Invertible Universal Hashing and the TET Encryption Mode*

Shai Halevi
IBM T.J. Watson Research Center,
Hawthorne, NY 10532, USA
shaih@alum.mit.edu

May 24, 2007

Abstract

This work describes a mode of operation, TET, that turns a regular block cipher into a length-preserving enciphering scheme for messages of (almost) arbitrary length. When using an n-bit block cipher, the resulting scheme can handle input of any bit-length between n and 2^n and associated data of arbitrary length.

The mode TET is a concrete instantiation of the generic mode of operation that was proposed by Naor and Reingold, extended to handle tweaks and inputs of arbitrary bit length. The main technical tool is a construction of invertible "universal hashing" on wide blocks, which is as efficient to compute and invert as polynomial-evaluation hash.

1 Introductions

Adding secrecy protection to existing (legacy) protocols and applications raises some unique problems. One of these problems is that existing protocols sometimes require that the encryption be "transparent", and in particular preclude length-expansion. One example is encryption of storage data "at the sector level", where both the higher-level operating system and the lower-level disk expect the data to be stored in blocks of 512 bytes, and so any encryption method would have to accept 512-byte plaintext and produce 512-byte ciphertext.

Clearly, insisting on a length-preserving (and hence deterministic) transformation has many drawbacks. Indeed, even the weakest common notion of security for "general purpose encryption" (i.e., semantic security [GM84]) cannot be achieved by deterministic encryption. Still, there may be cases where length-preservation is a hard requirement (due to technical, economical or even political constrains), and in such cases one may want to use some encryption scheme that gives better protection than no encryption at all. The strongest notion of security for a length-preserving transformation is "strong pseudo-random permutation" (SPRP) as defined by Luby and Rackoff [LR88], and its extension to "tweakable SPRP" by Liskov et al. [LRW02]. A "tweak" is an additional input to the enciphering and deciphering procedures that need not be kept secret. This report uses the terms "tweak" and "associated data" pretty much interchangeably, except that "associated data" hints that it can be of arbitrary length.

Motivated by the application to "sector level encryption", many modes of operation that implement tweakable SPRP on wide blocks were described in the literature in the last few years.

*Extended abstract appeard in the proceedings of CRYPTO 2007

Currently there are at least eight such proposals, following three different approaches: The "encrypt-mix-encrypt" approach is used for CMC, EME and EME* [HR03, HR04, Hal04], the "hash-ECB-hash" (due to Naor and Reingold [NR97]) is used in PEP [CS06b], and the "hash-CTR-hash" approach is used by XCB [FM04], HCTR [WFW05] and HCH [CS06a] (and some variation of the last approach is used in ABL4 [MV04]). Among these proposals, the "encrypt-mix-encrypt" modes are the most efficient (at least in software), the "hash-CTR-hash" modes are close behind, and PEP and ABL4 are considerably less efficient (more on efficiency in Section 3.5).

This work presents a ninth mode called TET (for linear-Transformation; ECB; linear-Transformation). TET belongs to the "hash-ECB-hash" family, but in terms of efficiency it is similar to the modes of the "hash-CTR-hash" family, thus complementing the current lineup of modes. We also mention that TET may have some practical advantage with respect to intellectual-property concerns, see further discussion in the appendix.

The main technical contribution of this work is a construction of an efficient invertible universal hashing for wide blocks, which is needed in the "hash-ECB-hash" approach. Given the wide range of applications of universal hashing in general, this invertible universal hashing may find applications beyond the TET mode itself. Another small contribution is a slight modification of the OMAC construction for pseudorandom function due to Iwata and Korasawa [IK03]. (In TET we use that pseudorandom function to handle the message-length and the tweak). This construction too can find other applications.

The Naor-Reingold construction and TET. Recall that the Naor-Reingold construction from [NR97] involves a layer of ECB encryption, sandwiched between two layers of universal hashing, as described in Figure 1. The universal hashing layers must be invertible (since they need to be inverted upon decryption), and their job is to ensure that different queries of the attacker will almost never result in "collisions" at the ECB layer. Namely, for any two plaintext vectors $\vec{p} = \langle p_1, \ldots, p_m \rangle$, $\vec{q} = \langle q_1, \ldots, q_m \rangle$ and two indexes i, j (such that $(\vec{p}, i) \neq (\vec{q}, j)$) it should hold with high probability (over the hashing key) that the i'th block of hashing $\vec{p}$ is different from the j'th block of hashing $\vec{q}$ (and similarly for any two ciphertext vectors).

As mentioned above, the main contribution of this note is a construction of an invertible universal hashing on wide blocks, which is as efficient to compute and invert as polynomial-evaluation hash. In a nutshell, the hashing family works on vectors in $\mathrm{GF}(2^n)^m$, and it is keyed by a single random element $\tau \in_R \mathrm{GF}(2^n)$, which defines the following $m \times m$ matrix:

$$A_\tau \stackrel{\text{def}}{=} \begin{pmatrix} \tau & \tau^2 & & \tau^m \\ \tau & \tau^2 & & \tau^m \\ & & \ddots & \\ \tau & \tau^2 & & \tau^m \end{pmatrix}$$

Set $\sigma \stackrel{\text{def}}{=} 1 + \tau + \tau^2 + \ldots + \tau^m$, we observe that if $\sigma \neq 0$ then the matrix $M_\tau = A_\tau + I$ is invertible and its inverse is $M_\tau^{-1} = I - (A_\tau / \sigma)$. Thus multiplying by M_τ for a random τ (subject to $\sigma \neq 0$) is an invertible universal hashing, and computing and inverting this hash function is about as efficient as computing polynomial evaluation.

The starting point of this work is an implementation of the generic Naor-Reingold mode of operation using the above for the universal hashing layers. We then extend that mode to handle associated data and input of arbitrary length, thus getting the TET mode. Specifically, TET takes

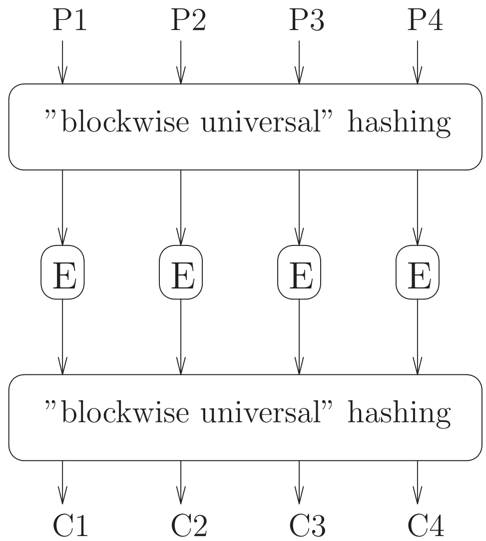

Figure 1: The Naor-Reingold generic mode: the universal hashing must be invertible, and its job is to prevent collisions in the ECB layer.

a standard cipher with n-bit blocks and turns it into a tweakable enciphering scheme with message space $\mathcal{M} = \{0,1\}^{n..2^n-1}$ (i.e., any string of at least n and at most $2^n - 1$ bits) and tweak space $\mathcal{T} = \{0,1\}^*$. The key for TET consists of two keys of the underlying cipher (roughly one to process the tweak and another to process the data). As we mentioned, TET offers the same performance characteristics as XCB, HCTR and HCH, making it significantly more efficient than PEP and ABL4, and almost as efficient as CMC, and EME/EME*.

Organization. Some standard definitions are recalled in Appendix A (which is taken almost verbatim from [HR04, Hal04]). Section 2 describes the hashing scheme that underlies TET, Section 3 describes the TET mode itself, and Section 4 contains a proof of security for this mode.

Acknowledgments. I would like to thank the participants of the IEEE SISWG working group for motivating me to write this note. I also thank Doug Whiting and Brian Gladman for some discussions about this mode, and the anonymous CRYPTO reviewers for their comments.

2 The underlying hashing scheme

The universality property that is needed for the Naor-Reingold mode of operation is defined next.

Definition 1 *Let $H : \mathcal{K} \times \mathcal{D} \to \mathcal{R}^m$ be a hashing family from some domain $\mathcal{D}$ to m-vectors over the range $\mathcal{R}$, with keys chosen uniformly from $\mathcal{K}$. We denote by $H_k(x)$ the output of H (which is an m-vector over $\mathcal{R}$) on key $k \in \mathcal{K}$ and input $x \in \mathcal{D}$. We also denote by $H_k(x)_i$ the i'th element of that output vector.*

For a real number $\epsilon \in (0,1)$, we say that $\mathcal{H}$ is "ϵ-blockwise-universal" if for every $x, x' \in \mathcal{D}$ and integers $i, i' \leq m$ such that $(x,i) \neq (x',i')$, it holds that $\Pr_k[H_k(x)_i = H_k(x')_{i'}] \leq \epsilon$, where the probability is taken over the uniform choice of $k \in \mathcal{K}$.

We say that $\mathcal{H}$ is "ϵ-xor-blockwise-universal" if in addition for all fixed $\Delta \in \mathrm{GF}(2^n)$ it holds that $\Pr_k[H_k(x)_i \oplus H_k(x')_{i'} = \Delta] \leq \epsilon$.

It was proven in [NR99] that the construction from Figure 1 is a strong PRP on wide blocks provided that the hashing layers are blockwise universal and invertible, and the underlying cipher E is a strong PRP on narrow blocks.

2.1 BPE: A blockwise universal hashing scheme

To get an invertible blockwise universal hash function, Naor and Reingold proposed in [NR97] to use an unbalanced Feistel network with standard universal hashing. For example, use polynomial-evaluation hash function applied to the first $m-1$ blocks, xor the result to the last block, and then derive $m-1$ "pairwise independent" values from the last block and xor them back to the first $m-1$ blocks. This solution, however, is somewhat unsatisfying in that it entails inherent asymmetry (which is likely to raise problems with implementations).

Below we propose a somewhat more elegant blockwise universal hashing based on a simple algebraic trick. Let $\mathcal{F}$ be a field (with more than $m+2$ elements) and consider an $m \times m$ matrix over $\mathcal{F}$, $M_\tau \stackrel{\text{def}}{=} A_\tau + I$ for an element $\tau \in \mathcal{F}$, where

$$A_\tau \stackrel{\text{def}}{=} \begin{pmatrix} \tau & \tau^2 & & \tau^m \\ \tau & \tau^2 & & \tau^m \\ & & \ddots & \\ \tau & \tau^2 & & \tau^m \end{pmatrix} \tag{1}$$

It is easy to check that the determinant of M_τ is $\sigma \stackrel{\text{def}}{=} \sum_{i=0}^m \tau^i$, and so M_τ is invertible if and only if $\sigma \neq 0$. We observe that when it is invertible, the structure of M_τ^{-1} is very similar to the structure of M_τ itself.

Observation 1 *Let $\mathcal{F}$ be a field and let $\tau \in \mathcal{F}$ be such that $\sigma \stackrel{\text{def}}{=} \sum_{i=0}^m \tau^i \neq 0$, let A_τ be an $m \times m$ matrix with $A_{i,j} = \tau^j$, and let $M_\tau \stackrel{\text{def}}{=} A_\tau + I$. Then $M_\tau^{-1} = I - (A_\tau/\sigma)$.*

Proof We first note that $A_\tau^2 = A_\tau(\sigma - 1)$, since for all i, j we have

$$(A_\tau^2)_{i,j} \;=\; \sum_{k=1}^m \tau^{k+j} \;=\; \tau^j \left(\sum_{k=1}^m \tau^k \right) \;=\; \tau^j \left(\sum_{k=0}^m \tau^k - 1 \right) \;=\; (A_\tau)_{i,j} \cdot (\sigma - 1)$$

Therefore, assuming $\sigma \neq 0$ we get

$$(A_\tau + I) \cdot (I - \frac{A_\tau}{\sigma}) \;=\; A_\tau + I - \frac{A_\tau^2}{\sigma} - \frac{A_\tau}{\sigma} \;=\; I + \frac{A_\tau \sigma - A_\tau(\sigma - 1) - A_\tau}{\sigma} \;=\; I$$

■

It follows that computing $\mathbf{y} = M_\tau \mathbf{x}$ and $\mathbf{x} = M_\tau^{-1} \mathbf{y}$ can be done as efficiently as computing polynomial-evaluation hash. Namely, to compute $\mathbf{y} = M_\tau \mathbf{x}$ we first compute $s = \sum_{i=1}^m x_i \tau^i$ and set $y_i = x_i + s$, and to invert $\mathbf{x} = M_\tau^{-1} \mathbf{y}$ we re-compute s as $s = \sum_{i=1}^m y_i(\tau^i/\sigma)$ and set $x_i = y_i - s$. Moreover, since τ and σ depend only the hashing key, one can speed up the multiplication by τ and τ/σ by pre-computing some tables (cf. [Sho96]).

The blockwise-universal family BPE. Given the observation from above, we define the hashing family BPE (for Blockwise Polynomial-Evaluation) and its inverse BPE^{-1} as follows: Let $\mathcal{F}$ be a finite field with $m+3$ or more elements.

Input: An m-vector of elements from $\mathcal{F}$, $\mathbf{x} = \langle x_1, \ldots, x_m \rangle \in \mathcal{F}^m$.
Keys: Two elements $\tau, \beta \in \mathcal{F}$, such that $\sum_{i=0}^{m} \tau^m \neq 0$.
Output: Let α be some fixed primitive element of $\mathcal{F}$, and denote by $\mathbf{b} \stackrel{\text{def}}{=} \langle \beta, \alpha\beta, \ldots, \alpha^{m-1}\beta \rangle$ the m-vector over $\mathcal{F}$ whose i'th entry is $\alpha^{i-1}\beta$. The two hash functions $\mathrm{BPE}_{\tau,\beta}(\mathbf{x})$ and $\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x})$ are defined as

$$\mathrm{BPE}_{\tau,\beta}(\mathbf{x}) \stackrel{\text{def}}{=} M_\tau \mathbf{x} + \mathbf{b} \quad \text{and} \quad \mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}) \stackrel{\text{def}}{=} M_\tau^{-1}(\mathbf{x} - \mathbf{b}) \tag{2}$$

By construction if follows that $\mathrm{BPE}^{-1}_{\tau,\beta}(\mathrm{BPE}_{\tau,\beta}(\mathbf{x})) = \mathbf{x}$ for all $\mathbf{x}$ and all τ, β (provided that $\sum_{i=0}^{m} \tau^m \neq 0$). We now prove that these two families (BPE and its inverse) are indeed "blockwise universal".

Claim 1 *Fix a finite field $\mathcal{F}$ and an integer $m \leq |\mathcal{F}| - 3$, and also fix $\mathbf{x}, \mathbf{x}' \in \mathcal{F}^m$ and indexes $i, i' \leq m$ such that $(x, i) \neq (x', i')$, and any $\delta \in \mathcal{F}$.*

(i) If $i \neq i'$ then

$$\Pr_{\tau,\beta}\left[[\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_{i'} - [\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_{i'} = \delta\right] \;=\; \Pr_{\tau,\beta}\left[[\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_{i'} - [\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_{i'} = \delta\right] \;=\; \frac{1}{|\mathcal{F}|}$$

(ii) If $i = i'$ and $\mathbf{x} \neq \mathbf{x}'$ then both

$$\Pr_{\tau,\beta}\left[[\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_{i'} - [\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_{i'} = \delta\right] \quad \text{and} \quad \Pr_{\tau,\beta}\left[[\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_{i'} - [\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_{i'} = \delta\right]$$

are bounded by $\frac{m}{|\mathcal{F}|-g}$, where $g = GCD(m+1, |\mathcal{F}|-1)$ if the characteristic of the field $\mathcal{F}$ divides $m+1$, and $g = GCD(m+1, |\mathcal{F}|-1) - 1$ otherwise.

Proof Case (i), $i \neq i'$. In this case we have $[\mathrm{BPE}_{\tau,\beta}(\mathbf{x})]_i - [\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_{i'} = (\alpha^{i-1} - \alpha^{i'-1})\beta + ((M_\tau \mathbf{x})_i - (M_\tau \mathbf{x}')_{i'})$ which is equal to any fixed δ with probability exactly $1/|\mathcal{F}|$ over the choice of $\beta \in_R \mathcal{F}$ (since α is primitive and so $\alpha^{i-1} \neq \alpha^{i'-1}$). Similarly

$$\begin{aligned}
[\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x})]_i - [\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_{i'} \;&=\; \left((I - \frac{A_\tau}{\sigma})(\mathbf{x} - \mathbf{b})\right)_i - \left((I - \frac{A_\tau}{\sigma})(\mathbf{x}' - \mathbf{b})\right)_{i'} \\
&=\; ((\frac{A_\tau}{\sigma}\mathbf{b})_i - \mathbf{b}_i) - ((\frac{A_\tau}{\sigma}\mathbf{b})_{i'} - \mathbf{b}_{i'}) + \left((I - \frac{A_\tau}{\sigma})\mathbf{x}\right)_i - \left((I - \frac{A_\tau}{\sigma})\mathbf{x}'\right)_{i'} \\
&=\; (\alpha^{i'-1} - \alpha^{i-1})\beta + \left((I - \frac{A_\tau}{\sigma})\mathbf{x}\right)_i - \left((I - \frac{A_\tau}{\sigma})\mathbf{x}'\right)_{i'}
\end{aligned}$$

where the last equality follows since $(A_\tau \mathbf{b})_i = (A_\tau \mathbf{b})_{i'}$ (because all the rows of A_τ are the same). Again, this sum equals δ with probability exactly 2^{-n}.

Case (ii), $i = i'$ **and** $\mathbf{x} \neq \mathbf{x}'$**.** In this case we have $[\mathrm{BPE}_{\tau,\beta}(\mathbf{x})]_i - [\mathrm{BPE}_{\tau,\beta}(\mathbf{x}')]_i - \delta = (x_i - x'_i - \delta) + \sum_{j=1}^{m}(x_j - x'_j)\tau^j$, which is zero only when τ is a root of this specific non-zero degree-m polynomial. Similarly for $\mathrm{BPE}^{-1}_{\tau,\beta}$ we have

$$\begin{aligned}
[\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x})]_i - [\mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}')]_i - \delta &= \left((I - \frac{A_\tau}{\sigma})(\mathbf{x} - \mathbf{b})\right)_i - \left((I - \frac{A_\tau}{\sigma})(\mathbf{x}' - \mathbf{b})\right)_i - \delta \\
= \left((I - \frac{A_\tau}{\sigma})\mathbf{x}\right)_i - \left((I - \frac{A_\tau}{\sigma})\mathbf{x}'\right)_i - \delta &= (x_i - x'_i - \delta) + \sum_{j=1}^{m} \frac{\tau^j}{\sigma}(x_j - x'_j) \\
\stackrel{*}{=} \frac{1}{\sigma}\left((x_i - x'_i - \delta)(\sum_{j=0}^{m}\tau^j) + \sum_{j=1}^{m}\tau^j(x_j - x'_j)\right) \\
= \frac{1}{\sigma}\left((x_i - x'_i - \delta) + \sum_{j=1}^{m}\tau^j((x_j - x'_j) + (x_i - x'_i - \delta))\right)
\end{aligned}$$

where the equality $\stackrel{*}{=}$ holds since $\sigma = \sum_{i=0}^{m}\tau^j$. The last expression is zero when τ is a root of the parenthesized polynomial. That polynomial is non-zero since (a) if $x_i - x'_i \neq \delta$ then it has non-zero constant term, and (b) if $x_i - x'_i = \delta$ then there is some index j such that $x_j \neq x'_j$, and thus the coefficient $((x_j - x'_j) + (x_i - x'_i - \delta))$ of τ^j is non-zero.

We conclude that for both $\mathrm{BPE}_{\tau,\beta}$ and $\mathrm{BPE}^{-1}_{\tau,\beta}$, a collision in this case implies that τ must be a root of some fixed non-zero degree-m polynomial. Such polynomials have at most m roots, and τ is chosen at random in $\mathrm{GF}(2^n)$ subject to the constraint that $\sigma \neq 0$. Since σ itself is a non-zero degree-m polynomial, then there are at least $2^n - m$ elements $\tau \in \mathrm{GF}(2^n)$ for which $\sigma \neq 0$, and so the collision probability is at most $m/(2^n - m)$.

Moreover, for most values of m we can actually show that there are fewer than m values of τ for which $\sigma = 0$. Specifically, we note that $\sigma = (\tau^{m+1} - 1)/(\tau - 1)$, so $\sigma = 0$ implies that also $\tau^{m+1} - 1 = 0$, which means that τ is an $m+1$'st root of unity in $\mathcal{F}$. We know that the number of $m+1$'st roots of unity in $\mathcal{F}$ is exactly $GCD(m+1, |\mathcal{F}|-1)$, and one of them is the trivial root $\tau = 1$. The trivial root $\tau = 1$ is also a root of σ if and only if the characteristic of $\mathcal{F}$ divides $m + 1$ (since there are $m + 1$ terms in the sum that defines σ), and all the other $m + 1$'st roots of unity are also root of σ. Hence τ is chosen at random from a set of size $|\mathcal{F}| - g$, where $g = GCD(m + 1, |\mathcal{F}| - 1)$ if the characteristic of $\mathcal{F}$ divides $m + 1$ and $g = GCD(m + 1, 2^n - 1) - 1$ otherwise. ■

Corollary 1 *Both the family* BPE *and the family* BPE^{-1} *are ϵ-xor-blockwise universal with* $\epsilon \leq \frac{m}{2^n - g}$ *where* $g = GCD(m + 1, |\mathcal{F}| - 1)$ *if the characteristic of* $\mathcal{F}$ *divides* $m + 1$ *and* $g = GCD(m + 1, |\mathcal{F}| - 1) - 1$ *otherwise.*

A variant of BPE**.** It is easy to see that the same claim can be proven also for the variant of BPE that subtracts the vector $\mathbf{b}$ before multiplying by M_τ, namely if we define

$$\widetilde{\mathrm{BPE}}_{\tau,\beta}(\mathbf{x}) \stackrel{\mathrm{def}}{=} M_\tau(\mathbf{x} - \mathbf{b}) \quad \text{and} \quad \widetilde{\mathrm{BPE}}^{-1}_{\tau,\beta}(\mathbf{x}) \stackrel{\mathrm{def}}{=} M_\tau^{-1}\mathbf{x} + \mathbf{b} \tag{3}$$

then also the hash families $\widetilde{\mathrm{BPE}}$ and $\widetilde{\mathrm{BPE}}^{-1}$ are ϵ-blockwise universal for the same ϵ.

Variable input length. Claim 1 refers only to the fixed-input length scenario, where $\text{BPE}_{\tau,\beta}$ is applied always to inputs of the same length. Similar arguments can show that the four variations BPE, BPE^{-1}, $\widetilde{\text{BPE}}$, and $\widetilde{\text{BPE}}$ are also ϵ-blockwise universal in the variable-input-length scenario, where the same τ and β are used for all the different input lengths.

One complication is that in the variable-input-length scenario, the element $\tau \in \mathcal{F}$ must be chosen such that *for all* m it holds that $1+\tau+\ldots+\tau^m \neq 0$. This can be achieved by choosing τ as a primitive element in $\mathcal{F}$, which means that it is not an $m+1$'ts root of unity for any $m < |\mathcal{F}|-2$, and therefore also not a root of $1+\tau+\ldots+\tau^m$. As the number of primitive elements in $\mathcal{F}$ is $\phi(|\mathcal{F}|-1)$ (where ϕ is Euler's totient function), it follows that in this case we choose τ from a set of size exactly $\phi(|\mathcal{F}|-1)$. Hence the collision probability for any $\mathbf{x}, \mathbf{x}'$ is bounded by $\epsilon = m/\phi(|\mathcal{F}|-1)$ where m is the length of the longer of $\mathbf{x}, \mathbf{x}'$.

3 The TET mode of operation

The BPE hashing scheme immediately implies a mode of operation for implementing a fixed-input-length, non-tweakable enciphering scheme for block-sizes that are a multiple of n bits: namely the Naor-Reingold construction from [NR97] with BPE for the hashing layers (over the field $\text{GF}(2^n)$, where n is the block size of the underlying cipher). In this section I describe how to extend this construction to get a tweakable scheme that supports arbitrary input lengths (and remains secure also when using the same key for different input lengths).

A word on notations. The TET mode uses BPE over the field $\text{GF}(2^n)$, so below we always use $\oplus$ to denote field addition (i.e., an exclusive-or), and we use $+$ to denote addition in other domains (e.g., integer addition). The sum operator $\sum$ is used below to denote characteristic-two addition, so $\sum_{i=1}^{m} x_i = x_1 \oplus x_2 \oplus \cdots \oplus x_m$.

3.1 Tweaks and variable input length

Incorporating a tweak into the basic mode turns out to be almost trivial: Instead of having the element β be part of the key, we derive it from the tweak using the underlying cipher. For example, if we are content with n-bit tweaks then we can just set $\beta \leftarrow E_K(T)$ where k is the cipher key and T is the tweak. Intuitively, this is enough since the multiples of β will be used to mask the input values before they arrive at the ECB layer, so using different pseudo-random values of β for different tweak values means that the ECB layer will be applied on different blocks.

To handle longer tweaks we can replace the simple application of the underlying cipher E with a variable-input-length cipher-based pseudo-random function (e.g., CBC-MAC, PMAC, etc.), using a key which is independent of the cipher key that is used for the ECB layer. In Section 3.3 I describe a particular CBC-MAC-like implementation that suits our needs.

The same fix can be applied also to handle variable input length: namely we derive β from both the tweak and the input length. If we are content with input length of no more than 2^ℓ and tweaks of size $n-\ell$ bits, then we can use $\beta \leftarrow E_K(L,T)$ where T is the tweak value and L is the input length, or else we can use $\beta \leftarrow \text{PRF}_K(L,T)$ for some variable-input-length pseudo-random function. As noted above, using the same hashing key for different input lengths implies that the element τ must satisfy $\sigma_m = 1 \oplus \tau \oplus \ldots \oplus \tau^m \neq 0$ for every possible input length m, and this can be ensured by choosing τ as a random primitive element in $\text{GF}(2^n)$.

3.2 Partial blocks

It appears harder to extend the mode to handle inputs whose length is not a multiple of n bits. Ideally, we would have liked an elegant way of extending BPE to handle such lengths, and then handle partial blocks in the ECB layer using ciphertext-stealing (cf. [MM82, Fig.2-23]). Unfortunately, I do not know how to extend BPE to handle input length that is not a multiple of n bits while maintaining invertability (except going back to the unbalanced Feistel idea).

Instead, I borrowed a technique that was used also in EME*: When processing an input whose length is not a multiple of n bits, one of the block cipher applications in the ECB layer is replaced with two consecutive applications of the cipher, and the middle value (between the two calls to the underlying cipher) is xor-ed to the partial block. (In addition, the partial block is added to the polynomial-evaluation, so that its value effects all the other blocks.)

In more details, let $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$ be all the full input blocks and let x_{m+1} be a partial block, $\ell = |x_{m+1}|$, $0 < \ell < n$. Instead of just computing $\mathbf{y} = \text{BPE}(\mathbf{x})$, we set the i'th full block to $y_i \leftarrow \text{BPE}(\mathbf{x})_i \oplus (x_{m+1}10..0)$, while leaving x_{m+1} itself unchanged. Then we apply the ECB layer, computing $z_i \leftarrow E_K(y_i)$ for the first $m-1$ full blocks, and computing $u \leftarrow E_K(y_m)$ and $z_m \leftarrow E_K(u)$ for the last full block. The first bits of u are then xor-ed into the partial block, setting $w_{m+1} = x_{m+1} \oplus u|_{1..\ell}$. Then we do the final BPE layer (adding $(w_{m+1}10..0)$ to each full block), thus getting $w_i \leftarrow \text{BPE}(\mathbf{z})_i \oplus (w_{m+1}10..0)$ and the TET output is the vector $w_1, \ldots, w_m, w_{m+1}$.

3.3 The PRF function

It is clear that any secure pseudo-random function can be used to derive the element β. We describe now a specific PRF, which is a slight adaptation of the OMAC construction of Iwata and Korasawa [IK03], that seems well suited for our application. The slight modification to OMAC can be thought of as constructing a "tweakable PRF", with an on-line/off-line optimization for the tweak.[1] (In our case, the input-length of TET is the "tweak" for the PRF and the tweak of TET is the input to the PRF.)

We assume that the input length of TET is less than 2^n bits, and we denote by L the input length in bits encoded as an n-bit integer. Also denote the tweak for TET (which is the input to the PRF) by $T = \langle T_1, \ldots, T_{m'} \rangle$ where $|T_1| = \cdots = |T_{m'-1}| = n$ and $1 \le |T_{m'}| \le n$.

To compute $\beta \leftarrow \text{PRF}_K(L, T)$ we first compute $X \leftarrow E_K(L)$, then compute β as a CBC-MAC of T, but before the last block-cipher application we xor either the value αX or the value $\alpha^2 X$ (depending on whether the last block is a full block or a partial block). In more details, we set $V_0 = 0$ and then $V_i \leftarrow E_K(V_{i-1} \oplus T_i)$ for $i = 1, \ldots, m'-1$. Then, if the last block is a full block ($|T_{m'}| = n$) then we set $\beta \leftarrow E_K(\alpha X \oplus V_{m'-1} \oplus T_{m'})$, and if the last block is a partial block ($|T_{m'}| < n$) then we set $\beta \leftarrow E_K(\alpha^2 X \oplus V_{m'-1} \oplus (T_{m'}10..0))$.

Notice that the only difference between this function and the OMAC construction is that OMAC does not have the additional input L and it sets $X \leftarrow E_K(0)$. Proving that this is a secure pseudo-random function is similar to the proof of OMAC [IK03], and is omitted here.

We point out that on one hand, the length L is needed only before processing the last tweak block, so this pseudo-random function is suited for streaming applications where the length of

[1] Formally there is not much difference between a "tweakable" and "non-tweakable" PRF, one can always process the tweak by concatenating it to the input. But here it is convenient to make the distinction since we can offer some tweak-specific performance optimization.

If the input length is	then these elements are bad values for τ	Bad key probability
512 bytes	$\alpha^{(2^{128}-1)/3}$, $\alpha^{2\cdot(2^{128}-1)/3}$	2^{-127}
1024 bytes	$\alpha^{i\cdot(2^{128}-1)/5}$ for $i = 1, 2, 3, 4$	2^{-126}
4096 bytes	$\alpha^{i\cdot(2^{128}-1)/257}$ for $i = 1, 2, \ldots, 256$	2^{-120}
65536 bytes	$\alpha^{i\cdot(2^{128}-1)/17}$ for $i = 1, 2, \ldots, 16$	2^{-124}

Table 1: Bad τ values for various input lengths, assuming $n = 128$

messages is not known in advance.[2] On the other hand, if used with a fixed input length (where L is known ahead of time) then the computation of X can be done off line, in which case we save one block-cipher application during the on-line phase.

3.4 Some other details

To get a fully-specified mode of operation one needs to set many other small details. Below I explain my choices for the details that I set, and describe those that are still left unspecified.

The element $\alpha \in \mathrm{GF}(2^n)$. Recall that BPE uses a fixed primitive element $\alpha \in \mathrm{GF}(2^n)$. If the field $\mathrm{GF}(2^n)$ is represented with a primitive polynomial, then this fixed element should be set as the polynomial x (or $1/x$), in which case a multiplication by α can be implemented with an n-bit shift and a conditional xor.[3]

The two hashing layers. I chose to use the same hashing keys τ, β for both hashing layers. The security of the mode does not seem to be effected by this. (In particular this has no effect on the proof in Section 4.1). On the other hand, having different keys for the two hashing layers adds a considerable burden to an implementation, especially it if optimizes the GF multiplications by preparing some tables off line.

The hashing key τ. I also chose to derive the hashing key τ from the same cipher key as the hashing key β, rather than being a separate key. (This decision is rather arbitrary, I made it because I could not see any reason to keep τ as a separate key.) Specifically, it can be set as $\tau \leftarrow \mathrm{PRF}_K(0, 0^n) = E_K(\alpha \cdot E_K(0))$. Note that this is not a duplicate of any $\mathrm{PRF}_K(L, T)$, since the input length L is always at least n bits.[4]

Of course, τ must be chosen so that for any message length m it holds that $\sigma_m \neq 0$ (where $\sigma_m = \sum_{i=0}^{m} \tau^m$). Hence if setting $\tau \leftarrow \mathrm{PRF}_K(0, 0)$ results in a bad value for τ then we can keep trying $\mathrm{PRF}_K(0, 1)$, $\mathrm{PRF}_K(0, 2)$, etc. When using TET with fixed input length (containing m complete blocks), we can just include a list of all the "bad τ values" for which $\sigma_m = 0$ with the implementation. This list is fairly easy to construct: Denoting $g = GCD(m + 1, 2^n - 1)$, when m is even the lists consists of $\alpha^{i\cdot(2^n-1)/g}$ for $i = 1, 2, \ldots, g - 1$ (where α is a primitive

[2] As explained in Section 3.5, TET is not a very good fit for such cases, but this PRF functions can perhaps be used in applications other than TET.

[3] The choice between setting $\alpha = x$ or $\alpha = 1/x$ depends on the endianess of the field representation, and it should be made so that multiplication by α requires left shift and not right shift.

[4] Setting $\tau \leftarrow E_K(0)$ would work just as well in this context, but the effort in proving it is too big for the minuscule saving in running time.

element). When m is odd it consists of the same elements and also of the element $\alpha^0 = 1$. In Table 1 we list the "bad τ values" for various input lengths assuming $n = 128$.

The approach of having a fixed list of "bad τ values" may not work as well when using TET with variable-input length. One way to handle this case is to insist on τ being a primitive element in GF(2^n), in which case we know that $\sigma_m \neq 0$ for all length m. (We can efficiently test is τ is a primitive element given the prime factorization of $2^n - 1$). But a better way of handling variable length is to allow different τ's for different input lengths. Specifically, when handling a message of with m full blocks, we try $\mathrm{PRF}_K(0,0)$, $\mathrm{PRF}_K(0,1)$, $\ldots$ and set τ to the first value for which $\sigma_m \neq 0$. It is not hard to see that this is just as secure as insisting on the same τ for all lengths (since we only use τ to argue about collisions between messages of the same length, cf. item 1 in the list on page 19 in the proof of Theorem 1).

Ordering the blocks for polynomial-evaluation. I chose to order the blocks at the input of BPE in "reverse order", evaluating the polynomial as $\sum_{i=1}^{m} x_i \tau^{m-i+1}$. The reason is to allow processing to start as soon as possible in the case where the input arrives one block at a time. We would like to use Horner's rule when computing BPE($\mathbf{x}$), processing the blocks in sequence as

$$s = (\ldots((x_1\tau \oplus x_2)\tau \oplus x_3)\tau \ldots \oplus x_m)\tau$$

which means that x_1 is multiplied by τ^m, x_2 is multiplied by τ^{m-1}, etc. Similarly when computing $\mathrm{BPE}^{-1}(\mathbf{y})$ we would implement the polynomial-evaluation as

$$s = (\ldots((y_1\tau \oplus y_2)\tau \oplus y_3)\tau \ldots \oplus y_m)(\tau/\sigma)$$

which means that y_1 is multiplied by τ^m/σ, y_2 is multiplied by τ^{m-1}/σ, etc.

The hashing direction. For each of the two hashing layers, one can use either of BPE, BPE^{-1}, $\widetilde{\mathrm{BPE}}$, or $\widetilde{\mathrm{BPE}}^{-1}$. For the encryption direction, I chose to use $\widetilde{\mathrm{BPE}}^{-1}$ for the first hashing layer and BPE^{-1} for the second layer. This means that on decryption we use BPE as the first hashing layer and $\widetilde{\mathrm{BPE}}$ for the second layer.

I chose the inverse hash function on encryption and the functions themselves on decryption because inverting the functions may be less efficient than computing them in the forward direction (since one needs to multiply also by τ/σ). In a typical implementation for storage, one would use encryption when writing to storage and decryption when reading back from storage. As most storage is optimized for read (at the expense of the less-frequent write operations), it makes sense to allocate the faster operations for read in this case too.

As for the choice between BPE and $\widetilde{\mathrm{BPE}}$, I chose to add the vector $\mathbf{b}$ in the middle, right before and after the ECB layer. The rationale here is that it is possible to do the computation $\beta \leftarrow \mathrm{PRF}_K(L, T)$ concurrently with the multiplication by M_τ (or its inverse).

Given the choices above, the specification of the TET mode is given in Figure 2. Other details that are not specified here are the choice of the underlying cipher and the block-size n, and the representation of the field GF(2^n) (including endianess issues).

function $\mathrm{PRF}_K(L, T_1 \cdots T_{m'})$ // $|L| = |T_1| = \cdots = |T_{m'-1}| = n,\ 1 \le |T_{m'}| \le n$

001 $V_0 \leftarrow 0,\ X \leftarrow E_K(L)$
002 **for** $i \leftarrow 1$ **to** $m' - 1$ **do** $V_i \leftarrow E_K(V_{i-1} \oplus T_i)$
003 **if** $|T_{m'}| = n$ **then return** $E_K(V_{m'-1} \oplus T_{m'} \oplus \alpha X)$
004 **else return** $E_K(V_{m'-1} \oplus T_{m'} \oplus \alpha^2 X)$

Algorithm $\mathrm{TET}_{K_1,K_2}(T; P_1 \cdots P_m P_{m+1})$
// $|P_1| = \cdots = |P_m| = n,\ 0 \le |P_{m+1}| < n$

101 $L \leftarrow mn + |P_{m+1}|$ // input size (bits)
102 $i = 0$
103 $\tau \leftarrow \mathrm{PRF}_{K_1}(0, i),\ \sigma \leftarrow 1 \oplus \tau \oplus \ \dots \ \oplus \tau^m$
104 **if** $\sigma = 0$ **then** $i \leftarrow i + 1$, **goto** 103
105 $\beta \leftarrow \mathrm{PRF}_{K_1}(L, T),\ SP \leftarrow 0,\ SC \leftarrow 0$

110 **for** $i \leftarrow 1$ **to** m **do** $SP \leftarrow (SP \oplus P_i) \cdot \tau$
111 $SP \leftarrow SP / \sigma$
112 **if** $|P_{m+1}| > 0$ **then**
113 $\quad SP \leftarrow SP \oplus P_{m+1}$ padded with 10..0

120 **for** $i \leftarrow 1$ **to** m **do**
121 $\quad PP_i \leftarrow P_i \oplus SP$
122 $\quad PPP_i \leftarrow PP_i \oplus \alpha^{i-1}\beta$
123 **for** $i \leftarrow 1$ **to** $m - 1$ **do**
124 $\quad CCC_i \leftarrow E_{K_2}(PPP_i)$
125 **if** $|P_{m+1}| > 0$ **then**
126 $\quad MM \leftarrow E_{K_2}(PPP_m)$
127 $\quad CCC_m \leftarrow E_{K_2}(MM)$
128 $\quad C_{m+1} \leftarrow P_{m+1} \oplus (MM$ truncated$)$
129 **else** $CCC_m \leftarrow E_{K_2}(PPP_m)$

130 **for** $i \leftarrow 1$ **to** m **do**
131 $\quad CC_i \leftarrow CCC_i \oplus \alpha^{i-1}\beta$
132 $\quad SC \leftarrow (SC \oplus CC_i) \cdot \tau$
133 $SC \leftarrow SC / \sigma$
134 **if** $|P_{m+1}| > 0$ **then**
135 $\quad SC \leftarrow SC \oplus C_{m+1}$ padded with 10..0

140 **for** $i \leftarrow 1$ **to** m **do**
141 $\quad C_i \leftarrow CC_i \oplus SC$

150 **return** $C_1 \dots C_m C_{m+1}$

Algorithm $\mathrm{TET}^{-1}_{K_1,K_2}(T; C_1 \cdots C_m C_{m+1})$
// $|C_1| = \cdots = |C_m| = n,\ 0 \le |C_{m+1}| < n$

201 $L \leftarrow mn + |C_{m+1}|$ // input size (bits)
202 $i = 0$
203 $\tau \leftarrow \mathrm{PRF}_{K_1}(0, i),\ \sigma \leftarrow 1 \oplus \tau \oplus \ \dots \ \oplus \tau^m$
204 **if** $\sigma = 0$ **then** $i \leftarrow i + 1$, **goto** 203
205 $\beta \leftarrow \mathrm{PRF}_{K_1}(L, T),\ SP \leftarrow 0,\ SC \leftarrow 0$

210 **for** $i \leftarrow 1$ **to** m **do** $SC \leftarrow (SC \oplus C_i) \cdot \tau$
212 **if** $|C_{m+1}| > 0$ **then**
213 $\quad SC \leftarrow SC \oplus C_{m+1}$ padded with 10..0

220 **for** $i \leftarrow 1$ **to** m **do**
221 $\quad CC_i \leftarrow C_i \oplus SC$
222 $\quad CCC_i \leftarrow CC_i \oplus \alpha^{i-1}\beta$
223 **for** $i \leftarrow 1$ **to** $m - 1$ **do**
224 $\quad PPP_i \leftarrow E^{-1}_{K_2}(CCC_i)$
225 **if** $|C_{m+1}| > 0$ **then**
226 $\quad MM \leftarrow E^{-1}_{K_2}(CCC_m)$
227 $\quad PPP_m \leftarrow E^{-1}_{K_2}(MM)$
228 $\quad P_{m+1} \leftarrow C_{m+1} \oplus (MM$ truncated$)$
229 **else** $PPP_m \leftarrow E^{-1}_{K_2}(CCC_m)$

230 **for** $i \leftarrow 1$ **to** m **do**
231 $\quad PP_i \leftarrow PPP_i \oplus \alpha^{i-1}\beta$
232 $\quad SP \leftarrow (SP \oplus PP_i) \cdot \tau$
234 **if** $|C_{m+1}| > 0$ **then**
235 $\quad SP \leftarrow SP \oplus P_{m+1}$ padded with 10..0

240 **for** $i \leftarrow 1$ **to** m **do**
241 $\quad P_i \leftarrow PP_i \oplus SP$

250 **return** $P_1 \dots P_m P_{m+1}$

Figure 2: Enciphering and deciphering under TET, with plaintext $P = P_1 \dots P_m P_{m+1}$, ciphertext $C = C_1 \cdots C_m C_{m+1}$, and tweak T. The element $\alpha \in \mathrm{GF}(2^n)$ is a fixed primitive element.

Mode	CMC	EME*	XCB	HCH	TET
Block-cipher calls	$2m+1$	$2m+1+\lceil m/n \rceil$	$m+1$	$m+3$	$m+1$
GF multiplies	–	–	$2(m+3)$	$2(m-1)$	$2m$ or $2(m-1)$

Table 2: Workload for enciphering an m-block input with a 1-block tweak.

3.5 Performance of TET

As specified above, the TET mode can be used with variable input length, and in Section 4 we prove that it is secure when used in this manner. However, its efficiency (at least in software) depends crucially on pre-processing that is only possible when used with fixed input length (or at least with a small number of possible lengths). The reason is that on encryption one needs to multiply by τ/σ, which depends on the message length (since $\sigma = \sum_{i=0}^{m} \tau^i$). When used with fixed input length, the value τ/σ can be computed off line, and some tables can be derived to speed up the multiplication by τ/σ. When used with variable input length, however, the value τ/σ must be computed on-line, which at least for software implies a considerable cost. Hence, TET is not very appealing as a variable-input-length mode.

We stress, however, that the motivating application for TET, namely "sector-level encryption", is indeed a fixed-input-length application. Also, there are some limited settings where one can use variable input length without suffering much from the drawback above. For example, a "write once / read many times" application, where the data is encrypted once and then decrypted many times, would only need to worry about computing σ in the initial encryption phase (since σ is not used during decryption). Also, the same value of σ is used for every bit-length from mn to $(m+1)n-1$, so length variability within this limited range in not effected.[5]

Below we analyze the performance characteristics of TET only for fixed input length. With this assumption, the computation of the PRF function from above takes exactly m' applications of the cipher, where m' is the number of blocks of associated data (full or partial). (This is because the computation of the mask value $X \leftarrow E_K(L)$ can be done off line.) Then we need either m or $m-1$ GF-multiplies for the polynomial evaluation (depending if we have m or $m-1$ full blocks), followed by m block-cipher applications for the ECB layer, and again m or $m-1$ GF multiplies. Altogether, we need $m+m'$ block-cipher applications and either $2m$ or $2m-2$ GF multiplies. (The shift and xor operations that are also needed are ignored in this description, since they are insignificant in comparison.)

Table 2 compares the number of block-cipher calls and GF multiplies in CMC, EME*, XCB, HCH, and TET.[6] It is expected that software efficiency will be proportional to these numbers. (As far as I know, the current "common wisdom" is that computing a GF(2^{128}) multiplication in software using the approach from [Sho96] with reasonable-size tables, is about as fast as a single application of AES-128.)

As for hardware implementations, all the modes except CMC are parallelizable and pipelinable, so they can be made to run as fast as needed using sufficiently large hardware. Table 3 describes a somewhat speculative efficiency comparison of hypothetical "fully pipelined" implementations of the the modes from above (except CMC). In that table I assume (following [YMK05]) that

[5] For example, an implementation can handle both 512-byte blocks and 520-byte blocks with a single value of σ (assuming block length of $n = 128$ bits).

[6] The other modes are not included since EME is essentially a special case of EME*, PEP an ABL4 are significantly less efficient than the others, and HCTR is almost identical to HCH.

Mode	EME*	XCB	HCH	TET
Latency	$m + 30$	$m + 13$	$m + 31$	$2m + 11$
Time	$2m + 10(\lceil m/n \rceil + 2)$	$2m + 27$	$2m + 31$	$2m + 11$
Size	10	13	13	13

Table 3: Hardware efficiency: A speculative comparison of pipelined implementations for m-block input and 1-block tweak. Latency is number of cycles until first output block, time is number of cycles until last output block, and size is measured in the equivalent of number of AES-round modules.

a one-cycle GF(2^{128}) multiplication is about three times the size of a module for computing the AES round function and that AES-128 is implemented as 10 such modules. A few other relevant characteristics of these modes are discussed next.

Any input length. All of these modes except CMC support any input length from n bits and up. CMC supports only input length which is a multiple of n bits (but it should be relatively straightforward to extended it using ciphertext-stealing).

Associated data. The modes EME*, XCB and TET support tweaks of arbitrary length. CMC and HCH support only n-bit tweaks (but it is straightforward to extended them to support arbitrary-size tweaks).

Security proofs. The security of XCB was not proved formally, only a sketch was given, and CMC was only proven secure with respect to fixed input-length. The other modes were proven secure with respect to variable input-length. Providing the missing proof seems fairly straightforward to me (but one never knows for sure until the proof is actually written).

Number of keys. Although in principle it is always possible to derive all the needed key material from just one key, different modes are specified with different requirements for key material. In fact using two keys as in TET (one for the ECB layer and one for everything else) offers some unexpected practical advantages over using just one key.

Specifically, implementations sometimes need to be "certified" by standard bodies (such as NIST), and one criterion for certification is that the implementation uses an "approved mode of operation" for encryption. Since standard bodies are slow to approve new modes, it may be beneficial for a TET implementation to claim that it uses the "approved" ECB mode with pre-and post-processing, and moreover the pre- and post-processing is independent of the ECB key. Also, detaching the ECB key from the key that is used elsewhere may make it easier to use a hardware accelerator that supports ECB mode.

3.6 Roads not taken

Before moving to the security analysis, I would like to explicitly discuss some alternatives to the design choices that I made in TET.

Unbalanced Feistel. As mentioned above, the original note of Naor and Reingold [NR97] proposed using unbalanced Feistel to get an invertible blockwise-universal hashing. This is also somewhat similar to the approach that was taken in the "hash-ECB-hash" modes. The main reason that I did not choose this approach was that BPE looks more elegant to me, but it is clear that one can devise a workable mode using the unbalanced Feistel idea.

Using BPE **also for the tweak.** Rather than processing the tweak with the PRF function, one could also process it via the polynomial evaluation, similarly to the way this is done in XCB. Namely, instead of computing $s \leftarrow \sum_{i=0}^{m-1} x_{i+1}\tau^{m-i}$, we can set $s' \leftarrow s \oplus \sum_{i=0}^{\ell-1} t_{i+1}\tau^{m+\ell-i}$, where $T = \langle t_1, \dots t_\ell \rangle$ is the tweak. (It should be clear that this has no effect on the invertability of the hash function.)

An advantage of using this alternative approach is that we do away with the need to specify also a PRF function. Some drawbacks of this approach, however, is that it leads to slightly weaker security bounds (since the polynomial is of higher degree and so it may have more roots), and more importantly it is no longer possible to process the tweak and the input concurrently.

4 Security of TET

We relate the security of TET to the security of the underlying primitives from which it is built as follows:

Theorem 1 [TET security] *Fix $n, s \in \mathbb{N}$. Consider an adversary attacking the* TET *mode with a truly random permutation over $\{0,1\}^n$ in place of the block cipher and a truly random function instead of* PRF*, such that the total length of all the queries that the attacker makes is at most s blocks altogether.*

The advantage of this attacker in distinguishing TET *from a truly random tweakable length-preserving permutation is at most $1.5s^2/\phi(2^n-1)$ (where ϕ is Euler's totient function). Using the notations from Appendix A, we have*

$$\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{TET}}(s) \le \frac{3s^2}{2\phi(2^n-1)}$$

A minor comment. Note that the value $\phi(2^n-1)$ in the denominator refers to a variable-input-length implementation where the hashing key τ is chosen as a primitive element in GF(2^n). As we explained in Section 3.4, it is probably better to allow using different values of τ for different lengths, in which case we choose τ from a set larger than the primitive elements, and the denominator improves accordingly. (Note that for $n = 128$ we have $\phi(2^{128}-1) \approx 2^{-127}$, so using the weaker bound costs us only about a factor of two in the probability.)

Corollary 2 *With the same setting as in Theorem 1, consider an attacker against* TET *with a specific cipher E and a specific* PRF *F, where the attack uses at most total of s' blocks of associated data. Then*

$$\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{TET}[E]}(t,s,s') \;\; \le \;\; \frac{3s^2}{2\phi(2^n-1)} + \; 2\big(\, \mathbf{Adv}^{\pm\mathrm{prp}}_{E}\left(t',s\right) + \mathbf{Adv}^{\mathrm{prf}}_{\mathrm{PRF}}\left(t',s'\right)\big)$$

where $t' = t + O(n(s+s'))$. □

4.1 Proof of Theorem 1

The intuition for the proof is that as long as there are no block collisions in the hash function, then the random permutation in the ECB layer will be applied to new blocks, so it will will output random blocks and the answer that the attacker will see is therefore random. To make this formal, we go through the usual "game hopping" argument as follows:

A random process. Fix the parameters $n, s \in \mathbb{N}$ and an attacker A that makes queries of total length at most s blocks altogether (full and partial). Assume (wlog) that the attacker A never makes a query for which it already knows the answer (such as a duplicate of previous query, a decryption of some value that it got from the encryption oracle with the same tweak, etc.).

Given this assumption, consider a random process in which all the queries of the attacker are answered with just random and independent bits. Clearly this experiment differs from a truly random tweakable permutation only in that it can return the same output on different queries, which happens rarely (we will account for this difference at the end of the proof). For the rest of the proof we show that interacting with TET is indistinguishable from interacting with this random process (upto the specified error).

We first describe a specific implementation of the random process. Namely, we choose at random some element $\tau \in \mathrm{GF}(2^n)$ subject to the condition that $1 \oplus \tau \oplus \ldots \oplus \tau^m \neq 0$ for all valid input-lengths m (this can be done, e.g., by choosing τ as a primitive element). Then for any encryption query (of length L and with tweak T), we choose at random $\beta_{L,T}$ (or use a previous value if these T, L where used before) and then simulate the "second half" of the TET mode as follows: We choose at random all the blocks that are supposed to be the output of the underlying block cipher (i.e., the CCC blocks on encryption or the PPP blocks on decryption, and in addition the MM block if there is a partial block). Given these random bits and the values of τ and $\beta_{L,T}$ we compute the output just as it is done by the TET mode itself. Note that for any fixed τ (such that $\sigma \neq 0$) and any $\beta_{L,T}$, the transformation from the cipher-outputs to the output of the mode is bijective. It follows that since the "cipher outputs" are all chosen at random then the output that is returned to the adversary is just L random bits, so this is indeed an implementation of the random process.

Next we add to this implementation of the random process a notion of a "bad event". To define this event, we add to the implementation as described above also the "first half" of TET. Namely, after setting τ and $\beta_{L,T}$, we also compute the PPP blocks on encryption or the CCC blocks on decryption. Note that these blocks are uniquely determined by all the choices that we did before, and that at this point we do not modify any of the other values (so in particular this is still an implementation of the random process). Roughly, we define the "bad event" to be the case where any of the block values already appeared earlier in the execution. Specifically we consider the following conditions:

- One of the PPP blocks equals any previous PPP or MM block.
- One of the CCC blocks equals any previous CCC or MM block.
- The MM block equals any previous PPP, CCC or MM block.

Subroutine Choose-$\pi(X)$:

$Y \xleftarrow{\$} \{0,1\}^n$; **if** $Y \in \text{Range}$ **then** $bad \leftarrow \texttt{true}$ $, Y \xleftarrow{\$} \overline{\text{Range}}$

if $X \in \text{Domain}$ **then** $bad \leftarrow \texttt{true}$ $, Y \leftarrow \pi(X)$

$\pi(X) \leftarrow Y$, $\text{Domain} \leftarrow \text{Domain} \cup \{X\}$, $\text{Range} \leftarrow \text{Range} \cup \{Y\}$; **return** Y

Subroutine Choose-$\pi^{-1}(Y)$:

$X \xleftarrow{\$} \{0,1\}^n$; **if** $X \in \text{Domain}$ **then** $bad \leftarrow \texttt{true}$ $, X \xleftarrow{\$} \overline{\text{Domain}}$

if $Y \in \text{Range}$ **then** $bad \leftarrow \texttt{true}$ $, X \leftarrow \pi^{-1}(Y)$

$\pi(X) \leftarrow Y$, $\text{Domain} \leftarrow \text{Domain} \cup \{X\}$, $\text{Range} \leftarrow \text{Range} \cup \{Y\}$; **return** X

Figure 3: The procedures for choosing π values in the random and TET processes. The shaded statements are executed in TET but not in the random process.

The TET **process.** We now modify the random process, starting by just re-arranging the computation of the various blocks to match the order in which they are assigned values by the TET mode. Namely, on encryption we first compute the PPP blocks, then choose the MM block (if needed) and then choose the CCC blocks. Similarly, on decryption we first compute the CCC blocks, then choose the MM block (if needed) and then choose the PPP blocks.

Also, we introduce a table π (which is meant to eventually hold a "random permutation", but for now does yet not have any semantics): When processing a query, we fill the entry corresponding to $\pi(PPP_i)$ with the value of CCC_i for all $i < m$, and for $i = m$ we do as above if there is no partial block and otherwise we set $\pi(PPP_m) \leftarrow MM$ and $\pi(MM) \leftarrow CCC_m$. Note that so far π may not correspond to any well-defined function (since we may reset entries of π), and also there may be duplicate entries in it. However, both of these cases only happen if the "bad event" from above occurs.

So far we still did not change the random process at all. Next we modify this process, so that when the "bad event" happens it ensures that π is consistent with a well-defined permutation. Specifically, we do the following changes:

- On encryption, whenever we pick a new value for any CCC_i, $i < m$, we check if the entry $\pi(PPP_i)$ is already defined, and if it is we use that value for the value CCC_i rather than choosing a new random value. We do a similar test before picking a value for CCC_m and for MM (if applicable) by checking either $\pi(PPP_m)$ or $\pi(MM)$, as appropriate.

- Still on encryption, if the entry of π is not defined yet, then we pick a new random value for the CCC or MM block, and then we check to see if this value already exists in the table. If if does, we try again, this time choosing it at random from the co-range (i.e., from the set of values that are not yet in the table π).

- On decryption, whenever we pick a new value for any PPP or MM block, $i < m$, we check if the corresponding entry $\pi(PPP)$ or $\pi(MM)$ is already defined, and if it is we reset our choice, this time choosing at random from the co-domain (i.e., , from the set of values X for which $\pi(X)$ is not defined yet).

- Still on decryption, after choosing PPP_i for $i < m$ we check if the value of CCC_i already appears in the table π, and if it does we reset the value of PPP_i to $\pi^{-1}(CCC_i)$ (i.e, the

Initialization:

Domain $\leftarrow$ Range $\leftarrow \emptyset$; $bad \leftarrow$ false
for all $X \in \{0,1\}^n$ **do** $\pi(X) \leftarrow$ undef
for all $T, L \in \{0,1\}^*$ **do** $\beta_{L,T} \leftarrow$ undef
$\tau \xleftarrow{\$} \{0,1\}^n$ subject to $1 \oplus \tau \oplus \ldots \oplus \tau^m \neq 0$ for all m

Respond to the j-th adversary query as follows:

Encryption query $(T^j; P_1^j \cdots P_{m^j}^j P_{m^j+1}^j)$

$L^j \leftarrow m^j n + |P_{m^j+1}^j|$ // input size (bits)
if $\beta_{L^j,T^j} =$ undef **then** $\beta_{L^j,T^j} \xleftarrow{\$} \{0,1\}^n$
$\sigma^j \leftarrow 1 \oplus \tau \oplus \ldots \oplus \tau^{m^j}$
$SP^j \leftarrow 0$, $SC^j \leftarrow 0$

for $i \leftarrow 1$ **to** m^j **do** $SP^j \leftarrow (SP^j \oplus P_i^j) \cdot \tau$
$SP^j \leftarrow SP^j / \sigma^j$
if $|P_{m^j+1}^j| > 0$ **then**
  $SP^j \leftarrow SP^j \oplus P_{m^j+1}^j$ padded with 10..0

for $i \leftarrow 1$ **to** m^j **do**
  $PP_i^j \leftarrow P_i^j \oplus SP^j$
  $PPP_i^j \leftarrow PP_i^j \oplus \alpha^{i-1} \beta_{L^j,T^j}$
for $i \leftarrow 1$ **to** $m^j - 1$ **do**
  $CCC_i^j \leftarrow$ Choose-$\pi(PPP_i^j)$
if $|P_{m^j+1}^j| > 0$ **then**
  $MM^j \leftarrow$ Choose-$\pi(PPP_{m^j}^j)$
  $CCC_{m^j}^j \leftarrow$ Choose-$\pi(MM^j)$
  $C_{m^j+1}^j \leftarrow P_{m^j+1}^j \oplus (MM^j$ truncated)
else $CCC_{m^j}^j \leftarrow$ Choose-$\pi(PPP_{m^j}^j)$

for $i \leftarrow 1$ **to** m^j **do**
  $CC_i^j \leftarrow CCC_i^j \oplus \alpha^{i-1} \beta_{L^j,T^j}$
  $SC^j \leftarrow (SC^j \oplus CC_i^j) \cdot \tau$
$SC^j \leftarrow SC^j / \sigma^j$
if $|P_{m^j+1}^j| > 0$ **then**
  $SC^j \leftarrow SC^j \oplus C_{m^j+1}^j$ padded with 10..0

for $i \leftarrow 1$ **to** m^j **do**
  $C_i^j \leftarrow CC_i^j \oplus SC^j$

return $C_1^j \ldots C_{m^j}^j C_{m^j+1}^j$

Decryption query $(T^j; C_1^j \cdots C_{m^j}^j C_{m^j+1}^j)$

$L^j \leftarrow m^j n + |C_{m^j+1}^j|$ // input size (bits)
if $\beta_{L^j,T^j} =$ undef **then** $\beta_{L^j,T^j} \xleftarrow{\$} \{0,1\}^n$
$SP^j \leftarrow 0$, $SC^j \leftarrow 0$

for $i \leftarrow 1$ **to** m^j **do** $SC^j \leftarrow (SC^j \oplus C_i^j) \cdot \tau$
if $|C_{m^j+1}^j| > 0$ **then**
  $SC^j \leftarrow SC^j \oplus C_{m^j+1}^j$ padded with 10..0

for $i \leftarrow 1$ **to** m^j **do**
  $CC_i^j \leftarrow C_i^j \oplus SC^j$
  $CCC_i^j \leftarrow CC_i^j \oplus \alpha^{i-1} \beta_{L^j,T^j}$
for $i \leftarrow 1$ **to** $m^j - 1$ **do**
  $PPP_i^j \leftarrow$ Choose-$\pi^{-1}(CCC_i^j)$
if $|C_{m^j+1}^j| > 0$ **then**
  $MM^j \leftarrow$ Choose-$\pi^{-1}(CCC_{m^j}^j)$
  $PPP_{m^j}^j \leftarrow$ Choose-$\pi^{-1}(MM^j)$
  $P_{m^j+1}^j \leftarrow C_{m^j+1}^j \oplus (MM^j$ truncated)
else $PPP_{m^j}^j \leftarrow$ Choose-$\pi^{-1}(CCC_{m^j}^j)$

for $i \leftarrow 1$ **to** m^j **do**
  $PP_i^j \leftarrow PPP_i^j \oplus \alpha^{i-1} \beta_{L^j,T^j}$
  $SP^j \leftarrow (SP^j \oplus PP_i^j) \cdot \tau$
if $|C_{m^j+1}^j| > 0$ **then**
  $SP^j \leftarrow SP^j \oplus P_{m^j+1}^j$ padded with 10..0

for $i \leftarrow 1$ **to** m^j **do**
  $P_i^j \leftarrow PP_i^j \oplus SP^j$

return $P_1^j \ldots P_{m^j}^j P_{m^j+1}^j$

Figure 4: The random TET processes. The only differences between them is in the implementation of Choose-π and Choose-π^{-1} procedures from Figure 3.

unique value such that $\pi(PPP_i) = CCC_i$). We similarly check after choosing the value for the MM block (if appropriate) and for PPP_m.

If any of these checks forces us to reset our choices, then we record that the "bad event" occurred. Also, we always update the table π with the new values that were chosen.

The resulting process proceeds just like the TET mode (with the cipher replaced by a random permutation), so we call it the TET process. Figure 3 contains a pseudo-code describing the way the random and TET processes pick values for π. In that figure, the shaded statements are executed in the TET process but not the random process. Figure 4 include pseudo-code for the random and TET processes (which are identical except for the differences in choosing values of π). In this pseudo-code, all the quantities that correspond to processing the j'th query of the adversary are marked with superscript j. (For example, the query contains m^j full blocks, the plaintext is $P_1^j, \ldots, P_{m^j}^j, P_{m^j+1}^j$, etc.)

Since the random and TET processes differ only when the bad event occurs, then the attacker A can only distinguish between them when that event occurs. Namely we have

$$\left| \Pr_{\text{random}}[A \Rightarrow 1] - \Pr_{\text{TET}}[A \Rightarrow 1] \right| \leq \Pr_{\text{random}}[bad = \mathsf{true}] \tag{4}$$

The rest of the proof is therefore devoted to bounding the probability that the bad event occurs (in the random process).

Bounding the bad-event probability. We partition the bad event into two sub-events: one is where we choose a block at random and discover that the same value was already used before (cf. lines 010 and 020 in Figure 3), and the other where it turns out that the point for which we intend to define π or its inverse is already defined (cf. lines 011 and 021 in Figure 3). In more details, we have the following two events:

Bad1. A value of CCC or MM that we choose on encryption is equal to some previous CCC or MM value, or a value of PPP or MM that we choose on decryption is equal to some previous PPP or MM value.

Bad2. On encryption, some values of PPP that are computed are equal to a previous PPP or MM value, or the MM value is equal to a previous PPP value. On decryption, some values of CCC that are computed are equal to a previous CCC or MM value, or the MM value is equal to a previous CCC value.

Bounding the probability of the Bad1 sub-event is easy. Over the course of the attack we choose at random at most s blocks, and each of them is chosen as a random n-bit string. Hence

$$\Pr_{\text{random}}[\mathsf{Bad1}] \leq \frac{\binom{s}{2}}{2^n} \tag{5}$$

To bound the probability of the Bad2 sub-event we use the blockwise-universality of the BPE function. Before we can do that, however, we must address the problem that the input to the BPE hash function may not appear to be independent of the hashing key. (This is because the input to the next query is chosen by the adversary after seeing the output from the previous one, which depends on the hashing key.) We observe that in fact the input to the hash function is independent of its key, since even when the hashing keys are all fixed, the random process still returns uniformly random bits to the adversary.

We can make this even more explicit by switching back to the implementation of the random process in which the answers to the attacker are chosen at random and the other quantities are computed from them (and from the hashing keys). In this alternative implementation we can choose all the random bits that the attacker sees ahead of time (thus fixing also the queries of the attacker) and only then choose the hashing keys.[7] Once we are back in this implementation of the random process, we observe the following:

1. Two blocks that correspond to queries of different lengths or with different tweaks collide with probability 2^{-n}, since in this case we use different random values for the $\beta_{L,T}$ hashing key.

2. Collisions of the form $MM^j = PPP_i^{j'}$ or $MM^j = CCC_i^{j'}$ also have probability 2^{-n}, since the PPP or CCC block depends on the $\beta_{L,T}$ that was used in query j' while MM^j does not.

3. Collisions of the form $PPP_i^j = PPP_{i'}^{j'}$ where $i \neq i'$ have probability 2^{-n}, even for the same query length and the same tweak (this is Case (i) in Claim 1). The same holds for $CCC_i^j = CCC_{i'}^{j'}$ with $i \neq i'$.

4. Collisions of the form $PPP_i^j = PPP_i^{j'}$ where queries j, j' have the same length (with m full blocks) and the same tweak, occur with probability at most $m/\phi(2^n-1)$ (this is Case (ii) in Claim 1). The same holds for $CCC_i^j = CCC_i^{j'}$.

Note that the last two items above actually use the fact that the hash functions are xor-universal (not just universal), since when partial blocks are present then the attacker by selecting partial block $P_{m+1}^j\ P_{m+1}^{j'}$ can add the constant $\Delta = (P_{m+1}^j 10..0) \oplus (P_{m+1}^{j'} 10..0)$ to the sum $PPP_i^j \oplus PPP_{i'}^{j'}$.

We therefore have bounds for all the possible collision events in $\mathsf{Bad2}$, and the only thing left is to sum them up using the union bound. Clearly, we have exactly $\binom{s}{2}$ potential collision events, but now some of these events have probability of upto $m/\phi(2^n-1)$ (where m is the length of the relevant query), while the others are still bounded by 2^{-n}.

Denote by $\mathcal{M}$ the set of message lengths that the attacker used in the attack (expressed in number of full blocks in the query). For any $m \in \mathcal{M}$, let q_m be the number of queries that have exactly m full blocks, and denote $s_m = m \cdot q_m$. (That is, s_m is the total number of full blocks in all the queries that have m full blocks.) Then we know that $\sum_{m \in \mathcal{M}} s_m \leq s$. The number of potential collision events corresponding to item 4 above for messages with m full blocks is at most $m \cdot \binom{q_m}{2} = m \cdot \binom{s_m/m}{2} \leq \binom{s_m}{2}/m$, and the probability of all these events sums up to at most $\frac{\binom{s_m}{2}}{m} \cdot \frac{m}{\phi(2^n-1)} = \frac{\binom{s_m}{2}}{\phi(2^n-1)}$. Recalling that $\sum_m s_m \leq s$ we get that the sum over all the potential collision events corresponding to item 4 (over all message lengths) is at most

$$\frac{\sum_{m \in \mathcal{M}} \binom{s_m}{2}}{\phi(2^n-1)} \leq \frac{\binom{s}{2}}{\phi(2^n-1)}$$

[7] See the comment after the proof for some further discussion of this argument.

Clearly, the sum over all the other potential collision events in $\mathsf{Bad2}$ is at most $\binom{s}{2}/2^n$, so the total probability of the event $\mathsf{Bad2}$ is bounded by

$$\Pr_{\text{random}}[\mathsf{Bad2}] \le \frac{\binom{s}{2}}{2^n} + \frac{\binom{s}{2}}{\phi(2^n - 1)} \tag{6}$$

Putting this all together we get that the advantage of the adversary in distinguishing between the TET process and the random process is at most $2 \cdot \frac{\binom{s}{2}}{2^n} + \frac{\binom{s}{2}}{2^n - \phi(2^n-1)}$.

We still need to account for the distinguishing probability between the random process and a random tweakable permutation, but we note that the distinguishing event (where the random process returns the same answer on two different queries) implies that the bad event occurred in the random process, so we do not need to count it again. Thus, the total advantage of the attacker can be bounded by

$$2 \cdot \frac{\binom{s}{2}}{2^n} + \frac{\binom{s}{2}}{\phi(2^n - 1)} < \frac{3s^2}{2\phi(2^n - 1)}$$

■

A comment. At a first glance, one may feel uneasy about the argument that switches back to the view of the random process as choosing the answers to the attacker at random and computing the relevant *CCC*, *PPP* and *MM* blocks from that output (after Eq. (5)), since in this implementation it is not clear that our bound on the probability of $\mathsf{Bad1}$ holds. Indeed, the various blocks are now computed using a hash function that is only ϵ-blockwise-universal for some $\epsilon > 2^{-n}$, so how can we claim that the previous bound that we derived by assigning 2^{-n} probability for each collision event still holds in this case?

A formalistic answer is that because these hash functions are bijective, then the two implementations induce identical probability spaces over their variables, so the bound that we proved with respect to one implementation must hold also for the other. A more informative answer is that in this case we do not need to look at the collision probability for any two messages, but only at collisions between a fixed message and another message that is chosen at random. Indeed, it is not hard to see that in this case the "collision probability" that we get is exactly 2^{-n} (and this indeed follows just from the fact that the hash function is bijective, regardless of its "universality").

5 Conclusions

We presented a new method for invertible "blockwise universal" hashing which is about as efficient as polynomial-evaluation hash, and used it in a construction of a tweakable enciphering scheme called TET. This complements the current lineup of tweakable enciphering schemes by providing a scheme in the family of "hash-ECB-hash" which is as efficient as the schemes in the "hash-CTR-hash" family. We also expect that the hashing scheme itself will find other uses beyond TET.

References

[CS06a] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In *INDOCRYPT'06*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006.

[CS06b] Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In *The 13th International Workshop on Fast Software Encryption – FSE'06*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.

[FM04] Scott R. Fluhrer and David A. McGrew. The extended codebook (XCB) mode of operation. Technical Report 2004/278, IACR ePrint archive, 2004. `http://eprint.iacr.org/2004/278/`.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Hal04] Shai Halevi. EME*: extending EME to handle arbitrary-length messages with associated data. In *INDOCRYPT'04*, volume 3348 of *LNCS*, pages 315–327. Springer, 2004.

[Hal07] Shai Halevi. Invertible Universal Hashing and the TET Encryption Mode. In *Advances in Cryptology – CRYPTO '07*, 2007. Long version available on-line at `http://eprint.iacr.org/2007/014/`.

[HR03] S. Halevi and P. Rogaway. A tweakable enciphering mode. In D. Boneh, editor, *Advances in Cryptology – CRYPTO '03*, volume 2729 of *LNCS*, pages 482–499. Springer, 2003.

[HR04] Shai Halevi and Phil Rogaway. A parallelizable enciphering mode. In *The RSA conference – Cryptographer's track, RSA-CT'04*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer-Velrag, 2004.

[IK03] Tetsu Iwata and Kaoru Kurosawa. "omac: One-key cbc mac". In *Fast Software Encryption - FSE'03*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing*, 17(2), April 1988.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *Advances in Cryptology – CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[MM82] Carl H. Meyr and Stephen M. Matyas. *Cryptography: A New Dimension in Computer Data Security*. John Wiley & Sons, 1982.

[MV04] David A. McGrew and John Viega. Arbitrary block length mode. Available on-line from `http://grouper.ieee.org/groups/1619/email/pdf00005.pdf`, 2004.

[NR97] Moni Naor and Omer Reingold. A pseudo-random encryption mode. Manuscript, available from `http://www.wisdom.weizmann.ac.il/~naor/`, 1997.

[NR99] Moni Naor and Omer Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.

[Sho96] V. Shoup. On fast and provably secure message authentication based on universal hashing. In *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 74–85. Springer-Verlag, 1996.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In *Information Security and Cryptology – CISC'05*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[YMK05] Bo Yang, Sambit Mishra, and Ramesh Karri. "a high speed architecture for galois/counter mode of operation (gcm)". Technical Report 2005/146, IACR ePrint archive, 2005. `http://eprint.iacr.org/2005/146/`.

A Preliminaries

A *tweakable enciphering scheme* is a function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ where $\mathcal{M} = \bigcup_{i\in I}\{0,1\}^i$ is the *message space* (for some nonempty index set $I \subseteq \mathbb{N}$) and $\mathcal{K} \neq \emptyset$ is the *key space* and $\mathcal{T} \neq \emptyset$ is the *tweak space*. We require that for every $K \in \mathcal{K}$ and $T \in \mathcal{T}$ we have that $\mathbf{E}(K, T, \cdot) = \mathbf{E}_K^T(\cdot)$ is a length-preserving permutation on $\mathcal{M}$. The inverse of an enciphering scheme $\mathbf{E}$ is the enciphering scheme $\mathbf{D} = \mathbf{E}^{-1}$ where $X = \mathbf{D}_K^T(Y)$ if and only if $\mathbf{E}_K^T(X) = Y$. A *block cipher* is the special case of a tweakable enciphering scheme where the message space is $\mathcal{M} = \{0,1\}^n$ (for some $n \geq 1$) and the tweak space is the singleton set containing the empty string. The number n is called the *blocksize*. By $\mathrm{Perm}(n)$ we mean the set of all permutations on $\{0,1\}^n$. By $\mathrm{Perm}^{\mathcal{T}}(\mathcal{M})$ we mean the set of all functions $\pi : \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ where $\pi(T, \cdot)$ is a length-preserving permutation.

An *adversary* A is a (possibly probabilistic) algorithm with access to some oracles. Oracles are written as superscripts. By convention, the running time of an algorithm includes its description size. The notation $A \Rightarrow 1$ describes the event that the adversary A outputs the bit one.

Security measure. For a tweakable enciphering scheme $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ we consider the advantage that the adversary A has in distinguishing $\mathbf{E}$ and its inverse from a random tweakable permutation and its inverse: $\mathbf{Adv}_{\mathbf{E}}^{\pm\widetilde{\mathrm{prp}}}(A) =$

$$\Pr\left[K \xleftarrow{\$} \mathcal{K} :\ A^{\mathbf{E}_K(\cdot,\cdot)\,\mathbf{E}_K^{-1}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) :\ A^{\pi(\cdot,\cdot)\,\pi^{-1}(\cdot,\cdot)} \Rightarrow 1\right]$$

The notation shows, in the brackets, an experiment to the left of the colon and an event to the right of the colon. We are looking at the probability of the indicated event after performing the specified experiment. By $X \xleftarrow{\$} \mathcal{X}$ we mean to choose X at random from the finite set $\mathcal{X}$. In writing $\pm\widetilde{\mathrm{prp}}$ the tilde serves as a reminder that the PRP is tweakable and the $\pm$ symbol is a reminder that this is the "strong" (chosen plaintext/ciphertext attack) notion of security. For a block cipher, we omit the tilde.

Without loss of generality we assume that an adversary never repeats an encipher query, never repeats a decipher query, never queries its deciphering oracle with (T, C) if it got C in response to some (T, M) encipher query, and never queries its enciphering oracle with (T, M) if it earlier got M in response to some (T, C) decipher query. We call such queries *pointless* because the adversary "knows" the answer that it should receive.

When $\mathcal{R}$ is a list of resources and $\mathbf{Adv}_{\Pi}^{\mathrm{xxx}}(A)$ has been defined, we write $\mathbf{Adv}_{\Pi}^{\mathrm{xxx}}(\mathcal{R})$ for the maximal value of $\mathbf{Adv}_{\Pi}^{\mathrm{xxx}}(A)$ over all adversaries A that use resources at most $\mathcal{R}$. Resources of interest are the running time t, the number of oracle queries q, and the total number of n-bit blocks in all the queries s. The name of an argument (e.g., t, q, s) will be enough to make clear what resource it refers to.

Finite fields. We interchangeably view an n-bit string as: a string; a nonnegative integer less than 2^n; a formal polynomial over GF(2); and an abstract point in the finite field GF(2^n). To do addition on field points, one xors their string representations. To do multiplication on field points, one must fix a degree-n irreducible polynomial. For example, a popular choice is to use the lexicographically first primitive polynomial of minimum weight. (For $n = 128$ this is the polynomial $\mathsf{x}^{128}+\mathsf{x}^7+\mathsf{x}^2+\mathsf{x}+1$.) We note that with this choice of field-point representation, the point α corresponding to the polynomial x (that has encoding $\alpha = 0^{n-2}10$) has order $2^n - 1$ in the multiplicative group of GF(2^n), meaning that $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2^n-1}$ are all distinct. Finally, we note that given any point $x = x_{n-1}\cdots x_1x_0 \in \{0,1\}^n$ it is easy to compute αx. We illustrate the procedure for $n = 128$, in which case $\alpha x == x{\ll}1$ if $\mathsf{firstbit}(x) = 0$, and $\alpha x = (L{\ll}1) \oplus \text{Const87}$ if $\mathsf{firstbit}(x) = 1$. Here $\text{Const87} = 0^{120}10^41^3$ and $\mathsf{firstbit}(x)$ means x_{n-1} and $x{\ll}1$ means $x_{n-2}x_{n-3}\cdots x_1x_00$.

B Intellectual-Property Issues

The original motivation for devising the TET mode was to come up with a reasonably efficient mode that is "clearly patent-free". The IEEE security-in-storage working group (SISWG) was working on a standard for length-preserving encryption for storage, and some of the participants expressed the wish to have such a mode. (**Disclaimer:** Not being a patent lawyer, I can only offer my educated guesses for the IP status of the various modes. The assessment below reflects only my opinion about where things stand.)

The modes CMC/EME/EME* from the "encrypt-mix-encrypt" family are all likely to be patent-encumbered, due to US Patent Application 20040131182A1 from the University of California (which as of this writing was not yet issued). Similarly, the XCB mode – which is the first proposed mode in the "hash-CTR-hash" family – is likely to be patent-encumbered due to a US patent application US20070081668A1 from Cisco Systems (also still not issued as of this writing). The status of the other members of the "hash-CTR-hash" family is unclear: they may or may not be covered by the claims of the Cisco patent when it is issued.

This state of affairs left the "hash-ECB-hash" approach as the best candidate for finding patent-free modes: this approach is based on the paper of Naor and Reingold [NR97] that pre-dates all these modes by at least five years, and for which no patent was filed. Specifically for TET, I presented the basic construction from Section 2 in an open meeting of the IEEE SISWG in October of 2002 (as well as in an email message that I sent the SISWG mailing list on October 9, 2002), and I never filed for any patent related to this construction. Thus it seems unlikely to me that there are any patents that cover either the general "hash-ECB-hash" approach or TET in particular.