaningful data which was not originally stored on the disk.

To prevent the above possibility, the encryption of the content of a sector needs to be somehow tied to the sector address. Decryption of any adversarially modified content of a sector should result in a random looking string which is unlikely to be meaningful data.

Viewed in this manner, a disk encryption mechanism is an example of a length preserving encryption where the length of the ciphertext is equal to the length of the plaintext. Further, there is another quantity (which is the sector address in case of disk encryption) which determines the ciphertext but, is itself not encrypted. In the literature this quantity has been called a tweak. The functionality of a tweak-based length preserving encryption has been called a tweakable enciphering scheme (TES) [22].

While disk encryption is a very important application of a TES, the full functionality of a TES is much more broader than just disk encryption. For the specific case of disk encryption, messages are contents of a sector and so are fixed length strings. A TES can have a more general message space consisting of binary strings of different lengths. Similarly, in the case of disk encryption, the tweak is a sector address and can be encoded using a short fixed length string. More generally, the tweak space in a TES can also consist of strings of different lengths or even consist of vectors of strings.

Our Contributions

This paper describes a new family of tweakable enciphering schemes called FAST which is built using a pseudo-random function (PRF) and a hash function with provably low collision and differential probabilities. The domain and the range of the pseudo-random function are both equal to the set of all n-bit binary strings for an appropriately chosen n. The hash function is built using arithmetic over the finite field $GF(2^n)$. Some of the salient aspects of FAST are described below.

Wide range of applications: FAST can be used in the following settings.

Fixed length setting: This setting is targeted towards disk encryption application. It supports an n-bit tweak and messages whose lengths are a fixed multiple of the block size n.

General setting: This setting is very general. Messages are allowed to have different lengths and tweaks are allowed to be vectors of binary strings where the numbers of components in the vectors can vary. The richness of the tweak space provides considerable flexibility in applications where there is a message and an associated set of attributes. The message is to be encrypted while the attributes are to be in the clear but the ciphertext needs to be bound to the attributes. We mention two possible applications for such a functionality.

1. The message is a data packet that is to be stored at a destination node while the vector of attributes encode the path taken by the data packet to reach the destination node with the components of the vector identifying the intermediate nodes.
2. The message consists of biometric information while the attributes are date-time, gender and other related information. A possible application would be to the Aadhaar database mentioned earlier.

We note that the idea of having associated data to be a vector of strings was earlier proposed [33] in the context of deterministic authenticated encryption. AEZ [24] provides a conceptual level description of how to handle a vector of strings as tweak using an almost XOR universal hash function to process the vector. A generic security bound is provided in terms of the collision probability of the hash function. No concrete proposal for the hash function is provided. Consequently, the efficiency of processing the tweak cannot be determined and neither it is possible to obtain a concrete security bound. In contrast, following our objective of practical implementation we put forward several concrete designs for hashing a vector of strings with associated concrete secuity bounds and detailed software implementations.

Software implementations: A major objective of the paper consists of rigorous software implementations of FAST and the most important TES schemes in the literature. The goal of such implementations is to perform a comparative study of the performances of FAST with those of the previous schemes in software. To this end, we have carried out detailed software implementations of the IEEE standards XCB and EME2 as well as AEZ (instantiated with the encryption function of the AES block cipher) along with similar implementations of variants of FAST. For a fair comparison, we have incorporated similar efficiency measures in all the implementations.

This implementation is targeted towards modern Intel processors and is in Intel intrinsics using the specialised AES-NI instructions and the `pclmulqdq` instruction. The code for the software implementation of FAST is publicly available (the link to it has been dropped from this version due to anonymity constraints and will be included in the final version). The implementation of FAST covers both the fixed length and the general settings. We provide timing results for the Skylake and the Kabylake processors of Intel.

Results arising from the implementations show that the new proposal compares favourably to the most important previous constructions in software. For the fixed length setting, the best speed achieved by FAST on the Intel Skylake platform is 1.24 cycles per byte (cpb). In comparison, XCB, EME2 and AEZ achieve speeds of 1.92 cpb, 2.07 cpb and 1.74 cpb respectively. The corresponding figures on Kabylake for FAST, XCB, EME2 and AEZ are 1.19, 1.85, 1.99 and 1.70 cpb respectively. Further timing details are provided later.

Dispensing with invertibility: There are several concrete TES proposals in the literature. Most of these proposals including the ones that have been standardised are modes of operations of a block cipher and use both the encryption and the decryption functions of the underlying block cipher. FAST, on the other hand, uses a PRF and does not require the invertibility property of a block cipher. The PRF itself may be instantiated using the encryption function of a block cipher such as AES. This provides two distinct advantages.

1. From a practical point of view, the advantage is that the decryption function of the block cipher does not require to be implemented. This is an advantage in hardware implementation since it results in a smaller hardware. A software implementation also benefits by requiring a smaller size code.

2. From a theoretical point of view, a block cipher is modelled as a strong pseudo-random permutation (SPRP). A PRF assumption on the encryption function of a block cipher is a weaker assumption than an SPRP assumption on the block cipher. So security of FAST can be based on a weaker assumption on the underlying block cipher.

We note that a previous work [36] had pointed out the possibility of using only the encryption function of a block cipher to build a TES. The work was more at a conceptual level using generic components and some unnecessary operations. It did not provide any specific instantiation or implementation. Subsequent to [36], the constructions AEZ [24] and FMix [6] proposed single key TESs using only the encryption function of the block cipher. FMix is a sequential scheme while AEZ is parallelisable. Later we discuss in more details several issues regarding the comparison of FAST to previous schemes.

Parallelisable: At a top level, the construction applies a Feistel layer of encryption on the first two message blocks and sandwiches a counter type mode of operation in-between two layers of hashing for the rest of the message. The counter mode is fully parallelisable. This leads to efficient implementations in both software and hardware.

Design of hash functions: We provide instantiations using two kinds of hash functions both of which are based on arithmetic over the finite field $GF(2^n)$. The first kind of hash function is based on the usual polynomial based hashing using Horner's rule. The second kind is based on a class of polynomials [5] which was later called BRW polynomials [35]. For tackling variable length inputs, a combination of BRW and Horner based hashing called Hash2L [7] turns out to be advantageous. For the fixed length setting, we show instantiations using Horner and BRW while for the general setting, we use the vector version vecHorner of Horner and the vector version vecHash2L of Hash2L.

Provable security treatment: The security of the proposed scheme is analysed following the standard provable security methodology. The theoretical notion of security of a TES is shown to hold under the assumption that the encryption function of the underlying block cipher is a PRF. The proof requires the hash functions to satisfy certain properties. We show that the hash functions obtained from Horner, vecHorner, BRW and vecHash2L satisfy the required properties. Concrete security bounds are derived for the different instantiations. These bounds show that the security of FAST is adequate for practical purposes and is comparable to those achieved in previous designs.

Previous Works on TES

The first proposal for the construction of a strong pseudorandom permutation using a hash-ECB-hash approach was by Naor and Reingold [30]. This work, though, did not consider tweaks since the paper predates the formal introduction of the notion of tweaks and hence of a TES. The notion of a tweakable block cipher and its security was formalised by Liskov, Rivest and Wagner [25]. This was followed by a formalisation of the notion of a tweakable enciphering scheme by Halevi and Rogaway [22]. The paper also described a TES called CMC which is based on the CBC mode of operation. A subsequent work [23] by the same authors introduced a TES called EME which is a parallelisable mode of operation of a block cipher. EME was extended to handle arbitrary length messages by Halevi [19] and the resulting scheme was called EME*. The EME family of TESs does not require finite field multiplication. The main cost of encryption is roughly two block cipher calls per block of the message.

Construction of a TES using a counter based mode of operation of a block cipher and a Horner type hash function was first proposed by McGrew and Fluhrer [26]. This scheme was called XCB. A later variant [27] of XCB was proposed to improve efficiency and reduce key size. Various security problems for XCB have been pointed out [8].

There have been a number of works proposing different constructions of TESs. Examples are PEP [12], ABL [28], HCTR [37], HCH [13], TET [20] and HEH [35]. An improved security analysis of HCTR has been done later [11]. A generalisation of EME using a general masking scheme has been proposed [34]. As mentioned earlier, the conceptual possibility of constructing a TES from a PRF (and hence using only the encryption function of a block cipher) has been suggested earlier [36]. The constructions AEZ [24] and FMix [6] are also TESs constructed from a PRF. The possibility of constructing TESs from stream ciphers has been considered [36]. Concrete proposals and detailed FPGA implementations of stream cipher based TESs have been described [10].

Another line of investigation has been the construction of ciphers that can securely encipher their own keys [21, 4]. A generic method is known [4] which converts a conventional TES to one which can be proved to be secure even under the possibility of encrypting its own key. This generic method has been applied to EME2 [4]. We note that the method can equally well be applied to the construction FAST proposed in the present work.

Standards and Patents

IEEE [3] has standardised two tweakable enciphering schemes, namely EME2 and XCB. Essentially, EME2 is the variant EME* [19] while the standardised version of XCB is a variant [27] of the original scheme [26]. Both EME2 and XCB are patented algorithms. Till date there is no unpatented algorithm which has been standardised. Apart from offering superior performance guarantees with respect to previous schemes XCB, EME2 and AEZ, it is our hope that FAST will also fill the gap of providing an attractive solution which is unencumbered by intellectual property claims.

An earlier IEEE standard is XTS [2] which has also been standardised [16] by NIST of USA. This is based on the XEX construction of Rogaway [32]. XTS is not a TES and the security provided by XTS is not adequate for disk encryption application. Rogaway [1] himself mentioned that XTS only provides light security and should be preferred only when there is an overriding concern for speed.

2 Preliminaries

Throughout the paper, we fix a positive integer n and a positive integer $\eta \geq 3$.

Notation: Let X and Y be binary strings.

- The length of X will be denoted as $\mathsf{len}(X)$.
- The concatenation of X and Y will be denoted as $X||Y$.
- For an integer i with $0 \leq i < 2^n$, $\mathsf{bin}_n(i)$ denotes the n-bit binary representation of i.

We define the following terminology.

$\mathsf{first}_i(X)$: For a binary string X, $0 < i \leq \mathsf{len}(X)$, $\mathsf{first}_i(X)$ will denote the first (or, the most significant) i bits of X.

$\mathsf{pad}_n(X)$: For a binary string X and $n > 0$, if X is the empty string, then $\mathsf{pad}_n(X)$ will denote the string 0^n; while if X is non-empty, then $\mathsf{pad}_n(X)$ will denote $X||0^i$, where $i \geq 0$ is the minimum integer such that n divides $\mathsf{len}(X||0^i)$.

$\mathsf{parse}_n(X)$: For a binary string X such that $\mathsf{len}(X) \geq 2n$, $\mathsf{parse}_n(X)$ denotes (X_1, X_2, X_3) where $\mathsf{len}(X_1) = \mathsf{len}(X_2) = n$ and $X = X_1||X_2||X_3$. In other words, $\mathsf{parse}_n(X)$ divides the string X into three parts with the first two parts having length n bits each with the remaining bits of X (if any) forming the third part.

$\mathsf{format}_n(X)$: For a non-empty binary string X and a positive integer n, $\mathsf{format}_n(X)$ denotes $(X_1, X_2, \ldots, X_m)$ where $X = X_1||X_2||\cdots||X_m$, $m = \lceil \mathsf{len}(X)/n \rceil$, $\mathsf{len}(X_i) = n$ for $1 \leq i \leq m-1$ and $1 \leq \mathsf{len}(X_m) \leq n$. In other words, $\mathsf{format}_n(X)$ divides the string X into $m-1$ n-bit blocks $X_1, \ldots, X_{m-1}$ and a possibly partial last block X_m.

Number of n-bit blocks: Let X be a binary string and suppose that $\mathsf{format}_n(\mathsf{pad}_n(X))$ returns $X_1||\cdots||X_m$. We will say that the number of n-bit blocks in X is m. Note that if X is the empty string, then $\mathsf{pad}(X)$ is 0^n and so $\mathsf{format}_n(\mathsf{pad}_n(X))$ is also 0^n whence $m = 1$, i.e., as per our formalism, the empty string has one n-bit block. The number of n-bit blocks in X will be denoted by $\mathfrak{l}(X)$.

For a vector of binary strings $Y = (Y_1, \ldots, Y_k)$, by the number of n-bit blocks in Y we will mean the sum of the numbers of n-bit blocks in the strings $Y_1, \ldots, Y_k$. The number of n-bit blocks in Y will be denoted by $\mathfrak{t}(Y)$.

$\mathsf{superBlks}_{n,\eta}(Z)$: For a binary string Z, $\mathsf{superBlks}_{n,\eta}(Z)$ denotes the vector of strings $(Z_1, \ldots, Z_\ell)$ obtained as $(Z_1, \ldots, Z_\ell) \leftarrow \mathsf{format}_{n\eta}(\mathsf{pad}_n(Z))$. For $1 \leq i \leq \ell - 1$, Z_i is an $n\eta$-bit string while Z_ℓ is a string whose length is at most $n\eta$ and is divisible by n. The strings $Z_1, \ldots, Z_\ell$ are called super-blocks. The first $\ell - 1$ of these super-blocks consist of exactly η n-bit blocks while the last super-block consists of at most η n-bit blocks. *We will say that the number of super-blocks in Z is ℓ.*

Finite field: Let $\mathbb{F} = GF(2^n)$ be the finite field of 2^n elements. Using a fixed irreducible polynomial of degree n over $GF(2)$ to represent $\mathbb{F}$, the elements of $\mathbb{F}$ can be identified with the binary strings of length n. Viewed in this manner, an n-bit binary string will be considered to be an element of $\mathbb{F}$. The addition operation over $\mathbb{F}$ will be denoted by $\oplus$; for $X, Y \in \mathbb{F}$, the product will be denoted as XY. The additive identity of $\mathbb{F}$ will be denoted as $\mathbf{0}$ and will be represented as 0^n; the multiplicative identity of $\mathbb{F}$ will be denoted as $\mathbf{1}$ and will be represented as $0^{n-1}1$.

For $n = 128$, let $\mathbb{F}$ be represented as $GF(2)[\alpha]/\psi(\alpha)$ where $\psi(\alpha) = \alpha^{128} \oplus \alpha^7 \oplus \alpha^2 \oplus \alpha \oplus 1$. The 128-bit string X is considered to be a polynomial $X(\alpha) \in GF(2)[\alpha]$. Let Y be a 128-bit string representing the polynomial $Y(\alpha) = \alpha X(\alpha) \bmod \psi(\alpha)$. The string Y can be obtained from the string X as $Y = (X \ll 1) \oplus (\mathsf{msb}(X) \cdot 135)$, where $\mathsf{msb}(X)$ denotes the most significant bit of X. Over $\mathbb{F}$, this operation corresponds to the 'multiply by α' map and has been called a doubling operation [32].

Pseudo-random function: The construction requires a family of functions where each function in the family maps n-bit strings to n-bit strings. More precisely, let $\{\mathbf{F}_K\}_{K \in \mathcal{K}}$ be a family of functions, where for $K \in \mathcal{K}$, $\mathbf{F}_K : \{0,1\}^n \to \{0,1\}^n$. Here $\mathcal{K}$ is the key space of $\mathbf{F}$. The security requirement on $\{\mathbf{F}_K\}_{K \in \mathcal{K}}$ is that of a pseudo-random function family. Informally this means, for a randomly chosen K, on distinct inputs, the outputs of $\mathbf{F}_K(\cdot)$ appear independent and uniformly distributed to a computationally bounded adversary. We provide the formal definition later. It is possible to instantiate $\mathbf{F}$ using the encryption (or the decryption) function of a block cipher. In particular, one may use the encryption function of AES to instantiate $\mathbf{F}$. This, however, is an overkill, since the invertibility property of the block cipher is not required by the construction.

Counter mode: The PRF $\mathbf{F}$ can handle only n-bit strings. Longer strings are handled in the following manner. Let X be a non-empty binary string. For $K \in \mathcal{K}$ and $S \in \{0,1\}^n$, we define

$\mathsf{Ctr}_{K,S}(X)$ in the following manner.

$$\mathsf{Ctr}_{K,S}(X) = (S_1 \oplus X_1, \ldots, S_{m-1} \oplus X_{m-1}, \mathsf{first}_r(S_m) \oplus X_m) \tag{1}$$

where $(X_1, \ldots, X_m) \leftarrow \mathsf{format}_n(X)$, $\mathsf{len}(X_m) = r$ and $S_i = \mathbf{F}_K(S \oplus \mathsf{bin}_n(i))$. This variant of the counter mode was originally used in HCTR [37]. Note that the PRF $\mathbf{F}$ is used to define the counter mode, but, the counter mode itself as defined here is not a PRF.

3 Construction

Formally, $\mathsf{FAST} = (\mathsf{FAST.Encrypt}, \mathsf{FAST.Decrypt})$ where

$$\mathsf{FAST.Encrypt}, \mathsf{FAST.Decrypt} : \mathcal{K} \times \mathcal{T} \times \mathcal{P} \to \mathcal{P}, \tag{2}$$

- $\mathcal{K}$ is a finite non-empty set called the key space,
- $\mathcal{T}$ is a finite non-empty set called the tweak space and
- $\mathcal{P}$ denotes both the message and the ciphertext spaces such that for any string $P \in \mathcal{P}$, $\mathsf{len}(P) > 2n$. So, for any $P \in \mathcal{P}$, the number of n-bit blocks in $\mathsf{pad}_n(P)$ is at least three. This requirement will be called the *length condition* on $\mathcal{P}$.

We emphasise that $\mathcal{P}$ does not necessarily contain all strings of lengths greater than $2n$. We provide the precise definitions of $\mathcal{P}$ for specific instantiations later.

For $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $P \in \mathcal{P}$, we write $\mathsf{FAST.Encrypt}_K(T, P)$ to denote $\mathsf{FAST.Encrypt}(K, T, P)$; for $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $C \in \mathcal{P}$, we write $\mathsf{FAST.Decrypt}_K(T, C)$ to denote $\mathsf{FAST.Decrypt}(K, T, C)$. The definitions of $\mathsf{FAST.Encrypt}_K(T, P)$ and $\mathsf{FAST.Decrypt}_K(T, C)$ are given in Table 1. These definitions use the functions $\mathbf{H}_\tau$, $\mathbf{G}_\tau$, $\mathbf{H}'_\tau$ and $\mathbf{G}'_\tau$ which themselves are defined using two hash functions h and h' in the following manner.

$$\begin{aligned}
\mathbf{H}_\tau(P_1, P_2, P_3, T) &= (P_1 \oplus h_\tau(T, P_3), P_2 \oplus \tau(P_1 \oplus h_\tau(T, P_3)));\\
\mathbf{G}_\tau(X_1, X_2, X_3, T) &= (X_1 \oplus h_\tau(T, X_3), X_2 \oplus \tau X_1);\\
\mathbf{H}'_\tau(C_1, C_2, C_3, T) &= (C_1 \oplus \tau(C_2 \oplus h'_\tau(T, C_3)), C_2 \oplus h'_\tau(T, C_3));\\
\mathbf{G}'_\tau(Y_1, Y_2, Y_3, T) &= (Y_1 \oplus \tau Y_2, Y_2 \oplus h'_\tau(T, Y_3)).
\end{aligned} \tag{3}$$

From the definitions of $\mathbf{H}_\tau, \mathbf{G}_\tau$ and $\mathbf{H}'_\tau, \mathbf{G}'_\tau$ it is easy to verify the following properties.

$$\begin{aligned}
\mathbf{H}_\tau(P_1, P_2, P_3, T) = (A_1, F_1) &\text{ implies } \mathbf{G}_\tau(A_1, F_1, P_3, T) = (P_1, P_2);\\
\mathbf{H}'_\tau(C_1, C_2, C_3, T) = (F_2, B_2) &\text{ implies } \mathbf{G}'_\tau(F_2, B_2, C_3, T) = (C_1, C_2).
\end{aligned} \tag{4}$$

Note that for fixed τ, P_3 and T, $\mathbf{H}_\tau(\cdot, \cdot, P_3, T)$ and $\mathbf{G}_\tau(\cdot, \cdot, P_3, T)$ are inverses of one another and similarly, $\mathbf{H}'_\tau(\cdot, \cdot, P_3, T)$ and $\mathbf{G}'_\tau(\cdot, \cdot, P_3, T)$ are inverses of one another.

The hash functions h and h' required in the definitions of $\mathbf{H}, \mathbf{G}$ and $\mathbf{H}', \mathbf{G}'$ respectively and the other components used in Table 1 are given below.

1. The two hash functions h and h' are defined in the following manner.

$$h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \to \mathbb{F}, \tag{5}$$

where

$$\mathcal{M} = \{x : w||x \in \mathcal{P} \text{ for some } w \in \{0,1\}^{2n}\}; \tag{6}$$

$\mathbb{F}$ is the key space and also the digest space, $\mathcal{T}$ is the tweak space and $\mathcal{M}$ is the message space for the hash functions. For $\tau \in \mathbb{F}$, $T \in \mathcal{T}$ and $M \in \mathcal{M}$, we will write $h_\tau(T, M)$ (resp. $h'_\tau(T, M)$) to denote $h(\tau, T, M)$ (resp. $h'(\tau, T, M)$). Note that in FAST, both h and h' share the same key τ. Later we discuss the properties required of the pair of hash functions (h, h') and how to construct such pairs using standard hash functions.

2. A PRF $\{\mathbf{F}_K\}_{K \in \mathcal{K}}$ where for $K \in \mathcal{K}$, $\mathbf{F}_K : \{0,1\}^n \to \{0,1\}^n$. The PRF is used in the Ctr mode as given in (1). Since strings in $\mathcal{P}$ are of length greater than $2n$, the Ctr mode is applied to non-empty strings.
3. A fixed n-bit string fStr.
4. Sub-routines $\mathsf{Feistel}$ and $\mathsf{Feistel}^{-1}$ which are shown in Table 2.

From the descriptions of $\mathsf{FAST.Encrypt}_K(T, P)$ and $\mathsf{FAST.Decrypt}_K(T, C)$ in Table 1, the following two facts are easy to verify. For $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $P \in \mathcal{P}$,

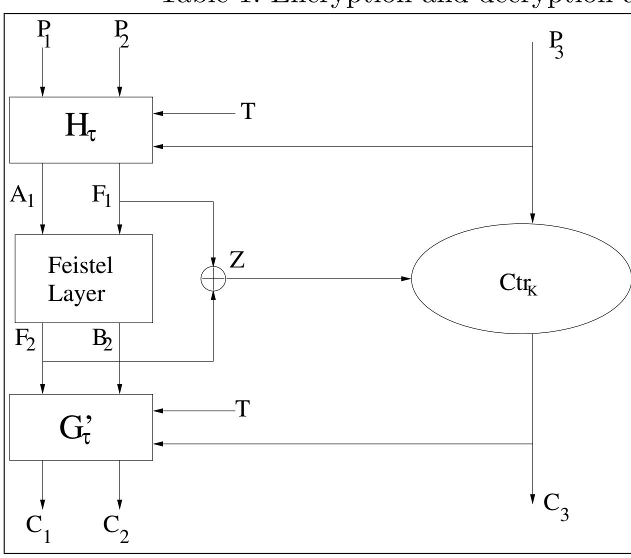
$$\mathsf{FAST.Decrypt}_K(T, \mathsf{FAST.Encrypt}_K(T, P)) = P; \tag{7}$$

$$\mathsf{len}\,(\mathsf{FAST.Encrypt}_K(T, P)) = \mathsf{len}(P). \tag{8}$$

From (7), it follows that the decryption function of FAST is the inverse of the encryption function, while (8) shows that the length of the ciphertext produced by the encryption function is equal to the length of the plaintext.

Note: From the schematic diagram of the encryption algorithm in Table 1, $\mathsf{FAST.Encrypt}_K(T, P)$ can be seen as consisting of three distinct layers – a hashing layer using the hash function H_τ; an encryption layer consisting of the two-round Feistel and the counter mode; and the hashing layer using the hash function G'_τ. The quantity τ is the key for the hashing layers and is not used in the encryption layer while K is used only in the encryption layer and not in the hashing layers.

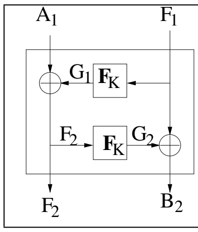
Table 1: Encryption and decryption algorithms for FAST.



Algorithm $\mathsf{FAST.Encrypt}_K(T, P)$
1. $\tau \leftarrow \mathbf{F}_K(\mathsf{fStr})$;
2. $(P_1, P_2, P_3) \leftarrow \mathsf{parse}_n(P)$;
3. $(A_1, F_1) \leftarrow \mathbf{H}_\tau(P_1, P_2, P_3, T)$;
4. $(F_2, B_2) \leftarrow \mathsf{Feistel}_K(A_1, F_1)$;
5. $Z \leftarrow F_1 \oplus F_2$;
6. $C_3 \leftarrow \mathsf{Ctr}_{K,Z}(P_3)$;
7. $(C_1, C_2) \leftarrow \mathbf{G}'_\tau(F_2, B_2, C_3, T)$;
8. return $(C_1 || C_2 || C_3)$.

Algorithm $\mathsf{FAST.Decrypt}_K(T, C)$
1. $\tau \leftarrow \mathbf{F}_K(\mathsf{fStr})$;
2. $(C_1, C_2, C_3) \leftarrow \mathsf{parse}_n(C)$;
3. $(F_2, B_2) \leftarrow \mathbf{H}'_\tau(C_1, C_2, C_3, T)$;
4. $(A_1, F_1) \leftarrow \mathsf{Feistel}_K^{-1}(F_2, B_2)$;
5. $Z \leftarrow F_1 \oplus F_2$;
6. $P_3 \leftarrow \mathsf{Ctr}_{K,Z}(C_3)$;
7. $(P_1, P_2) \leftarrow \mathbf{G}_\tau(A_1, F_1, P_3, T)$;
8. return $(P_1 || P_2 || P_3)$.

Table 2: A two-round Feistel construction required in Table 1.

	$\mathsf{Feistel}_K(A_1, F_1)$ 1. $F_2 \leftarrow A_1 \oplus \mathbf{F}_K(F_1)$; 2. $B_2 \leftarrow F_1 \oplus \mathbf{F}_K(F_2)$; return (F_2, B_2).	$\mathsf{Feistel}_K^{-1}(F_2, B_2)$ 1. $F_1 \leftarrow B_2 \oplus \mathbf{F}_K(F_2)$; 2. $A_1 \leftarrow F_2 \oplus \mathbf{F}_K(F_1)$; return (A_1, F_1).

4 Instantiations of FAST

Certain properties are required from the pair of hash functions (h, h'). These properties will be used in the security argument to show that in an information theoretic setting, the adversary's probability of breaking the security of FAST is low. The specific properties that will be required are formalised below.

Definition 1. *Let (h, h') be a pair of hash functions where $h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \to \mathbb{F}$ satisfy the following properties. For any $(T, M), (T', M') \in \mathcal{T} \times \mathcal{M}$, with $(T, M) \neq (T', M')$; any $\alpha, \beta \in \mathbb{F}$; and τ chosen uniformly at random from $\mathbb{F}$:*

$$\Pr[\tau(h_\tau(T, M) \oplus \alpha) = \beta] \leq \epsilon_1(\mathfrak{t}, \mathfrak{l}); \tag{9}$$

$$\Pr[\tau(h'_\tau(T, M) \oplus \alpha) = \beta] \leq \epsilon_1(\mathfrak{t}, \mathfrak{l}); \tag{10}$$

$$\Pr[\tau(h_\tau(T, M) \oplus h_\tau(T', M') \oplus \alpha) = \beta] \leq \epsilon_2(\mathfrak{t}, \mathfrak{l}, \mathfrak{t}', \mathfrak{l}'); \tag{11}$$

$$\Pr[\tau(h'_\tau(T, M) \oplus h'_\tau(T', M') \oplus \alpha) = \beta] \leq \epsilon_2(\mathfrak{t}, \mathfrak{l}, \mathfrak{t}', \mathfrak{l}'). \tag{12}$$

For any $(T, M), (T', M') \in \mathcal{T} \times \mathcal{M}$; any $\alpha, \beta \in \mathbb{F}$; and τ chosen uniformly at random from $\mathbb{F}$:

$$\Pr[\tau(h_\tau(T, M) \oplus h'_\tau(T', M') \oplus \alpha) = \beta] \leq \epsilon_2(\mathfrak{t}, \mathfrak{l}, \mathfrak{t}', \mathfrak{l}'). \tag{13}$$

Here $\mathfrak{t} \equiv \mathfrak{t}(T)$, $\mathfrak{t}' \equiv \mathfrak{t}(T')$, $\mathfrak{l} \equiv \mathfrak{l}(M)$, $\mathfrak{l}' \equiv \mathfrak{l}(M')$; and ϵ_1 and ϵ_2 are functions of $\mathfrak{t}, \mathfrak{l}, \mathfrak{t}'$ and $\mathfrak{l}'$. Then (h, h') is said to be an (ϵ_1, ϵ_2)-eligible pair of hash functions.

Before considering the specific instantiations of FAST, we briefly discuss the hash functions used for the purpose.

4.1 Hash Functions

Let $\mathcal{D}$ and $\mathcal{G}$ be finite non-empty sets. Let $\{H_\tau\}_{\tau \in \mathsf{T}}$ be an indexed family of functions such that for each τ, $H_\tau : \mathcal{D} \to \mathcal{G}$.

Collision and differential probabilities:

- For distinct $x, x' \in \mathcal{D}$, the *collision probability* of $\{H_\tau\}_{\tau \in \mathsf{T}}$ for the pair (x, x') is defined to be $\Pr_\tau[H_\tau(x) = H_\tau(x')]$.
- Suppose $\mathcal{G}$ is an additively written group. For distinct $x, x' \in \mathcal{D}$ and any $y \in \mathcal{G}$, the *differential probability* of $\{H_\tau\}_{\tau \in \mathsf{T}}$ for the triplet (x, x', y) is defined to be $\Pr_\tau[H_\tau(x) - H_\tau(x') = y]$.

The above probabilities are taken over uniform random choices of τ from T.

Almost universal function family: The family $\{H_\tau\}$ is said to be ϵ-almost universal (ϵ-AU) if for all distinct x, x' in $\mathcal{D}$, the collision probability for the pair (x, x') is at most ϵ.

Almost XOR universal function family: The family $\{H_\tau\}$ is said to be ϵ-almost XOR universal (ϵ-AXU) if for all distinct x, x' in $\mathcal{D}$ and any $y \in \mathcal{G}$, the differential probability for the triplet (x, x', y) is at most ϵ.

We define two standard hash functions.

Polynomials: For $m \geq 0$, let $\mathsf{Horner} : \mathbb{F} \times \mathbb{F}^m \to \mathbb{F}$ be defined as follows.

$$\mathsf{Horner}(\tau, X_1, \ldots, X_m) = \begin{cases} \mathbf{0}, & \text{if } m = 0; \\ X_1\tau^{m-1} \oplus X_2\tau^{m-2} \oplus \cdots \oplus X_{m-1}\tau \oplus X_m, & \text{if } m > 0. \end{cases}$$

We write $\mathsf{Horner}_\tau(X_1, \ldots, X_m)$ to denote $\mathsf{Horner}(\tau, X_1, \ldots, X_m)$. $\mathsf{Horner}_\tau(X_1, \ldots, X_m)$, as a polynomial in τ, has degree at most $m-1$. For $m > 0$, $\mathsf{Horner}_\tau(X_1, \ldots, X_m) = \tau\mathsf{Horner}_\tau(X_1, \ldots, X_{m-1}) \oplus X_m$ and so $\mathsf{Horner}_\tau(X_1, \ldots, X_m)$ can be evaluated using $m-1$ field multiplications.

For a fixed value of m, $\{\mathsf{Horner}_\tau\}$ is $((m-1)/2^n)$-AU and the family $\{\tau \cdot \mathsf{Horner}_\tau\}$ is $(m/2^n)$-AXU.

BRW polynomials: Bernstein [5] defined a family of polynomials based on a previous work by Rabin and Winograd [31]. (These have been called the BRW polynomials [35]). For $m \geq 0$, let $\mathsf{BRW} : \mathbb{F} \times \mathbb{F}^m \to \mathbb{F}$ be defined as follows. We write $\mathsf{BRW}_\tau(\cdots)$ to denote $\mathsf{BRW}(\tau, \cdots)$.

- $\mathsf{BRW}_\tau() = \mathbf{0}$;
- $\mathsf{BRW}_\tau(X_1) = X_1$;
- $\mathsf{BRW}_\tau(X_1, X_2) = X_1\tau \oplus X_2$;
- $\mathsf{BRW}_\tau(X_1, X_2, X_3) = (\tau \oplus X_1)(\tau^2 \oplus X_2) \oplus X_3$;
- $\mathsf{BRW}_\tau(X_1, X_2, \cdots, X_m)$
  $= \mathsf{BRW}_\tau(X_1, \cdots, X_{t-1})(\tau^t \oplus X_t) \oplus \mathsf{BRW}_\tau(X_{t+1}, \cdots, X_m)$;
  if $t \in \{4, 8, 16, 32, \cdots\}$ and $t \leq m < 2t$.

From the definition it follows that for $m \geq 3$, $\mathsf{BRW}_\tau(X_1, X_2, \cdots, X_m)$ is a monic polynomial and for $m = 0, 1, 2$, $\mathsf{BRW}_\tau(X_1, \ldots, X_m) = \mathsf{Horner}_\tau(X_1, \ldots, X_m)$. We further note the following points about BRW polynomials [5].

1. For $m \geq 3$, $\mathsf{BRW}_\tau(X_1, \ldots, X_m)$ can be computed using $\lfloor m/2 \rfloor$ field multiplications and $\lfloor \lg m \rfloor$ additional field squarings to compute $\tau^2, \tau^4, \ldots$.
2. *Let* $\mathfrak{d}(m)$ *denote the degree of* $\mathsf{BRW}_\tau(X_1, \ldots, X_m)$. For $m \geq 3$, $\mathfrak{d}(m) = 2^{\lfloor \lg m \rfloor + 1} - 1$ and so $\mathfrak{d}(m) \leq 2m - 1$; equality is achieved if and only if $m = 2^a$; and $\mathfrak{d}(m) = m$ if and only if $m = 2^a - 1$ for some integer $a \geq 2$.
3. The map from $\mathbb{F}^m$ to $\mathbb{F}[\tau]$ given by $(X_1, \ldots, X_m) \longmapsto \mathsf{BRW}_\tau(X_1, \ldots, X_m)$ is injective.

For a fixed value of m, $\{\mathsf{BRW}_\tau\}$ is $((2m-1)/2^n)$-AU and the family $\{\tau \cdot \mathsf{BRW}_\tau\}$ is $(2m/2^n)$-AXU.

Hash Function $\mathsf{vecHorner}$ Let

$$\mathcal{VD} = \bigcup_{k=0}^{255} \{(M_1, \ldots, M_k) : M_i \in \{0,1\}^*,\ 0 \leq \mathsf{len}(M_i) \leq 2^{n-16} - 1\}. \tag{14}$$

The upper bound of 255 on k ensures that the value of k fits in a byte and the upper bound of $2^{n-16} - 1$ on the lengths of strings ensures that the lengths of such strings fit into an $(n - 16)$-bit binary

Table 3: Computations of vecHorner and vecHash2L. The string 1^n denotes the element of $\mathbb{F}$ whose binary representation consists of the all-one string.

$\mathsf{vecHorner}_\tau(M_1, \ldots, M_k)$	$\mathsf{vecHash2L}_\tau(M_1, \ldots, M_k)$
if $k = 0$ return $1^n\tau$;	if $k = 0$ return $1^n\tau$;
$\mathsf{digest} \leftarrow \mathbf{0}$;	$\mathsf{digest} \leftarrow \mathbf{0}$;
for $i \leftarrow 1, \ldots, k-1$ do	for $i \leftarrow 1, \ldots, k-1$ do
$\quad (M_{i,1}, \ldots, M_{i,m_i}) \leftarrow \mathsf{format}_n(\mathsf{pad}_n(M_i))$;	$\quad (M_{i,1}, \ldots, M_{i,\ell_i}) \leftarrow \mathsf{superBlks}_{n,\eta}(M_i)$;
$\quad L_i \leftarrow \mathsf{bin}_n(\mathsf{len}(M_i))$;	$\quad L_i \leftarrow \mathsf{bin}_n(\mathsf{len}(M_i))$;
for $j \leftarrow 1, \ldots, m_i$ do	$\quad$ for $j \leftarrow 1, \ldots, \ell_i$ do
$\quad \mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus M_{i,j}$;	$\quad\quad \mathsf{digest} \leftarrow \tau^{\mathfrak{d}(\eta)+1}\mathsf{digest} \oplus \mathsf{BRW}_\tau(M_{i,j})$;
$\quad$ end for;	$\quad$ end for;
$\quad \mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus L_i$;	$\quad \mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus L_i$;
end for;	end for;
$(M_{k,1}, \ldots, M_{k,m_k}) \leftarrow \mathsf{format}_n(\mathsf{pad}_n(M_k))$;	$(M_{k,1}, \ldots, M_{k,\ell_k}) \leftarrow \mathsf{superBlks}_{n,\eta}(M_k)$;
$L_k \leftarrow \mathsf{bin}_8(k) \Vert 0^8 \Vert \mathsf{bin}_{n-16}(\mathsf{len}(M_k))$;	$L_k \leftarrow \mathsf{bin}_8(k) \Vert 0^8 \Vert \mathsf{bin}_{n-16}(\mathsf{len}(M_k))$;
for $j \leftarrow 1, \ldots, m_k$ do	for $j \leftarrow 1, \ldots, \ell_k$ do
$\quad \mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus M_{k,j}$;	$\quad \mathsf{digest} \leftarrow \tau^{\mathfrak{d}(\eta)+1}\mathsf{digest} \oplus \mathsf{BRW}_\tau(M_{k,j})$;
end for;	end for;
$\mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus L_k$;	$\mathsf{digest} \leftarrow \tau\mathsf{digest} \oplus L_k$;
$\mathsf{digest} \leftarrow \tau\mathsf{digest}$;	$\mathsf{digest} \leftarrow \tau\mathsf{digest}$;
return digest.	return digest.

string. The definition of $\mathsf{vecHorner} : \mathbb{F} \times \mathcal{VD} \to \mathbb{F}$ is shown in Table 3 where we write $\mathsf{vecHorner}_\tau(\cdot)$ to denote $\mathsf{vecHorner}(\tau, \cdot)$. The degree of $\mathsf{vecHorner}_\tau(M_1, \ldots, M_k)$ is at most $k + \sum_{i=1}^{k} m_i$ and its constant term is $\mathbf{0}$. Here $m_i = \mathfrak{l}(M_i)$, $i = 1, \ldots, k$.

The following result shows that vecHorner is an AXU family.

Proposition 1. *Let $k \geq k' \geq 0$; $\mathbf{M} = (M_1, \ldots, M_k)$ and $\mathbf{M}' = (M'_1, \ldots, M'_{k'})$ be two distinct vectors in $\mathcal{VD}$ and $\alpha \in \mathbb{F}$. For a uniform random $\tau \in \mathbb{F}$,*

$$\Pr_\tau\left[\mathsf{vecHorner}_\tau(\mathbf{M}) \oplus \mathsf{vecHorner}_\tau(\mathbf{M}') = \alpha\right] \leq \frac{\max\left(k + \sum_{i=1}^{k} m_i, k' + \sum_{j=1}^{k'} m'_j\right)}{2^n} \tag{15}$$

where m_i (resp. m'_j) is the number of n-bit blocks in $\mathsf{pad}_n(M_i)$ (resp. $\mathsf{pad}_n(M'_j)$).

Proof. Let $p(\tau) = \mathsf{vecHorner}_\tau(\mathbf{M}) \oplus \mathsf{vecHorner}_\tau(\mathbf{M}') \oplus \alpha$. If $p(\tau)$ is a non-zero polynomial, then the degree of $p(\tau)$ is at most $\max\left(k + \sum_{i=1}^{k} m_i, k' + \sum_{j=1}^{k'} m'_j\right)$. The probability that a uniform random τ is a root of $p(\tau)$ is at most the stated bound. So, it is sufficient to argue that $p(\tau)$ is non-zero.

If $k' = 0$, then, as $\mathbf{M} \neq \mathbf{M}'$, $k > 0$. In this case, $\mathsf{vecHorner}_\tau(\mathbf{M}') = 1^n\tau$ and the coefficient of τ in $\mathsf{vecHorner}_\tau(\mathbf{M})$ is $L_k \neq 1^n$. Hence, in this case $p(\tau)$ is a non-zero polynomial.

Let M_{i_1,i_2} (resp. M'_{j_1,j_2}) be the n-bit blocks obtained from $\mathbf{M}$ (resp. $\mathbf{M}'$) using format. If $k > k' > 0$, then the coefficient of τ in $p(\tau)$ is $L_k \oplus L'_{k'} \neq \mathbf{0}$ and so $p(\tau)$ is a non-zero polynomial. So, suppose $k = k'$. If there is an i such that $L_i \neq L'_i$, let i be the maximum such index. Using the maximality of i it is possible to argue that $L_i \oplus L'_i$ occurs as a coefficient of some power of τ in $p(\tau)$ and again it follows that $p(\tau)$ is a non-zero polynomial. So, now suppose that $L_i = L'_i$ for all $1 \leq i \leq k = k'$. Since $\mathbf{M} \neq \mathbf{M}'$, there must be an i and j such that $M_{i,j} \neq M'_{i,j}$ again showing that $p(\tau)$ is a non-zero polynomial. $\square$

Hash Function $\mathsf{vecHash2L}$ **[7]** Two hash functions, namely $\mathsf{Hash2L}$ and $\mathsf{vecHash2L}$, have been defined earlier [7]. Here we only recall the definition of $\mathsf{vecHash2L}$ since we will not be using the hash function $\mathsf{Hash2L}$ in this work. The definition of $\mathsf{vecHash2L} : \mathbb{F} \times \mathcal{VD} \to \mathbb{F}$ is given in Table 3 where we write $\mathsf{vecHash2L}_\tau(\cdot)$ to denote $\mathsf{vecHash2L}(\tau, \cdot)$. The degree of $\mathsf{vecHash2L}_\tau(M_1, \ldots, M_k)$ is at most $(\mathfrak{d}(\eta)+1)(\ell_1 + \cdots + \ell_k) + k$, and its constant term is 0. The values of $\ell_1, \ldots, \ell_k$ are defined by the algorithm given in Table 3. Theorem 2 of [7] shows that $\mathsf{vecHash2L}$ is an AXU family. More precisely, the following is proved. Let $k \geq k' \geq 0$; $\mathbf{M} = (M_1, \ldots, M_k)$ and $\mathbf{M}' = (M'_1, \ldots, M'_{k'})$ be two distinct vectors in $\mathcal{VD}$. For a uniform random $\tau \in \mathbb{F}$ and for any $\alpha \in \mathbb{F}$,

$$\Pr_\tau\left[\mathsf{vecHash2L}_\tau(\mathbf{M}) \oplus \mathsf{vecHash2L}_\tau(\mathbf{M}') = \alpha\right] \leq \frac{\max\left(k + (\mathfrak{d}(\eta)+1)\Lambda, k' + (\mathfrak{d}(\eta)+1)\Lambda'\right)}{2^n} \tag{16}$$

where $\Lambda = \sum_{i=1}^{k} \ell_i$ and $\Lambda' = \sum_{j=1}^{k'} \ell'_j$; ℓ_i (resp. ℓ'_j) is the number of super-blocks in M_i (resp. M'_j).

Note that the hash function $\mathsf{vecHash2L}$ is parameterised by the value of η. *In the rest of the paper, we will assume that $\eta+1$ is a power of two so that the degree $\mathfrak{d}(\eta)$ of $\mathsf{BRW}_\tau(X_1, \ldots, X_\eta)$ is η.*

4.2 Specific instantiations

For specific instantiations of FAST, we consider the following two scenarios.

Fixed length setting Fx_m for some positive integer $m > 2$: For this setting, in (2), we define

$$\mathcal{T} = \{0,1\}^n, \quad \mathcal{P} = \{0,1\}^{mn} \text{ and so } \mathcal{M} = \{0,1\}^{n(m-2)}. \tag{17}$$

In other words, a tweak T is an n-bit string while plaintexts and ciphertexts consist of m n-bit strings. This particular setting is suited for disk encryption application, where for a fixed n, the number of blocks m in a message is determined by the size of a disk sector. In this case, for $M \in \mathcal{M}$, $\mathfrak{l}(M) = m-2$ and for $T \in \mathcal{T}$, $\mathfrak{t}(T) = 1$. By the length condition on $\mathcal{P}$, we must have $m \geq 3$.

Consider the encryption and decryption algorithms of FAST. The number of n-bit blocks in P (resp. C) is m and so the number of n-bit blocks in P_3 (resp. C_3) is $m-2$. The hash functions h and h' are invoked in FAST as $h(T, P_3)$ and $h'(T, C_3)$. So, in Definition 1, $\mathcal{T} = \mathbb{F}$ and $\mathcal{M} = \mathbb{F}^{m-2}$ and we have

$$h, h' : \mathbb{F} \times \mathbb{F} \times \mathbb{F}^{m-2} \to \mathbb{F}. \tag{18}$$

For the setting of Fx_m, we describe two instantiations of h and h', one with Horner and the other with BRW. The corresponding instantiations of FAST will be denoted as $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$.

General setting Gn: Let $\mathfrak{k}$ be a fixed integer in the range $\{0, \ldots, 254\}$. For this setting, in (2), we define

$$\mathcal{T} = \bigcup_{k=0}^{\mathfrak{k}} \{(T_1, \ldots, T_k) : 0 \leq \mathsf{len}(T_i) \leq 2^{n-16} - 1\}; \tag{19}$$

$$\mathcal{P} = \bigcup_{i>2n}^{2^{n-16}-1} \{0,1\}^i; \text{ and so} \tag{20}$$

$$\mathcal{M} = \bigcup_{i>0}^{2^{n-16}-2n-1} \{0,1\}^i. \tag{21}$$

A tweak T is a vector $T = (T_1, \ldots, T_k)$ where $0 \leq k \leq \mathfrak{k}$ and each T_i is a binary string. Since $\mathfrak{k} \leq 254$, $\mathfrak{k} + 1 \leq 255$ and so the binary representation of $\mathfrak{k} + 1$ will fit in a byte.

For $P \in \mathcal{P}$, suppose $M \in \mathcal{M}$ is such that $P = X||M$ for some binary string X of length $2n$. Then $\mathfrak{l}(M) = m - 2$, where m is the number of blocks in $\mathsf{pad}_n(P)$. For a tweak $T = (T_1, \ldots, T_k)$, $\mathfrak{t}(T) = \sum_{i=1}^{k} m_i$, where m_i is the number of blocks in $\mathsf{pad}_n(T_i)$.

The parameter $\mathfrak{k}$ controls the maximum number of components that can appear in a tweak. This does not imply that the number of components in all the tweaks is equal to $\mathfrak{k}$. Rather, the number of components in a tweak is between 0 and $\mathfrak{k}$. So, the above definition of the tweak space models tweaks as vectors having variable number of components. Since we put an upper bound of 254 on $\mathfrak{k}$, one possibility is to do away with the parameter $\mathfrak{k}$ and replace it with the value 254. The reason we do not do this is the following. The parameter $\mathfrak{k}$ enters the security bound. If we replace $\mathfrak{k}$ by 254, then this value would enter the security bound. If in practice, the actual value of $\mathfrak{k}$ is much less than 254 (as it is likely to be), then using 254 instead of $\mathfrak{k}$ will lead to a looser security bound than what it should actually be. It is to avoid this unnecessary looseness in the security bound that we introduce and work with the parameter $\mathfrak{k}$.

For the setting of Gn, we describe two instantiations of h and h'. One of these is based on $\mathsf{vecHorner}$ while the other is based on $\mathsf{vecHash2L}$. The parameter $\mathfrak{k}$ is required in both cases while the parameter η is required only in the case of $\mathsf{vecHash2L}$. The instantiations of FAST in the general setting with $\mathsf{vecHorner}$ and $\mathsf{vecHash2L}$ will be denoted as $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ and $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$ respectively.

In the general setting, the lengths of the plaintexts can vary. Also, the tweak space has a rich structure which provides considerable flexibility in applications. Examples of such applications have been mentioned in the introduction. On the downside, the specific instantiations of the general setting are somewhat slower than the corresponding instantiations for the fixed length setting. So, for targeted applications such as disk encryption, it would be preferable to use the fixed length setting leaving out some of the extra overheads incurred in the general setting.

Hash Functions h and h' for $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ Fix a positive integer $m \geq 3$ so that the length condition on $\mathcal{P}$ is satisfied. The hash functions h, h' are defined using Horner as follows:

$$h_\tau(T, X_1||\cdots||X_{m-2}) = \tau\mathsf{Horner}_\tau(\mathbf{1}, X_1, \ldots, X_{m-2}, T); \tag{22}$$
$$h'_\tau(T, X_1||\cdots||X_{m-2}) = \tau^2\mathsf{Horner}_\tau(\mathbf{1}, X_1, \ldots, X_{m-2}, T). \tag{23}$$

Note that $\mathsf{Horner}_\tau(\mathbf{1}, X_1, \ldots, X_{m-2}, T)$ is a monic polynomial in τ of degree $m-1$. Consequently, h and h' are monic polynomials in τ of degrees m and $m+1$ respectively whose constant terms are zero.

Proposition 2. *Let $m \geq 3$ be an integer. The pair (h, h') of hash functions defined in (22) and (23) for the construction $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ is an (ϵ_1, ϵ_2)-eligibile pair, where $\epsilon_1 = \epsilon_2 = (m+2)/2^n$.*

Proof. In this case, for $M \in \mathcal{M}$, $\mathfrak{l}(M) = m - 2$ and for $T \in \mathcal{T}$, $\mathfrak{t}(T) = 1$. We write $\mathfrak{l}$ and $\mathfrak{t}$ instead of $\mathfrak{l}(M)$ and $\mathfrak{t}(T)$.

The polynomials $\tau(h_\tau(T, X_1||\cdots||X_{m-2}) \oplus \alpha) \oplus \beta$ and $\tau(h'_\tau(T, X_1||\cdots||X_{m-2}) \oplus \alpha) \oplus \beta$ are monic polynomials of degrees $\mathfrak{l} + \mathfrak{t} + 2 = m + 1$ and $\mathfrak{l} + \mathfrak{t} + 3 = m + 2$ in τ respectively. So, the probability that a uniform random τ in $\mathbb{F}$ is a root of $\tau(h_\tau(T, X_1||\cdots||X_{m-2}) \oplus \alpha) \oplus \beta$ (resp. $\tau(h'_\tau(T, X_1||\cdots||X_{m-2}) \oplus \alpha) \oplus \beta$) is $(\mathfrak{l}+\mathfrak{t}+2)/2^n = (m+1)/2^n$ (resp. $(\mathfrak{l}+\mathfrak{t}+3)/2^n = (m+2)/2^n$). This shows the value of ϵ_1.

Let $X = X_1||\cdots||X_{m-2}$ and $X' = X'_1||\cdots||X'_{m-2}$ and T, T' be such that $(T, X) \neq (T', X')$. Then $h_\tau(T, X) \oplus h_\tau(T', X')$ is a non-zero polynomial of degree at most $\mathfrak{l} + \mathfrak{t} = m - 1$ whose constant term is zero. This is because the leading terms of $h_\tau(T, X)$ and $h_\tau(T', X')$ will cancel out in the sum $h_\tau(T, X) \oplus h_\tau(T', X')$ so that its degree will be at most $m - 1$; $(T, X) \neq (T', X')$ ensures that $h_\tau(T, X) \oplus h_\tau(T', X')$ is a non-zero polynomial; and the constant terms of both $h_\tau(T, X)$ and $h_\tau(T', X')$ are zero. As a result, $\tau(h_\tau(T, X) \oplus h_\tau(T', X') \oplus \alpha) \oplus \beta$ is a non-zero polynomial in τ of degree at most m. So, the probability that a uniform random τ is a root of this polynomial is at most $(\mathfrak{l} + \mathfrak{t} + 1)/2^n = m/2^n$. A similar reasoning shows that the probability that a uniform random τ is a root of $\tau(h'_\tau(T, X) \oplus h'_\tau(T', X') \oplus \alpha) \oplus \beta$ is at most $(\mathfrak{l} + \mathfrak{t} + 2)/2^n = (m + 1)/2^n$.

For any (T, X) and (T', X'), the polynomial $h_\tau(T, X) \oplus h'_\tau(T', X')$ is a monic polynomial of degree $\mathfrak{l} + \mathfrak{t} + 2 = m + 1$ whose constant term is zero. Consequently, the polynomial $\tau(h_\tau(T, X) \oplus h'_\tau(T', X') \oplus \alpha) \oplus \beta$ is a monic polynomial of degree $m + 2$ and so the probability that a uniform random τ is a root of this polynomial is $(\mathfrak{l} + \mathfrak{t} + 3)/2^n = (m + 2)/2^n$. This shows the value of ϵ_2. □

Hash Functions h and h' for $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ Fix an integer $m \geq 4$. From the length condition on $\mathcal{P}$, we only need $m \geq 3$ and the condition $m \geq 4$ is a special requirement for $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ as we explain below. In this case, the hash functions h, h' are defined using BRW as follows:

$$h_\tau(T, X_1||\cdots||X_{m-2}) = \tau\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T); \tag{24}$$

$$h'_\tau(T, X_1||\cdots||X_{m-2}) = \tau^2\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T). \tag{25}$$

Note that from the definition of BRW polynomials, for $m = 3$, $\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T)$ is not necessarily monic, while for $m \geq 4$, $\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T)$ is necessarily monic. It is to ensure the monic property that we enforce the condition $m \geq 4$ for $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$. An alternative would have been to prepend $\mathbf{1}$ as in the case of $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$. This though would create complications which do not seem to be necessary for the fixed length setting. Instead, we use this technique later in the context of the general setting.

Recall that the degree of $\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T)$ is denoted as $\mathfrak{d}(m - 1)$. So, h and h' are also monic polynomials of degrees $1 + \mathfrak{d}(m - 1)$ and $2 + \mathfrak{d}(m - 1)$ respectively whose constant terms are zero.

Proposition 3. *Let $m \geq 4$ be an integer. The pair (h, h') of hash functions defined in (24) and (25) for the construction $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ is an (ϵ_1, ϵ_2)-eligibile pair, where $\epsilon_1 = \epsilon_2 = (3 + \mathfrak{d}(m - 1))/2^n$. Further, if m is a power of two, then (h, h') is an $((m + 2)/2^n, (m + 2)/2^n)$-eligibile pair.*

Proof. The proof is analogous to the proof of Proposition 2. It is required to use the expression $\mathfrak{d}(m - 1)$ for the degree of $\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T)$ and further the injectivity of the map $(X_1, \ldots, X_{m-2}, T) \longmapsto \mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T)$ ensures that for $(T, X) \neq (T', X')$, the polynomial $\mathsf{BRW}_\tau(X_1, \ldots, X_{m-2}, T) \oplus \mathsf{BRW}_\tau(X'_1, \ldots, X'_{m-2}, T')$ is not zero.

The last statement follows from the previously mentioned fact that $\mathfrak{d}(m - 1) = m - 1$ if and only if $m \geq 4$ is a power of two. □

Remark: In the case where m is a power of two, (h, h') is an $((m + 2)/2^n, (m + 2)/2^n)$-eligible pair for both $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$.

Hash Functions h and h' for $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ In this setting, we define $h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathbb{F}$ where $\mathcal{T}$ and $\mathcal{M}$ are given by (19) and (21) respectively. For $T = (T_1, \ldots, T_k) \in \mathcal{T}$ and

$M \in \mathcal{M}$, let $d = \mathfrak{t}(T) + \mathfrak{l}(M) + k + 2$. We define

$$h_\tau(T, M) = \tau^d \oplus \mathsf{vecHorner}_\tau(T_1, \ldots, T_k, M); \tag{26}$$
$$h'_\tau(T, M) = \tau(\tau^d \oplus \mathsf{vecHorner}_\tau(T_1, \ldots, T_k, M)). \tag{27}$$

It is easy to see that h and h' are monic polynomials of degrees d and $d+1$ respectively whose constant terms are zero. The computation of $\tau^d \oplus \mathsf{vecHorner}_\tau(T_1, \ldots, T_k, M)$ can be done by the following simple modification of the algorithm for computing vecHorner shown in Table 3. *The initialisation of* digest *using* digest $= \mathbf{0}$ *is to be replaced with* digest $= \mathbf{1}$.

Proposition 4. *The hash functions h and h' defined in (26) and (27) respectively for the construction* $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ *form an (ϵ_1, ϵ_2)-eligibile pair, where*

$$\epsilon_1 = \frac{\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 4}{2^n}; \qquad \epsilon_2 = \frac{\max(\mathfrak{t} + \mathfrak{l}, \mathfrak{t}' + \mathfrak{l}') + \mathfrak{k} + 4}{2^n}.$$

Proof. For $T = (T_1, \ldots, T_k)$, recall that $\mathfrak{t} = \mathfrak{t}(T) = \sum_{i=1}^{k} m_i$ where m_i is the number of n-bit blocks in $\mathsf{pad}_n(T_i)$. Also, $\mathfrak{l} = \mathfrak{l}(M)$ is the number of n-bit blocks in $\mathsf{pad}_n(M)$.

The degree of $h_\tau(T, M)$ is $d = \mathfrak{t} + \mathfrak{l} + k + 2 \leq \mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 2$ and the degree of $h'_\tau(T, M)$ is $d + 1 = \mathfrak{t} + \mathfrak{l} + k + 3 \leq \mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 3$. So, the polynomial $\tau(h_\tau(T, M) \oplus \alpha) \oplus \beta$ is a monic polynomial of degree at most $\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 3$ and the polynomial $\tau(h'_\tau(T, M) \oplus \alpha) \oplus \beta$ is a monic polynomial of degree at most $\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 4$. This shows the value of ϵ_1.

Consider $(T', M') \neq (T, M)$ where $T' = (T'_1, \ldots, T'_{k'})$, $\mathfrak{t}' = \mathfrak{t}(T')$ and $\mathfrak{l}' = \mathfrak{l}(M')$. Without loss of generality assume that $k \geq k'$. Let $p(\tau) = \tau(h_\tau(T, M) \oplus h_\tau(T', M') \oplus \alpha) \oplus \beta$. If the degrees of $h_\tau(T, M)$ and $h_\tau(T', M')$ are not equal, then $p(\tau)$ is a polynomial of degree $\max(\mathfrak{t} + \mathfrak{l} + k + 3, \mathfrak{t}' + \mathfrak{l}' + k' + 3) \leq \max(\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 3, \mathfrak{t}' + \mathfrak{l}' + \mathfrak{k} + 3)$. So, suppose that the degrees of $h_\tau(T, M)$ and $h_\tau(T', M')$ are equal. The leading monic terms of the two polynomials cancel out. If $p(\tau)$ is a non-zero polynomial, then it has maximum degree $\max(\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 2, \mathfrak{t}' + \mathfrak{l}' + \mathfrak{k} + 2)$. So, it is sufficient to show that $p(\tau)$ is a non-zero polynomial. This argument is similar to that of Proposition 1. Further, a similar argument applies for h' where the degree of $\tau(h'_\tau(T, M) \oplus h'_\tau(T', M') \oplus \alpha) \oplus \beta$ is at most $\max(\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 4, \mathfrak{t}' + \mathfrak{l}' + \mathfrak{k} + 4)$.

Now consider (T, M) and (T', M') which are not necessarily distinct and let $p(\tau) = \tau(h_\tau(T, M) \oplus h'_\tau(T', M') \oplus \alpha) \oplus \beta$. The coefficient of τ in $h_\tau(T, M)$ is $L = \mathsf{bin}_8(k + 1) || 0^8 || \mathsf{bin}_{n-16}(\mathsf{len}(M)) \neq \mathbf{0}$ which is the coefficient of τ^2 in τh_τ. The coefficient of τ^2 in $\tau h'_\tau(T', M')$ is $\mathbf{0}$ and so the coefficient of τ^2 in $p(\tau)$ is $L \neq \mathbf{0}$. So, $p(\tau)$ is a non-zero polynomial. The degree of $p(\tau)$ is at most $\max(\mathfrak{t} + \mathfrak{l} + \mathfrak{k} + 3, \mathfrak{t}' + \mathfrak{l}' + \mathfrak{k} + 4)$.

This completes the proof. $\square$

Hash Functions h and h' for $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$ In this setting, we define $h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \to \mathbb{F}$ where $\mathcal{T}$ and $\mathcal{M}$ are given by (19) and (21) respectively. For $T = (T_1, \ldots, T_k) \in \mathcal{T}$ and $M \in \mathcal{M}$, let the number of super-blocks in $\mathsf{pad}_n(T_i)$ be ℓ_i and the number of super-blocks in $\mathsf{pad}_n(M)$ be ℓ. Let $d = (\mathfrak{d}(\eta) + 1)(\ell_1 + \cdots + \ell_k + \ell) + k + 2$. We define

$$h_\tau(T, M) = \tau^d \oplus \mathsf{vecHash2L}_\tau(T_1, \ldots, T_k, M); \tag{28}$$
$$h'_\tau(T, M) = \tau(\tau^d \oplus \mathsf{vecHash2L}_\tau(T_1, \ldots, T_k, M)). \tag{29}$$

The definition of vecHash2L requires choosing the value η. As mentioned earlier, we will assume that η is chosen so that $\eta + 1$ is a power of two and so $\mathfrak{d}(\eta) = \eta$. The computation of $\tau^d \oplus$

$\mathsf{vecHash2L}_\tau(T_1,\ldots,T_k,M)$ can be done by the following simple modification of the algorithm for computing $\mathsf{vecHash2L}$ shown in Table 3. *The initialisation of* digest *using* $\mathsf{digest} = \mathbf{0}$ *is to be replaced with* $\mathsf{digest} = \mathbf{1}$.

Proposition 5. *Let the parameter $\eta \geq 3$ required in the definition of* $\mathsf{vecHash2L}$ *be such that $\eta+1$ is a power of two. The hash functions h and h' defined in (28) and (29) respectively for the construction* $\mathsf{FAST}[\mathsf{Gn},\mathfrak{k},\eta,\mathsf{vecHash2L}]$ *form an (ϵ_1,ϵ_2)-eligibile pair, where*

$$\epsilon_1 = \frac{((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l}) + (\mathfrak{k}+1)(\eta+2)+3}{2^n};$$
$$\epsilon_2 = \frac{((\eta+1)/\eta)\max(\mathfrak{t}+\mathfrak{l},\mathfrak{t}'+\mathfrak{l}') + (\mathfrak{k}+1)(\eta+2)+3}{2^n}.$$

Proof. Since $\eta+1\geq 4$ is a power of two, $\mathfrak{d}(\eta)=\eta$.

For $i=1,\ldots,k$, let the number of super-blocks in $\mathsf{pad}_n(T_i)$ be ℓ_i and the number of super-blocks in $\mathsf{pad}_n(M)$ be ℓ. For $i=1,\ldots,k$, let the number of n-bit blocks in $\mathsf{pad}_n(T_i)$ be m_i, so that $\mathfrak{t}=\mathfrak{t}(T)=\sum_{i=1}^k m_i$ and the number of n-bit blocks in $\mathsf{pad}_n(M)$ be $\mathfrak{l}$. In $\mathsf{pad}_n(T_i)$, each of the first ℓ_i-1 super-blocks contains exactly η n-bit blocks and the last super-block contains at most η blocks. Since the total number of n-bit blocks in $\mathsf{pad}_n(T_i)$ is m_i, we have $m_i > \eta(\ell_i-1)$ from which we obtain $(\eta+1)\ell_i < m_i((\eta+1)/\eta)+\eta+1$. Similarly, $(\eta+1)\ell < \mathfrak{l}((\eta+1)/\eta)+\eta+1$. We have

$$\begin{aligned}
d &= (\mathfrak{d}(\eta)+1)(\ell_1+\cdots+\ell_k+\ell)+k+2\\
&\leq (\mathfrak{d}(\eta)+1)(\ell_1+\cdots+\ell_k+\ell)+\mathfrak{k}+2\\
&= (\eta+1)(\ell_1+\cdots+\ell_k+\ell)+\mathfrak{k}+2 \quad (\text{since } \mathfrak{d}(\eta)=\eta)\\
&< ((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(k+1)(\eta+1)+\mathfrak{k}+2\\
&\leq ((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(\mathfrak{k}+1)(\eta+2)+1.
\end{aligned} \tag{30}$$

So, the degree of $\tau h_\tau(T,M)$ is $d+1$ which is at most $((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(\mathfrak{k}+1)(\eta+2)+2$ and the degree of $\tau h'_\tau(T,M)$ is $d+2$ which is at most $((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(\mathfrak{k}+1)(\eta+2)+3$. This shows the value of ϵ_1.

Consider $(T',M')\neq(T,M)$ where $T'=(T'_1,\ldots,T'_{k'})$, $\mathfrak{t}'=\mathfrak{t}(T')$ and $\mathfrak{l}'=\mathfrak{l}(M')$. Let d' be the degree of $h_\tau(T',M')$. For any $\alpha,\beta\in\mathbb{F}$, we wish to bound the probability (over uniform random choice of τ in $\mathbb{F}$) that the polynomial $p_1(\tau)=\tau(h_\tau(T,M)\oplus h_\tau(T',M')\oplus\alpha)\oplus\beta$ is zero. If $d\neq d'$, then $p_1(\tau)$ is a monic polynomial of degree $\max(d+1,d'+1)$ and so the probability that it is zero is at most $\max(d+1,d'+1)/2^n$. If $d=d'$, then $p_1(\tau)$ is zero if and only if the polynomial

$$p_2(\tau)=\tau(\mathsf{vecHash2L}_\tau(T_1,\ldots,T_k,M)\oplus\mathsf{vecHash2L}_\tau(T'_1,\ldots,T'_{k'},M')\oplus\alpha)\oplus\beta$$

is zero. Using Theorem 2 of [7] (see (16)), we have this probability to be at most $\max(d,d')/2^n$. So, the probability that $p_1(\tau)$ is zero is at most $\max(d+1,d'+1)/2^n$. Similarly, the probability that $\tau(h'_\tau(T,M)\oplus h'_\tau(T',M')\oplus\alpha)\oplus\beta$ is zero is at most $\max(d+2,d'+2)/2^n$.

Consider (T,M) and (T',M') which are not necessarily distinct. Fix $\alpha,\beta\in\mathbb{F}$ and consider $p(\tau)=\tau(h_\tau(T,M)\oplus h'_\tau(T',M')\oplus\alpha)\oplus\beta$. The coefficient of τ in

$$h_\tau(T,M)=\tau^d\oplus\mathsf{vecHash2L}_\tau(T_1,\ldots,T_k,M)$$

is

$$L=\mathsf{bin}_8(k+1)||0^8||\mathsf{bin}_{n-16}(\mathsf{len}(M))\neq\mathbf{0},$$

which is the coefficient of τ^2 in $\tau h_\tau(T,M)$. The coefficient of τ^2 in $\tau h'_\tau(T',M')$ is $\mathbf{0}$ and so the coefficient of τ^2 in $p(\tau)$ is $L\neq\mathbf{0}$. So, $p(\tau)$ is a non-zero polynomial. The degree of $p(\tau)$ is at most $\max(d+1,d'+2)\leq\max(d+2,d'+2)$. This shows the value of ϵ_2. $\square$

5 Security

In this section, we provide the formal definitions and the formal security statement for FAST.

An adversary $\mathcal{A}$ is a possibly probabilistic algorithm with access to one or more oracles. The output of an adversary is a single bit. The notation $\mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\ldots} \Rightarrow 1$ denotes the fact that $\mathcal{A}$ outputs the bit 1 after interacting with the oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots$. The interaction of $\mathcal{A}$ with its oracles is allowed to be adaptive, i.e., the adversary is allowed to choose an oracle and a query to be made to this oracle based on the responses it has received to its previous queries.

The important parameters of an adversary are its running time $\mathfrak{T}$, the number of queries q that it makes to all its oracles and its query complexity σ. The query complexity is defined later.

The bulk of the actual security analysis will be in the information theoretic sense which in particular means that there is no restriction on the resources of the adversary. For such analysis, it is sufficient to consider the adversary to be a deterministic algorithm.

5.1 Pseudo-Random Function

Let $\{\mathbf{F}_K\}_{K\in\mathcal{K}}$ be a family of functions where for each $K \in \mathcal{K}$, $\mathbf{F}_K : \mathcal{D} \to \mathcal{R}$. Here $\mathcal{K}$ is the keyspace, $\mathcal{D}$ is the domain and $\mathcal{R}$ is the range. We require that all three of $\mathcal{K}$, $\mathcal{D}$ and $\mathcal{R}$ are finite non-empty sets.

Let ρ be a function chosen uniformly at random from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$. An equivalent and more convenient view of ρ is the following. For distinct elements $X_1, \ldots, X_q$ from $\mathcal{D}$, the elements $\rho(X_1), \ldots, \rho(X_q)$ are independent and uniformly distributed elements of $\mathcal{R}$.

Roughly speaking, the function family $\{\mathbf{F}_K\}_{K\in\mathcal{K}}$ is said to be a PRF if for a uniform random $K \in \mathcal{K}$, the outputs of $\mathbf{F}_K(\cdot)$ on distinct elements of $\mathcal{D}$ are computationally indistinguishable from the outputs of ρ. This is formalised in the following manner. An adversary $\mathcal{A}$ has access to an oracle and can make queries to its oracle in an adaptive manner. We will assume that $\mathcal{A}$ does not repeat a query. The advantage of the adversary $\mathcal{A}$ in breaking the PRF-property of $\{\mathbf{F}_K\}_{K\in\mathcal{K}}$ is defined as follows.

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho(\cdot)} \Rightarrow 1\right]. \tag{31}$$

Suppose $\mathcal{A}$ makes a total of q queries which are $X^{(1)}, \ldots, X^{(q)}$. The query complexity of $\mathcal{A}$ is the sum total of the number of n-bit blocks in $\mathsf{pad}_n(X^{(s)})$, $s = 1, \ldots, q$.

By $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}, q, \sigma)$ we denote the maximum of $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A})$ over all adversaries $\mathcal{A}$ which run in time $\mathfrak{T}$, make q queries and have query complexity σ. The function family $\{\mathbf{F}_K\}_{K\in\mathcal{K}}$ (or, more simply $\mathbf{F}$) is said to be a $(\mathfrak{T}, q, \sigma, \varepsilon)$-PRF if $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}, q, \sigma) \leq \varepsilon$.

5.2 Tweakable Enciphering Scheme

A tweakable enciphering scheme is a pair $\mathsf{TES} = (\mathsf{TES.Encrypt}, \mathsf{TES.Decrypt})$ where

$$\mathsf{TES.Encrypt}, \mathsf{TES.Decrypt} : \mathcal{K} \times \mathcal{T} \times \mathcal{P} \to \mathcal{P}$$

for finite non-empty sets $\mathcal{K}, \mathcal{T}$ and $\mathcal{P}$. The set $\mathcal{K}$ is called the key space, $\mathcal{T}$ is called the tweak space and $\mathcal{P}$ is called the message/ciphertext space. We write $\mathsf{TES.Encrypt}_K(\cdot,\cdot)$ ($\mathsf{TES.Decrypt}_K(\cdot,\cdot)$) to denote $\mathsf{TES.Encrypt}(K,\cdot,\cdot)$ (resp. $\mathsf{TES.Decrypt}(K,\cdot,\cdot)$). The functions $\mathsf{TES.Encrypt}$ and $\mathsf{TES.Decrypt}$ satisfy the following two properties. For $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $P \in \mathcal{P}$,

1. $\mathsf{TES.Decrypt}_K(T, \mathsf{TES.Encrypt}_K(T, P)) = P$;
2. $\mathsf{len}\left(\mathsf{TES.Encrypt}_K(T, P)\right) = \mathsf{len}(P)$.

The first property states that the encrypt and the decrypt functions are inverses while the second property states that the length of the ciphertext is equal to the length of the plaintext. In other words, $\mathsf{TES.Encrypt}_K(T, \cdot)$ is a length preserving permutation of $\mathcal{P}$.

The notion of security for a TES that we consider is that of indistinguishability from uniform random strings. This implies other notions of security (see [22]). Let $\mathfrak{F}$ be the set of all functions f from $\mathcal{T} \times \mathcal{P}$ to $\mathcal{P}$ such that for any $T \in \mathcal{T}$ and $P \in \mathcal{P}$, $\mathsf{len}(f(T, P)) = \mathsf{len}(P)$. Let ρ_1 and ρ_2 be two functions chosen independently and uniformly at random from $\mathfrak{F}$.

An adversary $\mathcal{A}$ attacking a TES has access to two oracles which we will call the left and the right oracles. Both the oracles are functions from $\mathfrak{F}$. An input to the left oracle is of the form (T, P) and the response is C, while an input to the right oracle is of the form (T, C) and the response is P. The adversary $\mathcal{A}$ adaptively queries its oracles possibly interweaving its queries to its left and right oracles. At the end, $\mathcal{A}$ outputs a bit.

We assume that the adversary does not make any pointless query. This means that $\mathcal{A}$ does not repeat a query to any of its oracles; does not query the right oracle with (T, C) if it received C in response to a query (T, P) made to its left oracle; and does not query the left oracle with (T, P) if it received P in response to a query (T, C) made to its right oracle. The advantage of $\mathcal{A}$ in breaking TES is defined as follows:

$$\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{TES}}(\mathcal{A}) \\
&= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{TES.Encrypt}_K(\cdot,\cdot), \mathsf{TES.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot), \rho_2(\cdot,\cdot)} \Rightarrow 1\right]. \qquad (32)
\end{aligned}$$

Suppose $\mathcal{A}$ makes q queries $(T^{(1)}, X^{(1)}), \ldots, (T^{(q)}, X^{(q)})$ where $T^{(s)} = (T^{(s)}_1, \ldots, T^{(s)}_{k^{(s)}})$ and $X^{(s)}$ is either $P^{(s)}$ or $C^{(s)}$. For $j = 1, \ldots, k^{(s)}$, let $m^{(s)}_j$ be the number of n-bit blocks in $\mathsf{pad}_n(T^{(s)}_j)$ and let $m^{(t)}$ be the number of n-bit blocks in $\mathsf{pad}_n(X^{(t)})$. We will write $\mathfrak{t}^{(s)}$ to denote $\mathfrak{t}(T^{(s)})$. Let $X^{(s)} = X^{(s)}_1 || X^{(s)}_2 || X^{(s)}_3$ with $\mathsf{len}(X^{(s)}_1) = \mathsf{len}(X^{(s)}_2) = n$ and $\mathsf{len}(X^{(s)}_3) \geq 1$. Then $X^{(s)}_3 \in \mathcal{M}$ and $\mathfrak{l}(X^{(s)}_3) = m^{(s)} - 2$. We will write $\mathfrak{l}^{(s)}$ to denote $\mathfrak{l}(X^{(s)}_3)$.

The tweak query complexity θ, the message query complexity ω and the total query complexity σ are defined as follows.

$$\theta = \sum_{s=1}^{q} \mathfrak{t}(T^{(s)}) = \sum_{s=1}^{q} \mathfrak{t}^{(s)}; \qquad (33)$$

$$\omega = \sum_{t=1}^{q} m^{(t)}; \qquad (34)$$

$$\sigma = \theta + \omega. \qquad (35)$$

By $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{TES}}(\mathfrak{T}, q, \theta, \omega)$ we denote the maximum of $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{TES}}(\mathcal{A})$ over all adversaries $\mathcal{A}$ which run in time $\mathfrak{T}$, make q queries and have tweak query complexity θ and message query complexity ω. TES is said to be $(\mathfrak{T}, q, \theta, \omega, \varepsilon)$-secure if $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{TES}}(\mathcal{A}) \leq \varepsilon$ for all $\mathcal{A}$ running in time $\mathfrak{T}$, making a total of q queries with tweak query complexity θ and message query complexity ω.

5.3 Security of FAST

The security proof for FAST is the following.

Theorem 1. *Let n be a positive integer; $\mathbb{F} = GF(2^n)$ is represented using some fixed irreducible polynomial of degree n over $GF(2)$; $\{\mathbf{F}_K\}_{K\in\mathcal{K}}$ where for $K \in \mathcal{K}$, $\mathbf{F}_K : \{0,1\}^n \to \{0,1\}^n$; (h, h') is an (ϵ_1, ϵ_2)-eligible pair of hash functions, where $h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \to \mathbb{F}$; and fStr is a fixed n-bit string used to build the TES*

$$\mathsf{FAST} = (\mathsf{FAST.Encrypt}, \mathsf{FAST.Decrypt})$$

given in Table 1. Fix q, $\omega \geq q$ to be positive integers and θ to be a non-negative integer. For all $\mathfrak{T} > 0$,

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathfrak{T}, q, \theta, \omega) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega + 1, \omega + 1) + \Delta(\mathsf{FAST}), \qquad \textit{where}$$
$$\Delta(\mathsf{FAST}) = 2\omega \left(\sum_{s=1}^{q} \epsilon_1^{(s)} \right) + 3 \sum_{1 \leq s < t \leq q} \epsilon_2^{(s,t)} + \sum_{s=1}^{q} \epsilon_2^{(s,s)} + \frac{3\omega^2}{2^n}; \tag{36}$$

$\mathfrak{T}'$ is the time required to answer q queries with tweak query complexity θ and message query complexity ω plus some bookkeeping time; and for $1 \leq s, t \leq q$, $\epsilon_1^{(s)} = \epsilon_1(\mathfrak{t}^{(s)}, \mathfrak{l}^{(s)})$, $\epsilon_2^{(s,t)} = \epsilon_2(\mathfrak{t}^{(s)}, \mathfrak{l}^{(s)}, \mathfrak{t}^{(t)}, \mathfrak{l}^{(t)})$.

Proof. Let $\mathcal{A}$ be an adversary attacking FAST. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ attacking the PRF-property of $\mathbf{F}$. $\mathcal{B}$ has access to an oracle which is either $\mathbf{F}_K(\cdot)$ for a uniform random K in $\mathcal{K}$, or, the oracle is ρ which is a uniform random function from $\{0,1\}^n$ to $\{0,1\}^n$. Adversary $\mathcal{B}$ uses the (ϵ_1, ϵ_2)-eligible pair of hash functions (h, h') to set up an instance of FAST and invokes $\mathcal{A}$ to attack this instance. $\mathcal{A}$ makes a number of oracle queries to the encryption and decryption oracles of FAST. $\mathcal{B}$ uses its own oracle and the hash functions h and h' to compute the responses which it provides to $\mathcal{A}$. At the end, $\mathcal{A}$ outputs a bit and $\mathcal{B}$ outputs the same bit. Note that both encryption and decryption queries by $\mathcal{A}$ can be answered using the oracle of $\mathcal{B}$ and the hash functions h, h'.

The running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ along with the time required to compute the responses to the queries made by $\mathcal{A}$ using $\mathcal{B}$'s oracle plus some bookkeeping time which includes the time for set-up. So, the total running time of $\mathcal{B}$ is $\mathfrak{T} + \mathfrak{T}'$ as desired. Further, to answer $\mathcal{A}$'s queries, $\mathcal{B}$ needs to make a query to its oracle on fStr and to answer the s-th query, it needs to make $m^{(s)}$ queries to its oracle. So, the total number of times $\mathcal{B}$ queries its oracle is $1 + \sum_{s=1}^{q} m^{(s)} = \omega + 1$. Since each query of $\mathcal{B}$ consists of a single n-bit block, the query complexity is also $\omega + 1$.

If the oracle to $\mathcal{B}$ is the real oracle, i.e., the oracle is $\mathbf{F}_K$, then $\mathcal{A}$ gets to interact with the real encryption and decryption oracles of FAST. So,

$$\Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right]. \tag{37}$$

Denote by FAST_ρ the instance of FAST where $\mathbf{F}_K$ is replaced by ρ. If the oracle to $\mathcal{B}$ is the random oracle, i.e., the oracle is ρ, then

$$\Pr\left[\mathcal{B}^{\rho(\cdot)} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right]. \tag{38}$$

So,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\rho(\cdot)} \Rightarrow 1\right] \\ &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] \\ &\qquad - \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right]. \end{aligned} \tag{39}$$

The advantages of $\mathcal{A}$ and $\mathcal{B}$ are related as follows. Recall that $\mathfrak{F}$ is the set of all functions f from $\mathcal{T} \times \mathcal{P}$ to $\mathcal{P}$ such that for any $T \in \mathcal{T}$ and $P \in \mathcal{P}$, $\mathsf{len}(f(T, P)) = \mathsf{len}(P)$. Let $\rho_1(\cdot,\cdot)$ and $\rho_2(\cdot,\cdot)$ be two independent and uniform random functions from $\mathfrak{F}$.

$$
\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathcal{A}) \\
&= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right] \\
&= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] \\
&\quad - \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right] \\
&\quad + \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right] \\
&= \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) + \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right] \\
&\quad - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right]. \qquad (40)
\end{aligned}
$$

There are two events to consider, namely,

$$\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1 \text{ and } \mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1.$$

Consider the event $\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1$. Suppose $\mathcal{A}$ makes a total of q queries with tweak query complexity θ and message query complexity ω. For $1 \leq s \leq q$, let $\mathsf{ty}^{(s)} = \mathsf{enc}$ if the s-th query is an encryption query and $\mathsf{ty}^{(s)} = \mathsf{dec}$ if the s-th query is a decryption query. Denote the tweak, the plaintext and the ciphertext associated with the s-th query by $T^{(s)}$, $P^{(s)} = P_1^{(s)}||P_2^{(s)}||P_3^{(s)}$ and $C^{(s)} = C_1^{(s)}||C_2^{(s)}||C_3^{(s)}$ respectively. We have $\mathfrak{t}^{(s)} = \mathfrak{t}(T^{(s)})$ and $m^{(s)}$ is the number of n-bit blocks in $\mathsf{pad}_n(P^{(s)})$ and $\mathsf{pad}_n(C^{(s)})$. Also, $\mathfrak{l}^{(s)} = \mathfrak{l}(P_3^{(s)}) = \mathfrak{l}(C_3^{(s)}) = m^{(s)} - 2$.

The interaction of $\mathcal{A}$ with the oracle in this setting is given by the game G_{real} which is shown in Table 4. In this game, the random function ρ is built incrementally. Whenever a "new" input to ρ is received, the output is chosen independently and uniformly at random. The variable bad is set to true if it turns out that two inputs to ρ collide. Let $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$ be the event that bad is set to true in the game G_{real}. Also, by $\mathcal{A}^{G_{\mathsf{real}}} \Rightarrow 1$ we denote the event that $\mathcal{A}$ outputs 1 in the game G_{real}. Note that $\mathcal{A}^{G_{\mathsf{real}}} \Rightarrow 1$ is exactly the event $\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1$.

If $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$ does not occur, then the boxed instruction in game G_{real} is not executed. The absence of the boxed instruction does not affect the probability of $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$. We consider the distributions of the plaintexts and ciphertexts when $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$ does not occur. Let $Y_1^{(s)}$ denote the output of $\mathsf{Ch}\text{-}\rho(F_1^{(s)})$ and $Y_2^{(s)}$ denote the output of $\mathsf{Ch}\text{-}\rho(F_2^{(s)})$. Suppose $\mathsf{ty}^{(s)} = \mathsf{enc}$, i.e., the s-th query is an encryption query. Then from game G_{real}, we can write

$$
\begin{aligned}
C_1^{(s)} &= Y_1^{(s)} \oplus P_1^{(s)} \oplus \tau C_2^{(s)} \oplus \tau h'_\tau(T^{(s)}, C_3^{(s)}) \oplus h_\tau(T^{(s)}, P_3^{(s)}); \\
C_2^{(s)} &= Y_2^{(s)} \oplus P_2^{(s)} \oplus \tau P_1^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}) \oplus \tau h_\tau(T^{(s)}, P_3^{(s)}); \\
C_{3,i}^{(s)} &= S_i^{(s)} \oplus P_{3,i}^{(s)} \text{ for } i = 1, \ldots, m^{(s)} - 3; \\
C_{3,m^{(s)}-2}^{(s)} &= \mathsf{first}_{r^{(s)}}(D^{(s)}) \oplus P_{3,m^{(s)}-2}^{(s)};
\end{aligned}
$$

When $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$ does not occur, $Y_1^{(s)}, Y_2^{(s)}, S_i^{(s)}, (i = 1, \ldots, m^{(s)} - 3), D^{(s)}$ are independent and uniform random strings. From the above relations, it is easy to argue that the ciphertext $C^{(s)}$ is also independent and uniform random. A similar argument shows that when $\mathsf{ty}^{(s)} = \mathsf{dec}$, i.e.,

Table 4: Game G_{real}.

<table>
<tr><td colspan="2">Subroutine $\mathsf{Ch}\text{-}\rho(X)$
$Y \xleftarrow{\$} \{0,1\}^n$;
if $X \in \mathcal{D}$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{Y \leftarrow \rho(X);}$ endif;
$\rho(X) \leftarrow Y$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{X\}$; $\mathbf{return}(Y)$;
<u>Initialization</u>:
$\tau \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} \leftarrow \{\mathsf{fStr}\}$; $\mathsf{bad} \leftarrow \mathsf{false}$.</td></tr>
<tr><td><u>$\mathsf{ty}^{(s)} = \mathsf{enc}$: input $(T^{(s)}, P^{(s)})$</u>
$(P_1^{(s)}, P_2^{(s)}, P_3^{(s)}) \leftarrow \mathsf{parse}_n(P^{(s)})$;
$A_1^{(s)} \leftarrow P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})$;
$F_1^{(s)} \leftarrow \tau A_1^{(s)} \oplus P_2^{(s)}$;
$F_2^{(s)} \leftarrow A_1^{(s)} \oplus \mathsf{Ch}\text{-}\rho(F_1^{(s)})$;
$B_2^{(s)} \leftarrow F_1^{(s)} \oplus \mathsf{Ch}\text{-}\rho(F_2^{(s)})$;
$C_1^{(s)} \leftarrow \tau B_2^{(s)} \oplus F_2^{(s)}$;
$Z^{(s)} \leftarrow F_1^{(s)} \oplus F_2^{(s)}$;
for $i = 1$ to $m^{(s)} - 3$ do
$J_i^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(i)$;
$S_i^{(s)} \leftarrow \mathsf{Ch}\text{-}\rho(J_i^{(s)})$;
$C_{3,i}^{(s)} \leftarrow P_{3,i}^{(s)} \oplus S_i^{(s)}$;
end for;
$J_{m^{(s)}-2}^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(m^{(s)} - 2)$;
$D^{(s)} \leftarrow \mathsf{Ch}\text{-}\rho(J_{m^{(s)}-2}^{(s)})$;
$C_{3,m^{(s)}-2}^{(s)} \leftarrow P_{3,m^{(s)}-2}^{(s)} \oplus \mathsf{first}_{r^{(s)}}(D^{(s)})$;
$C_2^{(s)} \leftarrow B_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$;
return $(C_1^{(s)} \| C_2^{(s)} \| C_3^{(s)})$.</td><td><u>$\mathsf{ty}^{(s)} = \mathsf{dec}$: input $(T^{(s)}, C^{(s)})$</u>
$(C_1^{(s)}, C_2^{(s)}, C_3^{(s)}) \leftarrow \mathsf{parse}_n(C^{(s)})$;
$B_2^{(s)} \leftarrow C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$;
$F_2^{(s)} \leftarrow C_1^{(s)} \oplus \tau B_2^{(s)}$;
$F_1^{(s)} \leftarrow B_2^{(s)} \oplus \mathsf{Ch}\text{-}\rho(F_2^{(s)})$;
$A_1^{(s)} \leftarrow F_2^{(s)} \oplus \mathsf{Ch}\text{-}\rho(F_1^{(s)})$;
$P_2^{(s)} \leftarrow \tau A_1^{(s)} \oplus F_1^{(s)}$;
$Z^{(s)} \leftarrow F_1^{(s)} \oplus F_2^{(s)}$;
for $i = 1$ to $m^{(s)} - 3$ do
$J_i^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(i)$;
$S_i^{(s)} \leftarrow \mathsf{Ch}\text{-}\rho(J_i^{(s)})$;
$P_{3,i}^{(s)} \leftarrow C_{3,i}^{(s)} \oplus S_i^{(s)}$;
end for;
$J_{m^{(s)}-2}^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(m^{(s)} - 2)$;
$E^{(s)} \leftarrow \mathsf{Ch}\text{-}\rho(J_{m^{(s)}-2}^{(s)})$;
$P_{3,m^{(s)}-2}^{(s)} \leftarrow C_{3,m^{(s)}-2}^{(s)} \oplus \mathsf{first}_{r^{(s)}}(E^{(s)})$;
$P_1^{(s)} \leftarrow A_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})$;
return $(P_1^{(s)} \| P_2^{(s)} \| P_3^{(s)})$.</td></tr>
</table>

the query is a decryption query, then $P^{(s)}$ is an independent and uniform random string. So, if $\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})$ does not occur, then the adversary obtains independent and uniform random strings as responses to all its queries.

In the next step, the game G_{real} is modified to the game G_{int}. This game is shown in Table 5. In this game, the outputs of ρ are not chosen directly. Instead, these are defined from the plaintexts and the ciphertexts. For a enciphering query, the ciphertext is chosen independently and uniformly at random while for a deciphering query, the plaintext is chosen independently and uniformly at random. The outputs of ρ are defined from these in the following manner.

$$
\begin{aligned}
\rho(F_1^{(s)}) = Y_1^{(s)} &\leftarrow C_1^{(s)} \oplus P_1^{(s)} \oplus \tau C_2^{(s)} \oplus \tau h'_\tau(T^{(s)}, C_3^{(s)}) \oplus h_\tau(T^{(s)}, P_3^{(s)}); \\
\rho(F_2^{(s)}) = Y_2^{(s)} &\leftarrow C_2^{(s)} \oplus P_2^{(s)} \oplus \tau P_1^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}) \oplus \tau h_\tau(T^{(s)}, P_3^{(s)}); \\
\rho(J_i^{(s)}) &= C_{3,i}^{(s)} \oplus P_{3,i}^{(s)} \text{ for } i = 1, \ldots, m^{(s)} - 3; \\
\rho(J_{m^{(s)}-2}^{(s)}) &= \begin{cases} D^{(s)} \oplus (P_{3,m^{(s)}-2}^{(s)} \| 0^{n-r^{(s)}}) \text{ if } \mathsf{ty}^{(s)} = \mathsf{enc}; \\ E^{(s)} \oplus (C_{3,m^{(s)}-2}^{(s)} \| 0^{n-r^{(s)}}) \text{ if } \mathsf{ty}^{(s)} = \mathsf{dec}; \end{cases}
\end{aligned}
\tag{41}
$$

As for an encryption query, $C_1^{(s)}, C_2^{(s)}, C_{3,1}^{(s)}, \ldots, C_{3,m^{(s)}-3}^{(s)}, D^{(s)}$ are chosen independently and uniformly at random, from (41) it follows that the outputs of ρ are also independent and uniformly distributed. For a decryption query, $P_1^{(s)}, P_2^{(s)}, P_{3,1}^{(s)}, \ldots, P_{3,m^{(s)}-3}^{(s)}, E^{(s)}$ are chosen independently and uniformly at random. Again, from (41) it follows that the outputs of ρ are also independent

Table 5: Game G_{int}.

<table>
<tr><td colspan="2">Subroutine $\mathsf{ChkDom}(X)$
if $X \in \mathcal{D}$ then $\mathsf{bad} \leftarrow \mathsf{true}$; endif;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{X\}$;
<u>Initialization</u>:
$\tau \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} \leftarrow \{\mathsf{fStr}\}$; $\mathsf{bad} \leftarrow \mathsf{false}$.</td></tr>
<tr><td><u>$\mathsf{ty}^{(s)} = \mathsf{enc}$: input $(T^{(s)}, P^{(s)})$</u>
$(P_1^{(s)}, P_2^{(s)}, P_3^{(s)}) \leftarrow \mathsf{parse}_n(P^{(s)})$;
$C_1^{(s)} \xleftarrow{\$} \{0,1\}^n$; $C_2^{(s)} \xleftarrow{\$} \{0,1\}^n$;
for $i = 1, \ldots, m^{(s)} - 3$ do $C_{3,i}^{(s)} \xleftarrow{\$} \{0,1\}^n$;
$W^{(s)} \xleftarrow{\$} \{0,1\}^n$; $C_{3,m^{(s)}-2} \leftarrow \mathsf{first}_{r^{(s)}}(W^{(s)})$;
$A_1^{(s)} \leftarrow P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})$;
$F_1^{(s)} \leftarrow \tau A_1^{(s)} \oplus P_2^{(s)}$; $\mathsf{ChkDom}(F_1^{(s)})$;
$Y_1^{(s)} \leftarrow C_1^{(s)} \oplus P_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}))$
$\oplus h_\tau(T^{(s)}, P_3^{(s)})$;
$\rho(F_1^{(s)}) \leftarrow Y_1^{(s)}$;
$F_2^{(s)} \leftarrow A_1^{(s)} \oplus Y_1^{(s)}$; $\mathsf{ChkDom}(F_2^{(s)})$;
$Y_2^{(s)} \leftarrow C_2^{(s)} \oplus P_2^{(s)} \oplus \tau P_1^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$
$\oplus \tau h_\tau(T^{(s)}, P_3^{(s)})$;
$\rho(F_2^{(s)}) \leftarrow Y_2^{(s)}$;
$B_2^{(s)} \leftarrow F_1^{(s)} \oplus Y_2^{(s)}$;
$C_1^{(s)} \leftarrow \tau B_2^{(s)} \oplus F_2^{(s)}$;
$Z^{(s)} \leftarrow F_1^{(s)} \oplus F_2^{(s)}$;
for $i = 1$ to $m^{(s)} - 3$ do
$J_i^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(i)$; $\mathsf{ChkDom}(J_i^{(s)})$;
$\rho(J_i^{(s)}) \leftarrow C_{3,i}^{(s)} \oplus P_{3,i}^{(s)}$;
end for;
$J_{m^{(s)}-2}^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(m^{(s)} - 2)$;
$\mathsf{ChkDom}(J_{m^{(s)}-2}^{(s)})$;
$D^{(s)} \leftarrow W^{(s)} \oplus (P_{3,m^{(s)}-2}^{(s)} \| 0^{n-r^{(s)}})$;
$\rho(J_{m^{(s)}-2}^{(s)}) \leftarrow D^{(s)}$;
return $(C_1^{(s)} \| C_2^{(s)} \| C_3^{(s)})$.</td>
<td><u>$\mathsf{ty}^{(s)} = \mathsf{dec}$: input $(T^{(s)}, C^{(s)})$</u>
$(C_1^{(s)}, C_2^{(s)}, C_3^{(s)}) \leftarrow \mathsf{parse}_n(C^{(s)})$;
$P_1^{(s)} \xleftarrow{\$} \{0,1\}^n$; $P_2^{(s)} \xleftarrow{\$} \{0,1\}^n$;
for $i = 1, \ldots, m^{(s)} - 3$ do $P_{3,i}^{(s)} \xleftarrow{\$} \{0,1\}^n$;
$V^{(s)} \xleftarrow{\$} \{0,1\}^n$; $P_{3,m^{(s)}-2} \leftarrow \mathsf{first}_{r^{(s)}}(V^{(s)})$;
$B_2^{(s)} \leftarrow C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$;
$F_2^{(s)} \leftarrow C_1^{(s)} \oplus \tau B_2^{(s)}$; $\mathsf{ChkDom}(F_2^{(s)})$;
$Y_2^{(s)} \leftarrow C_2^{(s)} \oplus P_2^{(s)} \oplus \tau P_1^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$
$\oplus \tau h_\tau(T^{(s)}, P_3^{(s)})$;
$\rho(F_2^{(s)}) \leftarrow Y_2^{(s)}$;
$F_1^{(s)} \leftarrow B_2^{(s)} \oplus Y_2^{(s)}$; $\mathsf{ChkDom}(F_1^{(s)})$;
$Y_1^{(s)} \leftarrow C_1^{(s)} \oplus P_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}))$
$\oplus h_\tau(T^{(s)}, P_3^{(s)})$;
$\rho(F_1^{(s)}) \leftarrow Y_1^{(s)}$;
$A_1^{(s)} \leftarrow F_2^{(s)} \oplus Y_1^{(s)}$;
$P_2^{(s)} \leftarrow \tau A_1^{(s)} \oplus F_1^{(s)}$;
$Z^{(s)} \leftarrow F_1^{(s)} \oplus F_2^{(s)}$;
for $i = 1$ to $m^{(s)} - 3$ do
$J_i^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(i)$; $\mathsf{ChkDom}(J_i^{(s)})$;
$\rho(J_i^{(s)}) \leftarrow C_{3,i}^{(s)} \oplus P_{3,i}^{(s)}$;
end for;
$J_{m^{(s)}-2}^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(m^{(s)} - 2)$;
$\mathsf{ChkDom}(J_{m^{(s)}-2}^{(s)})$;
$E^{(s)} \leftarrow V^{(s)} \oplus (C_{3,m^{(s)}-2}^{(s)} \| 0^{n-r^{(s)}})$;
$\rho(J_{m^{(s)}-2}^{(s)}) \leftarrow E^{(s)}$;
return $(P_1^{(s)} \| P_2^{(s)} \| P_3^{(s)})$.</td></tr>
</table>

and uniformly distributed. So, as in game G_{real}, in game G_{int} also the outputs of ρ are independent and uniformly distributed.

Let $\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})$ be the event that the variable bad is set to true in the game G_{int}. Let $\mathcal{A}^{G_{\mathsf{int}}} \Rightarrow 1$ denote the event that $\mathcal{A}$ outputs 1 in the game G_{int}. From the description of the games, it follows that if bad does not occur, then $\mathcal{A}$'s views in both G_{real} and G_{int} are the same. Also, the probabilities that bad occurs in the two games are equal. This gives the following.

Claim 1

$$\Pr\left[\left(\mathcal{A}^{G_{\mathsf{real}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})}\right] = \Pr\left[\left(\mathcal{A}^{G_{\mathsf{int}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})}\right];$$
$$\Pr\left[\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})\right] = \Pr\left[\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})\right].$$

Next, the game G_{int} is changed to the game G_{rnd} which is shown in Table 6. In this game, there is no ρ. For an enciphering query, the ciphertext is chosen independently and uniformly at random and

Table 6: Game G_{rnd}

Respond to the s^{th} adversary query as follows:

if $\mathsf{ty}^{(s)} = \mathsf{enc}$; $C_1^{(s)}||C_2^{(s)}||C_{3,1}^{(s)}||\ldots||C_{3,m^{(s)}-3}^{(s)}||D^{(s)} \xleftarrow{\$} \{0,1\}^{nm^{(s)}}$;

$C_{3,m^{(s)}-2}^{(s)} \leftarrow \mathsf{first}_{r^{(s)}}(D^{(s)})$ **return** $C^{(s)} = C_1^{(s)}||C_2^{(s)}||C_{3,1}^{(s)}||\ldots||C_{3,m^{(s)}-2}^{(s)}$;

if $\mathsf{ty}^{(s)} = \mathsf{dec}$; $P_1^{(s)}||P_2^{(s)}||P_{3,1}^{(s)}||\ldots||P_{3,m^{(s)}-3}^{(s)}||E^{(s)} \xleftarrow{\$} \{0,1\}^{nm^{(s)}}$;

$P_{3,m^{(s)}-2}^{(s)} \leftarrow \mathsf{first}_{r^{(s)}}(E^{(s)})$ **return** $P^{(s)} = P_1^{(s)}||P_2^{(s)}||P_{3,1}^{(s)}||\ldots||P_{3,m^{(s)}-2}^{(s)}$;

Finalisation:

$\mathcal{D} \leftarrow \{\mathsf{fStr}\}$; $\mathsf{bad} \leftarrow \mathsf{false}$; $\tau \xleftarrow{\$} \{0,1\}^n$;
for $s = 1$ to q do
  $A_1^{(s)} \leftarrow P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})$;
  $B_2^{(s)} \leftarrow C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})$;
  $F_1^{(s)} \leftarrow \tau A_1^{(s)} \oplus P_2^{(s)} = \tau(P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})) \oplus P_2^{(s)}$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{F_1^{(s)}\}$;
  $F_2^{(s)} \leftarrow C_1^{(s)} \oplus \tau B_2^{(s)} = C_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}))$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{F_2^{(s)}\}$;
  $Z^{(s)} \leftarrow F_1^{(s)} \oplus F_2^{(s)}$;
  for $i = 1$ to $m^{(s)} - 2$;
    $J_i^{(s)} \leftarrow Z^{(s)} \oplus \mathsf{bin}_n(i) = F_1^{(s)} \oplus F_2^{(s)} \oplus \mathsf{bin}_n(i)$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{J_i^{(s)}\}$;
  end for;
end for;
if (some value occurs more than once in $\mathcal{D}$) then $\mathsf{bad} \leftarrow \mathsf{true}$ endif;

for a deciphering query, the plaintext is chosen independently and uniformly at random. These are returned to $\mathcal{A}$. After the interaction is over, in the finalisation step, the internal random variables are included in $\mathcal{D}$ and bad is set to true if there is a collision in $\mathcal{D}$. Let $\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})$ be the event that the variable bad is set to true in the game G_{rnd}. Let $\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1$ denote the event that $\mathcal{A}$ outputs 1 in the game G_{rnd}. If bad does not occur, then in both the games G_{int} and G_{rnd}, $\mathcal{A}$ obtains independent and uniform random strings as responses to all its queries. Also, the probabilities that bad occurs in the two games are equal. So, we have the following.

Claim 2

$$\Pr\left[\left(\mathcal{A}^{G_{\mathsf{int}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})}\right] = \Pr\left[\left(\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})}\right];$$
$$\Pr\left[\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})\right] = \Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})\right].$$

Note that the event $\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1$ is exactly the event $\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1$.

Using (40) along with Claims 1 and 2, we have the following.

$$\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathcal{A}) \\
&= \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right] + \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) \\
&= \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) + \Pr\left[\mathcal{A}^{G_{\mathsf{real}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1\right] \\
&\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) + \Pr\left[\left(\mathcal{A}^{G_{\mathsf{real}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})}\right] + \Pr\left[\mathsf{Bad}_{\mathsf{real}}(\mathcal{A})\right] - \Pr\left[\left(\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})}\right] \\
&= \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) + \Pr\left[\left(\mathcal{A}^{G_{\mathsf{int}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})}\right] + \Pr\left[\mathsf{Bad}_{\mathsf{int}}(\mathcal{A})\right] - \Pr\left[\left(\mathcal{A}^{G_{\mathsf{rnd}}} \Rightarrow 1\right) \wedge \overline{\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})}\right] \\
&= \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) + \Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})\right]. \qquad (42)
\end{aligned}$$

Adversary $\mathcal{A}$ runs in time $\mathfrak{T}$. We instead consider an adversary $\mathcal{C}$ which is allowed unbounded runtime and also unbounded memory. Consider the interaction of $\mathcal{C}$ with the oracle in the game G_{rnd} and define the event $\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right]$ in a manner analogous to $\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})\right]$. Clearly, we have

$$\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{A})\right] \leq \Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right]. \qquad (43)$$

So, it is sufficient to upper bound $\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right]$. Since $\mathcal{C}$ has unbounded computational power, without loss of generality, we may assume that $\mathcal{C}$ is deterministic.

Upper bound on $\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right]$**:** An upper bound on $\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right]$ is obtained by showing that in the game G_{rnd} the event that two random variables in $\mathcal{D}$ are equal occurs with low probability. The main crux of the whole proof is to argue this in the various cases that arise in considering the different pairs of random variables from $\mathcal{D}$. The claims below tackle all the different cases that can arise.

Claim 3 *For* $1 \leq s \leq q$, $\Pr[F_1^{(s)} = \mathsf{fStr}] \leq \epsilon_1^{(s)}$.

Proof.

$$\begin{aligned}
\Pr[F_1^{(s)} = \mathsf{fStr}] &= \Pr[\tau P_1^{(s)} \oplus \tau h_\tau(T^{(s)}, P_3^{(s)}) \oplus P_2^{(s)} = \mathsf{fStr}] \\
&= \Pr[\tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus P_1^{(s)}) = P_2^{(s)} \oplus \mathsf{fStr}] \\
&\leq \epsilon_1^{(s)}.
\end{aligned}$$

The last inequality follows from (9). □

Claim 4 *For* $1 \le s \le q$, $\Pr[F_2^{(s)} = \mathsf{fStr}] \le \epsilon_1^{(s)}$.

Proof.

$$\begin{aligned}\Pr[F_2^{(s)} = \mathsf{fStr}] &= \Pr[\tau C_2^{(s)} \oplus \tau h'_\tau(T^{(s)}, C_3^{(s)}) \oplus C_1^{(s)} = \mathsf{fStr}] \\ &= \Pr[\tau(h'_\tau(T^{(s)}, C_3^{(s)}) \oplus C_2^{(s)}) = C_1^{(s)} \oplus \mathsf{fStr}] \\ &\le \epsilon_1^{(s)}.\end{aligned}$$

The last inequality follows from (10). □

Claim 5 *For* $1 \le s \le q$, $1 \le i \le m^{(s)} - 2$, $\Pr[J_i^{(s)} = \mathsf{fStr}] = 1/2^n$.

Proof.

$$\begin{aligned}J_i^{(s)} \oplus \mathsf{fStr} &= Z^{(s)} \oplus \mathsf{bin}_n(i) \oplus \mathsf{fStr} \\ &= F_1^{(s)} \oplus F_2^{(s)} \oplus \mathsf{bin}_n(i) \oplus \mathsf{fStr} \\ &= \tau(P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})) \oplus P_2^{(s)} \oplus C_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})) \\ &\quad \oplus \mathsf{bin}_n(i) \oplus \mathsf{fStr}.\end{aligned}$$

When $\mathsf{ty}^{(s)} = \mathsf{enc}$, then $C_1^{(s)}$ is an n-bit uniform random string which is independent of the other quantities and when $\mathsf{ty}^{(s)} = \mathsf{dec}$, then $P_2^{(s)}$ is an n-bit uniform random string which is independent of the other quantities. So in both cases we have the required probability. □

Claim 6 *For* $s \ne t$, $\Pr[F_1^{(s)} = F_1^{(t)}] \le \max\{\epsilon_2^{(s,t)}, 1/2^n\}$.

Proof.

$$F_1^{(s)} \oplus F_1^{(t)} = \tau(P_1^{(s)} \oplus P_1^{(t)}) \oplus \tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus h_\tau(T^{(t)}, P_3^{(t)})) \oplus P_2^{(s)} \oplus P_2^{(t)}.$$

There are four cases to consider.

Case 1: $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{enc}$. There are two sub-cases.

(a) **Case 1a:** $(T^{(s)}, P_1^{(s)}, P_3^{(s)}) = (T^{(t)}, P_1^{(t)}, P_3^{(t)})$.
As the adversary is not allowed to repeat a query, hence $(T^{(s)}, P_1^{(s)}, P_3^{(s)}) = (T^{(t)}, P_1^{(t)}, P_3^{(t)})$ implies $P_2^{(s)} \ne P_2^{(t)}$ and so $\Pr[F_1^{(s)} = F_1^{(t)}] = 0$.

(b) **Case 1b:** $(T^{(s)}, P_1^{(s)}, P_3^{(s)}) \ne (T^{(t)}, P_1^{(t)}, P_3^{(t)})$.
If $(T^{(s)}, P_3^{(s)}) = (T^{(t)}, P_3^{(t)})$, then $P_1^{(s)} \ne P_1^{(t)}$ and so $F_1^{(s)} \oplus F_1^{(t)} = \tau(P_1^{(s)} \oplus P_1^{(t)}) \oplus P_2^{(s)} \oplus P_2^{(t)}$ is a non-zero polynomial in τ of degree 1. Thus, $\Pr[F_1^{(s)} = F_1^{(t)}] = 1/2^n$.
If $(T^{(s)}, P_3^{(s)}) \ne (T^{(t)}, P_3^{(t)})$, then

$$\begin{aligned}&\Pr[F_1^{(s)} = F_1^{(t)}] \\ &= \Pr[\tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus h_\tau(T^{(t)}, P_3^{(t)}) \oplus P_1^{(s)} \oplus P_1^{(t)}) = P_2^{(s)} \oplus P_2^{(t)}] \\ &\le \epsilon_2^{(s,t)}.\end{aligned}$$

The last inequality follows from (11).

Case 2: $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{dec}$. In this case all of $P_1^{(s)}, P_1^{(t)}, P_2^{(s)}, P_2^{(t)}$ are independent and uniformly distributed n-bit strings and so $\Pr[F_1^{(s)} = F_1^{(t)}] = 1/2^n$.
Case 3: $\mathsf{ty}^{(s)} = \mathsf{enc}$ and $\mathsf{ty}^{(t)} = \mathsf{dec}$. In this case $P_1^{(t)}$ and $P_2^{(t)}$ are independent and uniformly distributed n-bit strings and so $\Pr[F_1^{(s)} = F_1^{(t)}] = 1/2^n$.
Case 4: $\mathsf{ty}^{(s)} = \mathsf{dec}$ and $\mathsf{ty}^{(t)} = \mathsf{enc}$. In this case $P_1^{(s)}$ and $P_2^{(s)}$ are independent and uniformly distributed n-bit strings and so $\Pr[F_1^{(s)} = F_1^{(t)}] = 1/2^n$.

□

Claim 7 *For $s \neq t$, $\Pr[F_2^{(s)} = F_2^{(t)}] \leq \max\{\epsilon_2^{(s,t)}, 1/2^n\}$.*

The proof is almost the same as the proof of Claim 6.

Claim 8 *For $1 \leq s, t \leq q$, $\Pr[F_1^{(s)} = F_2^{(t)}] \leq \max\{\epsilon_2^{(s,t)}, 1/2^n\}$.*

Proof.

$$F_1^{(s)} \oplus F_2^{(t)} = \tau(P_1^{(s)} \oplus C_2^{(t)}) \oplus \tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus h'_\tau(T^{(t)}, C_3^{(t)})) \oplus (P_2^{(s)} \oplus C_1^{(t)}).$$

There are four cases.

Case 1: $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{enc}$. In this case, $C_1^{(t)}$ is an independent and uniform random n-bit string and so $\Pr[F_1^{(s)} = F_2^{(t)}] = 1/2^n$.
Case 2: $\mathsf{ty}^{(s)} = \mathsf{enc}$ and $\mathsf{ty}^{(t)} = \mathsf{dec}$. We have

$$\begin{aligned}
&\Pr[F_1^{(s)} = F_2^{(t)}] \\
&= \Pr[\tau(P_1^{(s)} \oplus C_2^{(t)}) \oplus \tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus h'_\tau(T^{(t)}, C_3^{(t)})) = P_2^{(s)} \oplus C_1^{(t)}] \\
&= \Pr[\tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus h'_\tau(T^{(t)}, C_3^{(t)}) \oplus P_1^{(s)} \oplus C_2^{(t)}) = P_2^{(s)} \oplus C_1^{(t)}] \\
&\leq \epsilon_2^{(s,t)}.
\end{aligned}$$

The last inequality follows from (13).
Case 3: $\mathsf{ty}^{(s)} = \mathsf{dec}$ and $\mathsf{ty}^{(t)} = \mathsf{enc}$. In this case, $P_2^{(s)}$ is an independent and uniform random n-bit string and so $\Pr[F_1^{(s)} = F_2^{(t)}] = 1/2^n$.
Case 4: $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{dec}$. In this case also, $P_2^{(s)}$ is an independent and uniform random n-bit string and so $\Pr[F_1^{(s)} = F_2^{(t)}] = 1/2^n$.

□

Claim 9 *For $1 \leq s, t \leq q$ and $1 \leq i \leq m^{(t)} - 2$, $\Pr[F_1^{(s)} = J_i^{(t)}] \leq \epsilon_1^{(s)}$.*

Proof.

$$\begin{aligned}
\Pr[F_1^{(s)} = J_i^{(t)}] &= \Pr[F_1^{(s)} = F_1^{(t)} \oplus F_2^{(t)} \oplus \mathsf{bin}_n(i)] \\
&= \Pr[\tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus P_1^{(s)}) \oplus P_2^{(s)} \\
&\qquad = \tau(P_1^{(t)} \oplus h_\tau(T^{(t)}, P_3^{(t)})) \oplus P_2^{(t)} \\
&\qquad\quad \oplus C_1^{(t)} \oplus \tau(C_2^{(t)} \oplus h'_\tau(T^{(t)}, C_3^{(t)})) \oplus \mathsf{bin}_n(i)].
\end{aligned}$$

First suppose that $s \neq t$. If $\mathsf{ty}^{(t)} = \mathsf{enc}$, then $C_1^{(t)}$ is a uniform n-bit string which is independent of the other quantities and if $\mathsf{ty}^{(t)} = \mathsf{dec}$, then $P_2^{(t)}$ is a uniform n-bit string which is independent of the other quantities. In both cases, the above probability is $1/2^n$.

So, suppose that $s = t$. Then the required probability reduces to

$$\begin{aligned}
\Pr[F_1^{(s)} = J_i^{(s)}] &= \Pr[F_2^{(s)} = \mathsf{bin}_n(i)] \\
&= \Pr[\tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})) = C_1^{(s)} \oplus \mathsf{bin}_n(i)] \\
&\leq \epsilon_1^{(s)}.
\end{aligned}$$

If $\mathsf{ty}^{(s)} = \mathsf{enc}$, then $C_1^{(s)}$ is a uniform n-bit string which is independent of the other quantities and so the probability is equal to $1/2^n$; on the other hand, if $\mathsf{ty}^{(s)} = \mathsf{dec}$, then the last inequality follows from (10). □

Claim 10 *For $1 \leq s, t \leq q$ and $1 \leq i \leq m^{(t)} - 2$, $\Pr[F_2^{(s)} = J_i^{(t)}] \leq \epsilon_1^{(s)}$.*

The proof is almost the same as the proof of Claim 9.

Claim 11 *For $1 \leq s, t \leq q$, $1 \leq i \leq m^{(s)} - 2$, $1 \leq j \leq m^{(t)} - 2$ and $(s, i) \neq (t, j)$, $\Pr[J_i^{(s)} = J_j^{(t)}] \leq 1/2^n$.*

Proof.

$$\begin{aligned}
J_i^{(s)} \oplus J_j^{(t)} &= F_1^{(s)} \oplus F_2^{(s)} \oplus F_1^{(t)} \oplus F_2^{(t)} \oplus \mathsf{bin}_n(i) \oplus \mathsf{bin}_n(j) \\
&= \tau(P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})) \oplus P_2^{(s)} \oplus C_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)})) \\
&\quad \oplus \tau(P_1^{(t)} \oplus h_\tau(T^{(t)}, P_3^{(t)})) \oplus P_2^{(t)} \oplus C_1^{(t)} \oplus \tau(C_2^{(t)} \oplus h'_\tau(T^{(t)}, C_3^{(t)})) \\
&\quad \oplus \mathsf{bin}_n(i) \oplus \mathsf{bin}_n(j).
\end{aligned}$$

If $s = t$, then $i \neq j$ and so $\Pr[J_i^{(s)} = J_j^{(t)}] = \Pr[\mathsf{bin}_n(i) = \mathsf{bin}_n(j)] = 0$. Suppose that $s \neq t$. There are four cases to consider.

– If $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{enc}$, then both $C_1^{(s)}$ and $C_1^{(t)}$ are independent and uniform random strings which are independent of the other quantities.
– If $\mathsf{ty}^{(s)} = \mathsf{ty}^{(t)} = \mathsf{dec}$, then both $P_2^{(s)}$ and $P_2^{(t)}$ are independent and uniform random strings which are independent of the other quantities.
– If $\mathsf{ty}^{(s)} = \mathsf{enc}$ and $\mathsf{ty}^{(t)} = \mathsf{dec}$, then both $C_1^{(s)}$ and $P_2^{(t)}$ are independent and uniform random strings which are independent of the other quantities.
– If $\mathsf{ty}^{(s)} = \mathsf{dec}$ and $\mathsf{ty}^{(t)} = \mathsf{enc}$, then both $P_2^{(s)}$ and $C_1^{(t)}$ are independent and uniform random strings which are independent of the other quantities.

From the above it follows that if $s \neq t$, then in all cases the probability is equal to $1/2^n$. Thus, the claim follows. □

By Claims 3 to 11 and the union bound we have

$$\begin{aligned}
&\Pr\left[\mathsf{Bad}_{\mathsf{rnd}}(\mathcal{C})\right] \\
&\leq 2\sum_{s=1}^{q} \epsilon_1^{(s)} + \sum_{s=1}^{q} \left(\frac{m^{(s)} - 2}{2^n}\right) + \sum_{1 \leq s < t \leq q} 2\left(\epsilon_2^{(s,t)} + \frac{1}{2^n}\right)
\end{aligned}$$

$$
\begin{aligned}
&+\sum_{1\le s\le t\le q}\left(\epsilon_2^{(s,t)}+\frac{1}{2^n}\right)+2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)\left(\sum_{t=1}^{q}(m^{(t)}-2)\right)+\frac{1}{2^n}\binom{\sum_{s=1}^{q}(m^{(s)}-2)}{2}\\
&=2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)\left(1+\sum_{t=1}^{q}(m^{(t)}-2)\right)+\frac{3}{2^n}\frac{q(q-1)}{2}+\frac{q}{2^n}+3\sum_{1\le s<t\le q}\epsilon_2^{(s,t)}+\sum_{s=1}^{q}\epsilon_2^{(s,s)}\\
&\quad+\sum_{s=1}^{q}\left(\frac{m^{(s)}-2}{2^n}\right)+\frac{1}{2^n}\binom{\sum_{s=1}^{q}(m^{(s)}-2)}{2}\\
&\le 2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)(1+\omega-2q)+\frac{2q^2}{2^n}+3\sum_{1\le s<t\le q}\epsilon_2^{(s,t)}+\sum_{s=1}^{q}\epsilon_2^{(s,s)}\\
&\quad+\frac{\omega-2q}{2^n}+\frac{1}{2^n}\frac{\omega(\omega-1)}{2}\\
&\le 2\omega\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)+3\sum_{1\le s<t\le q}\epsilon_2^{(s,t)}+\sum_{s=1}^{q}\epsilon_2^{(s,s)}+\frac{3\omega^2}{2^n}. \qquad (44)
\end{aligned}
$$

Putting together (42), (43) and (44) we obtain

$$
\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathcal{A})\le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B})+2\omega\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)+3\sum_{1\le s<t\le q}\epsilon_2^{(s,t)}+\sum_{s=1}^{q}\epsilon_2^{(s,s)}+\frac{3\omega^2}{2^n}.
$$

The relations between the resources of $\mathcal{A}$ and $\mathcal{B}$ have been stated earlier. Maximising the left hand side on the resources shows the required result and completes the proof of Theorem 1. □

We have the following consequences of Theorem 1 for the specific instantiations of FAST.

Corollary 1. *Let $m\ge 3$ be a fixed positive integer. Let q and $\sigma\ge q$ be positive integers and $\omega=\sigma-q$. Consider the instantiations $\mathsf{FAST}[\mathsf{Fx}_m,\mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_m,\mathsf{BRW}]$ of FAST. Then for all $\mathfrak{T}>0$,*

$$
\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m,\mathsf{Horner}]}(\mathfrak{T},q,q,\omega)\\
&\le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}',\omega+1,\omega+1)+\frac{1}{2^n}\left(5\omega^2+\omega+\frac{11\omega q}{2}+3q^2+2q\right). \qquad (45)
\end{aligned}
$$

If $m\ge 4$, then for all $\mathfrak{T}>0$,

$$
\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m,\mathsf{BRW}]}(\mathfrak{T},q,q,\omega)\\
&\le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}',\omega+1,\omega+1)\\
&\quad+\frac{1}{2^n}\left(6\omega q+\frac{9q^2}{2}+3q+3\omega^2+\mathfrak{d}(m-1)\left(2\omega q+\frac{3q^2}{2}+q\right)\right)\\
&\le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}',\omega+1,\omega+1)+\frac{1}{2^n}\left(7\omega^2+3\omega q+2\omega\right). \qquad (46)
\end{aligned}
$$

Further, if $m\ge 4$ is a power of two, then for all $\mathfrak{T}>0$,

$$
\begin{aligned}
&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m,\mathsf{BRW}]}(\mathfrak{T},q,q,\omega)\\
&\le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}',\omega+1,\omega+1)+\frac{1}{2^n}\left(5\omega^2+\omega+\frac{11\omega q}{2}+3q^2+2q\right). \qquad (47)
\end{aligned}
$$

Proof. Let $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}])$ and $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}])$ denote the expressions for Δ in (36) when $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ are respectively used.

In the setting of Fx_m, each query consists of a single n-bit block for the tweak and m n-bit blocks for the plaintext or the ciphertext, i.e., $m^{(s)} = m$ for all $1 \leq s \leq q$. So, $\theta = q$, $\omega = qm$ and $\sigma = \theta + \omega = q(m+1)$. Also, $m \geq 3$ and $q \geq 1$ imply that $q < \omega$ and $m \leq \omega$.

From Proposition 2, we have $\epsilon_1^{(s)} = \epsilon_2^{(s,t)} = (m+2)/2^n$ for $1 \leq s, t \leq q$. Using these, $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}])$ can be upper bounded as given in (45).

From Proposition 3, we have $\epsilon_1^{(s)} = \epsilon_2^{(s,t)} = (3 + \mathfrak{d}(m-1))/2^n$ for $1 \leq s, t \leq q$. Using these, $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}])$ achieves the first bound and using $\mathfrak{d}(m-1) \leq 2(m-1) - 1$ yields the second bound.

In the case where $m \geq 4$ is a power of two, then $\mathfrak{d}(m-1) = m-1$. Also, from Proposition 3, we have $\epsilon_1^{(s)} = \epsilon_2^{(s,t)} = (m+2)/2^n$ and so $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]) = \Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}])$ which shows the required statement. $\square$

Corollary 2. *Let q, $\omega \geq q$ be positive integers and θ be a non-negative integer. Consider the instantiations* $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ *and* $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$ *of* FAST. *Then for all $\mathfrak{T} > 0$,*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Gn},\mathfrak{k},\mathsf{vecHorner}]}(\mathfrak{T}, q, \theta, \omega) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega+1, \omega+1) + \frac{5\omega^2 + 2\omega\theta + \omega + \theta}{2^n} + \frac{3q(\theta+\omega) + (\mathfrak{k}+2)((2\omega+1)q + 3q^2) + 6q^2}{2^n}; \tag{48}$$

$$\begin{aligned} &\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Gn},\mathfrak{k},\eta,\mathsf{vecHash2L}]}(\mathfrak{T}, q, \theta, \omega) \\ &\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega+1, \omega+1) \\ &\quad + \frac{(3 + 2(\eta+1)/\eta)\omega^2 + (2(\eta+1)/\eta)\omega\theta + ((\eta+1)/\eta)(\omega+\theta)}{2^n} \\ &\quad + \frac{3q((\eta+1)/\eta)(\theta+\omega) + ((2\omega+1)q + 3q^2)(\mathfrak{k}+1)(\eta+2) + 3q(2\omega+1) + 9q^2}{2^n}. \end{aligned} \tag{49}$$

Proof. Let $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}])$ and $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}])$ denote the expressions for Δ in (36) when $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ and $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$ are respectively used.

First consider $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$. From Proposition 4, $\epsilon_1^{(s)} = (\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)} + \mathfrak{k} + 4)/2^n$ and $\epsilon_2^{(s,t)} = (\max(\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)}, \mathfrak{t}^{(t)} + \mathfrak{l}^{(t)}) + \mathfrak{k} + 4)/2^n$. So, $\epsilon_2^{(s,s)} = \epsilon_1^{(s)}$. We have

$$\sum_{s=1}^{q} \epsilon_1^{(s)} = \sum_{s=1}^{q} \frac{\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)} + \mathfrak{k} + 4}{2^n} = \frac{\theta+\omega}{2^n} + \frac{q(\mathfrak{k}+2)}{2^n};$$

$$\sum_{1 \leq s < t \leq q} \epsilon_2^{(s,t)} \leq \sum_{1 \leq s < t \leq q} \frac{\max(\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)}, \mathfrak{t}^{(t)} + \mathfrak{l}^{(t)}) + \mathfrak{k} + 4}{2^n} \leq \frac{q(\theta+\omega)}{2^n} + \frac{q^2(\mathfrak{k}+4)}{2^n}.$$

Using these in Theorem 1, we obtain

$$\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]) \leq \frac{5\omega^2 + 2\omega\theta + \omega + \theta}{2^n} + \frac{3q(\theta+\omega) + (\mathfrak{k}+2)((2\omega+1)q + 3q^2) + 6q^2}{2^n}.$$

Now consider $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$. From Proposition 5,

$$\epsilon_1^{(s)} = \frac{((\eta+1)/\eta)(\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)}) + (\mathfrak{k}+1)(\eta+2) + 3}{2^n};$$
$$\epsilon_2^{(s,t)} = \frac{((\eta+1)/\eta)\max(\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)}, \mathfrak{t}^{(t)} + \mathfrak{l}^{(t)}) + (\mathfrak{k}+1)(\eta+2) + 3}{2^n}.$$

So, $\epsilon_2^{(s,s)} = \epsilon_1^{(s)}$. We have

$$\sum_{s=1}^{q} \epsilon_1^{(s)} \leq \frac{((\eta+1)/\eta)(\theta+\omega) + q(\mathfrak{k}+1)(\eta+2) + 3q}{2^n};$$
$$\sum_{1\leq s<t\leq q} \epsilon_2^{(s,t)} \leq \frac{q((\eta+1)/\eta)(\theta+\omega) + q^2(\mathfrak{k}+1)(\eta+2) + 3q^2}{2^n};$$

Using these in Theorem 1, we obtain

$$\begin{aligned}
&\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]) \\
&\leq \frac{(3+2(\eta+1)/\eta)\omega^2 + (2(\eta+1)/\eta)\omega\theta + ((\eta+1)/\eta)(\omega+\theta)}{2^n} \\
&\quad + \frac{3q((\eta+1)/\eta)(\theta+\omega) + ((2\omega+1)q + 3q^2)(\mathfrak{k}+1)(\eta+2) + 3q(2\omega+1) + 9q^2}{2^n}.
\end{aligned}$$

□

Some more explanations along with illustrations of the results obtained in these Corollaries are in Appendix A.

6 Comparison

This section provides a comparison of the design features of FAST with previously proposed TESs. As mentioned earlier, an important application of a TES is disk encryption. The literature, on the other hand, contains a number of proposals for disk encryption which do not provide provable security as TESs; our comparison does not include such schemes.

Several block cipher based TES constructions essentially use a layer of encryption using a mode of operation of the block cipher sandwiched between two layers of hashing. Differences arise in the choice of the mode of operation, the choice of the hash functions and other details.

1. For the mode of operation, the electronic codebook mode (ECB) has been suggested in TET [20] and HEH [35] while some form of the counter mode of operation has been used in XCB [26, 27], HCTR [37] and HCH [13]. In this paper, we use the counter mode of operation as described in the scheme HCTR [37].
2. For the hash functions, XCB, HCTR and HCH essentially use polynomial hashing based on Horner's rule. The cost of hashing in TET is higher. BRW based hashing has been suggested for HEH and implemented in hardware for fixed length messages [9].

All of the above mentioned TESs require both the encryption and the decryption functions of the block cipher. The possibility of using only the encryption function of a block cipher to build a TES has been reported [36] and for the convenience of description let us denote this scheme by TESX.

Table 7: Comparison of different tweakable enciphering schemes according to computational efficiency. [BC] denotes the number of block cipher calls; [M] denotes the number of field multiplications; [D] denotes the number of doubling ('multiplication by α') operations;

type	scheme	[BC]	[M]	[D]
enc-mix-enc	CMC [22]	$2m+1$	–	–
	EME2* [19]	$2m+1+m/n$	–	2
	AEZ [24]	$(5m+4)/2$	–	$\frac{m-2}{4}$
	FMix [6]	$2m+1$	–	–
hash-enc-hash	XCB [26]	$m+1$	$2(m+3)$	–
	HCTR [37]	m	$2(m+1)$	–
	HCHfp [13]	$m+2$	$2(m-1)$	–
	TET [20]	$m+1$	$2m$	$2(m-1)$
	HEH-BRW[35]	$m+1$	$2+2\lfloor(m-1)/2\rfloor$	$2(m-1)$
	TESX with BRW [36]	$m+1$	$4+2\lfloor(m-1)/2\rfloor$	$2(m-1)$
	FAST[Fx_m, Horner]	$m+1$	$2m+1$	–
	FAST[Fx_m, BRW]	$m+1$	$2+2\lfloor(m-1)/2\rfloor$	–

The present work is based upon the idea behind TESX. In terms of similarity, both TESX and the present work use a Feistel layer on the first two blocks and a counter mode of operation on the rest of the blocks. There are several differences in the two constructions.

1. For TESX, inside the Feistel layer, the hash function h is used to process A_1 and B_2. The key for this hash function is τ' which is independent of the key τ which is used for the hash function outside the Feistel layer. In contrast, FAST is organised such that the Feistel layer consists of only two encryption rounds and the entire hashing using a single key τ takes place outside the Feistel layer. In summary, TESX uses two hash keys while FAST uses a single hash key.
2. For TESX, the hash of P_3 is masked with β_1 and XORed to both P_1 and P_2 and the hash of C_3 is masked with β_2 and XORed to both C_1 and C_2. FAST does away with the maskings with β_1 and β_2; the hash of P_3 is XORed to only P_1; and the hash of C_3 is XORed to only C_2.
3. For TESX, the seed to the counter mode is generated as $A_1 \oplus P_2 \oplus C_1 \oplus B_2$. FAST generates the seed to the counter mode as $F_1 \oplus F_2$.
4. The counter mode suggested in TESX requires a doubling operation for each block. The counter mode used in FAST is given by (1) and is based on HCTR [37]; this counter mode does not require the doubling operation.

Some of the above differences such as reducing the hash key size and avoiding doubling operations are important from a practical point of view while the others are simplifications obtained by removing unnecessary operations. Keeping the similarities and the differences in mind, it would be proper to view the present work as a fine-tuned version of TESX. This fine tuning is required from an engineering point of view where the goal is to obtain an efficient and clean design. More importantly, this work presents detailed software implementations of different instantiations of FAST and thereby actually demonstrates the advantages of the new proposal in comparison to the previous works.

Another class of block cipher based TESs such as CMC [22], EME [23, 19] (the scheme EME2 is essentially EME* [19]), AEZ [24] and FMix [6] essentially uses several layers of encryption using a mode of operation of a block cipher. CMC, EME and FMix use two layers of encryption whereas AEZ uses three layers of encryption. These TESs do not use any hash function. Out of these CMC and FMix are sequential while EME and AEZ are parallelisable. AEZ and FMix require only the encryption function of the underlying block cipher whereas CMC and EME require both the encryption and the decryption functions of the block cipher. The costs of encryption for CMC, EME and FMix are roughly two block cipher calls per block of the message and for AEZ the cost is roughly two-and-half block cipher calls per block of the message. CMC and FMix do not use any doubling operation while both EME and AEZ use doubling operations. Since AEZ and EME are parallelisable while CMC and FMix are not, any reasonable implementations of AEZ and EME will be faster than both CMC and FMix. Later we provide software implementation results which show the superiority of FAST in comparison to both AEZ and EME and hence also imply the superiority of FAST over CMC and FMix.

Table 8: Comparison of different tweakable enciphering schemes according to practical and implementation simplicity. [BCK] denotes the number of block cipher keys; and [HK] denotes the number of blocks in the hash key.

type	scheme	[BCK]	[HK]	dec module	parallel
enc-mix-enc	CMC [22]	1	–	reqd	no
	EME2* [19]	1	2	reqd	yes
	AEZ [24]	1	2	not reqd	yes
	FMix [6]	1	–	not reqd	no
hash-enc-hash	XCB [26]	3	2	reqd	yes
	HCTR [37]	1	1	reqd	yes
	HCHfp [13]	1	1	reqd	yes
	TET [20]	2	3	reqd	yes
	HEH-BRW[35]	1	1	reqd	yes
	TESX with BRW [36]	1	2	not reqd	yes
	FAST[Fx_m, Horner]	1	–	not reqd	yes
	FAST[Fx_m, BRW]	1	–	not reqd	yes

Note: for both Tables 7 and 8, the block size is n bits, the tweak is a single n-bit block and the number of blocks $m \geq 3$ in the message is fixed.

Table 7 and 8 compares FAST to previously proposed schemes. From the viewpoint of efficiency, it is preferable to have schemes which are parallelisable. This would eliminate CMC and FMix from the comparison. Further, again from an efficiency point of view it would be preferable to have schemes which use only the encryption module of a block cipher. This restricts the comparison to AEZ and TESX. As explained above, the current construction is a fine-tuned version of TESX and Table 7 and 8 shows the comparative advantage in terms of operation counts and key size. The inherent simplification of the design of FAST over that of TESX is not captured by these parameters. Since the design approaches of FAST and AEZ are different, the comparison between FAST and AEZ cannot be determined only from the operation counts.

Among the various schemes that have been proposed, only XCB and EME2 (which is essentially EME* [19]) have been standardised. Further, AEZ is a more recent proposal and has received a fair amount of attention as part of the CAESAR[5] competition. So, it is important to provide more detailed comparison to XCB, EME2 and AEZ. Below we provide details of the software implementations of FAST and the performance results of these implementations in comparison to those of XCB, EME2 and AEZ. For the purpose of such comparison, we have made efficient

[5] https://competitions.cr.yp.to/caesar.html

software implementations of XCB, EME2 and AEZ. In Appendix B we provide a brief overview of the software implementation of AEZ. This is of some independent interest.

There have been proposals for constructing TESs from stream ciphers. The proposal [10] provides a scheme called STES which uses hardware oriented stream ciphers from the Estream portfolio to develop TES with small hardware footprint and low power consumption. A recent proposal [15] provides two TESs called Adiantum and HPolyC aimed at entry level processors. Adiantum uses four primitives, namely, AES, XChaCha12, NH and Poly1305 while HPolyC uses AES, XChaCHa12 and Poly1305. These schemes are targeted towards processors which do not provide processor support for AES and binary field multiplication. The target applications of STES and Adiantum/HPolyC are different from that of FAST. None of these schemes are expected to be as efficient as FAST on modern Intel processors, or more generally on processors which provide support for AES and binary field multiplication.

7 Software Implementation

In this section, we describe our implementations of the various instantiations of FAST in software. For the implementation, we set $n = 128$ and so $\mathbb{F} = GF(2^{128})$.

The implementation of the PRF $\mathbf{F}$ is done using the encryption function of AES. This may seem like a mismatch since the encryption function of AES is invertible while a PRF does not require invertibility. On the other hand, the PRF assumption on the encryption function of AES is widely believed to hold up to the birthday bound. Since we do not claim security beyond the birthday bound, the use of the encryption function of AES as a PRF is justified. The main reason for choosing AES is its universal acceptance.

Our target platforms for software implementation were modern Intel processors which support the AES-NI instructions which includes the 64-bit binary polynomial multiplication. The relevant Intel instructions for the two main tasks are the following.

Computation of AES: The relevant instructions are `aeskeygenassist` (for round key generation); `aesenc` (for one round of AES encryption) and `aesenclast` (for the last round of AES encryption). There are additional instructions for AES decryption. We do not mention these, since we do not require the AES decryption module.

Computation of polynomial multiplication: The relevant instruction is `pclmulqdq`. This instruction takes as input two 64-bit unsigned integers representing two polynomials each of maximum degree 63 over $GF(2)$ and returns a 128-bit quantity which represents the product of these two polynomials over $GF(2)$.

For software implementation, the two relevant parameters are latency and throughput. These are defined[6] as follows: "Latency is the number of processor clocks it takes for an instruction to have its data available for use by another instruction. Throughput is the number of processor clocks it takes for an instruction to execute or perform its calculations." On Skylake the latency/throughput figures of `aesenc`, `aesenclast` and `pclmulqdq` are 4/1.

7.1 Implementation of the Hash Functions

The several variants of the hash functions used in this work are all based on finite field computations over $\mathbb{F} = GF(2^n)$. As for the implementation, we choose $n = 128$, addition over this field is a simple XOR operation of 128-bit data types. Multiplication, on the other hand, is more involved.

[6] `https://software.intel.com/en-us/articles/measuring-instruction-latency-and-throughput`

As mentioned earlier in Section 2, we consider the field $\mathbb{F}$ to be represented using the irreducible polynomial $\psi(x) = x^{128} \oplus x^7 \oplus x^2 \oplus x \oplus 1$ over $GF(2)$ and the elements of $\mathbb{F}$ are represented using polynomials over $GF(2)$ of degrees at most 127. Let $a(x)$ and $b(x)$ be two such polynomials. The multiplication of $a(x)$ and $b(x)$ in $\mathbb{F}$ consists of the following two operations. Compute the polynomial multiplication of $a(x)$ and $b(x)$ over $GF(2)$ and let $c(x)$ be the result. Then $c(x)$ is a polynomial over $GF(2)$ of degree at most 254. The product of $a(x)$ and $b(x)$ over $\mathbb{F}$ is $c(x) \bmod \psi(x)$. The above computation consists of two distinct steps, namely polynomial multiplication followed by reduction.

Polynomial multiplication: The instruction `pclmulqdq` multiplies two polynomials over $GF(2)$ of degrees at most 63 each and returns a polynomial of degree at most 126. This is a 64-bit polynomial multiplication over $GF(2)$. Our requirement is a 128-bit polynomial multiplication over $GF(2)$. Using the direct schoolbook method, a 128-bit polynomial multiplication can be computed using four 64-bit polynomial multiplications and hence using four `pclmulqdq` calls. Karatsuba's algorithm, on the other hand, allows the computation of a 128-bit polynomial multiplication using three `pclmulqdq` calls at the cost of a few extra XOR operations. Due to the low latency of `pclmulqdq` on Skylake processors, it turns out that schoolbook is faster than Karatsuba. This has been reported by Gueron[7] and we also observed this in our experiments. So, we opted to implement 128-bit polynomial multiplication using the schoolbook method.

Reduction: Efficient computation of $c(x) \bmod \psi(x)$ has been discussed earlier [17]. It was shown that this operation can be efficiently computed using two `pclmulqdq` instructions along with a few other shifts and xors. A more detailed description of this procedure can also be found elsewhere [7]. We implemented reduction using this method.

Horner: As mentioned earlier, $\mathsf{Horner}_\tau(X_1, \ldots, X_m)$ can be computed using $m-1$ multiplications in $\mathbb{F}$. Each multiplication consists of a polynomial multiplication followed by a reduction. Doing this directly, would lead to a count of $m-1$ polynomial multiplications and $m-1$ reductions.

The efficiency can be improved by using a delayed (or lazy) reduction strategy. Let $i > 1$ be a positive integer and suppose the powers $1, \tau, \tau^2, \ldots, \tau^{i-1}, \tau^i$ are available (i.e., the powers $\tau^2, \ldots, \tau^{i-1}, \tau^i$ have been pre-computed and stored). The expression $X_1\tau^{i-1} + \cdots + X_{i-1}\tau + X_i$ over $\mathbb{F}$ can be computed using $i-1$ polynomial multiplications followed by a single reduction. Extension to handle arbitrary number of blocks is easy. For simplicity, assume that $i|m$ and $\lambda = m/i$. The m blocks are divided into λ groups of i blocks each. Each group of i blocks is processed and suppose the outputs are $Y_1, Y_2, \ldots, Y_\lambda$. Then $\mathsf{Horner}_\tau(X_1, \ldots, X_m) = \tau^i(\cdots\tau^i(\tau^i Y_1 \oplus Y_2) \oplus \cdots \oplus Y_{\lambda-1}) \oplus Y_\lambda$. Processing of a single such group of i blocks requires $i-1$ polynomial multiplications and a single reduction plus a multiplication by τ^i. Note that the computation of $\tau^i Y_1 \oplus Y_2$ is done by performing the polynomial multiplication of τ^i and Y_1, computing Y_2 without the final reduction, adding the two results and then performing a reduction. Further, this strategy is also carried out for the intermediate computations. So, processing a group of i blocks requires i polynomial multiplications and a single reduction except for the last group. In the case where i does not divide m, it is easy to modify this strategy to handle this case. We have implemented this strategy for $i = 8$ (for use in Horner and vecHorner) and $i = 9$ (for use in vecHash2L). This strategy of delayed reduction has been earlier used [18] in the context of evaluation of POLYVAL which is a variant of Horner.

There is another technique which can result in efficiency improvement. The sequence $X_1, \ldots, X_m$ is decimated into j subsequences $X_1, X_{j+1}, \ldots$; $X_2, X_{j+2}, \ldots$; $\ldots$; $X_j, X_{2j}, \ldots$; each subsequence is

[7] `https://github.com/Shay-Gueron/AES-GCM-SIV/`

computed as a polynomial in τ^j and then the results are combined together to obtain the final result. This is a well known technique. The advantage of this technique is that the j sub-sequences can be computed in parallel. In software the ability to batch j independent multiplications allows the processor to efficiently pre-fetch and pipeline the corresponding operations. We have experimented with $j = 1$, 2 and 3 and later we report timing results for $j = 3$. There are cases, however, where $j = 1$ provides slightly better performance than $j = 3$.

vecHorner: The computation of vecHorner essentially boils down to Horner computation on several different blocks. The implementation of Horner is extended to implement the computation of vecHorner.

BRW: Implementation of BRW arises as part of the implementation of FAST[Fx_m, BRW]. For this implementation, we chose $m = 256$ and $n = 128$ (corresponding to a 4096-byte disk sector to be encrypted using AES). With $m = 256$, BRW is invoked on $m - 2 + 1 = 255$ (the first two message blocks are not hashed while the single tweak block is hashed) blocks. The implementation of BRW on 255 blocks has been done in the following manner. Write

$$\mathsf{BRW}_\tau(X_1, \ldots, X_{255}) = \mathsf{BRW}_\tau(X_1, \ldots, X_{127})(X_{128} \oplus \tau^{128}) \oplus \mathsf{BRW}_\tau(X_{129}, \ldots, X_{255}).$$

This shows that the 255-block BRW computation can be broken down into 2 127-block BRW computations. Continuing, we break up the 255-block BRW computation into 8 31-block BRW computations. A completely loop unrolled 31-block BRW computation can be implemented using 15 polynomial multiplications and 8 reductions [7]. We use this implementation of 31-block BRW computation to build the 255-block BRW computation. This strategy requires 127 polynomial multiplications and 71 reductions. Following the delayed reduction strategy for BRW computation [7], it is possible to have a completely loop unrolled 255-block BRW computation requiring 127 polynomial multiplications and 64 reductions. The code for such a loop unrolled implementation would be quite complex and could lead to a substantial performance penalty. This is the reason why we have chosen to build the 255-block BRW computation from the (loop unrolled) implementation of the 31-block BRW computation.

vecHash2L: The hash function vecHash2L is parameterised by two quantities, namely the block size n and the super-block size η. The use of AES fixes n to be 128. We have taken $\eta = 31$. This requires the implementations of 31-block BRW and also i-block BRW for $i = 1, \ldots, 30$ to tackle the last super-block which can possibly have less than 31 blocks. As mentioned earlier, an implementation of Hash2L for $n = 128$ and $\eta = 31$ was reported earlier [7], but, the implementation of vecHash2L was not considered in that work. The computation of vecHash2L can be conceptually seen as 31-block BRW computations whose outputs are combined using Horner. Additionally, after each component, the length block is processed. As discussed above, the computation of Horner can be improved by using the delayed reduction strategy and the decimation technique. We have experimented with various combinations and later we report the results for 3-decimated implementations with and without the delayed reduction strategy.

7.2 Implementation of FAST

We have described several variants of FAST. Software implementations of these variants are built from the implementation of AES and the implementations of the various hash functions. The AES based parts consist of the Ctr mode and the Feistel layer while the hash functions are built from either Horner or BRW in case of the fixed length setting and are built from either vecHorner or vecHash2L in the general setting. We describe these aspects below.

Key schedule generation: All versions of FAST use a single key K which is the key to the underlying PRF $\mathbf{F}_K$. Instantiating $\mathbf{F}_K$ with the encryption function of AES requires generating the round keys. This is a one-time activity and is done using the instruction `aeskeygenassist`. The generated round keys are stored and used in both the Ctr mode and the $\mathsf{Feistel}$ layer.

Ctr: The Ctr mode defined in (1) requires a PRF $\mathbf{F}$ which is implemented using the encryption function of AES. Each invocation of the encryption function can be implemented using `aesenc` followed by `aesenclast`. The invocations of `aesenc` can be speeded up using an interleaving of multiple AES invocations. The AES encryption calls in the Ctr mode are fully parallelisable. Let $i \geq 1$ be a positive integer. The computations of the AES calls in the Ctr mode are done in batches of i calls each. The inputs to one batch of i encryptions are prepared; then the first rounds of AES encryptions of this batch of i encryptions are computed using i calls to `aesenc`; this is followed by the second rounds of this batch of i encryptions again using `aesenc` and so on. This ensures that the second round of any AES encryption does not have to wait to obtain the output of the first round. This interleaved strategy leads to substantial speed-up over computing the complete AES encryptions one after another. In our implementation, we have used $i = 8$ which follows the earlier work by Gueron[8].

$\mathsf{Feistel}$: The $\mathsf{Feistel}$ layer has two calls to AES encryptions. These calls are not parallelisable. So, these calls are implemented using a sequence of `aesenc` followed by a single call to `aesenclast` to perform the computation of a single AES encryption.

Hash key generation: The hash key τ is obtained by applying $\mathbf{F}_K$ to fStr. This is a one-time operation and the value of τ does not change during the life-time of K. So, it is possible to generate τ once and store it securely along with K. More generally, it is also possible to use a uniform random τ as the hash key instead of generating it by applying $\mathbf{F}_K$ to fStr. This will not affect the security analysis, but, will increase the key storage requirement. Alternatively, it is possible to generate τ once per session. The cost of generating τ is amortised over all the encryptions/decryptions per session and hence is negligible. Timing results provided later include the time for generating τ.

$\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$: In the setting of Fx_m, tweaks consist of a single n-bit block while plaintexts and ciphertexts consist of m n-bit blocks. In our implementation, we have taken $m = 256$ so that the total number of bytes in a plaintext/ciphertext is 4096. As mentioned earlier, this corresponds to the size of a modern disk sector. In this case, $P_3 = (P_{3,1}, \ldots, P_{3,m-2})$ consists of $(m-2)$ n-bit blocks and the hash function $\mathsf{Horner}_\tau(1, P_{3,1}, \ldots, P_{3,m-2}, T)$ needs to be computed. An implementation of Horner as mentioned above is used. This requires a total of 255 polynomial multiplications and a total of 32 reductions. Counting a single polynomial multiplication as 4 `pclmulqdq` and a reduction as 2 `pclumulqdq`, the total number of `pclmulqdq` calls required is 1084.

$\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$: As above, we take $m = 256$. The requirement is to compute

$$\mathsf{BRW}_\tau(P_{3,1}, \ldots, P_{3,m-2}, T).$$

This is done as described above which requires 127 polynomial multiplications and 71 reductions. The total number of `pclmulqdq` calls required is 650. This is 434 calls lesser than that required for computing $\mathsf{Horner}_\tau(1, P_{3,1}, \ldots, P_{3,m-2}, T)$. So, one would expect $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ to be faster

[8] Interleaving of 8 AES encryptions has been called a sweat point in `https://crypto.stanford.edu/RealWorldCrypto/slides/gueron.pdf`

than $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$. Our implementation shows a speed-up, but, not as much as one might expect from the counts of the `pclmulqdq` calls. Indeed instruction cache and pipelining are rather complicated issues and precise information about these issues for Intel processors are not easily available[9]. So, it is possible that the code for BRW that we have developed can be tuned further to obtain speed improvements.

$\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$**:** This requires the implementation of the hash function vecHorner which is an easy extension of the implementation of Horner.

$\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$**:** This requires the implementation of the hash function vecHash2L. The implementation of this hash function has been discussed above.

7.3 Timing Results

In this section, we provide timing results for the software implementations of all the four variants of FAST, i.e. for both the settings of Fx and Gn. As mentioned earlier the corresponding code is publicly available (the link is omitted from this version to maintain anonymity).

In the setting of Fx, messages are 4096 bytes long, i.e., each message consists of 256 128-bit blocks and the tweak is a single 128-bit block. The timing results are shown in Table 9 along with the timing results for XCB, EME2 and AEZ.

In the setting of Gn, timing measurements are separately reported for messages of lengths 512, 1024, 4096 and 8192 bytes. For tweaks, the number of components has been considered to be 2, 3 and 4 and the sum of the lengths of the components of the tweaks has been taken to be 1024 bytes: For tweaks with 2 components, each component has length 512 bytes; for tweaks with 3 components, the 3 components have lengths 336, 336 and 352 bytes; whereas for tweaks with 4 components, each component has length 256 bytes. Two columns of measurements are shown for $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}]$. The column with the heading 'delayed' reports measurements for the case where the Horner layer in vecHash2L has been implemented using the delayed reduction strategy while the column with the heading 'normal' reports measurements for the case where the Horner layer in vecHash2L has been implemented without using the delayed reduction strategy. The timing results are shown in Table 10.

The timing measurements were taken on two platforms.

- **Skylake:** The processor was Intel Core i7-6500U @ 2.5GHz. The operating system was 64-bit Ubuntu 14.04 LTS and the C codes were complied using GCC version 5.5.0.
- **Kabylake:** The processor was Intel Core i7-7700 @ 3.6GHz. The operating system was 64-bit Ubuntu 18.04 LTS and the C codes were complied using GCC version 7.3.0.

For the setting of Fx, we have carried out efficient implementations of XCB, EME2 and AEZ. In the implementation of AEZ, for the underlying block cipher, the full AES has been used unlike the reduced round versions considered earlier [24]. XCB uses hashing based on Horner's rule and we have used the same delayed reduction strategy in the implementation of this hashing as we did in the implementation of the hash function for $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$. The efficient software implementations of XCB, EME2 and AEZ are of independent interest. We provide a brief description of the software implementation of AEZ in Appendix B.1. Timing results from the setting of Fx show that all three of XCB, EME2 and AEZ are slower than FAST. Consequently, we do not compare FAST to XCB, EME2 and AEZ in the setting of Gn.

Based on Tables 9 and 10, we make the following observations.

[9] `https://blog.cr.yp.to/20140517-insns.html`

Table 9: Comparison of the cycles per byte measure of FAST with those of XCB, EME2 and AEZ in the setting of Fx_{256}.

scheme	Skylake	Kabylake
XCB	1.92	1.85
EME2	2.07	1.99
AEZ	1.74	1.70
$\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$	1.63	1.56
$\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$	1.24	1.19

Table 10: Report of cycles per byte measure for the setting of Gn for $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ and $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, 31, \mathsf{vecHash2L}]$.

		Skylake			Kabylake		
msg len (bytes)	$\mathfrak{k}$	vecHorner	vecHash2L (delayed)	vecHash2L (normal)	vecHorner	vecHash2L (delayed)	vecHash2L (normal)
512	2	1.51	1.38	1.59	1.42	1.32	1.56
	3	1.40	1.38	1.39	1.32	1.31	1.35
	4	1.34	1.37	1.36	1.26	1.31	1.33
1024	2	1.53	1.34	1.48	1.42	1.27	1.42
	3	1.45	1.34	1.34	1.35	1.27	1.30
	4	1.40	1.33	1.32	1.29	1.27	1.30
4096	2	1.57	1.30	1.35	1.45	1.24	1.30
	3	1.54	1.29	1.31	1.43	1.24	1.27
	4	1.51	1.29	1.30	1.40	1.24	1.26
8192	2	1.57	1.27	1.32	1.45	1.22	1.27
	3	1.56	1.27	1.30	1.44	1.22	1.25
	4	1.54	1.27	1.30	1.43	1.22	1.25

1. In the setting of Fx, $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ is faster than $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$. Moreover, $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ are both faster than all three of XCB, EME2 and AEZ.
2. In the setting of Gn for $\mathsf{vecHorner}$, the speed decreases with increase in message length while for $\mathsf{vecHash2L}$ the speed increases with increase in message length. In both cases, for the same message length, the speed mostly does not vary much with increase in the number of components in the tweak. In the case of $\mathsf{vecHash2L}$, using the delayed reduction strategy for implementing the Horner layer results in improved speed than an implementation without using delayed reduction. Overall, on Kabylake $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, 31, \mathsf{vecHash2L}]$ is faster than $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ while on Skylake $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, 31, \mathsf{vecHash2L}]$ is faster than $\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}]$ for longer messages.

8 Conclusion

In this paper we have presented a tweakable enciphering scheme called FAST. Instantiations of the scheme for both fixed length messages with single block tweaks and variable length messages with very general tweaks have been described. A detailed security analysis in the style of reductionist security proof has been provided. Software implementations of both kinds of instantiations have been made. The instantiation for fixed length messages with single block tweaks is appropriate for low-level disk encryption. The implementation results show that the new scheme outperforms previous schemes which makes the new scheme an attractive option for designers and standardisation bodies.

References

1. Public comments on the XTS-AES mode. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/collected_XTS_comments.pdf`.
2. IEEE Std 1619-2007: Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE Computer Society, 2008. Available at: `http://standards.ieee.org/findstds/standard/1619-2007.html`.
3. IEEE Std 1619.2-2010: IEEE standard for wide-block encryption for shared storage media. `http://standards.ieee.org/findstds/standard/1619.2-2010.html`, March 2011.
4. Mihir Bellare, David Cash, and Sriram Keelveedhi. Ciphers that securely encipher their own keys. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 423–432. ACM, 2011.
5. Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.
6. Ritam Bhaumik and Mridul Nandi. An inverse-free single-keyed tweakable enciphering scheme. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 159–180. Springer, 2015.
7. Debrup Chakraborty, Sebati Ghosh, and Palash Sarkar. A fast single-key two-level universal hash function. *IACR Trans. Symmetric Cryptol.*, 2017(1):106–128, 2017.
8. Debrup Chakraborty, Vicente Hernandez-Jimenez, and Palash Sarkar. Another look at XCB. *Cryptography and Communications*, 7(4):439–468, 2015.
9. Debrup Chakraborty, Cuauhtemoc Mancillas-López, Francisco Rodríguez-Henríquez, and Palash Sarkar. Efficient hardware implementations of BRW polynomials and tweakable enciphering schemes. *IEEE Trans. Computers*, 62(2):279–294, 2013.
10. Debrup Chakraborty, Cuauhtemoc Mancillas-López, and Palash Sarkar. STES: A stream cipher based low cost scheme for securing stored data. *IEEE Trans. Computers*, 64(9):2691–2707, 2015.
11. Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 2008.
12. Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.
13. Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. *IEEE Transactions on Information Theory*, 54(4):1683–1699, 2008.
14. Debrup Chakraborty and Palash Sarkar. On modes of operations of a block cipher for authentication and authenticated encryption. *Cryptography and Communications*, 8(4):455–511, 2016.
15. Paul Crowley and Eric Biggers. Adiantum: length-preserving encryption for entry-level processors. *IACR Trans. Symmetric Cryptol.*, 2018(4):39–61, 2018.
16. Dworkin, Morris J. SP 800-38E. Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices. Technical report, Gaithersburg, MD, United States, 2010.
17. Shay Gueron and Michael E. Kounavis. Efficient implementation of the galois counter mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.*, 110(14-15):549–553, 2010.
18. Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: specification and analysis. *IACR Cryptology ePrint Archive*, 2017:168, 2017.
19. Shai Halevi. EME^*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2004.
20. Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.
21. Shai Halevi and Hugo Krawczyk. Security under key-dependent inputs. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 466–475. ACM, 2007.
22. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.
23. Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

24. Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.
25. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
26. David A. McGrew and Scott R. Fluhrer. The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. `http://eprint.iacr.org/`.
27. David A. McGrew and Scott R. Fluhrer. The security of the extended codebook (xcb) mode of operation. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.
28. David A. McGrew and John Viega. Arbitrary block length mode, 2004. `http://grouper.ieee.org/groups/1619/email/pdf00005.pdf`.
29. Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2014.
30. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999.
31. Michael O. Rabin and Shmuel Winograd. Fast evaluation of polynomials by rational preparation. *Communications on Pure and Applied Mathematics*, 25:433–458, 1972.
32. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.
33. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.
34. Palash Sarkar. A general mixing strategy for the ECB-Mix-ECB mode of operation. *Inf. Process. Lett.*, 109(2):121–123, 2008.
35. Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Transactions on Information Theory*, 55(10):4749–4759, 2009.
36. Palash Sarkar. Tweakable enciphering schemes using only the encryption function of a block cipher. *Inf. Process. Lett.*, 111(19):945–955, 2011.
37. Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

A Additional Remarks and Explanations on Some Topics

A.1 Remark on the construction FAST

The quantity Z in $\mathsf{FAST.Encrypt}_K(T,P)$ and $\mathsf{FAST.Decrypt}_K(T,C)$, given in Table 1 is defined to be equal to $F_1 \oplus F_2$. In an earlier version, we had defined Z to be equal to $P_2 \oplus C_1$. The suggestion to define Z as $F_1 \oplus F_2$ is due to Mridul Nandi. This saves a few cycles in a pipelined hardware implementation when $\mathbf{F}$ is instantiated with AES; it has no effect on the efficiency of software implementation.

A.2 Remarks on bounds obtained in Corollary 1

1. As mentioned earlier, the query complexity σ is equal to the sum of the tweak query complexity θ and the message query complexity ω, i.e., $\sigma = \theta + \omega$. For the setting of Fx, $\theta = q$ so that $\sigma = q + \omega$. Using this, the bounds in (45), (46) and (47) are all of the form $\mathfrak{c}\sigma^2$ for some small constant $\mathfrak{c}$. This is the typical form of the bound that is obtained for other constructions in the literature. So, FAST does not suffer any security loss compared to previous constructions.
2. Consider the application of FAST to disk encryption of 4096 byte sectors. Then $m = 2^8$. Using $\omega = mq$ the bounds for $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}])$ $\Delta(\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}])$ given by (45) and (47) respectively both reduce to the following form.

$$\frac{1}{2^n}\left(q^2(5 \cdot 2^{16} + 1441) + 258q\right) < \frac{1}{2^n}\left(5\sigma^2 + 2\sigma\right).$$

A.3 Remarks on bounds obtained in Corollary 2

1. Recall that $\mathfrak{k}$ is the number of components in the tweak while η is the number of blocks in a BRW super-block. So, $\mathfrak{k}$ and η are constants, i.e., they are independent of n.
2. The query complexity σ is equal to the sum of the tweak query complexity θ and the message query complexity ω, i.e., $\sigma = \theta + \omega$. Using this, it can be seen that the bounds on $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}])$ and $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}])$ given by (48) and (49) respectively are of the form $\mathfrak{c}\sigma^2$ for some constant $\mathfrak{c}$. We provide illustrations below. As mentioned earlier, this is the typical form of the security bound for earlier constructions and so FAST does not have any security loss compared to the known constructions.
3. In our implementations, we take $\eta = 31$ and we use the bounds $(\eta+1)/\eta < 2$, $2(\eta+1)/\eta < 3$ and $3(\eta+1)/\eta < 4$. The value of $\mathfrak{k}$ and the lengths of the tweaks depend on the application. We provide two illustrations.

   Case $\mathfrak{k} = 1$ and each tweak is an n-bit string: The bounds for $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}])$ and $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}])$ can be shown to be respectively less than

$$\frac{5\sigma^2 + \sigma q + 12q^2 + \sigma + 3q}{2^n} \quad \text{and} \quad \frac{6\sigma^2 + 135\sigma q + 72q^2 + 2\sigma + 69q}{2^n}.$$

   Case $\mathfrak{k} = 8$ where the components of a tweak can have variable lengths:
   The bounds for $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \mathsf{vecHorner}])$ and $\Delta(\mathsf{FAST}[\mathsf{Gn}, \mathfrak{k}, \eta, \mathsf{vecHash2L}])$ can be shown to be respectively less than

$$\frac{1}{2^n}\left(5\omega^2 + 2\omega\theta + \omega + \theta\right) + \frac{1}{2^n}\left(3q(\omega+\theta) + 10((2\omega+1)q + 3q^2) + 6q^2\right) \tag{50}$$

$$\text{and } \frac{1}{2^n}\left(6\omega^2 + 3\omega\theta + 2(\omega+\theta) + 6q(\theta+\omega) + 300q(2\omega+1) + 900q^2\right). \tag{51}$$

The expressions in (50) and (51) are determined by the values of q, ω and θ. These quantities are in turn determined by the manner in which the adversary makes its queries. Since the adversary can make queries of varying lengths, it is not possible to obtain further simplifications of the expressions given in (50) and (51).

A.4 Remark on comparative performance of AEZ and EME2

From Table 9, it may be noted that AEZ is faster than EME2. From Table 7, it can be seen that the number of block cipher calls made by AEZ is more than that made by EME2. So, the fact that in practice AEZ turns out to be faster may be surprising. The explanation lies in the difference in the number of doubling operations made by EME2 and AEZ. From Table 7, EME2 requires roughly 2[BC]+2[D] operations per block whereas AEZ requires roughly 2.5[BC]+0.25[D] operations per block. Executing AES instructions in groups using pipelining results in very fast AES timings. Our experiment on the Skylake processor shows that AES requires about 0.65 cycles per byte. In contrast, while the doubling operation should in theory be much faster, there is no support for 128-bit shift and consequently doubling takes about 0.29 cycles per byte. (We refer to [14] for an elaborate discussion on various strategies for constant time doubling operation.) Using these figures, the operations count of 2[BC]+2[D] for EME2 translates to about 1.88 cycles per byte while the operations count of 2.5[BC]+0.25[D] for AEZ translates to about 1.70 cycles per byte. This provides an explanation of why AEZ is faster than EME2. Note that EME2 requires additional block cipher calls and so the actual observed time of 2.07 cycles per byte for EME2 is a bit higher than the estimated 1.88 cycles per byte whereas the observed and the estimated timings for AEZ are quite close.

B Implementation of AEZ

For $\alpha \in \{0,1\}^{128}$ and $i \in \mathbb{N}$, the following operation has been defined [24] in the context of AEZ:

$$i \cdot \alpha = \begin{cases} \mathbf{0} & \text{if } i = 0; \\ \alpha & \text{if } i = 1; \\ (\alpha \ll 1) \oplus (\mathsf{msb}(\alpha) \cdot 135) & \text{if } i = 2; \\ 2 \cdot (j \cdot \alpha) & \text{if } i = 2j > 2; \\ (2j \cdot \alpha) \oplus \alpha & \text{if } i = 2j + 1 > 2. \end{cases} \tag{52}$$

The operation in (52) corresponding to $i = 2$ is the doubling operation. (See Section 2).

There are different versions of AEZ [24] built from different variants of AES. The version which is relevant to our work is the one where the proper AES algorithm is used. Messages of lengths at least $2n$ bits are handled differently from messages of lengths less than $2n$ bits. For our purpose, we will be considering the portion which can handle messages of lengths at least $2n$ bits. This portion has been called AEZ-Core [24]. By AEZ, we will denote AEZ-Core[AES].

The length of the message is written as $2nk + \mu$ bits with $0 \leq \mu < 2n$ and $k \geq 1$. Below we provide an overview of the encryption algorithm of AEZ where $\mu = 0$. Let $m = 2\mathfrak{m} + 2$ and consider a message having m blocks with total length $n(2\mathfrak{m} + 2)$ bits. The message is partitioned into two parts: The first part consists of $2\mathfrak{m}n$ bits organised as $2\mathfrak{m}$ n-bit blocks $M_1, M'_1, \ldots, M_{\mathfrak{m}}, M'_{\mathfrak{m}}$ and the second part consists of $2n$ bits organised as 2 n-bit blocks M_{x} and M_{y}. The ciphertext blocks are C_i, C'_i, $i = 1, \ldots, \mathfrak{m}$ and $C_{\mathsf{x}}, C_{\mathsf{y}}$.

At a conceptual level, this encryption consists of three layers. The first and the third layers consist of a sequence of 2-round Feistel networks where each Feistel network requires 2 block cipher calls. The second layer is a mixing layer and requires one block cipher call for each i. Let E denote the encryption function of AES and for $\alpha \in \{0,1\}^n$, define $\widetilde{E}_K^{i,j}(\alpha) = E_K(\alpha \oplus (i+1) \cdot I \oplus j \cdot J)$ where $I = E_K(\mathbf{0})$ and $J = E_K(\mathbf{1})$ [24]. The encryption proceeds as follows:

First layer: for $i = 1, \ldots, \mathfrak{m}$, $W_i = M_i \oplus \widetilde{E}_K^{1,i}(M_i')$; $X_i = M_i' \oplus \widetilde{E}_K^{0,0}(W_i)$;
$S_{\mathsf{x}} = \widetilde{E}_K^{0,1}(M_{\mathsf{y}}) \oplus M_{\mathsf{x}} \oplus X \oplus \Delta$; $S_{\mathsf{y}} = \widetilde{E}_K^{-1,1}(S_{\mathsf{x}}) \oplus M_{\mathsf{y}}$;
Second layer: for $i = 1, \ldots, \mathfrak{m}$, $S_i' = \widetilde{E}_K^{2,i}(S)$; $Y_i = S_i' \oplus W_i$; $Z_i = S_i' \oplus X_i$;
Third layer: for $i = 1, \ldots, \mathfrak{m}$, $C_i' = Y_i \oplus \widetilde{E}_K^{0,0}(Z_i)$; $C_i = Z_i \oplus \widetilde{E}_K^{1,i}(C_i')$;
$C_{\mathsf{y}} = S_{\mathsf{x}} \oplus \widetilde{E}_K^{-1,2}(S_{\mathsf{y}})$; $C_{\mathsf{x}} = S_{\mathsf{y}} \oplus \widetilde{E}_K^{0,2}(C_{\mathsf{y}}) \oplus \Delta \oplus Y$;

Here $X = X_1 \oplus \cdots \oplus X_{\mathfrak{m}}$, $Y = Y_1 \oplus \cdots \oplus Y_{\mathfrak{m}}$, $S = S_{\mathsf{x}} \oplus S_{\mathsf{y}}$ and Δ is obtained by processing the tweak.

Remarks:

1. For implementation, we have considered $n = 128$ and $m = 256$, i.e., messages of lengths equal to 4096 bytes. So, writing $m = 2\mathfrak{m} + 2$ we have $\mathfrak{m} = 127$.
2. In our implementations, we have taken $\Delta = \mathbf{0}$, i.e., we have ignored the processing of the tweak. Since the resulting implementations of AEZ turn out to be less efficient than FAST, considering the processing of the tweak for AEZ will result in further slowdown compared to FAST.

B.1 Software Implementation

The design of AEZ is based on OTR [29] and the parallelism in AEZ is the same as that in OTR. The encryptions in the first layer can be divided into two classes – one class consisting of the encryptions of $W_1, W_2, \ldots$ and the other class consisting of the encryptions of $M_1', M_2', \ldots$. Following the pipelining strategy for AES described in Section 7.2, the encryptions in each class have to be bunched into groups of eight. The encryptions proceed as follows. One bunch of M_i''s is encrypted followed by the corresponding bunch of W_i's; then the next bunch of M_i''s is encrypted followed by the corresponding bunch of W_i's and so on. After all the encryptions in the first layer are over, the encryptions in the second layer are to be computed. These encryptions are independent and can be executed in groups of eight. The strategy for executing the encryptions in the third layer is similar to that of the first layer. Though somewhat complicated, the above mentioned strategy can be used to obtain the benefits of pipelined AES execution.

One important efficiency issue is that of computing the values $j \cdot J$. As briefly mentioned in the paper introducing AEZ [24], by storing some of the previously generated values, it is possible to efficiently generate the required values. We provide some details. A queue of values $2 \cdot J, 3 \cdot J, \ldots$ is maintained. When $j \cdot J$ is computed, it is added to the tail of the queue. If the head of the queue contains $\jmath \cdot J$, then this value is used to generate $2\jmath \cdot J$ and $(2\jmath + 1) \cdot J$ and then the entry $\jmath \cdot J$ is deleted from the queue. Using this strategy, the computation of $2\jmath \cdot J$ from $\jmath \cdot J$ requires one doubling and the computation of $(2\jmath+1) \cdot J$ from $\jmath \cdot J$ requires one XOR. Overall, the computations of $2\jmath \cdot J$ and $(2\jmath+1) \cdot J$ require one doubling and one XOR. For the $2\mathfrak{m}$ blocks in the first part, the operation (52) will be required to be applied $\mathfrak{m}$ times. This will require $\lfloor \mathfrak{m}/2 \rfloor$ doubling operations and $\lfloor (\mathfrak{m}-1)/2 \rfloor$ XOR operations. Since $m = 2\mathfrak{m} + 2$, the computations of all the masks require $\lfloor (m-2)/4 \rfloor$ doubling operations.