FAST: A New Family of Secure and Efficient Tweakable Enciphering Schemes

Debrup Chakraborty[1] and Sebati Ghosh[1] and 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 describes a new family of cryptographic constructions called FAST. Several instantiations of FAST are described. These are targeted towards two goals, the specific task of disk encryption on one hand and a more general scheme suitable for a wide variety of applications on the other. Formally, FAST is a new family of tweakable enciphering schemes. It is built as a mode of operation of a fixed input length pseudo-random function and an appropriate hash function. FAST uses a single-block key, is parallelisable and can be instantiated using only the encryption function of an appropriate block cipher. The hash function can be instantiated using either the Horner's rule based usual polynomial hashing or the more efficient Bernstein-Rabin-Winograd polynomials. Security is rigorously analysed using the standard provable security approach and concrete security bounds are derived. Detailed and careful implementations of FAST in both software and hardware are presented. The results from the implementations show that FAST compares favourably to all previous schemes. Based on these results, we put forward FAST as a serious candidate for standardisation and deployment.
Keywords: 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 Aadhar runs into several petabytes[3]. In today's world, much of the data at rest is 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 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. Just sector-wise 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. 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) [25].

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 TES called FAST which is built using a pseudo-random function (PRF) and a hash function. The domain and the range of the pseudo-random function are both the set of all n-bit binary strings for an appropriately chosen n. The hash function is built using arithmetic over the finite field $GF(2^n)$. FAST can encrypt and decrypt strings of varying lengths and hence can be considered to be a mode of operation of the underlying PRF. Some of the salient features of FAST are described below.

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 pseduo-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 [38] 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 [38], a work [7] proposed a single key TES using only the encryption function of the block cipher. This construction, however, is essentially sequential and again practical implementation results for either hardware or software were not provided. 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 hardware and software.

Wide range of applications: We provide two different types of instantiations of FAST.

Fixed length setting: The first type of instantiation 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: The second type of instantiation is more 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. No previous construction of TES supports such a general tweak space. 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 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 is some biometric information while the attributes are date-time, gender and other related information.

We note that the idea of having associated data to be a vector of strings was earlier proposed in [35] in the context of deterministic authenticated encryption.

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 introduced in [6] and called BRW polynomials in [37]. For tackling variable length inputs, a combination of BRW and Horner based hashing called Hash2L [10] turns out to be advantageous. For the fixed length setting, we show instantiations using Horner and BRW while for the general setting, we use use the vector version vecHorner of Horner and the vector version vecHash2L of Hash2L.

Provable security treatment: The proposed scheme is analysed following the standard provable security methodology. The theoretical notion of security of 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.

Software and hardware implementations: A major contribution of this paper is to provide detailed software and hardware implementations of several variants of FAST.

1. The software implementation is targeted towards the Intel Skylake processor and is in Intel intrinsics using the specialised AES-NI instructions and the `pclmulqdq` instruction. The code for the software implementation is publicly available from `https://github.com/sebatighosh/FAST`.
2. The hardware implementation is based on FPGA and is targeted towards the disk encryption application.

Results arising from the implementations show that the new proposal compares favourably to the previous and standardised constructions in both software and hardware.

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 [32]. This work, though, did not consider tweaks since the paper predates the formal introduction of the notion of a TES. The notion of a tweakable block cipher and its security was formalised by Liskov, Rivest and Wagner [28]. This was followed by a formalisation of the notion of a tweakable enciphering scheme by Halevi and Rogaway in [25]. This paper also described a TES called CMC which is based on the CBC mode of operation. A subsequent work [26] 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 [22] 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 [29]. This scheme was called XCB. A later variant [30] of XCB was proposed to improve efficiency and reduce key size. Various security problems for XCB have been pointed out in [11].

There have been a number of works proposing different constructions of TESs. Examples are PEP [17], ABL [31], HCTR [39], HCH [18], TET [23] and HEH [37]. An improved security analysis of HCTR was presented in [15]. A generalisation of EME using a general masking scheme was proposed in [36]. As mentioned earlier, the work [38] suggested the conceptual possibility of constructing a TES from a PRF and hence using only the encryption function of a block cipher. A subsequent work [7] also proposed a TES construction from a PRF which is sequential in nature. The possibility of constructing TESs from stream ciphers was indicated in [38]. Concrete proposals and detailed FPGA implementations of stream cipher based TESs have been described in [14].

Another line of investigation has been the construction of ciphers that can securely enciphers their own keys [24, 4]. The work [4] provides a generic method to convert 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 in [4]. We note that the method can equally well be applied to the construction FAST proposed in the present work. So the present work complements the construction in [4] rather than being competitive with it.

Related Primitives

Authenticated encryption with associated data (AEAD) [27, 5] encrypts a message under a key and a nonce to produce a ciphertext. The ciphertext is longer than the message and contains a tag which serves the purpose of authentication. Generating and maintaining the security of nonces can be a problem. The notion of deterministic authenticated encryption with associated data (DAEAD) does away with the nonce [35]. In terms of known constructions, AEAD schemes turn out to be the most efficient, followed by DAEAD schemes and then TESs. The difficulty of using AEAD schemes for disk encryption and the possibility of indeed using DAEAD schemes for this purpose have been discussed in details in [13].

Standards and Patents

IEEE [3] has standardised two tweakable enciphering schemes, namely EME2 and XCB. Essentially the variant of EME described in [22] has been standardised as EME2 while the variant of XCB described in [30] has been standardised as XCB. Both EME 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 XCB and EME2, 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 [19] by NIST of USA. This is based on the XEX construction of Rogaway [34]. 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. We note, on the other hand, that if the speed of a TES (guaranteeing full security) is sufficient for achieving the disk read/write rate, then there is no justification for using XTS. Indeed the scheme FAST that we propose in this work provides adequate speed of encryption/decryption to match with the read/write speed of the presently fastest disks.

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 $0 \leq i < 2^n$, $\mathsf{bin}_n(i)$ denotes the n-bit binary representation of i.

We define the following terminology.

$\mathsf{first}_i(X)$: For a binary string X, $0 \leq i \leq \mathsf{len}(X)$, $\mathsf{first}_i(X)$ 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 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 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 block. For a vector of binary strings $Y = (Y_1, \ldots, Y_k)$, by the number of blocks in Y we will mean the sum total of all the blocks in the strings $Y_1, \ldots, Y_k$.
- $\mathsf{superBlks}_{n,\eta}(Z)$: For a binary string Z, by $\mathsf{superBlks}_{n,\eta}(Z)$, we denote 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: Fix a positive integer n and 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 $0^{n-1}1$.

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 $\{\mathbf{F}_K\}_{K \in \mathcal{K}}$ where for $K \in \mathcal{K}$, $\mathbf{F}_K : \{0,1\}^n \to \{0,1\}^n$ be a family of functions. 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 [39].

3 Hash Functions

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 $\mathsf{Horner}(\tau, X_1, \ldots, X_m)$. The degree of $\mathsf{Horner}_\tau(X_1, \ldots, X_m)$ as a polynomial in τ is 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.

BRW polynomials: In [6], Bernstein defined a family of polynomials based on a previous work by Rabin and Winograd [33] (these have been called the BRW polynomials [37]). 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 proved in [6].

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 $\mathfrak{d}(m) \leq 2m - 1$; the bound 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.

3.1 Hash Function vecHorner

Let

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

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 string. The definition of $\mathsf{vecHorner} : \mathbb{F} \times \mathcal{VD} \to \mathbb{F}$ is shown in Table 1 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 0. Note that the values of $m_1, \ldots, m_k$ are defined by the algorithm to compute vecHorner shown in Table 1.

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

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 $\tau \in \mathbb{F}_{2^n}$,*

$$\Pr_\tau\left[\mathsf{vecHorner}_\tau(\mathbf{M}) \oplus \mathsf{vecHorner}_\tau(\mathbf{M}') = \alpha\right] \leq \frac{\max\left(k + \sum_{i=1}^{k} m_i, k' + \sum_{j=1}^{k'} m'_j\right)}{2^n} \tag{3}$$

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. □

3.2 Hash Function vecHash2L [10]

Two hash functions, namely Hash2L and vecHash2L, have been defined in [10]. Here we only recall the definition of vecHash2L since we will not be using the hash function Hash2L in this work. The definition of $\mathsf{vecHash2L} : \mathbb{F} \times \mathcal{VD} \to \mathbb{F}$ is given in Table 1 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. Theorem 2 of [10] shows the following. 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 $\tau \in \mathbb{F}_{2^n}$ and for any $\alpha \in \mathbb{F}_{2^n}$,

$$\Pr_{\tau}\left[\mathsf{vecHash2L}_\tau(\mathbf{M}) \oplus \mathsf{vecHash2L}_\tau(\mathbf{M}') = \alpha\right] \leq \frac{\max\left(k + (\mathfrak{d}(\eta)+1)\Lambda, k' + (\mathfrak{d}(\eta)+1)\Lambda'\right)}{2^n} \tag{4}$$

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 Construction

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

$$\mathsf{FAST.Encrypt}, \mathsf{FAST.Decrypt} : \mathcal{K} \times \mathcal{T} \times \mathcal{P} \to \mathcal{P} \tag{5}$$

and

- $\mathcal{K}$ 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 will 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 will 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 2. These require the following components.

1. Two hash functions

$$h, h' : \mathbb{F} \times \mathcal{T} \times \mathcal{M} \to \mathbb{F}, \tag{6}$$

   where

$$\mathcal{M} = \{x : w || x \in \mathcal{P} \text{ for some } w \in \{0,1\}^{2n}\}; \tag{7}$$

   $\mathbb{F}$ 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. 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 3.

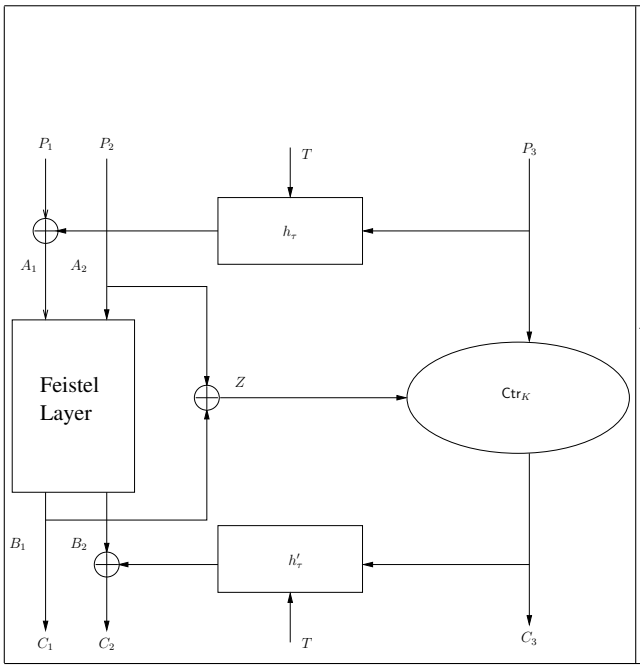
From the descriptions of $\mathsf{FAST.Encrypt}_K(T, P)$ and $\mathsf{FAST.Decrypt}_K(T, C)$ in Table 2, the following two facts are easy to verify. For $K \in \mathcal{K}$, $T \in \mathcal{T}$ and $P \in \mathcal{P}$,

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

$$\mathsf{len}\,(\mathsf{FAST.Encrypt}_K(T, P)) = \mathsf{len}(P). \tag{9}$$

From (8), it follows that the decryption function of FAST is the inverse of the encryption function, while (9) shows that the length of the ciphertext produced by the encryption function is equal to the length of the plaintext.

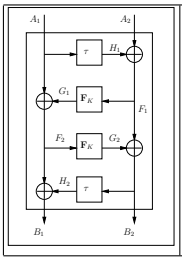
Table 2. Encryption and decryption algorithms for FAST.



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. $A_1 \leftarrow P_1 \oplus h_\tau(T, P_3)$;
4. $A_2 \leftarrow P_2$;
5. $(B_1, B_2) \leftarrow \mathsf{Feistel}_{K,\tau}(A_1, A_2)$;
6. $Z \leftarrow A_2 \oplus B_1$;
7. $C_3 \leftarrow \mathsf{Ctr}_{K,Z}(P_3)$;
8. $C_1 \leftarrow B_1$;
9. $C_2 \leftarrow B_2 \oplus h'_\tau(T, C_3)$;
10. 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. $B_1 \leftarrow C_1$;
4. $B_2 \leftarrow C_2 \oplus h'_\tau(T, C_3)$;
5. $(A_1, A_2) \leftarrow \mathsf{Feistel}^{-1}_{K,\tau}(B_1, B_2)$;
6. $Z \leftarrow A_2 \oplus B_1$;
7. $P_3 \leftarrow \mathsf{Ctr}_{K,Z}(C_3)$;
8. $P_1 \leftarrow A_1 \oplus h_\tau(T, P_3)$;
9. $P_2 \leftarrow A_2$;
10. return $(P_1||P_2||P_3)$.

Table 3. The four-round Feistel construction required in Table 2.



$\mathsf{Feistel}_{K,\tau}(A_1, A_2)$
1. $H_1 \leftarrow \tau A_1$;
2. $F_1 \leftarrow H_1 \oplus A_2$;
3. $F_2 \leftarrow A_1 \oplus \mathbf{F}_K(F_1)$;
4. $B_2 \leftarrow F_1 \oplus \mathbf{F}_K(F_2)$;
5. $H_2 \leftarrow \tau B_2$;
6. $B_1 \leftarrow H_2 \oplus F_2$;

return (B_1, B_2).

$\mathsf{Feistel}^{-1}_{K,\tau}(B_1, B_2)$
1. $H_2 \leftarrow \tau B_2$;
2. $F_2 \leftarrow B_1 \oplus H_2$;
3. $F_1 \leftarrow B_2 \oplus \mathbf{F}_K(F_2)$;
4. $A_1 \leftarrow F_2 \oplus \mathbf{F}_K(F_1)$;
5. $H_1 \leftarrow \tau A_1$;
6. $A_2 \leftarrow H_1 \oplus F_1$;

return (A_1, A_2).

5 Instantiations of FAST

Certain properties are required from the pair of hash functions (h, h'). These are formalised below.

For $M \in \mathcal{M}$, let $\mathfrak{l}(M)$ denote the number of n-bit blocks in M; for $T \in \mathcal{T}$, let $\mathfrak{t}(T)$ denote the number of n-bit blocks in T.

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 $\mathbb{F}$:*

$$\Pr[\tau(h_\tau(T, M) \oplus \alpha) = \beta] \leq \epsilon_1(\mathfrak{t}, \mathfrak{l}); \tag{10}$$

$$\Pr[\tau(h'_\tau(T, M) \oplus \alpha) = \beta] \leq \epsilon_1(\mathfrak{t}, \mathfrak{l}); \tag{11}$$

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

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

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 $\mathbb{F}$:

$$\Pr[\tau(h_\tau(T, M) \oplus h'_\tau(T', M') \oplus \alpha) = \beta] \leq \epsilon_2(\mathfrak{t}, \mathfrak{l}, \mathfrak{t}', \mathfrak{l}'). \tag{14}$$

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.

The actual definitions of h and h' vary depending on whether the intention is to handle fixed or variable length strings. We consider the following two scenarios for FAST.

Fixed length setting Fx_m for some positive integer $m > 2$: For this setting, in (5), we define

$$\mathcal{T} = \{0,1\}^n, \quad \mathcal{P} = \{0,1\}^{mn} \text{ and so } \mathcal{M} = \{0,1\}^{n(m-2)}. \tag{15}$$

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

$$h, h' : \mathbb{F} \times \mathbb{F} \times \mathbb{F}^{m-2} \to \mathbb{F}. \tag{16}$$

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 (5), we define

$$\mathcal{T} = \bigcup_{k=0}^{\mathfrak{k}} \{(T_1, \ldots, T_k) : 0 \leq \mathsf{len}(T_i) \leq 2^{n-16} - 1\}; \tag{17}$$

$$\mathcal{P} = \bigcup_{i>2n}^{2^{n-16}-1} \{0,1\}^i; \text{ and so} \tag{18}$$

$$\mathcal{M} = \bigcup_{i>0}^{2^{n-16}-2n-1} \{0,1\}^i. \tag{19}$$

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.

5.1 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:

$$h_\tau(T, X_1||\cdots||X_{m-2}) = \tau\mathsf{Horner}_\tau(\mathbf{1}, X_1, \ldots, X_{m-2}, T); \tag{20}$$

$$h'_\tau(T, X_1||\cdots||X_{m-2}) = \tau^2\mathsf{Horner}_\tau(\mathbf{1}, X_1, \ldots, X_{m-2}, T). \tag{21}$$

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 (20) and (21) 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. □

5.2 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:

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

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 (22) and (23) 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 $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$.

5.3 Hash Functions h and h' for FAST[Gn, $\mathfrak{k}$, vecHorner]

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 (17) and (19) 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

$$h_\tau(T, M) = \tau^d \oplus \mathsf{vecHorner}_\tau(T_1, \ldots, T_k, M); \tag{24}$$
$$h'_\tau(T, M) = \tau(\tau^d \oplus \mathsf{vecHorner}_\tau(T_1, \ldots, T_k, M)). \tag{25}$$

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 1. *The initialisation of* digest *using* $\mathsf{digest} = \mathbf{0}$ *is to be replaced with* $\mathsf{digest} = \mathbf{1}$.

Proposition 4. *The hash functions h and h' defined in (24) and (25) respectively for the construction* FAST[Gn, $\mathfrak{k}$, 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. □

5.4 Hash Functions h and h' for FAST[Gn, $\mathfrak{k}$, η, 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 (17) and (19) respectively. For $T = (T_1, \ldots, T_k) \in \mathcal{T}$ and $M \in \mathcal{M}$ let $d = (\mathfrak{d}(\eta) + 1)(\ell_1 + \cdots + \ell_k + \ell) + k + 2$. We define

$$h_\tau(T, M) = \tau^d \oplus \mathsf{vecHash2L}_\tau(T_1, \ldots, T_k, M); \tag{26}$$
$$h'_\tau(T, M) = \tau(\tau^d \oplus \mathsf{vecHash2L}_\tau(T_1, \ldots, T_k, M)). \tag{27}$$

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 vecHash2L shown in Table 1. *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* vecHash2L *be such that $\eta+1$ is a power of two. The hash functions h and h' defined in (26) and (27) 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 T_i be ℓ_i and the number of super-blocks in 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

$$\begin{aligned}
d &= (\mathfrak{d}(\eta)+1)(\ell_1+\cdots+\ell_k+\ell)+k+2\\
&\leq (\mathfrak{d}(\eta)+1)(\ell_1+\cdots+\ell_k+\ell)+\mathfrak{k}+2\\
&= (\eta+1)(\ell_1+\cdots+\ell_k+\ell)+\mathfrak{k}+2 \quad (\text{since } \mathfrak{d}(\eta)=\eta)\\
&< ((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(k+1)(\eta+1)+\mathfrak{k}+2\\
&\leq ((\eta+1)/\eta)(\mathfrak{t}+\mathfrak{l})+(\mathfrak{k}+1)(\eta+2)+1. \qquad (28)
\end{aligned}$$

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 [10] (see (4)), 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 $\max(d+2,d'+2)/2^n$.

Now 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. □

6 Security

In this section, we provide the formal definitions and the formal security statement for FAST. The detailed security proof is provided in the Appendix.

An adversary $\mathcal{A}$ is a possibly probabilistic algorithm with access to one or more oracles. The output of an algorithm is a single bit. The notation $\mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\cdots} \Rightarrow 1$ 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.

6.1 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.

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho(\cdot)} \Rightarrow 1\right]. \tag{29}$$

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 $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A}) \leq \varepsilon$ for all $\mathcal{A}$ running in time $\mathfrak{T}$ and making q queries with query complexity σ.

6.2 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 TES.Encrypt and 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 [25]). 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:

$$\begin{aligned}&\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{TES}}(\mathcal{A})\\ &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{TES.Encrypt}_K(\cdot,\cdot),\mathsf{TES.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right]. \qquad (30)\end{aligned}$$

Suppose $\mathcal{A}$ makes q queries $(T^{(1)}, X^{(1)}), \ldots, (T^{(q)}, X^{(q)})$ where $X^{(s)}$ is one of $P^{(s)}$ or $C^{(s)}$. We will write $\mathfrak{t}^{(s)}$ to denote $\mathfrak{t}(T^{(s)})$. Let $X^{(s)} = X_1^{(s)}||X_2^{(s)}||X_3^{(s)}$ with $\mathsf{len}(X_1^{(s)}) = \mathsf{len}(X_2^{(s)}) = n$ and $\mathsf{len}(X_3^{(s)}) \geq 1$. Then $X_3^{(s)} \in \mathcal{M}$ and $\mathfrak{l}(X_3^{(s)}) = m^{(s)} - 2$. We will write $\mathfrak{l}^{(s)}$ to denote $\mathfrak{l}(X_3^{(s)})$.

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

$$\theta = \sum_{s=1}^{q} \mathfrak{t}(T^{(s)}) = \sum_{s=1}^{q} \mathfrak{t}^{(s)}; \qquad (31)$$

$$\omega = \sum_{t=1}^{q} m^{(t)}; \qquad (32)$$

$$\sigma = \theta + \omega. \qquad (33)$$

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.

6.3 Security of FAST

The security statement 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 2. Fix q*,* $\omega \geq q$ *to be positive integers and* θ *to be a non-negative integer. For all* $\mathfrak{T} > 0$*,*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathfrak{T}, q, \theta, \omega) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega + 1, \omega + 1) + \Delta(\mathsf{FAST}), \quad \textit{where}$$
$$\Delta(\mathsf{FAST}) = 2\omega \left(\sum_{s=1}^{q} \epsilon_1^{(s)} \right) + 3 \sum_{1 \leq s < t \leq q} \epsilon_2^{(s,t)} + \sum_{s=1}^{q} \epsilon_2^{(s,s)} + \frac{3\omega^2}{2^n}; \tag{34}$$

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

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

Corollary 1. *Let* $m \geq 3$ *be a fixed positive integer. Let* q *and* $\sigma \geq q$ *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* $\mathfrak{T} > 0$*,*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]}(\mathfrak{T}, q, q, \omega) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega + 1, \omega + 1) + \frac{14\omega(\omega + 1)}{2^n}; \tag{35}$$

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

$$\begin{aligned} &\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]}(\mathfrak{T}, q, q, \omega) \\ &\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega + 1, \omega + 1) + \frac{1}{2^n} \left(14\omega(\omega + 1) + 4\omega^2 \mathfrak{d}(m - 1) \right). \end{aligned} \tag{36}$$

Further, if $m \geq 4$ *is a power of two, then*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]}(\mathfrak{T}, q, q, \omega) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T} + \mathfrak{T}', \omega + 1, \omega + 1) + \frac{14\omega(\omega + 1)}{2^n}. \tag{37}$$

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 (34) 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 by $(14\omega(\omega + 1))/2^n$.

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 stated bound.

In the case where $m \geq 4$ is a power of two, 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 second statement. □

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*

$$\begin{aligned}\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Gn},\mathfrak{k},\mathsf{vecHorner}]}(\mathfrak{T}, q, \theta, \omega) &\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}', \omega+1, \omega+1) \\ &\quad + \frac{5\omega^2 + 2\omega\theta + \omega + \theta}{2^n} + \frac{3q(\theta+\omega) + (\mathfrak{k}+2)((2\omega+1)q + 3q^2) + 6q^2}{2^n}; \quad (38)\\ \mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}[\mathsf{Gn},\mathfrak{k},\eta,\mathsf{vecHash2L}]}(\mathfrak{T}, q, \theta, \omega) &\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathfrak{T}+\mathfrak{T}', \omega+1, \omega+1) \\ &\quad + \frac{(3 + 2(\eta+1)/\eta)\omega^2 + (2(\eta+1)/\eta)\omega\theta + ((\eta+1)/\eta)(\omega+\theta)}{2^n} \\ &\quad + \frac{3q((\eta+1)/\eta)(\theta+\omega) + ((2\omega+1)q + 3q^2)(\mathfrak{k}+1)(\eta+2) + 3q(2\omega+1) + 9q^2}{2^n}. \quad (39)\end{aligned}$$

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 (34) 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

$$\begin{aligned}\sum_{s=1}^{q} \epsilon_1^{(s)} &= \sum_{s=1}^{q} \frac{\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)} + \mathfrak{k} + 4}{2^n} = \frac{\theta+\omega}{2^n} + \frac{q(\mathfrak{k}+2)}{2^n}; \\ \sum_{1\leq s<t\leq q} \epsilon_2^{(s,t)} &\leq \sum_{1\leq s<t\leq q} \frac{\max(\mathfrak{t}^{(s)} + \mathfrak{l}^{(s)}, \mathfrak{t}^{(t)} + \mathfrak{l}^{(t)}) + \mathfrak{k} + 4}{2^n} \leq \frac{q(\theta+\omega)}{2^n} + \frac{q^2(\mathfrak{k}+4)}{2^n}.\end{aligned}$$

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,

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

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

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

Using these in Theorem 1, we obtain

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

□

7 Comparison

This section provides a comparison of the design features of FAST with previously proposed TESs.

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 [23] and HEH [37] while some form of the counter mode of operation has been used in XCB [29, 30], HCTR [39] and HCH [18]. In this paper, we use the counter mode of operation as described in [39].
2. For the hash functions, XCB, HCTR and HCH essentially use Horner based hashing. The cost of hashing in TET is higher. BRW based hashing has been suggested for HEH and implemented in hardware for fixed length messages in [12].

All of the above mentioned TESs require both the encryption and the decryption functions of the block cipher. The work [38] suggested the possibility of using only the encryption function of a block cipher to build a TES. The present work is based upon the idea in [38]. In terms of similarity, both [38] 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. In [38], 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 replaces the applications of h to A_1 and B_2 with multiplications by τ and further this τ is the key for the hash function used outside the Feistel layer. In summary, [38] uses two hash keys while FAST uses a single hash key.
2. In [38], 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. In [38], the seed to the counter mode is generated as $A_1 \oplus A_2 \oplus B_1 \oplus B_2$. FAST generates the seed to the counter mode as $A_2 \oplus B_1 = P_2 \oplus C_1$.

Some of the above differences such as reducing the hash key size 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 the design in [38]. 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 implementations in both software and hardware 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 [25], EME [26, 22] and FMix [7] essentially uses two layers of encryption using a mode of operation of a block cipher. These TESs do not use any hash function. Out of these CMC and FMix are essentially sequential while EME is parallelisable. FMix requires 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 cost of encryption is roughly two block cipher calls per block of the message. Since EME is parallelisable while CMC and FMix are not, any reasonable implementation of EME will be faster than both CMC and FMix. Later we provide implemetation results which show the superiority of FAST in comparison to EME in both software and hardware and hence also imply the superiority of FAST over CMC and FMix.

Table 4. Comparison of different tweakable enciphering schemes. 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. [BC] denotes the number of block cipher calls; [M] denotes the number of field multiplications; [BCK] denotes the number of block cipher keys; and [HK] denotes the number of blocks in the hash key.

Scheme	[BC]	[M]	[BCK]	[HK]	dec module	parallel
CMC [25]	$2m+1$	–	1	–	reqd	no
EME* [22]	$2m+1+m/n$	–	1	2	reqd	yes
XCB [29]	$m+1$	$2(m+3)$	3	2	reqd	yes
HCTR [39]	m	$2(m+1)$	1	1	reqd	yes
HCHfp [16]	$m+2$	$2(m-1)$	1	1	reqd	yes
TET [23]	$m+1$	$2m$	2	3	reqd	yes
HEH-BRW[37]	$m+1$	$2+2\lfloor(m-1)/2\rfloor$	1	1	reqd	yes
[38] with BRW	$m+1$	$4+2\lfloor(m-1)/2\rfloor$	1	2	not reqd	yes
FMix [7]	$2m+1$	–	1	–	not reqd	no
FAST[Fx_m, Horner]	$m+1$	$2m+1$	1	–	not reqd	yes
FAST[Fx_m, BRW]	$m+1$	$2+2\lfloor(m-1)/2\rfloor$	1	–	not reqd	yes

Table 4 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 only the construction in [38]. As explained above, the current construction is a fine-tuned version of the construction in [38] and Table 4 shows the comparative advantage in terms of operation counts and key size. The inherent simplification of the design is not captured by these parameters.

While various schemes have been proposed, only XCB and EME2 (which is essentially EME*) have been standardised. So, it is important to provide more detailed comparison to these two schemes. Below we provide details of the implementations of FAST in both software and hardware and the performance results of these implementations in comparison to those of XCB and EME2. For the purpose of such comparison, we have made efficient implementations of XCB and EME2 in both software and hardware.

8 Software Implementation

In this section, we describe our implementations of the various versions of FAST in software. For the implementation, we set $n = 128$ and so $\mathbb{F} = GF(2^{128})$. The implementation of the PRF $\mathbf{F}$ was done using the encryption function of AES.

Our target platform for software implementation was the Intel processor Skylake. This processor provides instruction set support for computing the AES and also for computing polynomial multiplication. The Intel instructions for these 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[5] 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.

8.1 Implementation of the Hash Functions

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

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 $GF(2)$. 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 $c(x)$ mod $\psi(x)$. 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[6] 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 $c(x)$ mod $\psi(x)$ has been discussed in [20]. It was shown that this operation can be efficiently computed using two `pclmulqdq` instructions along with a few other shifts and xors. A description of this procedure can also be found in [10]. 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 $\mathsf{Horner}_\tau(X_1, \ldots, X_m) = \tau^i(\cdots \tau^i(\tau^i Y_1 \oplus Y_2) \oplus \cdots \oplus Y_{\lambda-1}) \oplus Y_\lambda$.

[5] `https://software.intel.com/en-us/articles/measuring-instruction-latency-and-throughput`
[6] `https://github.com/Shay-Gueron/AES-GCM-SIV/`

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 in [21] 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 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 and we provide further details as part of the discussion on hardware implementation. The advantage of this technique is that the j sub-sequences can be computed in parallel. This is a distinct advantage in hardware while 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 $j = 3$ and we later 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: The implementation of BRW arises as part of the implementation of $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$. 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 [10]. 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 described in [10], 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 in [10]. The implementation of vecHash2L was not given in [10]. 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.

8.2 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 Feistel layer.

Ctr: The Ctr mode 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[7].

Feistel: The 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. Additionally, Feistel also has two field multiplications by τ. These multiplications also need to be computed sequentially each using a polynomial multiplication followed by a reduction.

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.

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

$\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$**:** In the setting of Fx_m, tweaks consist of a single n-bit block and plaintexts/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 $m = 256$ and the requirement is to compute $\mathsf{BRW}_\tau(P_{3,1}, \ldots, P_{3,m-2}, T)$. This is done as described above requiring 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 than $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$. Our implementation shows a speed-up, but, not as much as one might expect from the count of `pclmulqdq` calls. Indeed instruction cache and pipelining are rather complicated issues and precise information about these issues for Intel processors are not easily available[8]. So, it is possible that the code for BRW that we have developed can be tuned further to obtain speed improvements.

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

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

8.3 Timing Results

In this section, we provide timing results for the software implementations of all the four variants of FAST. The corresponding code is available at `https://github.com/sebatighosh/FAST`. The timing results for FAST are for two settings. The first is in the setting of Fx while the second is in the setting of Gn.

1. 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.
2. 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: for tweaks with 2 components, each component contains 512 bytes; for tweaks with 3 components, the components contain 336, 336 and 352 bytes; whereas for tweaks with 4 components, each component contains 256 bytes. Two columns of measurements are shown for $\mathsf{FAST}[\mathsf{Gn}, \mathbb{t}, \eta, \mathsf{vecHash2L}]$. The column with the heading 'delayed' reports measurements for the case where the Horner layer in $\mathsf{vecHash2L}$ 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 $\mathsf{vecHash2L}$ has been implemented without using the delayed reduction strategy.

The timing measurements were on Intel Core i7-6500U Skylake @ 2.5GHz. The operating system was 64-bit Ubuntu-14.04-LTS and the C codes were complied using GCC version 4.8.4.

Based on Tables 5 and 6, we make the following observations.

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

scheme	performance
XCB	1.92
EME2	2.07
$\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$	1.66
$\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$	1.25

Table 5. Comparison of the cycles per byte measure of FAST with those of XCB and EME2 in the setting of Fx_{256}.

msg len in bytes	ℓ	vecHorner	vecHash2L (delayed)	vecHash2L (normal)
512	2	1.40	1.43	1.37
	3	1.40	1.57	1.44
	4	1.40	1.70	1.55
1024	2	1.43	1.37	1.33
	3	1.43	1.49	1.38
	4	1.44	1.56	1.47
4096	2	1.52	1.30	1.31
	3	1.53	1.35	1.33
	4	1.53	1.38	1.36
8192	2	1.55	1.28	1.30
	3	1.55	1.30	1.31
	4	1.55	1.34	1.32

Table 6. Report of cycles per byte measure for the setting of Gn for $\mathsf{FAST}[\mathsf{Gn}, \ell, \mathsf{vecHorner}]$ and $\mathsf{FAST}[\mathsf{Gn}, \ell, 31, \mathsf{vecHash2L}]$.

1. In the setting of Fx, $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ are faster than both XCB and EME2 with $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ being faster than $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$. We note that XCB also 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 $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$.
2. In the setting of Gn, for vecHorner the speed decreases with both increase in message length and increase in the number of components while for vecHash2L the speed increases with increase in message length and for the same message length the speed decreases a bit as the number of components in the tweak increases. In the case of vecHash2L, using the delayed reduction strategy for implementing the Horner layer in vecHash2L is slower than an implementation without using delayed reduction. This is due to the fact that the overhead of pre-computing the required powers of τ becomes significant. For obtaining the effect of delayed reduction, it is required to increase the message length beyond the 8192 bytes that has been considered. Overall, in most cases, $\mathsf{FAST}[\mathsf{Gn}, \ell, 31, \mathsf{vecHash2L}]$ is faster than $\mathsf{FAST}[\mathsf{Gn}, \ell, \mathsf{vecHorner}]$.

9 Hardware Implementation

We have implemented the fixed length variants of FAST, i.e., $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_m, \mathsf{BRW}]$ in reconfigurable hardware. The implementations were done keeping in mind the application of disk encryption. The design decisions that were made are as follows:

1. **PRF**: For the PRF $\mathbf{F}$, we have used the encryption function of AES. So, the block length $n = 128$.
2. **Message and tweak lengths**: We have assumed a message length of 4096 bytes. As mentioned earlier, this is the current sector size of commercially available hard disks. For $n = 128$, 4096

bytes correspond to 256 blocks, i.e., $m = 256$. For disk encryption application, the tweaks are sector addresses and we have assumed the tweak to be a single n-bit block. So, our implementations are those of $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$.

3. **Choice of FPGA**: The basic design goal was speed and so the implementations were optimized for speed. Nevertheless, we tried to keep the area metric reasonable. The target device was a high end fast FPGA. In particular, we have optimized our design for the Xilinx Virtex 5 and Virtex 7 families.

With $m = 256$ and a single block tweak, the numbers of blocks in the inputs to the hash functions h and h' are both 255. The 255 blocks comprise of 254 blocks arising from P_3 or C_3 and one block from the tweak. Since 255 blocks are to be hashed, for $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$, the requirement is to implement 255-block Horner while for $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$, the requirement is to implement 255-block BRW.

For both $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$, we have implemented two variants, one with a single core of the AES encryption module and the other with two cores of the AES encryption module. We will denote variants of $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ using a single AES core as FAST[AES,Horner]-1 and FAST[AES,BRW]-1 respectively. The variants of $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{Horner}]$ and $\mathsf{FAST}[\mathsf{Fx}_{256}, \mathsf{BRW}]$ using two AES cores will be denoted as FAST[AES,Horner]-2 and FAST[AES,BRW]-2 respectively.

The two basic building blocks for all of these designs are the encryption function of the AES and a finite field multiplier.

In our implementations, we have used pipelined AES encryption cores. An AES encryption core requires a key generation module. For the two-core designs the same key generation module is shared by both the cores. The latency of each AES core is 11 cycles, i.e., the first block of ciphertext is produced after a delay of 11 cycles and thereafter one cipher block is obtained in each cycle. The design of the AES cores adopts some interesting ideas reported in [8]. The design in [8] was that of a sequential AES design tailored for the Virtex 5 family of devices. An important aspect of this design is that the S-boxes are implemented as 256×8 multiplexers and one S-box fits into 32 six-input LUTs which are available in Virtex 5 FPGAs. We have used the same idea to design the S-boxes of our pipelined AES core.

With $n = 128$, the requirement is to compute products in $GF(2^{128})$. For this, we have used a 4-stage pipelined Karatsuba multiplier. The number of stages was selected to match the maximum frequency of the AES encryption core, which is the only other significant component in the circuits. The multiplier design is the same as reported in [13].

To use the pipelined multiplier efficiently, it is important to schedule the multiplications in such a way that pipeline delays are minimised. The BRW computation is amenable to a very efficient pipelined implementation. This requires identifying an "optimal" order of the multiplications so that both pipeline delays and the necessity to store intermediate results are minimised. In [12], a detailed study of such an optimal ordering was done. A circuit for computing BRW polynomials on 31 blocks of inputs using a 3-stage pipelined Karatsuba multiplier was presented in [12]. In this work, the requirement is to compute BRW polynomials on 255 blocks using a 4-stage pipelined multiplier. We scale up the design in [12] suitably for our purpose.

For computing Horner using a pipelined multiplier the idea of decimation is used. This has been briefly mentioned in the context of software implementation. We provide some more details here. Let $(X_1, X_2, \ldots, X_m)$ and a positive integer d be given. Let $\chi_i = m - i \pmod{d}$. The d-decimated

Horner computation [10] is based on the following observation.

$$\begin{aligned}&\mathsf{Horner}_\tau(X_1, X_2, \ldots, X_m)\\ &= \tau^{\chi_1}\mathsf{Horner}_{\tau^d}(X_1, X_{1+d}, X_{1+2d}, \ldots) \oplus \cdots \oplus \tau^{\chi_d}\mathsf{Horner}_{\tau^d}(X_d, X_{2d}, X_{3d}, \ldots).\end{aligned}$$

So, $\mathsf{Horner}_\tau(X_1, X_2, \ldots, X_m)$ can be computed by evaluating d *independent* polynomials at τ^d and then combining the results. This representation allows efficient use of a d-stage pipelined multiplier, as in each clock d independent multiplications can be scheduled.

In what follows, we give a detailed description of the architecture of FAST[AES,BRW]-2 followed by a short description of the architecture of FAST[AES,Horner]-2.

9.1 Architecture for FAST[AES,BRW]-2

FAST[AES,BRW]-2 uses two pipelined AES encryption cores and a 4-stage pipelined multiplier. An overview of the architecture is shown in Figure 1. We briefly describe its components and functioning.

The basic components of the architecture are the two AES encrytion cores which are denoted as **AESodd** and **AESeven**. The module for the BRW polynomial evaluation using a 4-stage Karatsuba multiplier is shown as **BRWPoly_eval**.

The two AES cores, two multiplexers **M1** and **M2** and a counter named **Counter** are enclosed inside a dashed rectangle. This constitutes a module which implements the counter mode. The module can also perform AES encryption of a single block. The **AESeven** core is used only in counter mode whereas the **AESodd** core is used for both encryption in the counter mode and to encrypt single blocks. According to the algorithms in Tables 2 and 3, encryption of a single block is required for the blocks F_1 and F_2 in the Feistel function and for fStr in the main function.

The counter has two outputs, one for the odd values and the other for the even ones. The even values are fed directly to the **AESeven** core and the odd values are fed to the **AESodd** core through the multiplexer **M1**. The block **BRWPoly_eval** performs the 255-block BRW computation. Additionally, this block also computes the single multiplications by τ to compute H_1 and H_2 which are required in the Feistel layer (see Table 3).

The registers **Z**, **A1**, **A2**, **F1**, **F2** and **B2** are used to store the intermediate values and these correspond to the variables Z, A_1, A_2, F_1, F_2 and B_2 respectively of the algorithms described in Tables 2 and 3.

The input ports **Podd** and **Peven** are used to feed in the odd numbered message blocks and even numbered message blocks respectively. The tweak is also fed in through **Podd**. The hash key is fed in through a separate port. The output ports **Ceven** and **Codd** output the even and odd numbered cipher blocks respectively.

The multiplexer **M5** selects the input to **AESodd** from one of the four possible inputs, namely, **Podd**, **F1**, **F2** or the string fStr. The multiplexer **M1** selects either the output of **M5** or $Z \oplus i$, where i is the output from the odd port of **Counter**. This input design to the **AESodd** core through the multiplexers **M1** and **M5** allows **AESodd** to encrypt in the counter mode and also to encrypt the required single blocks.

The BRW computation module **BRWPoly_eval** is required to be fed two blocks of plaintext or ciphertext in each cycle. The multiplexer **M3** provides the first input to **BRWPoly_eval**. This input is selected by **M3** to be one of **Podd**, **Codd**, **A1** or **B2**. The inputs **Podd** and **Codd** are relevant for BRW while the inputs **A1** and **B2** are relevant when a single-block multiplication is required. The second input to **BRWPoly_eval** is the output of the multiplexer **M4** and can be either **Peven** or **Ceven**.

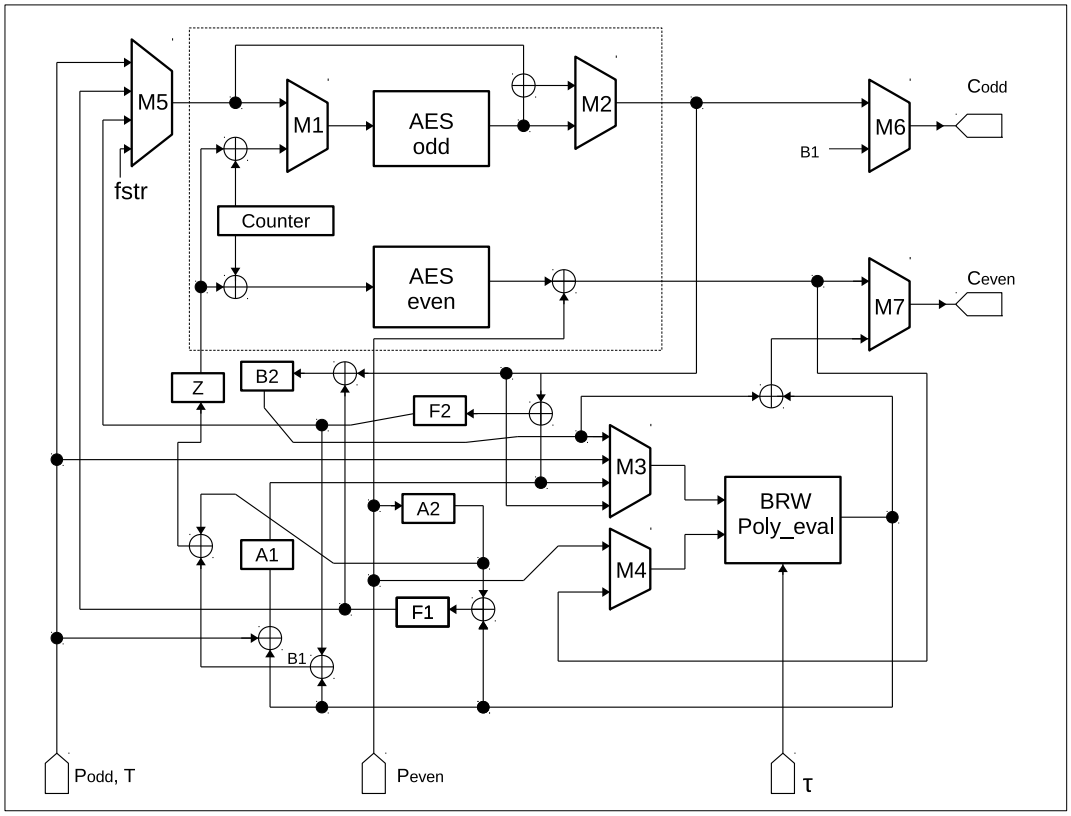


Fig. 1. Architecture for FAST[AES,BRW]-2.

The final outputs of the circuit are selected using multiplexers **M6** and **M7**. Control signals are generated using a finite state machine which follows the algorithm of FAST.

Timing analysis: In Figure 2 we show the timing diagram for the architecture for FAST[AES,BRW]-2. The first 11 clock cycles are required to compute the hash key τ by applying the AES encryption module to fStr. The computation of the hash function h (see (22)) requires computing a 255-block BRW computation and a field multiplication by τ for a total of 128 field multiplications. The 4-stage multiplier has a latency of 4 cycles. This latency arises twice, once for the first field multiplication and also at the end when the multiplication by τ is dependent on all the blocks. The computation of $h()$ requires a total of 128 field multiplications (127 multiplications for the 255-block BRW computation plus the multiplication by τ) and is completed in 136 cycles. The Feistel network is computed in 34 clock cycles: 22 cycles for the two AES encryption calls plus 8 cycles for the two multiplications and four additional cycles for synchronization. After the Feistel computation is over, C_3 can be computed with the counter mode. Eleven clock cycles after the counter mode is initiated, two blocks of C_3 become available and then onwards two blocks of the ciphertext become available at each clock cycle. So, 11 clock cycles after the counter mode is initiated, the second hash can start. After a total of 330 clock cycles the last block of the ciphertext is produced.

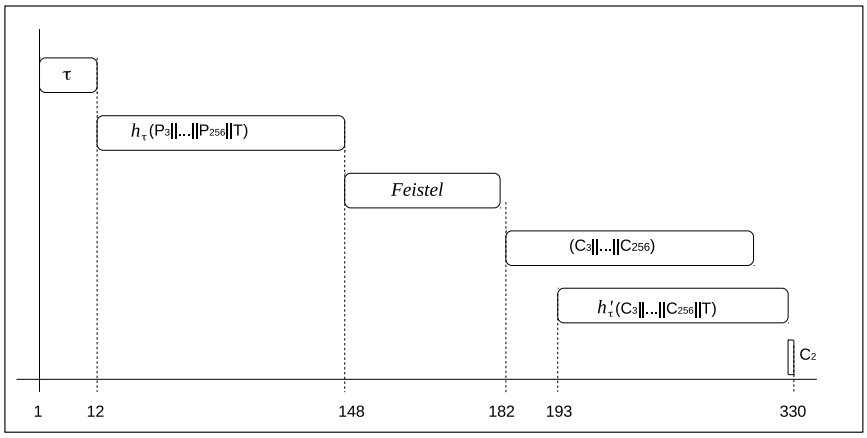


Fig. 2. Time diagram for encryption using FAST[AES,BRW]-2.

9.2 Architecture for FAST[AES,Horner]-2

To take the advantage of two AES cores in the design of FAST[AES,Horner]-2 it becomes necessary to use two multipliers. The reason is the following. The crucial parallelisation is in computing the second hash layer (see Line 9 in Table 2), where the hash of the ciphertexts produced by the counter mode is computed. Since two pipelined AES cores are used to implement the counter mode, after an initial delay, in each clock cycle two blocks of ciphertexts are produced. So, the hash module has to be capable of processing two ciphertext blocks in each cycle. For BRW based hashing, each multiplication involves two ciphertext blocks. On the other hand, in the case of Horner, each multiplication involves a single block. So, to process two ciphertext blocks in each cycle it is required to use two multipliers. Each multiplier operates in a 4-stage pipeline. For proper scheduling using the two multipliers, it is required to use a 8-decimated version of Horner. This allows the scheduling of four independent multiplications to each multiplier in every clock cycle.

9.3 Experimental Results

We present performance data for four implementations of FAST: The results are compared with the implementations of the disk encryption standards XCB and EME2 [3] as reported in [13]. In [13] the performance of two architectures each of EME2 and XCB are reported, they are named EME2-1, EME2-2 and XCB-1 and XCB-2 respectively. The hardware resources utilized in these constructions along with those used in the different architectures for FAST are summarized in Table 7.

Table 7. Summary of the main hardware resources in the architectures of FAST, EME2 and XCB

Scheme	Pipelined AES encryption core	Pipelined AES decryption core	Sequential AES decryption core	Pipelined multiplier
FAST[AES,BRW]-1	1	0	0	1
FAST[AES,Horner]-1	1	0	0	1
EME2-1	1	1	0	0
XCB-1	1	0	1	1
FAST[AES,BRW]-2	2	0	0	1
FAST[AES,Horner]-2	2	0	0	2
EME2-2	2	2	0	0
XCB-2	2	0	1	2

Some important aspects of the architectures of EME2 and XCB are as follows:

1. The encryption cores and the multipliers utilised in FAST are the same as those utilised in XCB and EME. The sequential decryption core required in XCB was optimised for speed. To match the critical path of the AES encryption core the sequential decryption core was implemented using T-boxes.
2. EME2 is a mask-encrypt-mask type construction which consists of two ECB layers with an intermediate masking. The ECB layers can be implemented with pipelined AES cores. For decryption, ECB in decryption mode is required, hence for efficient decryption functionality pipelined AES decryption cores are required to be used. The second layer of ECB in EME2 can only be computed once the first layer has been completed and so the intermediate results of the first layer of ECB encryption are required to be stored. Block RAMs are used for this purpose.
3. XCB is a hash-counter-hash type mode which involves a counter mode of operation sandwiched between two polynomial hash layers. The main encryption/decryption in XCB takes place through a variant of the counter mode (which is different from the counter mode used in FAST). The counter mode can be implemented using only the encryption module of AES. One call to the decryption module of AES is required in XCB for both encryption and decryption. For this, a sequential AES decryption core is utilised. Thus, XCB-2 uses two pipelined AES encryption cores which does the bulk encryption and in addition uses a sequential AES decryption core.
4. The polynomial hash layers in XCB are consist of Horner computations. The second Horner computation in XCB can be computed in parallel with the counter mode. As in case of FAST[AES,Horner]-

2 the counter mode in XCB-2 is implemented using two AES cores. So, in each clock cycle, two blocks of ciphertexts are obtained and to utilise this parallelisation two multipliers are required.

The performance results presented in Table 8 are obtained after place and route process in ISE 14.7. The target device was `xc5vlx330t-2ff1738`. We tried many timing restrictions and the best case is reported.

Construction	Area (slices)	Frequency (MHz)	Clock cycles	Throughput (Gbps)	TPA (Mbps/slice)
AES-PEC	2859	300.56	1	38.47	
AES-PDC	3110	239.34	1	30.72	
AES-SDC	1800	292.48	11	3.40	
128-bit Karatsuba multiplier	1650	298.43	1	38.20	
FAST[AES,BRW]-2	7175	289.56	330	28.75	4.00
FAST[AES,Horner]-2	8983	289.98	322	**29.51**	3.28
XCB-2	9752	270.52	316	28.05	2.87
EME2-2	10970	230.56	305	24.77	2.25
FAST[AES,BRW]-1	5064	290.57	466	20.43	**4.03**
FAST[AES,Horner]-1	**4411**	291.29	576	16.53	3.74
XCB-1	6070	272.75	569	15.70	2.58
EME2-1	6500	233.58	561	13.64	2.09

Table 8. Implementation results for Virtex 5 FPGA.

The first part of Table 8 shows the performance of the basic modules, i.e., the pipelined encryption core, the pipelined decryption core, the sequential decryption core and the pipelined multiplier. The decryption cores are not required in FAST. The pipelined decryption core is required for EME2 and the sequential decryption core is required for XCB. The results for individual AES cores in Table 8 include the area required for the key schedule module. For the implementations of modes of operation we have implemented only one key schedule, and it is shared between all the AES cores presented in the architecture.

From the results in Table 8 we observe the following:

1. Comparison of area.
   (a) Among the two-core architectures, FAST[AES, BRW]-2 is the smallest in terms of area while among the single-core architectures, FAST[AES, Horner]-1 is the smallest.
   (b) In comparison to Horner, the module for implementing BRW requires more registers and also circuits for squaring. As a result, FAST[AES, BRW]-1 requires 653 slices more than FAST[AES, Horner]-1.
   (c) For the two-core architectures, FAST[AES, Horner]-2 requires more area than FAST[AES, BRW]-2 since the implementation of FAST[AES, Horner]-2 requires two multipliers while the implementation of FAST[AES, BRW]-2 requires a single multiplier.
   (d) EME2 is the costliest in terms of area in both categories of single core and double core architectures. This is because it requires two AES decryption cores for its functionality. Further both EME2-1 and EME2-2 require four block RAMs in addition to the slices reported in Table 8.

(e) The overall architecture of XCB is very similar to that of FAST[AES, Horner]. The main difference is that XCB requires an additional sequential AES decryption core and this results in XCB being costlier than FAST[AES, Horner] in terms of area.

2. Comparison of throughput.
   (a) Among the two-core architectures, FAST[AES,Horner]-2 has the highest throughput while among the single-core architectures, FAST[AES,BRW]-1 has the highest throughput.
   (b) As computing BRW requires about half the number of multiplications required for computing Horner, in comparison to FAST[AES,Horner]-1, a significant number of clocks can be saved in computing the first hash in case of FAST[AES,BRW]-1. As a result, the total number of clocks required by FAST[AES,BRW]-1 is smaller than that required by FAST[AES,Horner]-1 and this leads to a better throughput for FAST[AES,BRW]-1.
   (c) FAST[AES,Horner]-2 is marginally better than FAST[AES,BRW]-2 in terms of throughput. This is due to the following reason. FAST[AES,Horner]-2 uses two multipliers which compensates for the gain from the use of BRW polynomials. Overall, FAST[AES,Horner]-2 requires slightly lesser number of clocks and slightly higher frequency.
   (d) Both version of XCB operate at a lower frequency than the corresponding versions of FAST. This leads to lower throughput of XCB compared to FAST. The lower frequency of XCB is essentially due to the use of the sequential AES decryption core which is not present in the architectures for FAST. EME2 has the lowest throughput due to its reduced frequency for the use of the pipelined decryption core, which is absent in all other architectures.

3. Comparison of throughput per area (TPA) metric.
   (a) The TPA is the highest for FAST[AES,BRW]-1 which is slightly higher than that of FAST[AES,BRW]-2.
   (b) XCB has a better TPA than EME for both versions. This is due to the significantly higher operation frequency (EME is limited by the frequency of pipelined AES decryption core).

To confirm the comparative performance of the different designes, we have also obtained result for the high end modern Virtex 7 FPGA. The target devise was `xc7vx690t-3fgg1930`. The results are presented in Table 9. Based on Table 9, we make the following observations.

Construction	Area (slices)	Frequency (MHz)	Clock cycles	Throughput (Gbps)	TPA (Mbps/slice)
AES-PEC	2093	405.02	1	51.84	
AES-PDC	2352	352.19	1	45.08	
AES-SDC	1575	390.056	11	4.54	
128-bit Karatsuba Ofman multiplier	1884	404.86	1	51.82	
FAST[AES,BRW]-2	7202	375.43	330	37.28	5.17
FAST[AES,Poly]-2	8906	377.03	322	**38.37**	4.30
XCB-2	9330	358.84	316	37.21	3.99
EME2-2	11800	315.58	305	33.90	2.87
FAST[AES,BRW]-1	5024	377.87	466	26.57	5.29
FAST[AES,Poly]-1	**4781**	379.25	576	21.57	**5.56**
XCB-1	5875	360.67	569	20.77	3.54
EME2-1	6350	319.74	561	18.67	2.94

Table 9. Implementation results for Virtex 7.

1. The frequency grows significantly in comparison with Virtex 5 results. This is basically a direct effect of the difference of the fabrication technology between the two families. While Virtex 5 family is built with 65 nm technology, Virtex 7 is built with 28 nm technology.
2. The number of slices for the AES cores is significantly lesser than the corresponding implementations in Virtex 5. This is due to the fact that slices in Virtex 7 include 8 Flip-Flops which is 4 more than in Virtex 5.
3. In some cases, the number of slices grows in comparison with the Virtex 5. Examples are the 128-bit multiplier and FAST[AES,Poly]-1. This behaviour can be attributed to the optimisation performed by the tool. In general, the designs were optimised for speed. We tried many restriction and have reported the best results. During the implementation experiments, we have observed that the differences in area and frequency are really significant for two different timing restrictions. When the area increases the reason is that the tool could find better paths for routing, using more slices just to complete such paths. Another strategy to increase the performance could be to replicate some parts of the circuit leading to an increase in the area.

10 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. An FPGA based hardware implementation has been done for this application. Both the software and the hardware implementations show that the new scheme outperforms previous schemes and make the 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. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
6. Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.
7. 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.
8. Philippe Bulens, François-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin, and Gaël Rouvroy. Implementation of the aes-128 on virtex-5 fpgas. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 16–26. Springer, 2008.

9. Debrup Chakraborty, Sebati Ghosh, and Palash Sarkar. Evaluating bernstein-rabin-winograd polynomials. *IACR Cryptology ePrint Archive*, 2017:328, 2017.
10. 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.
11. Debrup Chakraborty, Vicente Hernandez-Jimenez, and Palash Sarkar. Another look at XCB. *Cryptography and Communications*, 7(4):439–468, 2015.
12. 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.
13. Debrup Chakraborty, Cuauhtemoc Mancillas-Lopez, and Palash Sarkar. Disk encryption: Do we need to preserve length? *Journal of Cryptographic Engineering*.
14. 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.
15. Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR, 2008. To appear.
16. Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006. full version available at `http://eprint.iacr.org/2007/028`.
17. 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.
18. 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.
19. 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.
20. 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.
21. Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: specification and analysis. *IACR Cryptology ePrint Archive*, 2017:168, 2017.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.
28. 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.
29. 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/`.
30. 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.
31. David A. McGrew and John Viega. Arbitrary block length mode, 2004. `http://grouper.ieee.org/groups/1619/email/pdf00005.pdf`.
32. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999.
33. Michael O. Rabin and Shmuel Winograd. Fast evaluation of polynomials by rational preparation. *Communications on Pure and Applied Mathematics*, 25:433–458, 1972.
34. 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.

35. 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.
36. Palash Sarkar. A general mixing strategy for the ECB-Mix-ECB mode of operation. *Inf. Process. Lett.*, 109(2):121–123, 2008.
37. Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Transactions on Information Theory*, 55(10):4749–4759, 2009.
38. Palash Sarkar. Tweakable enciphering schemes using only the encryption function of a block cipher. *Inf. Process. Lett.*, 111(19):945–955, 2011.
39. 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 Proof of Theorem 1

This section provides the proof of Theorem 1.

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 all queries by $\mathcal{A}$ can be answered using the oracle of $\mathcal{B}$.

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,

$$\Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right]. \tag{40}$$

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

$$\Pr\left[\mathcal{B}^{\rho(\cdot)} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right]. \tag{41}$$

So,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{F}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\rho(\cdot)} \Rightarrow 1\right] \\
&= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] \\
&\quad - \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right].
\end{aligned} \tag{42}$$

We now 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 $1 \le s \le q$, let $\mathsf{ty}^{(s)} = \mathsf{enc}$ if the s-th query is an encryption query and let $\mathsf{ty}^{(s)} = \mathsf{dec}$ 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, $\mathfrak{l}^{(s)} = \mathfrak{l}(P_3^{(s)}) = \mathfrak{l}(C_3^{(s)}) = m^{(s)} - 2$.

Note that the tweak, the plaintext and the ciphertext are random variables. We write the internal random variables corresponding to the s-th query as $A_1^{(s)}, A_2^{(s)}, B_1^{(s)}, B_2^{(s)}, Z^{(s)}, H_1^{(s)}, H_2^{(s)}, F_1^{(s)}, F_2^{(s)}$.

Let $(P_{3,1}^{(s)}, \ldots, P_{3,m^{(s)}-2}^{(s)})$ be the output of $\mathsf{format}_n(P_3^{(s)})$, and $(C_{3,1}^{(s)}, \ldots, C_{3,m^{(s)}-2}^{(s)})$ be the output of $\mathsf{format}_n(C_3^{(s)})$. Then $\mathsf{len}(P_{3,1}^{(s)}) = \cdots = \mathsf{len}(P_{3,m^{(s)}-3}^{(s)}) = n = \mathsf{len}(C_{3,1}^{(s)}) = \cdots = \mathsf{len}(C_{3,m^{(s)}-3}^{(s)})$ and $1 \le \mathsf{len}(P_{3,m^{(s)}-2}^{(s)}) = \mathsf{len}(C_{3,m^{(s)}-2}^{(s)}) \le n$.

For $1 \leq i \leq m^{(s)} - 2$, let $J_i^{(s)} = Z^{(s)} \oplus \mathsf{bin}_n(i)$. So, $C_{3,i}^{(s)} = P_{3,i}^{(s)} \oplus \rho(J_i^{(s)})$ for $1 \leq i \leq m^{(s)} - 3$; $C_{3,m^{(s)}-2}^{(s)} = P_{3,m^{(s)}-2}^{(s)} \oplus \mathsf{first}_{r^{(s)}}(\rho(J_{m^{(s)}-2}^{(s)}))$ where $r^{(s)} = \mathsf{len}(P_{3,m^{(s)}-2}^{(s)})$.

The tweak and either the plaintext or the ciphertext are chosen by the adversary $\mathcal{A}$ and hence can depend on the possible internal randomness of $\mathcal{A}$. The other random variables (except A_2 and B_1) can also depend on the internal randomness of $\mathcal{A}$ and moreover, depend on the randomness of the random oracle ρ.

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 $\mathfrak{F}$.

For $1 \leq s \leq q$, let $\mathcal{D}^{(s)}(\mathcal{A}) = \{F_1^{(s)}, F_2^{(s)}\} \bigcup \{J_i^{(s)} : 1 \leq i \leq m^{(s)} - 2\}$ and

$$\mathcal{D}(\mathcal{A}) = \left(\bigcup_{s=1}^{q} \mathcal{D}^{(s)}(\mathcal{A}) \right) \bigcup \{\mathsf{fStr}\}. \tag{43}$$

Along with fStr, the set $\mathcal{D}(\mathcal{A})$ consists of all the random variables to which ρ is applied directly during the processing of the q queries made by the adversary.

Let $\mathsf{Coll}(\mathcal{A})$ be the event that the queries made by $\mathcal{A}$ to $\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot, \cdot)$ and $\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot, \cdot)$ result in two elements in $\mathcal{D}(\mathcal{A})$ being equal. We have

$$\begin{aligned} &\Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1\right] \\ &\leq \Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1 | \overline{\mathsf{Coll}(\mathcal{A})}\right] + \Pr\left[\mathsf{Coll}(\mathcal{A})\right]. \end{aligned} \tag{44}$$

From Tables 2 and 3, we have the following.

$$\begin{aligned} C_1^{(s)} &= H_2^{(s)} \oplus A_1^{(s)} \oplus \rho(F_1^{(s)}); \\ C_2^{(s)} &= F_1^{(s)} \oplus \rho(F_2^{(s)}) \oplus h'_\tau(T^{(s)}, C_3^{(s)}); \\ P_1^{(s)} &= F_2^{(s)} \oplus \rho(F_1^{(s)}) \oplus h_\tau(T^{(s)}, P_3^{(s)}); \\ P_2^{(s)} &= H_1^{(s)} \oplus B_2^{(s)} \oplus \rho(F_2^{(s)}); \\ C_{3,i}^{(s)} &= P_{3,i}^{(s)} \oplus \rho(J_i^{(s)}), \text{ for } i = 1, \ldots, m^{(s)} - 2. \end{aligned}$$

Conditioned on the event $\overline{\mathsf{Coll}(\mathcal{A})}$, if $\mathsf{ty}^{(s)} = \mathsf{enc}$ then ρ is applied to distinct inputs and so the outputs of ρ are independent and uniform random. From this it is easy to argue using the above expressions for $C_1^{(s)}, C_2^{(s)}, C_{3,i}^{(s)}$ that these are uniform random and independent of all other variables. Similarly, if $\mathsf{ty}^{(s)} = \mathsf{dec}$ then $P_1^{(s)}, P_2^{(s)}, P_{3,i}^{(s)}$ are uniform random and independent of all other variables. So,

$$\Pr\left[\mathcal{A}^{\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot,\cdot),\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot,\cdot)} \Rightarrow 1 | \overline{\mathsf{Coll}(\mathcal{A})}\right] = \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right]. \tag{45}$$

At this point, we wish to upper bound $\Pr[\mathsf{Coll}(\mathcal{A})]$. This is the probability that $\mathcal{A}$ is able to achieve a collision in $\mathcal{D}(\mathcal{A})$. 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 $\mathsf{FAST}_\rho.\mathsf{Encrypt}(\cdot, \cdot)$ and $\mathsf{FAST}_\rho.\mathsf{Decrypt}(\cdot, \cdot)$ and define $\mathcal{D}(\mathcal{C})$ and $\mathsf{Coll}(\mathcal{C})$ in a manner analogous to $\mathcal{D}(\mathcal{A})$ and $\mathsf{Coll}(\mathcal{A})$ respectively. Clearly, we have

$$\Pr\left[\mathsf{Coll}(\mathcal{A})\right] \leq \Pr\left[\mathsf{Coll}(\mathcal{C})\right]. \tag{46}$$

So, it is sufficient to upper bound $\Pr[\mathsf{Coll}(\mathcal{C})]$. Since $\mathcal{C}$ has unbounded computational power, without loss of generality, we may assume that $\mathcal{C}$ is deterministic. So, the only randomness in the event $\mathsf{Coll}(\mathcal{C})$ arises from the randomness in ρ. In other words, we need to upper bound $\Pr[\mathsf{Coll}(\mathcal{C})]$ based on the randomness in ρ.

We have the following.

$$\begin{aligned}
&A_1^{(s)} = P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)});\\
&A_2^{(s)} = P_2^{(s)};\\
&B_1^{(s)} = C_1^{(s)};\\
&B_2^{(s)} = C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)});\\
&Z^{(s)} = A_2^{(s)} \oplus B_1^{(s)} = P_2^{(s)} \oplus C_1^{(s)};\\
&F_1^{(s)} = H_1^{(s)} \oplus A_2^{(s)} = \tau A_1^{(s)} \oplus P_2^{(s)} = \tau(P_1^{(s)} \oplus h_\tau(T^{(s)}, P_3^{(s)})) \oplus P_2^{(s)};\\
&F_2^{(s)} = B_1^{(s)} \oplus H_2^{(s)} = C_1^{(s)} \oplus \tau B_2^{(s)} = C_1^{(s)} \oplus \tau(C_2^{(s)} \oplus h'_\tau(T^{(s)}, C_3^{(s)}));\\
&J_i^{(s)} = Z^{(s)} \oplus \mathsf{bin}_n(i) = P_2^{(s)} \oplus C_1^{(s)} \oplus \mathsf{bin}_n(i), \text{ for } 1 \le i \le m^{(s)} - 2.
\end{aligned}$$

To compute $\Pr[\mathsf{Coll}(\mathcal{C})]$ we consider the probability that two different random variables in $\mathcal{D}(\mathcal{C})$ are equal.

The hash functions h and h' are respectively applied to $P_3^{(s)}$ and $C_3^{(s)}$. The number of n-bit blocks in both these strings is equal to $m^{(s)} - 2$.

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

Proof.

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

The last inequality follows from (10). □

Claim 2. *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 (11). □

Claim 3. *For* $1 \le s \le q$, $1 \le i \le m^{(s)} - 2$, $\Pr[J_i^{(s)} = \mathsf{fStr}] = 1/2^n$.

Proof.

$$\begin{aligned}
J_i^{(s)} \oplus \mathsf{fStr} &= Z^{(s)} \oplus \mathsf{bin}_n(i) \oplus \mathsf{fStr}\\
&= P_2^{(s)} \oplus C_1^{(s)} \oplus \mathsf{bin}_n(i) \oplus \mathsf{fStr}.
\end{aligned}$$

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

Claim 4. *For $s \neq t$, $\Pr[F_1^{(s)} = F_1^{(t)}] \leq \max\{\epsilon_2^{(s,t)}, 1/2^n\}$.*

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 $P_2^{(s)} \neq P_2^{(t)}$ and so $\Pr[F_1^{(s)} = F_1^{(t)}] = 0$.

(b) **Case 1b:** $(T^{(s)}, P_1^{(s)}, P_3^{(s)}) \neq (T^{(t)}, P_1^{(t)}, P_3^{(t)})$.
If $(T^{(s)}, P_3^{(s)}) = (T^{(t)}, P_3^{(t)})$, then $P_1^{(s)} \neq P_1^{(t)}$ 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 $(T^{(s)}, P_3^{(s)}) \neq (T^{(t)}, P_3^{(t)})$, then

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

The last inequality follows from (12).

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 5. *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 4.

Claim 6. *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 (14).

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 7. *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.

$$\begin{aligned}\Pr[F_1^{(s)} = J_i^{(t)}] &= \Pr[\tau(h_\tau(T^{(s)}, P_3^{(s)}) \oplus P_1^{(s)}) = P_2^{(s)} \oplus P_2^{(t)} \oplus C_1^{(t)} \oplus \mathsf{bin}_n(i)] \\ &\leq \epsilon_1^{(s)}.\end{aligned}$$

The last inequality follows from (10). □

Claim 8. *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 7.

Claim 9. *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.

$$J_i^{(s)} \oplus J_j^{(t)} = (P_2^{(s)} \oplus P_2^{(t)}) \oplus (C_1^{(s)} \oplus C_1^{(t)}) \oplus \mathsf{bin}_n(i) \oplus \mathsf{bin}_n(j).$$

If $s = t$, then $i \neq j$ and so $\Pr[J_i^{(s)} = J_j^{(t)}] = 0$. If $s \neq t$, then either $(P_2^{(s)} \oplus P_2^{(t)})$ or $(C_1^{(s)} \oplus C_1^{(t)})$ is an independent and uniform random n-bit string. Thus the claim follows. □

By Claims 1 to 9 and the union bound we have

$$\begin{aligned}
\Pr[\mathsf{Coll}(\mathcal{C})] &\leq 2\sum_{s=1}^{q}\epsilon_1^{(s)} + \sum_{s=1}^{q}\left(\frac{m^{(s)}-2}{2^n}\right) + \sum_{1\leq s<t\leq q} 2\left(\epsilon_2^{(s,t)} + \frac{1}{2^n}\right) \\
&\quad + \sum_{1\leq s\leq t\leq q}\left(\epsilon_2^{(s,t)} + \frac{1}{2^n}\right) + 2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)\left(\sum_{t=1}^{q}(m^{(t)}-2)\right) + \frac{1}{2^n}\binom{\sum_{s=1}^{q}(m^{(s)}-2)}{2} \\
&= 2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)\left(1+\sum_{t=1}^{q}(m^{(t)}-2)\right) + \frac{3}{2^n}\frac{q(q-1)}{2} + \frac{q}{2^n} + 3\sum_{1\leq s<t\leq q}\epsilon_2^{(s,t)} + \sum_{s=1}^{q}\epsilon_2^{(s,s)} \\
&\quad + \sum_{s=1}^{q}\left(\frac{m^{(s)}-2}{2^n}\right) + \frac{1}{2^n}\binom{\sum_{s=1}^{q}(m^{(s)}-2)}{2} \\
&\leq 2\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right)(1+\omega-2q) + \frac{2q^2}{2^n} + 3\sum_{1\leq s<t\leq q}\epsilon_2^{(s,t)} + \sum_{s=1}^{q}\epsilon_2^{(s,s)} \\
&\quad + \frac{\omega-2q}{2^n} + \frac{1}{2^n}\frac{\omega(\omega-1)}{2} \\
&\leq 2\omega\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right) + 3\sum_{1\leq s<t\leq q}\epsilon_2^{(s,t)} + \sum_{s=1}^{q}\epsilon_2^{(s,s)} + \frac{3\omega^2}{2^n}. \qquad (47)
\end{aligned}$$

Putting together (42), (44), (45), (46) and (47), we obtain

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{B}) \\
&\geq \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{FAST.Encrypt}_K(\cdot,\cdot),\mathsf{FAST.Decrypt}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\rho_1(\cdot,\cdot),\rho_2(\cdot,\cdot)} \Rightarrow 1\right] \\
&\quad - \left(2\omega\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right) + 3\sum_{1\leq s<t\leq q}\epsilon_2^{(s,t)} + \sum_{s=1}^{q}\epsilon_2^{(s,s)} + \frac{3\omega^2}{2^n}\right) \\
&= \mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{FAST}}(\mathcal{A}) - \left(2\omega\left(\sum_{s=1}^{q}\epsilon_1^{(s)}\right) + 3\sum_{1\leq s<t\leq q}\epsilon_2^{(s,t)} + \sum_{s=1}^{q}\epsilon_2^{(s,s)} + \frac{3\omega^2}{2^n}\right). \qquad (48)
\end{aligned}$$

Rearranging we obtain

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

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. □