Summer 8-7-2015

Tweakable Ciphers: Constructions and Applications

Robert Seth Terashima
Portland State University

10.15760/etd.2481

Tweakable Ciphers: Constructions and Applications

by

Robert Seth Terashima

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Thomas Shrimpton, Chair
John Caughman IV
James Hook
Mark Jones
Charles V. Wright

Portland State University
2015

Abstract

Tweakable ciphers are a building block used to construct a variety of cryptographic algorithms. Typically, one proves (via a reduction) that a tweakable-cipher-based algorithm is about as secure as the underlying tweakable cipher. Hence improving the security or performance of tweakable ciphers immediately provides corresponding benefits to the wide array of cryptographic algorithms that employ them. We introduce new tweakable ciphers, some of which have better security and others of which have better performance than previous designs. Moreover, we demonstrate that tweakable ciphers can be used directly (as opposed to as a building block) to provide authenticated encryption with associated data in a way that (1) is robust against common misuses and (2) can, in some cases, result in significantly shorter ciphertexts than other approaches.

Table of Contents

List of Tables

List of Figures

Glossary of Abbreviations

The table below contains a list of acronyms, with references to the text. Acronyms for various wideblock tweakable ciphers appear in Table 4.1 (pg. 43).

AE	Pg. 3	**Authenticated Encryption:** A type of cryptographic algorithm that protects both the privacy and integrity of data.
AEAD	Pg. 3	**Authenticated Encryption with Associated Data:** An AE algorithm that also protects the integrity of unencrypted associated data.
AES	Pg. 8	**Advanced Encryption Standard:** A widely used 128-bit blockcipher.
AEZ	Pg. 3	**(Not an acronym):** Refers to both an AEAD algorithm and the wideblock tweakable cipher from which it's built.
CBC	Pg. 40	**Cipher block Chaining Mode:** An encryption algorithm.
CTR	Pg. 43	**Counter Mode:** An blockcipher-based encryption algorithm.
GCM	Pg. 3	**Galois Counter Mode:** An AEAD algorithm.
IV	Pg. 40	**Initialization Vector:** A input to an encryption algorithm that prevents similar/identical plaintexts from generating related ciphertexts.
LRW	Pg. 17	**Liskov Rivest Wagner:** A TBC with birthday-bound security. May also refer to a disk-encryption algorithm that uses this TBC.
NH	Pg. 56	**Non-Linear Hash:** A universal hash function.
OCB	Pg. 2	**Offset Tweak-based Code Book:** An extremely fast AEAD algorithm.
PIV	Pg. 47	**Protected IV Mode:** A modular framework for constructing wideblock tweakable ciphers.
PRP	Pg. 7	**Pseudo-Random Permutation:** A blockcipher is a PRP if it "looks random" to a chosen-plaintext attacker.
SPRP	Pg. 7	**Strong Pseudo-Random Permutation:** A blockcipher is a SPRP if it "looks random" to a a chosen-ciphertext attacker.
STPRP	Pg. 8	**Strong Tweakable Pseudo-Random Permutation:** A TBC is a STPRP if it "looks random" to a a chosen-ciphertext attacker .
TBC	Pg. 1	**Tweakable Blockcipher:** A generalization of a blockcipher, where each key provides a large number of pseudorandom permutations.
TCT	Pg. 42	**Tweak-Counter-Tweak:** TCT_1 and TCT_2 are wideblock tweakable ciphers offering birthday-bound and beyond-birthday bound security, respectively.
TCTR	Pg. 53	**Tweaked Counter Mode:** A wideblock tweakable cipher.
TPRP	Pg. 8	**Tweakable Pseudo-Random Permutation:** A TBC is a TPRP if it "looks random" to a chosen-plaintext attacker.
VCV	Pg. 66	**VHASH-Counter-VHASH:** A wideblock tweakable cipher construction.
VHASH	Pg. 67	**(Not an acronym):** A universal hash function.
XEX	Pg. 17	**XOR-Encrypted-XOR:** A TBC with birthday-bound security.
XTS	Pg. 15	**XEX-based tweaked-codebook mode with ciphertext stealing:** A TBC with birthday-bound security.

1. Introduction

A fundamental issue in cryptography is identifying assumptions that are both reasonable and useful. For example, the assumption that factoring large numbers is infeasible is generally regarded as satisfying both criteria.

In symmetric-key cryptography (i.e., cryptography where two or more parties start with a shared secret), one often begins by assuming there exists some secure *blockcipher*. Briefly, an n-bit blockcipher is an algorithm that takes two inputs: a (random, secret) key and an n-bit string X, and outputs a second n-bit string, Y. Anyone who knows the key can recover X from Y. The blockcipher is secure insofar as absent knowledge of this key, the mapping from X-values to Y-values appears random. Given a secure blockcipher, one can construct a variety of provably-secure cryptographic algorithms, including encryption schemes and message-authentication codes.

In a seminal paper [30], Liskov, Rivest, and Wagner argue that despite the past popularity of blockciphers, blockciphers are the "wrong" primitive to start with. The authors propose *tweakable blockciphers* (TBCs) as an alternative. A TBC functions similarly to a regular blockcipher, except that it takes an additional input: a τ-bit *tweak*. Instead of one random-looking mapping between n-bit strings, a TBC provides 2^τ such maps — one for each tweak. To a third party that does not know the key, each map should each appear random and independent of the others. This should remain true even if the third party knows the corresponding tweak values.

TBCs are thus a much more powerful primitive than traditional blockciphers, so it's natural to wonder if a secure TBC could be constructed without being prohibitively slow. Liskov, Rivest, and Wagner, however, showed how to build an efficient TBC from a blockcipher in such a way that the former inherits (most of) the latter's security, and doesn't incur too much of a performance loss.

Subsequent work has helped validate the paper's central thesis that TBCs are not only useful, but significantly *more* useful than traditional blockciphers. Rogaway's Offset Code Book (OCB) [48, 46] is an excellent case study, because it was initially conceived of as a blockcipher-based algorithm [48], rather than TBC-based one. When recast in terms of a TBC [46], an act which simply required viewing some of OCB's internal machinery as a single black box, the length of its security proof dropped from from 19 pages to two. Moreover, although one would expect starting with a more-powerful primitive to induce overhead, the change in perspective actually revealed opportunities for stream-lining the algorithm by removing unnecessary machinery. This provides a rather compelling case for further investigating TBCs as a cryptographic tool.

All TBCs are functionally equivalent in the sense that one could replace, e.g., OCB's TBC with essentially any other TBC, and the resulting OCB variant would inherit the new TBC's performance and security characteristics. Thus, any advancement in the design of TBCs has the potential to impact a wide array of cryptographic algorithms. This raises the question,

How can we improve the performance and/or security properties of TBCs?

We shed some light on the answer. We define and prove the security properties of LRW2 (which consists of two rounds of the TBC construction by Liskov, Rivest, and Wagner [30]), as well as the two Tweak-Counter-Tweak constructions, TCT_1, and

TCT_2. (We introduced all three of these algorithms in prior publications [29, 53].) Finally, we also refine TCT_1 to produce VHASH-Counter-VHASH (VCV), a novel TBC construction, and provide benchmarks for an implementation. These benchmarks demonstrate that VCV likely[1] outperforms competing "wideblock" TBCs in software when AES is used as the underlying blockcipher. (AES is a widely supported 128-bit blockcipher.) The recent AEZ [23] algorithm is faster than VCV, but AEZ offers a security proof only under a heuristic assumption.

Finally, we describe some of our other results from [53], which show how wideblock TBCs, such as VCV, can be used to provide authenticated encryption with associated data (AEAD). (AEAD not only prevents a third party from learning information about the plaintext, but it also prevents him from tampering with the ciphertext to induces changes in a perhaps partially known plaintext.) The resulting ciphertexts can, in some situations, be much shorter than could be achieved using traditional methods. This results in significantly less bandwidth overhead when sending numerous short messages, which in turn may lead to power savings for wireless devices. Moreover, the algorithm is "nonce-misuse resistant" — it can continue to provide a high degree of security even when programmers make certain types of mistakes when using it. VCV out-performs the industry-standard Galois Counter Mode (GCM) [35] by about 37% on older machines (Intel Core Duo), and is over twice as fast on newer ones (i7-4770), despite having a strictly stronger set of security properties. The aforementioned AEZ algorithm by Hoang, Krovetz, and Rogaway uses this TBC-to-AEAD approach, built from a stripped-down version of AES.

[1]With the exception of AEZ, discussed later, were unable to obtain any optimized software implementations of competing constructions, making comparisons difficult. However, these competing constructions are required to perform certain computations sequentially (i.e., without the possibility of software pipelining), permitting us to use a subset of these computations to establish a lower bound on running times.

1.1 Notes on source material

Parts of this work appear in previous publications [29, 53] that were produced with coauthors.

The LRW2 algorithm was first described in CRYPTO '12 [29], coauthored with Shrimpton and Landecker (this paper referred to the algorithm as CLRW2). The LRW2 design and its security proof were my contribution to the paper; Shrimpton assigned me the task of finding a TBC construction with beyond birthday-bound security, and edited my proof write-up. Both Shrimpton and Landecker provided valuable sounding boards in the search for viable constructions. The CRYPTO paper includes other TBC-related results to which I was not a contributer, but LRW2 is the paper's central result.

The PIV tweakable cipher framework and its instantiations, TCT_1 and TCT_2, originally appeared in an ASIACRYPT '13 paper [53], coauthored with Shrimpton. I had decided to pursue a "wideblock" tweakable cipher construction while interning at Voltage Security. I devised and wrote the proofs, while Shrimpton edited them and provided essential guidance in how to package and present the results. The other results in this paper concern using wideblock tweakable ciphers to provide AEAD. Although I again devised and wrote the proofs, this was a relatively minor task compared to deciding *what* to prove; as best I can recall, these were joint decisions that emerged over the course of several conversations. I believe the initial idea to investigate AEAD was Shrimpton's, while the idea for handling multiple error messages (and using these as a model for certain side channels) was mine.

The VCV variant is a modest but, I hope, worthwhile improvement on TCT_1; its design and implementation are my own work.

2. Preliminaries

This chapter introduces some standard cryptographic definitions and concepts. Readers with a background in cryptography may safely skim or skip it.

Section 2.1 introduces the syntax of various cryptographic primitives, as well as their associated security definitions. Next, Section 2.2 provides an example of a "game-playing proof". This type of proof is common in cryptography, and we shall make extensive use of it. Finally, Section 2.3 describes the so-called "birthday paradox", which plays a role in limiting how much data can be securely processed by various cryptographic algorithms. We discuss it here because finding TBC constructions that avoid the birthday paradox is one of our main contributions.

2.1 Cryptographic primitives

Blockciphers. An n-bit blockcipher is an invertible function that uses a secret key to encrypt an n-bit string, producing an n-bit string as output. This mapping between n-bit strings should appear essentially random to anyone who lacks knowledge of the key. As previously discussed, blockciphers are a primitive used to construct higher-level cryptographic algorithms (such as encryption algorithms that can encrypt arbitrarily long bit strings).

More formally, we denote the set of n-bit strings as $\{0,1\}^n$, and the set of all (finite) binary strings as $\{0,1\}^*$. Let $\mathrm{Perm}(n)$ denote the set of all permutations on $\{0,1\}^n$.

Definition 1 (Blockciphers)**.** *Let n and k be positive integers. A function $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is a* blockcipher *if for each* key *$K \in \{0,1\}^k$, $E_K(\cdot) := E(K, \cdot) \in Perm(n)$.*

This describes the syntax of blockciphers, but we still need to define what it means for a blockcipher to be secure. Informally, a blockcipher E is secure if, given a random key K, E_K "looks like" a random permutation to those who don't know the key. Because the number of keys, 2^k, is typically much less than $|\mathrm{Perm}(n)| = 2^n!$, E_K cannot *be* a uniformly random permutation; however, we can hope that as long as K remains secret, no one will be able to distinguish it from one in practice.

In order to make this statement formal, we need to introduce the notion of an *adversary.*

Adversaries and oracles. An *adversary* A is an algorithm that is provided with black-box access to zero or more functions, $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$, called *oracles.* That is, A can query an oracle $\mathcal{O}_i$ at a point x to learn $\mathcal{O}_i(x)$. However, A doesn't learn anything about the oracle's behavior at other points that couldn't be inferred from this information. We typically limit the number of queries an adversary is permitted to make in order to model the presumed computational resource limitations of an attack. Both adversaries and oracles may be probabilistic, and oracles may also retain state between invocations. We write $A^{\mathcal{O}_1,\ldots,\mathcal{O}_\ell} \Rightarrow b$ to denote the event that A outputs b when equipped with the indicated oracles. Note that in general, b will be a random variable whose distribution is governed by the (probabilistic) behavior of both A and its oracles.

An adversary is *adaptive* if each of its queries can depend on the results of previous queries. Adversaries are always assumed to be adaptive unless we explicitly state otherwise.

(Strong) PRP security. Given a finite set S, let $X \xleftarrow{\$} S$ denote that X is a random variable uniformly distributed on S; in particular, $K \xleftarrow{\$} \{0,1\}^k$ indicates that K is a random k-bit key and $\pi \xleftarrow{\$} \mathrm{Perm}(n)$ is a random permutation on the set of n-bit strings. Let K and π be so defined. To capture the notion that no one can efficiently distinguish E_K from π, we define the *pseudorandom permutation* (PRP) advantage of an adversary A against E as:

$$\mathbf{Adv}_E^{\mathrm{prp}}(A) := \Pr\left[\, A^{E_K} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\pi} \Rightarrow 1 \,\right].$$

The intuition here is that A is some algorithm that submits a sequence of queries $x_1, \ldots, x_q$ to an oracle $\mathcal{O}$, receiving $\mathcal{O}(x_1), \ldots, \mathcal{O}(x_q)$ as replies. Then A attempts to guess if $\mathcal{O} = E_K$, in which case A outputs 1, or if $\mathcal{O} = \pi$, in which case A outputs 0. If A always guesses correctly, then $\mathbf{Adv}_E^{\mathrm{prp}}(A) = 1$; if A doesn't do better than random guessing, then $\mathbf{Adv}_E^{\mathrm{prp}}(A) = 0$.

Similarly, we define the *strong pseudorandom permutation* (SPRP) security of A against E as:

$$\mathbf{Adv}_E^{\mathrm{sprp}}(A) := \Pr\left[\, A^{E_K, E_K^{-1}} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\pi, \pi^{-1}} \Rightarrow 1 \,\right].$$

Note that here, A is provided access to two oracles. So not only can A encrypt messages of its choosing, it can also use its second oracle to decrypt ciphertexts of its choosing.

Now, A could simply enumerate all 2^k possible key values and output 1 if and only if there is some key K' such that $E_{K'}(x_i) = \mathcal{O}(x_i)$ for each $i = 1, 2, \ldots, q$. For typical n and k (e.g., $n = k = 128$), $2^k \ll |\mathrm{Perm}(n)|$. In this case, $\Pr\left[\, A^{\pi} \Rightarrow 1 \,\right] \approx 0$ for even modest values of q, while $\Pr\left[\, A^{E_K} \Rightarrow 1 \,\right] = 1$, giving us $\mathbf{Adv}_E^{\mathrm{prp}}(A) \approx 1$.

But this “brute-force” strategy is not feasible for, e.g., 128-bit keys. Hence we will usually restrict the class of adversaries under consideration to those running in time t in some implicit model of computation; we informally deem E a PRP if $\mathbf{Adv}^{\mathrm{prp}}_{E}(A)$ is small (say, 2^{-60}) for any A in this class. The consensus among cryptographers is that there are blockciphers, such as AES, for which $\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(A)$ can safely be assumed to be negligible for any A with realistic time constraints.

Tweakable blockciphers We are now in a position to define tweakable blockciphers and their close cousins, tweakable ciphers.

Definition 2 (Tweakable blockcipher.). *Let k, τ, and n be positive integers. A function $\widetilde{E} : \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^n \times \{0,1\}^n$ is a* tweakable blockcipher *(TBC) if for each key $K \in \{0,1\}^k$ and* tweak *$T \in \{0,1\}^\tau$, $\widetilde{E}_K(T,\cdot) := \widetilde{E}(K,T,\cdot) \in \mathit{Perm}(n)$.*

Abusing notation, $\widetilde{E}^{-1}_K(T,Y)$ is the unique X such that $\widetilde{E}_K(T,X) = Y$.

In other words, once equipped with a key, a TBC provides a family of permutations — one for each tweak — whereas a traditional blockcipher provides only one. This means that a TBC with key space $\{0,1\}^k$ and tweak space $\{0,1\}^\tau$ is syntactically a blockcipher with key space $\{0,1\}^{k+\tau}$. The distinction arises in the security definition.

(Strong) TPRP security. Just as we defined a blockcipher's security in terms of its indistinguishability from a random permutation π, we define a tweakable blockcipher's security in terms of its indistinguishability from Π, a family of (independent, uniform) random permutations.

Fix positive integers τ, and n, and let

$$\Pi \xleftarrow{\$} \{f : \{0,1\}^\tau \times \{0,1\}^n \to \{0,1\}^n \ : \text{ For all } T \in \{0,1\}^\tau,\ f(T,\cdot) \in \mathrm{Perm}\,(n)\}\,.$$

Again abusing notation, let $\Pi^{-1}(T, Y)$ be the unique X such that $\Pi(T, X) = Y$.

We define the (strong) tweakable pseudo-random permutation (TPRP) advantage of an adversary A against $\widetilde{E}$ as:

$$\mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(A) := \Pr\left[\, A^{\widetilde{E}_K} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\Pi} \Rightarrow 1 \,\right],$$
$$\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\widetilde{E}}(A) := \Pr\left[\, A^{\widetilde{E}_K, \widetilde{E}_K^{-1}} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\Pi, \Pi^{-1}} \Rightarrow 1 \,\right],$$

with probabilities over the randomness of A and the random variables $K \xleftarrow{\$} \{0,1\}^k$ and Π. In other words, $\widetilde{E}$ is secure if, when equipped with a random secret key, A is unable to tell it apart from a family of independent, random permutations. Note that because A has oracle access to either $\widetilde{E}_K$ or Π, A's queries are of the form $(T, X) \in \{0,1\}^{\tau} \times \{0,1\}^{n}$ — A gets to choose which tweak is used on any given query.

Tweakable ciphers are a generalization of TBCs. While a TBC can only operate on fixed-length inputs, a tweakable cipher does not necessarily have this restriction; however, a tweakable cipher must preserve the lengths of its inputs.

Given $S \subseteq \mathbb{N}$, define $\{0,1\}^S = \bigcup_{n \in S} \{0,1\}^n$. Then:

Definition 3 (Tweakable ciphers)**.** *Fix a set $S \subseteq \mathbb{N}$ and positive integers k and τ. Then a function $\widetilde{E} : \{0,1\}^k \times \{0,1\}^{\tau} \times \{0,1\}^S \to \{0,1\}^S$ is a* tweakable cipher *if for any key $K \in \{0,1\}^k$, tweak $T \in \{0,1\}^{\tau}$ and positive integer $n \in S$, $\widetilde{E}_K(T, \cdot) \in$ Perm(n) when restricted to n-bit inputs.*

Hence, if $\widetilde{E}$ is a tweakable cipher, then $\left|\widetilde{E}_K(T, X)\right| = |X|$ for all $(K, T, X) \in \mathsf{dom}(\widetilde{E})$.

Definition 4 ((Almost) Δ-universal hash functions)**.** *Fix positive integers k, ℓ, and n, and let ϵ be a positive real number. Let $+$ be a group operator on $\{0,1\}^n$. Let $H : \{0,1\}^k \times \{0,1\}^{\ell} \to \{0,1\}^n$, and define $H_K(\cdot) = H(K, \cdot)$. Then H is* ϵ-almost

Δ-universal *(ϵ-AΔU) with respect to* $+$ *if, for any* $C \in \{0,1\}^n$ *and any distinct* $X, X' \in \{0,1\}^\ell$*,* $\Pr[\, H_K(X) - H_K(X') = C \,] \le \epsilon$*. The probability is over the random variable* $K \xleftarrow{\$} \{0,1\}^k$*.*

This is a strictly combinatorial property of H; there are no adversaries or computational assumptions in this definition. If H is ϵ-AΔU with respect to $\oplus$ (bitwise XOR; i.e., componentwise addition modulo two), then we refer to H as being ϵ-almost XOR-universal (ϵ-AXU). Note that, in this case, addition and subtraction are the same operation.

2.2 Game-playing proofs

We will often need to bound an expression of the form $\Pr\left[\, A^{\mathcal{R}} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\mathcal{I}} \Rightarrow 1 \,\right]$, where A is some adversary and $\mathcal{R}$ and $\mathcal{I}$ are oracles (typically $\mathcal{I}$ will be some mathematically ideal oracle, such as a random permutation, while $\mathcal{R}$ will be some real-world object, such as a blockcipher equipped with a random key, whose security we are investigating).

To bound these expressions, we will make extensive use of so-called *game-playing proofs*. The game-playing framework [5] was introduced by Bellare and Rogaway as a means to facilitate cryptographic proofs. The framework is especially helpful in reasoning about the sorts of conditional probabilities that arise when trying to prove the (in)effectiveness of arbitrary adversaries. Because adversaries are permitted to choose queries based on the results of earlier queries, this sort of reasoning can be difficult and error-prone.

We first write an explicit program that, given A, describes the operation $A^{\mathcal{R}}$. Because this program captures an interaction between A and an oracle, we refer to it as a *game*. Then, starting from a first game, $G_0(A)$, we construct a sequence of games

$G_0(A), G_1(A), \ldots, G_n(A)$ such that $G_n(A)$ behaves identically to $A^{\mathcal{I}}$ (i.e., for any A, the outputs of $G_n(A)$ and $A^{\mathcal{I}}$ are identically distributed). Typically, $G_0(A)$ will just consist of "glue" that relays queries from A to $\mathcal{R}$ and responses from $\mathcal{R}$ to A, while $G_n(A)$ will do the same with respect to $\mathcal{I}$. The interesting work lies in designing the remaining games.

We have immediately that

$$\Pr\left[\, A^{\mathcal{R}} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\mathcal{I}} \Rightarrow 1 \,\right] = \sum_{i=0}^{n-1} \left(\Pr\left[\, G_i(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_{i+1}(A) \Rightarrow 1 \,\right]\right).$$

Now, instead of finding an upperbound for the left-hand side directly, we can find upperbounds for each individual term on the right-hand side. We construct intermediate games $G_1(A), G_2(A), \ldots, G_{n-1}(A)$ with the aim of simplifying this task.

Example: Distinguishing a random function from a random permutation. As an example, let $\rho \xleftarrow{\$} \{f : \{0,1\}^n \to \{0,1\}^n\}$ be a random function mapping $\{0,1\}^n$ to $\{0,1\}^n$, and let $\pi \xleftarrow{\$} \mathrm{Perm}\,(n)$ be a random permutation. We will show that for any adversary A making at most q queries, $\Pr\left[\, A^{\pi} \Rightarrow 1 \,\right] = \Pr\left[\, A^{\rho} \Rightarrow 1 \,\right] \leq q^2/2^n$. What follows is essentially an annotated and semi-formal version of Black and Rogaway's proof [5].

The outline of our proof is as follows: We can define the random permutation π by pulling values from the set $\{0,1\}^n$ at random one at a time without replacement to generate $\pi(0), \pi(1), \ldots, \pi(2^n - 1)$ (we identify elements of $\{0,1\}^n$ with integers in the appropriate range). So let $G_0(A)$ generate π in this manner, and then use it to respond to A's queries. Alternatively, we could define π though *lazy sampling*: when the adversary submits a new query X, we pull a value Y from $\{0,1\}^n$ at random without replacement and set $\pi(X) = Y$. Because this doesn't change the distribution on π, the adversary can't tell the difference; hence, if we let $G_1(A)$ define π in this manner, then we have

$$\Pr[\, A^{\pi} \Rightarrow 1 \,] = \Pr[\, G_0(A) \Rightarrow 1 \,] = \Pr[\, G_1(A) \Rightarrow 1 \,] .$$

Working from the other end, we can choose ρ by sampling $\rho \overset{\$}{\leftarrow} \{f : \{0,1\}^n \to \{0,1\}^n\}$ and use this to create game $G_4(A)$. Or, we could define ρ through lazy sampling—this time choosing its output values by drawing from $\{0,1\}^n$ *with* replacement to create $G_3(A)$. We have:

$$\Pr[\, A^{\rho} \Rightarrow 1 \,] = \Pr[\, G_4(A) \Rightarrow 1 \,] = \Pr[\, G_3(A) \Rightarrow 1 \,] .$$

Now, the only difference between games G_1 and G_3 is that the former samples values without replacement, whereas the latter samples them with replacement. But for even modest values of n (say, $n = 128$) and a large number of queries (say, $q = 2^{40}$), A will only be able to query a small fraction of the domain $\{0,1\}^n$. So intuitively, it seems like our decision about whether or not to sample with replacement would be unlikely to affect A's output. We now proceed to make this argument quantitative.

Suppose that instead of sampling values without replacement, we sample values until we happen upon one that hasn't been chosen before, and then return that one (we assume $q \leq 2^n$). It may take us a longer time to return a result, but we haven't actually changed distribution of π. And recall that an adversary accesses oracles in a black-box fashion: it learns the value returned by a query, but nothing else. So let's use this method of defining π for $G_2(A)$.

Suppose, however, that you were the adversary, and, as a thought experiment, that you were in fact able to watch some Oracle literally pulling numbers out of a hat. How could easily could you tell the Oracle of $G_2(A)$ (which "loops" until it finds a new value) apart from the Oracle of $G_3(A)$ (which always returns the first value it draws)? The *only* way you could learn any information would be if the Oracle happened to sample a previously chosen value, at which point the distinction would be obvious. The Oracle's behavior in the two games doesn't diverge until this point is reached — one could imagine that he even isn't informed of what game he's playing unless this happens, so you can't learn anything by watching him until then.

```
                                                    GAMES $\boxed{G_2}$, $G_3$
Oracle $f(X)$:
$Y \xleftarrow{\$} \{0,1\}^n$
if $Y$ has previously been returned then
    bad ← true
    $\boxed{\text{Resample } Y \xleftarrow{\$} \{0,1\}^n \text{ until } Y \text{ has not been previously returned}}$
return $Y$
```

Listing 2.1: In Game G_2, which includes the boxed statement, the oracle f implements a random permutation using lazy sampling. In Game G_3, which excludes the boxed statement, f instead implements a random function. An adversary's ability to distinguish one from the other is upperbounded by the probability that bad is ever set to true.

We make this somewhat more formal by explicitly writing out the pseudocode for these two games in Listing 2.1. The code for the two games differs only within an **if** block, and this block begins by setting the boolean flag bad to true. (We adapt the convention, here and throughout, that boolean values are initialized to false). As this boolean value is *monotonic* — it can change from false to true but not the other way around — the *fundamental lemma of game-playing* [5] tells us that

$$|\Pr[\,G_2(A) \Rightarrow 1\,] - \Pr[\,G_3(A) \Rightarrow 1\,]| \le \Pr[\,G_2(A)\,;\,\mathsf{bad}\,] = \Pr[\,G_3(A)\,;\,\mathsf{bad}\,]\,.$$

That is, the probability of bad being set is the same in either game, and is an upper-bound for A's advantage in distinguishing the two games.

But the probability of setting bad on a particular query is at most $q/2^n$. Therefore, using a union bound, the probability that this flag will ever be set is $\Pr[\, G_3(A)\, ;\, \mathsf{bad}\,] \leq q^2/2^n$. Collecting earlier results gives us $\Pr[\, A^{\pi} \Rightarrow 1\,] - \Pr[\, A^{\rho} \Rightarrow 1\,] \leq \frac{q^2}{2^n}$, as desired.

2.3 The birthday bound

The advantage of an adversary is typically a function of the number of queries that it is permitted to make. For example, we will encounter a few TBCs $\widetilde{E}$ that are built from a blockcipher E and have the property that for any adversary A making q queries and running in time t, there exists some PRP adversary B making q queries and running in time $t' \approx t$ such that

$$\mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(A) \leq \mathbf{Adv}^{\mathrm{prp}}_{E}(B) + \frac{cq^2}{2^{n+1}},$$

for some modest constant c. That is, A cannot "break" $\widetilde{E}$ unless there is some B (with similar computational resources) that can "break" E. We have previously described this informally by saying that $\widetilde{E}$ inherits E's security.

But this statement tacitly assumes that $cq^2 \ll 2^{n+1}$. Security bounds that have an $\mathcal{O}(q^2/2^n)$ term are referred to as having "birthday bounds". This term is a reference to the so-called birthday-paradox, which states that, given a room with q people and a year with 2^n days, the probability that there will be two people who share a birthday is upperbounded by $q^2/2^n$. (This upperbound is a good approximation for smaller values of q, and assumes that dates of birth are independent).

Birthday bounds are ubiquitous in cryptography — we saw an example in Section 2.2. Roughly speaking, they arise when an algorithm with n bits of state "looks random" unless an adversary can cause an internal state to be repeated.

In many cases, a birthday bound is not a problem. Consider the 128-bit AES blockcipher: if $q < 2^{40}$, then $q^2/2^{128} < 2^{-48} \approx 0$.[1] So an encryption algorithm with birthday-bound security that uses AES can safely make 2^{40} AES calls (assuming AES is itself secure); because each AES block is 16 bytes, this corresponds to roughly 16 terabytes of data. However, there are circumstances where a birthday-bound becomes problematic: when q can become large, or when n is small.

The former could occur when large amounts of data are processed or stored. For example, the 16TB above becomes about 3.5TB if one uses the standard XTS [34] disk-encryption algorithm where $c = 4.5$ [37]; it drops to under a terabyte if one requires a 2^{-50} security bound instead of a 2^{-48} bound.

Similar problems emerge when n is small. Lightweight blockciphers have received much attention recently (see, for example, the survey by Cazorla, Marquet, and Minier [9]), where $n \in \{32, 48, 64\}$ is typical. For $n = 64$, a security bound of even 2^{-40} requires $q < 12$, which translates to only 32KB. A TBC with a $\theta(q^3/2^{2n})$ security bound, such as our LRW2 algorithm, raises this limit to about 5GB.

[1]We have asserted that $2^{-48} \approx 0$, begging the question of what numbers are "close enough" to zero. This is ultimately a question of context and judgement. One can view the security bound as the probability of a cryptographic attack succeeding; hence, a bound of 2^{-30} (roughly 1 in a billion) may be sufficient for encrypting innocent text messages, while one may want a significantly lower bound, such as 2^{-70}, if one is protecting nuclear launch codes.

3. The LRW2 Tweakable Blockcipher

The previous chapter outlined scenarios in which a TBC with beyond birthday-bound security would prove valuable. This chapter describes our LRW2 construction, which guarantees such a bound. We begin in Section 3.1 with an overview of prior TBC constructions. Section 3.2 introduces LRW2 and provides a proof of its security; the proof is rather involved, and constitutes the bulk of this chapter.

3.1 Prior TBC constructions

Blockcipher-based constructions. In their seminal paper [30], Liskov, Rivest, and Wagner propose two n-bit tweakable blockcipher constructions, both based on some underlying n-bit blockcipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$.

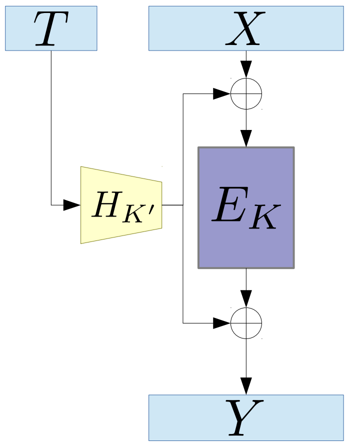

Figure 3.1: The LRW TBC.

The first construction $\widetilde{E} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is defined as $\widetilde{E}(T,X) = E_K(T \oplus E_K(X))$. This TBC has birthday-bound security: given an adversary A making q queries and running in time t, there exists some other adversary B making q queries and running in time $t' \approx t$ such that $\mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(A) \leq \Theta\left(\frac{q^2}{2^n}\right) + \mathbf{Adv}^{\mathrm{prp}}_{E}(B)$. However, $\widetilde{E}$ has since received little attention because the two blockcipher invocations prevent it from being competitive with the second construction in terms of performance. (Also note that the length of the tweak must be the length of the block.)

The second construction, $\mathsf{LRW} : \{0,1\}^{k+k'} \times \{0,1\}^{\tau} \times \{0,1\}^{n} \to \{0,1\}^{n}$, requires some ϵ-AXU hash function $H : \{0,1\}^{k'} \times \{0,1\}^{\tau} \to \{0,1\}^{n}$. Given H and E, we define $\mathsf{LRW}[H,E]_{K \,\|\, K'}(T,X) = H_{K'}(T) \oplus E_K(H_{K'}(T) \oplus E_K(X))$. See Figure 3.1.

This LRW construction has a similar security bound to the first, provided that $\epsilon = \theta(2^{-n})$. Such a bound is obtainable using, for example, the polynomial hash [56] $\mathsf{PolyHash} : \{0,1\}^{n} \times \{0,1\}^{nm} \to \{0,1\}^{n}$ given by

$$\mathsf{PolyHash}_K(X_1 X_2 \cdots X_m) = \sum X_i K^i.$$

Here, multiplication and addition take place in the finite field $\mathbb{F}_{2^n}$; $\mathsf{PolyHash}$ is $m/2^n$-AXU by the Fundamental Theorem of Algebra.

The XE and XEX [46] TBCs likewise obtain birthday-bound security.

Minematsu's TBC [38] can obtain better than birthday-bound security, but the tweak length τ must be less than half of the block length n. Moreover, it requires two blockcipher invocations to compute, and must re-key one of the underlying blockciphers each time the tweak changes. This incurs a significant performance penalty because re-keying is an expensive operation intended to be amortized over multiple blockcipher calls. On Intel's Haswell chips, for example, the `aeskeygenassist` instruction must be invoked ten times to setup a 128-bit AES key; it has an inverse throughput of eight cycles. (That is, although more than eight cycles may elapse between when a single `aeskeygenassist` instruction is issued and when it completes, the hardware pipeline can complete pending `aeskeygenassist` instructions at a rate of one every eight cycles.) In contrast, the `aesenc` and `aesenclast` instructions, which must also be invoked a combined ten times per AES call, have an inverse throughput of only a single cycle [7].

Dedicated constructions. Building a TBC from a blockcipher allows the latter to inherit the security of the former, with some loss. However, it's natural to wonder if constructing a TBC "from scratch" would provide a more efficient construction (and perhaps provide more security, as well). The difficulty here is that establishing the security of blockciphers from first principles has eluded cryptographers; instead, specific blockciphers are only deemed trustworthy after withstanding years of expert scrutiny. This problem is at least as hard in the context of TBCs, because TBCs are more powerful objects.

Nonetheless, there have been attempts at constructing TBCs without using some underlying blockcipher. The Hasty Pudding [52] cipher predates the formalization of tweakable ciphers, and was an entry in the AES competition. Ferguson et al. invented Threefish [17] for use with the Skein hash function, a finalist in the SHA-3 competition. More recently, Jean, Nikolić, and Peyrin [26] proposed constructions that modify AES to support tweaks directly.

3.2 The LRW2 TBC

The centerpiece of this section is a TBC construction that provides beyond birthday-bound security, admits a large tweakspace, and does not require re-keying of any underlying object.

Given a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and a hash function family $H\colon \mathcal{K}_H \times \mathcal{D} \to \{0,1\}^n$, the LRW2 construction $\mathsf{LRW2}[H,E]\colon (\mathcal{K}_H)^2 \times (\{0,1\}^k)^2 \times \mathcal{D} \times \{0,1\}^n \to \{0,1\}^n$ is given by

$$\begin{aligned}\mathsf{LRW2}[H,E]_{h_1,h_2,K_1,K_2}(T,X) &= \mathsf{LRW}[H,E]_{h_2,K_2}(T,\mathsf{LRW}[H,E]_{h_1,K_1}(T,X))\\ &= E_{K_2}(E_{K_1}(X \oplus H_{h_1}(T)) \oplus H_{h_1}(T) \oplus H_{h_2}(T)) \oplus H_{h_2}(T).\end{aligned}$$

See Figure 3.2.

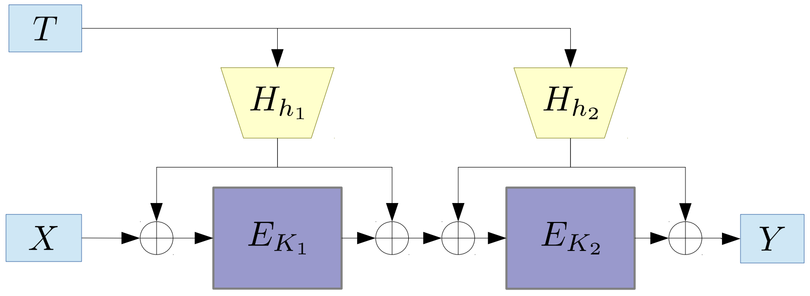

Figure 3.2: The LRW2TBC consists of two independently-keyed rounds of LRW chained together. Both rounds use the same tweak.

The following theorem establishes LRW2's security.

Theorem 1. *Fix $k, n > 0$ and let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Fix a non-empty set $\mathcal{K}_H$, and let $\mathcal{D} \subseteq \{0,1\}^*$. Let $H\colon \mathcal{K}_H \times \mathcal{D} \to \{0,1\}^n$ be an ϵ-AXU function family. Let $\widetilde{E} = \mathsf{LRW2}[H,E]$ be the LRW2 construction, defined above. Let A be an adversary asking a total of q queries to its oracles, running in time t. Let $\hat{\epsilon} = \max\{\epsilon, 1/(2^n - 2q)\}$. Then there exists an adversary B using the same resources, such that*

$$\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\widetilde{E}}(A) \le 2\mathbf{Adv}^{\mathrm{sprp}}_{E}(B) + \frac{4q^3\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}^2}.$$

This bound requires some interpretation. Consider $\epsilon \approx 2^{-n}$ (since there are efficient constructions meeting this, such as PolyHash), and assume $q \le 2^{n-2}$. Then $\hat{\epsilon} \le 1/2^{n-1} \approx 2^{-n}$ for interesting values of n. Now the additive term in the bound is at most p when $q \le (p/(p+6))^{1/3}\hat{\epsilon}^{-2/3}$, so for any small constant p we have $q = \mathcal{O}(2^{2n/3})$. Thus when $\mathbf{Adv}^{\mathrm{sprp}}_{E}(B)$ is sufficiently small, LRW2 is secure as a tweakable-SPRP up to about $2^{2n/3}$ queries.[1] Figure 3.3 gives a graphical comparison of our bound and

[1] We note that $\mathbf{Adv}^{\mathrm{sprp}}_{E}(B)$ will be at least $t/2^k \approx q/2^k$ by exhaustive key search so, $q = 2^{2n/3}$ requires $k > 2n/3$, which is met by AES ($k = n = 128$) and DES ($k = 56, n = 64$).

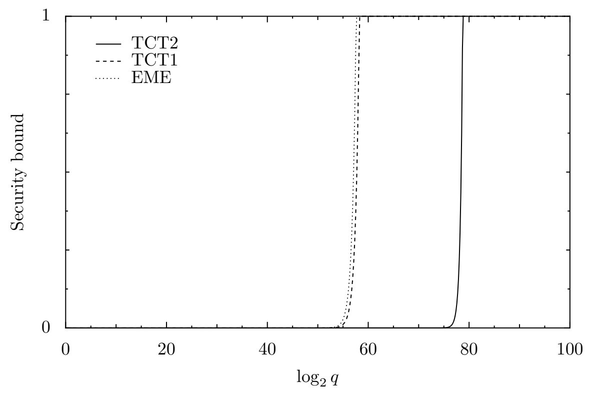

Figure 3.3: The maximum advantage of an adversary making q queries against $\mathsf{LRW2}$ (solid line) and constructions limited by the birthday bound, $q^2/2^n$ (dashed line). Here, $n = 128$, $\epsilon = 2^{-n}$, and we have assumed the $\mathbf{Adv}_E^{\mathrm{sprp}}(B)$ term is negligible.

the standard birthday bound.

Proof overview. The proof of Theorem 1 is quite long and involved, so we'll start by giving a high-level overview of it. Proofs demonstrating birthday-bound security for TBC constructions typically "give up" if the adversary can cause a collision at a blockcipher input. In constructions like LRW and XEX, the TBC output is no longer random, even when the blockcipher has been replaced by a random permutation. We overcome this problem by using two rounds of $\mathsf{LRW2}$, and showing that it takes two independent collisions *on the same query* to force non-random $\mathsf{LRW2}$ outputs.

The chief difficulty is ensuring that the second $\mathsf{LRW2}$ round can withstand a collision so long as there was not also one on the first round. To this end, we argue that given a collision-free first round, the resulting distribution of $\mathsf{LRW2}$ output values — including those which *require* a second-round collision to obtain — is extremely close to that of an ideal TBC.

The bulk of the proof is a sequence of games bounding the success probability of an adversary in the information-theoretic setting, where the blockciphers have been replaced by random permutations. The first three games address first-round collisions, and show that the distribution of $\mathsf{LRW2}$ outputs is consistent with that of an ideal cipher unless there is simultaneous a second-round collision. Our next three games address the case in which there is no first-round collision. By swapping the order in which dependent random variables are assigned values, we can choose the output early on in the game, and gain insight into the distribution by which it is governed. This distribution is shown to be very close to the ideal one. The final two games are used to derive an upper bound for the probability that the adversary can set a "bad flag", which would force the game to exhibit non-ideal behavior. In the end, we are able to assume that the adversary is non-adaptive by giving it explicit control over oracle return values. At that point, the ϵ-AXU property can be applied.

Proof. For notational simplicity, we write h_1 for H_{h_1}, and h_2 for H_{h_2}; this should cause no confusion. The majority of the proof will consider the construction $\mathsf{LRW2}$ with E_{K_1} and E_{K_2} replaced with random permutations π_1 and π_2, which we write as $\mathsf{LRW2}_{h_1,h_2,\pi_i,\pi_2}$. At the end we can make a standard move to lift to the fully complexity theoretic setting.

Let A be an adversary making q queries. If the i^{th} query is to the left (encryption) oracle, we denote the query with (T_i, X_i) and the response with Y_i; if the query is to the right (decryption) oracle, the roles of X_i and Y_i are reversed. We denote by $\mathcal{Y}_i$ the set of permissible (tweak-respecting) return values for an encryption oracle query, and similarly, $\mathcal{X}_i$ is the set of permissible return values for a decryption oracle query. That is,

$$\mathcal{Y}_i = \{0,1\}^n \setminus \{Y_j \,:\, j < i, T_j = T_i\}$$
$$\mathcal{X}_i = \{0,1\}^n \setminus \{X_j \,:\, j < i, T_j = T_i\}\,.$$

Given a set $S \subseteq \{0,1\}^n$ and a string $x \in \{0,1\}^n$ we define $S \oplus x = \{s \oplus x \,:\, s \in S\}$.

The permutations π_1 and π_2 are constructed lazily, while h_1 and h_2 are already defined. Initially, boolean variables have the value false, integers are zero, and all other variable types are undefined (equal to $\perp$).

Game G_4 of Listing 3.1 (pg. 33) simulates $\mathsf{LRW2}$ exactly by lazily sampling values for π_1 and π_2. Note that there is a certain symmetry between the encryption and decryption oracles. This symmetry arises from the fact that $\mathsf{LRW2}$ is the dual of $\mathsf{LRW2}^{-1}$, in the sense that $\mathsf{LRW2}^{-1}_{h_1,h_2,\pi_1,\pi_2}(Y,T) = \mathsf{LRW2}_{h_2,h_1,\pi_2^{-1},\pi_1^{-1}}(Y,T)$.

The bulk of this proof concerns showing that a sequence of games are identical, or are identical until a specified event occurs (a boolean variable is set to true). When arguing that transitions between games are correct in this sense, we will exploit the above symmetry by limiting our discussion to changes in the encryption oracle, and hence to queries made to that oracle; the arguments used to justify the corresponding changes in the decryption oracle are practically identical. Therefore fix some value $i \in [1..q]$, and assume the i^{th} query is to the encryption oracle.

In Game G_5 of Listing 3.2 (pg. 34), we change what happens when there is a collision at the first block cipher: we sample Y_i from the ideal distribution, but raise a bad flag if we also encounter a collision at the input of second block cipher (bad_1) or if Y_i is already in the defined range (bad_2). Game G_6, in the same listing, is identical to Game G_5, except Y_i is not reassigned after a bad flag is set. Hence $\Pr[\, G_4(A) \Rightarrow 1\,] = \Pr[\, G_5(A) \Rightarrow 1\,] \leq \Pr[\, G_6(A) \Rightarrow 1\,] + \Pr[\, G_6(A) : \mathsf{bad}_1 \vee \mathsf{bad}_2\,]$.

Next we modify the section of code in Game G_6 that is executed when no collision occurs at π_1; i.e., when $X_i \oplus h_1(T_i) \neq X_j \oplus h_1(T_j)$ for all $j < i$. Note that the random variables Z and Y_i are dependent. In Game G_6, Z is chosen before Y_i, but as long as the joint distribution as preserved we may reverse this order. The resulting game will be equivalent to Game G_6. As always, the decryption oracle will be modified in a similar manner.

To describe the correct distribution for Y_i, partition $\{0,1\}^n$ into four sets, S_1, S_2, S_3 and S_4. These sets are defined with respect to an oracle query (T_i, X_i) such that no collision occurs at π_1; that is, such that $X_i \oplus h_1(T_i) \notin \text{Dom}\,(\pi_1)$. (When referring to $\text{Dom}\,(\cdot)$ outside of pseudocode, we refer to the set of points at which the function is defined at the instant the adversary makes its i^{th} oracle call [and similarly for $\text{Rng}\,(\cdot)$]; the game currently being used to define the oracle should be clear from context). For $y \in \{0,1\}^n$, we say y is *permissible* when $y \in \mathcal{Y}_i$, and y is *possible* when $\Pr[\,Y_i = y\,] > 0$, given our assumption that $X_i \oplus h_1(T_i) \notin \text{Dom}\,(\pi_1)$ and the oracles' execution histories for the first $i-1$ queries.

Let S_4 be the set of all non-permissible values. Note that if y is not permissible (it has been returned on a query that used tweak T_i), then y is not possible (since $\mathsf{LRW2}(T_i, \cdot)$ is a permutation and queries are distinct); hence S_4 is a subset of the impossible values. Let S_3 be the set of impossible values that *are* permissible.

We now subdivide the set of possible values based on the conditional branch on Line 6 in Game G_6. Some values for Y_i will only be returned if the choice of Z causes a collision at π_2, while others can only be assigned in the absence of such a collision; the former will be S_2, the latter S_1. More formally, one can see that y is not possible if and only if $y \oplus h_2(T_i) \in \text{Rng}\,(\pi_2)$ and $\pi_2^{-1}(y \oplus h_2(T_i)) \oplus h_2(T_i) \oplus h_1(T_i) \in \text{Rng}\,(\pi_1)$ Therefore let $S_1 = \{y : y \oplus h_2(T_i) \notin \text{Rng}\,(\pi_2)\}$, and let S_2 be the set of all other possible values.

In summary,

$$
\begin{aligned}
S_1 &= \{y \;:\; y \oplus h_2(T_i) \notin \mathrm{Rng}\,(\pi_2)\} \\
S_2 &= \left\{y \;:\; y \oplus h_2(T_i) \in \mathrm{Rng}\,(\pi_2)\,, \pi_2^{-1}(y \oplus h_2(T_i)) \oplus h_2(T_i) \oplus h_1(T_i) \in \overline{\mathrm{Rng}\,(\pi_1)}\right\} \\
S_3 &= \mathcal{Y}_i \setminus (S_1 \cup S_2) \\
&= \left\{y \;:\; y \oplus h_2(T_i) \in \mathrm{Rng}\,(\pi_2)\,,\; \pi_2^{-1}(y \oplus h_2(T_i)) \oplus h_2(T_i) \oplus h_1(T_i) \in \mathrm{Rng}\,(\pi_1)\right\} \setminus \overline{\mathcal{Y}_i} \\
S_4 &= \overline{\mathcal{Y}_i} = \{Y_j \;:\; j < i, T_j = T_i\}\,.
\end{aligned}
$$

When these sets are used in pseudocode, it is understood that they are defined at the time the oracle call is made; although $\mathrm{Rng}\,(\pi_1)$ (for example) may change as code executes, S_2 does not change until the next query. When referred to by a decryption oracle, the definitions for these sets are the same up to the previously mentioned duality.

We will now compute the probability that Y_i will be in each of these sets (again, under the assumption that there is no collision at the first block cipher; i.e, that $L_i = X_i \oplus h_1(T_i) \notin \mathrm{Dom}\,(\pi_1)$). Since S_3 and S_4 contain only impossible values, $\Pr[\, Y_i \in S_3 \cup S_4 \mid L_i \notin \mathrm{Dom}\,(\pi_1)\,] = 0$. Let $N = \left|\overline{\mathrm{Rng}\,(\pi_1)}\right|$. Given $y \in S_2$ and $L_i \notin \mathrm{Dom}\,(\pi_1)$, $Y_i = y$ if and only if $Z = \pi_2^{-1}(y \oplus h_2(T_i)) \oplus h_2(T_i) \oplus h_2(T_i)$. This value is in $\overline{\mathrm{Rng}\,(\pi_1)}$ by definition of S_2, and so this event happens with probability $1/N$. Hence,

$$
\begin{aligned}
\Pr[\, Y_i \in S_2 \mid L_i \notin \mathrm{Dom}\,(\pi_1)\,] &= |S_2|\,/N, \text{ and} \\
\Pr[\, Y_i \in S_1 \mid L_i \notin \mathrm{Dom}\,(\pi_1)\,] &= (N - |S_2|)/N.
\end{aligned}
$$

Ideally, Y_i would be distributed as $p_{\mathsf{TBC}}(y) := \Pr\left[\, Y \xleftarrow{\$} \mathcal{Y}_i\,;\, Y = y\,\right] = 1/(2^n - |S_4|)$ (for $y \notin S_4$) and zero otherwise. However, we have shown that if there is no collision at π_1 on the i^{th} query, then Y_i is distributed as

$$p_{\mathsf{lazy}}(y) := \Pr\left[\, Y_i = y \mid L_i \notin \mathrm{Dom}\,(\pi_1)\,\right] = \begin{cases} \frac{N-|S_2|}{N|S_1|} & \text{if } y \in S_1 \\ \frac{1}{N} & \text{if } y \in S_2 \\ 0 & \text{if } y \in S_3 \cup S_4 \end{cases}$$

See Figure 3.4.

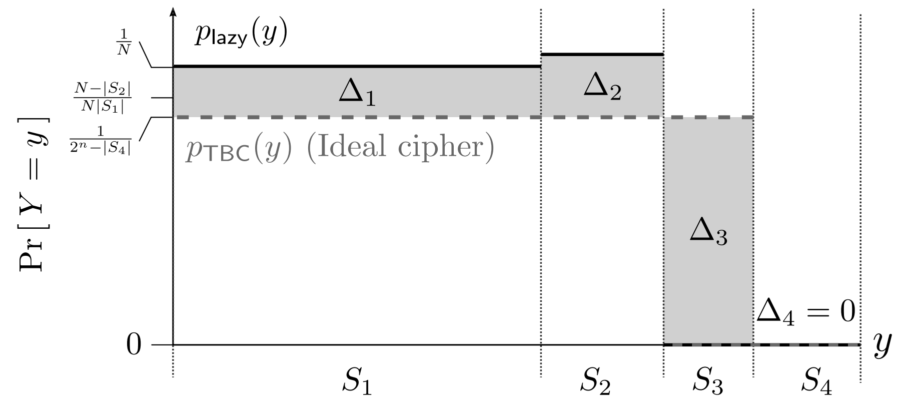

Figure 3.4: The distribution governing an oracle output Y in Game G_6 given a collision at the first blockcipher input (solid line) compared to the distribution an ideal cipher would provide (dashed line). For most parameters of interest, the statistical distance between these distributions will be negligible.

Although this distribution is not quite what we want, we will show that it is close enough (even against birthday-type attacks). In particular, the statistical distance

$$\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) := \frac{1}{2} \sum_{y \in \{0,1\}^n} |p_{\mathsf{lazy}}(y) - p_{\mathsf{TBC}}(y)| = \max_{S \subseteq \{0,1\}^n} \sum_{y \in S} (p_{\mathsf{lazy}}(y) - p_{\mathsf{TBC}}(y))$$

will be on the order of $q^2/2^{2n}$. Geometrically, this quantity corresponds to half the shaded area in Figure 3.4. It can also be viewed as the area above p_{TBC} but below p_{lazy} (or the other way around).

We integrate statistical distance into the game-playing proof by exploiting the existence of a optimal *coupling distribution* $\Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ (see, e.g., Lemma 11.3 of [40]). This distribution samples points from $\{0,1\}^n \times \{0,1\}^n$ and has the property that when $(Y, Y') \xleftarrow{\$} \Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$:

1. For all $y \in \{0,1\}^n$, $\Pr[\, Y = y \,] = p_{\mathsf{lazy}}(y)$ and $\Pr[\, Y' = y \,] = p_{\mathsf{TBC}}(y)$.
2. Except with probability $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$, $Y = Y'$.

So in Game G_7 (pg. 35), we sample $(Y, Y') \xleftarrow{\$} \Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ and return Y. But we also need to lazily sample a point Z for π_1 that is consistent with our choice. If $Y \in S_2$, then our decision is forced. On the other hand, if $Y \in S_1$, then we can choose any value for $Z \in S' = \overline{\mathrm{Rng}\,(\pi_1)} \cap \{z \,:\, z \oplus h_1(T_i) \oplus h_2(T_i) \notin \mathrm{Dom}\,(\pi_2)\}$. Sampling $Z \xleftarrow{\$} S'$ uniformly satisfies the need for the joint distribution on (Z, Y_i) to be identical in Games G_6 and G_7; e.g., in Game G_7, for any fixed $z \in S'$:

$$\Pr[\, Z = z \,] = \Pr[\, Z = z \mid Y_i \in S_1 \,] \Pr[\, Y \in S_1 \,] = \frac{1}{|S'|}\left(\frac{N - |S_2|}{N}\right) = \frac{1}{N}.$$

(By construction, $|S'| = N - |S_2|$).

Thus $\Pr[\, G_6(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,] = \Pr[\, G_7(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,]$ and $\Pr[\, G_6(A) \Rightarrow 1 \,] = \Pr[\, G_7(A) \Rightarrow 1 \,]$.

Then in Game G_8, we return Y' instead of Y after sampling $(Y, Y') \xleftarrow{\$} \Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$. Since $Y \neq Y'$ only with probability $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$, most of the time these two games will behave identically. We set bad_3 when they do not:

$$\Pr[\, G_7(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,] - \Pr[\, G_8(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,] \le \Pr[\, G_8(A)\,;\, \mathsf{bad}_3 \,],$$

$$\Pr[\, G_7(A) \Rightarrow 1 \,] - \Pr[\, G_8(A) \Rightarrow 1 \,] \le \Pr[\, G_8(A)\,;\, \mathsf{bad}_3 \,].$$

Now that we return Y' instead of Y, the random variable Y serves no direct function. So in Game G_9, we forgo assigning Y. Instead, we sample $Y_i \xleftarrow{\$} \mathcal{Y}_i$, and then set bad_3 with probability $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$. Values for π_1 and π_2 are chosen as before. Thus, Games G_8 and G_9 are equivalent.

At this point, Y_i is always sampled from $\mathcal{Y}_i$, and once assigned, its value is never changed.

In Game G_{10}, we give the adversary control over what value is assigned to Y_i (or X_i, in the case of decryption queries), but insist that the value be in $\mathcal{Y}_i$ or $\mathcal{X}_i$, as appropriate. Because the adversary can compute $\mathcal{Y}_i$ and $\mathcal{X}_i$, it may simulate the oracles of Game G_9 if desired; hence, he can set the bad flags in Game G_{10} with probability at least as high as any adversary can set the corresponding flags in Game G_9. The oracle's outputs are now deterministic, and may be (trivially) computed by the adversary in advance. Hence, we may assume without loss of generality that the adversary is non-adaptive.

For the rest of this proof, all probabilities will be with respect to the experiment $G_{10}(A)$ (unless the experiment is explicitly stated).

Let $\mathcal{Q}$ be the event that for there exist i, j, and k (with $j, k \neq i$) such that $X_i \oplus h_1(T_i) = X_j \oplus h_1(T_j)$ and $Y_i \oplus h_2(T_i) = Y_k \oplus h_2(T_k)$. That is, $\mathcal{Q}$ indicates the i^{th} query is responsible for collisions at both π_1 and π_2. Our strategy is to show that $\mathcal{Q}$ is extremely unlikely, since it requires two independent collisions involving a single query. Barring such a query, we can show that the probability of a bad flag being set is very small.

By definition of $\mathcal{Q}$ and the ϵ-AXU property of H,

$$\begin{aligned}\Pr[\,\mathcal{Q}\,] &\leq \sum_{i=1}^{q} \sum_{j,k\neq i} \Pr[\,h_1(T_j) \oplus h_1(T_i) = X_j \oplus X_i\,] \Pr[\,h_2(T_k) \oplus h_2(T_i) = Y_k \oplus Y_i\,] \\ &< q^3\epsilon^2.\end{aligned}$$

Define

$$\beta_j = \max_{\widetilde{A}} \left(\Pr\left[\widetilde{A}^{G_{10}} : \mathsf{bad}_j \mid \neg\mathcal{Q} \right] \right), \text{ and}$$
$$\beta_j(i) = \max_{\widetilde{A}} \left(\Pr\left[\widetilde{A}^{G_{10}} : \mathsf{bad}_j \text{ on query } i \mid \neg\mathcal{Q} \right] \right).$$

We consider the event in the latter definition to "trigger" even if it has also triggered on an earlier query. (This definition assumes q is not so large that $\Pr[\neg\mathcal{Q}] = 0$, but since our bound becomes vacuous before this threshold, this is not an issue.) When bounding $\beta_j(i)$, we will assume the i^{th} query is made to the encryption oracle; as before, the other case is symmetric.

Because bad_2 can only be set if the conditions for $\mathcal{Q}$ are met, we immediately have that $\beta_2 \leq \Pr[\mathcal{Q}] \leq q^3\epsilon^2$.

Note that bad_1 is set on query i if and only if there exist $j, k < i$ such that

$$X_i \oplus h_1(T_i) = X_j \oplus h_1(T_j) \text{ and } \pi_1(L_i) \oplus h_1(T_i) \oplus h_2(T_i) = \pi_1(L_k) \oplus h_1(T_k) \oplus h_2(T_k),$$

where we remind the reader that $L_i = X_i \oplus h_1(T_i)$. Our goal now is to bound

$$\beta_1(i) = \Pr[\exists k < i \,:\, \pi_1(L_i) \oplus \pi_1(L_k) = R(i,k) \mid \exists j < i \,:\, L_i = L_j \wedge \neg\mathcal{Q}]$$
$$\cdot \Pr[\exists j < i \,:\, L_i = L_j \mid \neg\mathcal{Q}],$$

where, for brevity, we introduce $R(i,k) = h_1(T_i) \oplus h_2(T_i) \oplus h_1(T_k) \oplus h_2(T_k)$.

Because queries are unique and $\mathsf{LRW2}(T_i, \cdot)$ is a permutation, $L_i = L_j$ is only possible if $T_i \neq T_j$, bringing the ϵ-AXU property into scope. Hence

$$\begin{aligned}\Pr[\,\exists j < i \,:\, L_i = L_j \mid \neg\mathcal{Q}\,] &= \frac{\Pr[\,\exists j < i \,:\, L_i = L_j \wedge \neg\mathcal{Q}\,]}{\Pr[\,\neg\mathcal{Q}\,]} \\ &\leq \frac{\Pr[\,\exists j < i \,:\, L_i = L_j\,]}{1 - q^3\epsilon^2} \leq \frac{q\epsilon}{1 - q^3\epsilon^2}.\end{aligned}$$

We now wish to bound

$$\Pr[\,\exists k < i \,:\, \pi_1(L_i) \oplus \pi_1(L_k) = R(i,k) \mid \exists j < i \,:\, L_i = L_j \wedge \neg\mathcal{Q}\,]$$

(this is other factor in our bound for $\beta_1(i)$), so assume that there is some $j < i$ such that $L_i = L_j$ and that $\neg\mathcal{Q}$.

Fix $k \in [1..i-1]$. Consider the case that $L_i = L_k$. Then $\pi_1(L_i) = R(i,k)$ is equivalent to $h_1(T_i) \oplus h_1(T_k) = h_2(T_i) \oplus h_2(T_k)$. Because queries must respect per-tweak permutivity, $T_i \neq T_k$; hence by the ϵ-AXU property of H, in this case $\beta_1(i) \leq \epsilon$.

On the other hand, if $L_i \neq L_k$, we will trace the execution history of the game backwards to when the values eventually assigned to $\pi_1(L_i)$ and $\pi_1(L_k)$ become determined. Define $\operatorname{root}(x) = \min\{m \,:\, L_x = L_m\}$. Let $i' = \operatorname{root}(i)$, and let $k' = \operatorname{root}(k)$. Since $L_i = L_j$ for some $j < i$, it follows that $i' < i$. Therefore, by our assumption that $\mathcal{Q}$ does not occur, there is no $\ell \neq i'$ such that $Y_\ell \oplus h_2(T_\ell) = Y_{i'} \oplus h_2(T_{i'})$. Hence on query i', $\pi_1(L_i)$ is sampled from a set of size at least $2^n - 2q$; this sampling occurs indirectly through the random variable Z, itself sampled either on Line 813 or 836, depending on which oracle receives query i'.

Now we compute when the value of $\pi_1(L_k) = \pi_1(L_{k'})$ is determined. If there is no $\ell < k'$ such that $Y_\ell \oplus h_2(T_\ell) = Y_{k'} \oplus h_2(T_{k'})$, then $\pi_1(L_{k'})$ is likewise sampled indirectly from a set of size at least $2^n - 2q$. However, if such an ℓ exists, then $\pi_1(L_k) = \pi_2^{-1}(Y_{k'} \oplus h_2(T_{k'})) \oplus h_2(T_{k'}) \oplus h_1(T_{k'})$, and we are forced to backtrack further to when

$\pi_2^{-1}(Y_\ell \oplus h_2(T_\ell)) = \pi_2^{-1}(Y_{k'} \oplus h_2(T_{k'}))$ was defined. Fortunately, our assumption that the conditions for $\mathcal{Q}$ are not met saves us from having to backtrack far. Let $\ell' = \min\{m : Y_\ell \oplus h_2(T_\ell) = Y_m \oplus h_2(T_m)\}$. Then $\neg\mathcal{Q}$ implies $\ell' = \mathrm{root}(\ell')$. Hence on query ℓ', $\pi_2^{-1}(Y_{\ell'} \oplus h_2(T_{\ell'})) = \pi_2^{-1}(Y_\ell \oplus h_2(T_\ell))$ is sampled, through Z, from a set of size at least $2^n - 2q$. In the first of these two cases, let $r = k'$; in the second, let $r = \ell'$. After query r completes, the value which will be assigned to $\pi_1(L_k)$ is deterministic.

Suppose without loss of generality that $i' > r$. Then $\pi_1(L_i) = \pi_1(L_k) \oplus R(i,k)$ only if on query i', $\pi_1(L_i) = \pi_1(L_{i'})$ is assigned the unique value that makes the former equation true; this happens with probability at most $1/(2^n - 2q)$.

Let $\hat{\epsilon} = \max(\epsilon, 1/(2^n - 2q))$. Then

$$\Pr[\, \pi_1(L_i) \oplus \pi_1(L_k) = R(i,k) \mid \exists j < i \,:\, L_i = L_j \wedge \neg\mathcal{Q} \,] \le \hat{\epsilon}.$$

We have

$$\beta_1 \le \sum_{i=1}^{q} \beta_1(i) \le \sum_{i=1}^{q} \sum_{k=1}^{i-1} \frac{q\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}} < \frac{q^3\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}^2}.$$

Our final task is to bound $\Pr[\,\mathsf{bad}_3\,] = \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$. For $j = 1,2,3$, define $\Delta_j = \sum_{y \in S_j} (p_{\mathsf{lazy}}(y) - p_{\mathsf{TBC}}(y))$. Since for any $y, y' \in S_j$, $p_{\mathsf{lazy}}(y) = p_{\mathsf{lazy}}(y')$ (and p_{TBC} is constant in these sets), we have:

$$\beta_3(i) = \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) = \frac{1}{2}\left(|\Delta_1| + |\Delta_2| + |\Delta_3|\right).$$

The law of total probability tells us that $\Delta_1 + \Delta_2 + \Delta_3 = 0$, and, by construction of S_3, $\Delta_3 \le 0$. Further, since $\pi_2^{-1}(S_4) \oplus h_2(T_i) \oplus h_1(T_i) \subseteq \mathrm{Rng}(\pi_1)$, we have that for $y \in S_2$: $p_{\mathsf{lazy}}(y) - p_{\mathsf{TBC}}(y) = 1/N - 1/(2^n - |S_4|) \ge 0$. That is, $\Delta_2 \ge 0$.

Therefore either $\Delta_1 \le 0$ (in which case $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) = \Delta_2$), or $\Delta_1 > 0$ (in which case $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) = -\Delta_3$). Hence, $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) \le \max(\Delta_2, -\Delta_3)$.

The quantity Δ_2 can be bounded as follows:

$$\begin{aligned}
\Delta_2 &= |S_2| \left(\frac{1}{N} - \frac{1}{2^n - |S_4|} \right) \leq \frac{q}{N(2^n - |S_4|)} \left(2^n - |S_4| - N\right) \\
&\leq \frac{q}{N(2^n - q)} \left(2^n - N\right) \leq \frac{q}{N(2^n - q)} \left(2^n - (2^n - q)\right) \\
&= \frac{q^2}{(2^n - q)^2}.
\end{aligned}$$

It remains to bound $-\Delta_3 = \Pr\left[\, Y \xleftarrow{\$} \mathcal{Y}_i \,;\, Y \in S_3 \,\right]$. Note that $Y \in S_3$ only if there exists $j, k < i$ such that $X_j \oplus h_1(T_j) = X_i \oplus h_1(T_i)$ and $Y_k \oplus h_2(T_k) = Y \oplus h_1(T_i)$. We appeal to the ϵ-AXU property as before to argue that the probability of such j and k existing is at most $(q\epsilon)^2$.

In both cases, $\beta_3(i) = \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}}) \leq q^2\hat{\epsilon}^2$. (Recall that $\hat{\epsilon} = \max\{\epsilon, 1/(2^n - 2q)\}$.) Therefore $\beta_3 \leq q^3\hat{\epsilon}^2$. By the fundamental lemma of game playing,

$$\begin{aligned}
\Pr\left[\, A^{\mathsf{LRW2}_{h_1,h_2,\pi_1,\pi_2}(\cdot,\cdot),\mathsf{LRW2}^{-1}_{h_1,h_2,\pi_1,\pi_2}(\cdot,\cdot)} \Rightarrow 1 \,\right] &= \Pr\left[\, G_4(A) \Rightarrow 1 \,\right] \\
&\leq \Pr\left[\, G_6(A) \Rightarrow 1 \,\right] + \Pr\left[\, G_6(A) : \mathsf{bad}_1 \vee \mathsf{bad}_2 \,\right] \\
&\leq \Pr\left[\, G_9(A) \Rightarrow 1 \,\right] + \Pr\left[\, G_9(A) : \mathsf{bad}_1 \vee \mathsf{bad}_2 \,\right] + 2\Pr\left[\, G_9(A) : \mathsf{bad}_3 \,\right] \\
&\leq \Pr\left[\, G_9(A) \Rightarrow 1 \,\right] + \Pr\left[\, G_{10}(A) : \mathsf{bad}_1 \vee \mathsf{bad}_2 \,\right] + 2\Pr\left[\, G_{10}(A) : \mathsf{bad}_3 \,\right] \\
&\leq \Pr\left[\, G_9(A) \Rightarrow 1 \,\right] + \beta_1 + \Pr\left[\, \mathcal{Q} \,\right] + 2\beta_3 \\
&\leq \Pr\left[\, A^{\Pi(\cdot,\cdot),\Pi^{-1}(\cdot,\cdot)} \Rightarrow 1 \,\right] + \frac{4q^3\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}^2}.
\end{aligned}$$

Thus by a standard reduction argument, there exists a B such that

$$\mathbf{Adv}^{\widetilde{\text{sprp}}}_{\mathsf{LRW2}}(A) \leq 2\mathbf{Adv}^{\text{sprp}}_{E}(B) + \frac{4q^3\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}^2}.$$

This completes the proof. □

A note on an error in a previous version. An earlier version of this proof [29] attempted to, in effect, construct $\Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ explicitly — but failed to do so correctly: we erroneously made a tacit assumption that $\Delta_1 \geq 0$. We thank Gordon Procter for bringing this mistake to our attention. Procter also helpfully provided a suggested patch [44], which uses an **if/else** clause in the game-playing proof to address the two cases $\Delta_1 \geq 0$ and $\Delta_1 < 0$ separately. We believe Procter's solution succeeds in fixing the problem. Ultimately, however, we decided to use a coupling distribution to abstract the details of transitioning from p_{lazy} to p_{TBC}. This simplifies some of the arguments, but admittedly sacrifices some of the explicitness present in Procter's proof.

Attacks on simpler variants. Having seen our construction, one may wonder if simpler variants work. For example, consider $\mathsf{LRW2}$ without the first $H_{h_2}(T)$ XOR operation, leaving

$$\mathsf{LRW2}_{h_1,h_2,K_1,K_2}(T, X) = E_{K_2}(E_{K_1}(H_{h_1}(T) \oplus X) \oplus H_{h_1}(T)) \oplus H_{h_2}(T).$$

This variation permits a birthday-bound attack. Namely, an adversary could submit queries in pairs, (T_i, X') and (T_i, X''), where X' and X'' are fixed, and a new random tweak is used for each pair. By remembering the values $\mathsf{LRW2}(T_i, X') \oplus \mathsf{LRW2}(T_i, X'')$, which are independent of H_{h_2}, it could detect collisions in H_{h_1}, say by using a hash table. That is, if $H_{h_1}(T_i) = H_{h_1}(T_j)$, then $\mathsf{LRW2}(T_i, X') \oplus \mathsf{LRW2}(T_i, X'') = \mathsf{LRW2}(T_j, X') \oplus \mathsf{LRW2}(T_j, X'')$. The converse is false, but false positives could be weeded out by testing a small number of X-values. Such an adversary would gain advantage close to one. Similar variations on $\mathsf{LRW2}$ permit analogous attacks, though we believe (but have not proven) that omitting the second $H_{h_1}(T)$ XOR operation yields

GAME G_4

```
Oracle LRW2$(T, X)$:
  $i \leftarrow i + 1$; $X_i \leftarrow X$; $T_i \leftarrow T$
  $L_i \leftarrow X_i \oplus h_1(T_i)$
  if $L_i \in \mathrm{Dom}(\pi_1)$ then
    $M_i \leftarrow \pi_1(L_i) \oplus h_1(T_i) \oplus h_2(T_i)$
    if $M_i \in \mathrm{Dom}(\pi_2)$ then
      $Y_i \leftarrow \pi_2(M_i) \oplus h_2(T_i)$
    else
      $Y_i \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_2)} \oplus h_2(T_i)$
      $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
  else
    $Z \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_1)}$; $\pi_1(L_i) \leftarrow Z$
    $M_i \leftarrow \pi_1(L_i) \oplus h_1(T_i) \oplus h_2(T_i)$
    if $M_i \in \mathrm{Dom}(\pi_2)$ then
      $Y_i \leftarrow \pi_2(M_i) \oplus h_2(T_i)$
    else
      $Y_i \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_2)} \oplus h_2(T_i)$
      $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
  return $Y_i$
```

```
Oracle LRW2$^{-1}(T, Y)$:
  $i \leftarrow i + 1$; $Y_i \leftarrow Y$; $T_i \leftarrow T$
  $N_i \leftarrow Y_i \oplus h_2(T_i)$
  if $N_i \in \mathrm{Rng}(\pi_2)$ then
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus h_2(T_i) \oplus h_1(T_i)$
    if $M_i \in \mathrm{Rng}(\pi_1)$ then
      $X_i \leftarrow \pi_1^{-1}(M_i) \oplus h_1(T_i)$
    else
      $X_i \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_1)} \oplus h_1(T_i)$
      $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
  else
    $Z \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_2)}$; $\pi_2^{-1}(N_i) \leftarrow Z$
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus h_2(T_i) \oplus h_1(T_i)$
    if $M_i \in \mathrm{Rng}(\pi_1)$ then
      $X_i \leftarrow \pi_1^{-1}(M_i) \oplus h_1(T_i)$
    else
      $X_i \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_1)} \oplus h_1(T_i)$
      $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
  return $X_i$
```

Listing 3.1: Game G_4 simulates LRW2 by using lazy sampling to define the random permutations.

Games $\boxed{G_5}$, G_6

```
Oracle LRW2(T, X):
  $i \leftarrow i+1$; $X_i \leftarrow X$; $T_i \leftarrow T$
  $L_i \leftarrow X_i \oplus h_1(T_i)$
  if $L_i \in \mathrm{Dom}(\pi_1)$ then
    $M_i \leftarrow \pi_1(L_i) \oplus h_1(T_i) \oplus h_2(T_i)$
    $Y_i \xleftarrow{\$} \mathcal{Y}_i$
    if $M_i \in \mathrm{Dom}(\pi_2)$ then
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
      $\boxed{Y_i \leftarrow \pi_2(M_i) \oplus h_2(T_i)}$
    else
      if $Y_i \oplus h_2(T_i) \in \mathrm{Rng}(\pi_2)$ then
        $\mathsf{bad}_2 \leftarrow \mathsf{true}$
        $\boxed{Y_i \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_2)} \oplus h_2(T_i)}$
      $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
  else
    $Z \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_1)}$; $\pi_1(L_i) \leftarrow Z$
    $M_i \leftarrow \pi_1(L_i) \oplus h_1(T_i) \oplus h_2(T_i)$
    if $M_i \in \mathrm{Dom}(\pi_2)$ then
      $Y_i \leftarrow \pi_2(M_i) \oplus h_2(T_i)$
    else
      $Y_i \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_2)} \oplus h_2(T_i)$
      $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
  return $Y_i$
```

```
Oracle LRW2$^{-1}$(T, Y):
  $i \leftarrow i+1$; $Y_i \leftarrow Y$; $T_i \leftarrow T$
  $N_i \leftarrow Y_i \oplus h_2(T_i)$
  if $N_i \in \mathrm{Rng}(\pi_2)$ then
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus h_2(T_i) \oplus h_1(T_i)$
    $X_i \xleftarrow{\$} \mathcal{X}_i$
    if $M_i \in \mathrm{Rng}(\pi_1)$ then
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
      $\boxed{X_i \leftarrow \pi_1^{-1}(M_i) \oplus h_1(T_i)}$
    else
      if $X_i \oplus h_1(T_i) \in \mathrm{Dom}(\pi_1)$ then
        $\mathsf{bad}_2 \leftarrow \mathsf{true}$
        $\boxed{X_i \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_1)} \oplus h_1(T_i)}$
      $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
  else
    $Z \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_2)}$; $\pi_2^{-1}(N_i) \leftarrow Z$
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus h_2(T_i) \oplus h_1(T_i)$
    if $M_i \in \mathrm{Rng}(\pi_1)$ then
      $X_i \leftarrow \pi_1^{-1}(M_i) \oplus h_1(T_i)$
    else
      $X_i \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_1)} \oplus h_1(T_i)$
      $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
  return $X_i$
```

Listing 3.2: Game G_5 behaves identically to Game G_4, except we set a flag if either (1) there are collisions at the inputs to both blockciphers or (2) there is a collision at the input of the first and the output of the second. Game G_6 is the same, except we resample Y_i after setting one of these flags.

GAMES $\boxed{G_7}$, G_8

```
Oracle LRW2($T, X$):
$i \leftarrow i + 1$; $X_i \leftarrow X$; $T_i \leftarrow T$
$L_i \leftarrow X_i \oplus h_1(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $L_i \in \mathrm{Dom}(\pi_1)$ then
  $M_i \leftarrow \pi_1(L_i) \oplus H$
  $Y_i \xleftarrow{\$} \mathcal{Y}_i$
  if $M_i \in \mathrm{Dom}(\pi_2)$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  else
    if $Y_i \oplus h_2(T_i) \in \mathrm{Rng}(\pi_2)$ then
      $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
else
  $(Y, Y') \xleftarrow{\$} \Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$
  $Y_i \leftarrow Y'$;
  if $Y \neq Y'$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
    $\boxed{Y_i \leftarrow Y}$
  if $Y_i \in S_2$ then
    $Z \leftarrow \pi_2^{-1}(Y_i \oplus h_2(T_i)) \oplus H$
  else if $Y_i \in S_1$
    $Z \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_1)} \setminus (\mathrm{Dom}(\pi_2) \oplus H)$
    $\pi_2(Z \oplus H) \leftarrow Y_i \oplus h_2(T_i)$
  $\pi_1(L_i) \leftarrow Z$
  $M_i \leftarrow \pi_1(L_i) \oplus H$
return $Y_i$
```

```
Oracle LRW2$^{-1}$($T, Y$):
$i \leftarrow i + 1$; $Y_i \leftarrow Y$; $T_i \leftarrow T$
$N_i \leftarrow Y_i \oplus h_2(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $N_i \in \mathrm{Rng}(\pi_2)$ then
  $M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
  $X_i \xleftarrow{\$} \mathcal{X}_i$
  if $M_i \in \mathrm{Rng}(\pi_1)$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  else
    if $X_i \oplus h_1(T_i) \in \mathrm{Dom}(\pi_1)$ then
      $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
else
  $(X, X') \xleftarrow{\$} \Gamma(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$
  if $X \neq X'$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
    $X_i \leftarrow X'$;   $\boxed{X_i \leftarrow X}$
  if $X_i \in S_2$ then
    $Z \leftarrow \pi_1(X_i \oplus h_1(T_i)) \oplus H$
  else if $X_i \in S_1$
    $Z \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_2)} \setminus (\mathrm{Rng}(\pi_1) \oplus H)$
    $\pi_1^{-1}(Z \oplus H) \leftarrow X_i \oplus h_1(T_i)$
  $\pi_2^{-1}(N_i) \leftarrow Z$
  $M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
return $X_i$
```

Listing 3.3: In Game G_7, we use a coupling Γ to sample random variables from the distribution of Game G_6 (p_{lazy}) and the distribution of an ideal TBC (p_{TBC}). We return the former. In Game G_8, we return the latter instead.

Game G_9

```
Oracle LRW2$(T, X)$:
$i \leftarrow i+1$; $X_i \leftarrow X$; $T_i \leftarrow T$
$L_i \leftarrow X_i \oplus h_1(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $L_i \in \mathrm{Dom}(\pi_1)$ then
    $M_i \leftarrow \pi_1(L_i) \oplus H$
    $Y_i \xleftarrow{\$} \mathcal{Y}_i$
    if $M_i \in \mathrm{Dom}(\pi_2)$ then
        $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    else
        if $Y_i \oplus h_2(T_i) \in \mathrm{Rng}(\pi_2)$ then
            $\mathsf{bad}_2 \leftarrow \mathsf{true}$
        $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
else
    $Y_i \xleftarrow{\$} \mathcal{Y}_i$
    $V \xleftarrow{\$} \{w \in \mathbb{R} : 0 \leq w \leq 1\}$
    if $V < \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ then $\mathsf{bad}_3 \xleftarrow{\$} \mathsf{true}$
    if $Y_i \in S_2$ then
        $Z \leftarrow \pi_2^{-1}(Y_i \oplus h_2(T_i)) \oplus H$
    else if $Y_i \in S_1$
        $Z \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_1)} \setminus (\mathrm{Dom}(\pi_2) \oplus H)$
        $\pi_2(Z \oplus H) \leftarrow Y_i \oplus h_2(T_i)$
    $\pi_1(L_i) \leftarrow Z$
    $M_i \leftarrow \pi_1(L_i) \oplus H$
return $Y_i$
```

```
Oracle LRW2$^{-1}(T, Y)$:
$i \leftarrow i+1$; $Y_i \leftarrow Y$; $T_i \leftarrow T$
$N_i \leftarrow Y_i \oplus h_2(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $N_i \in \mathrm{Rng}(\pi_2)$ then
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
    $X_i \xleftarrow{\$} \mathcal{X}_i$
    if $M_i \in \mathrm{Rng}(\pi_1)$ then
        $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    else
        if $X_i \oplus h_1(T_i) \in \mathrm{Dom}(\pi_1)$ then
            $\mathsf{bad}_2 \leftarrow \mathsf{true}$
        $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
else
    $X_i \xleftarrow{\$} \mathcal{X}_i$
    $V \xleftarrow{\$} \{w \in \mathbb{R} : 0 \leq w \leq 1\}$
    if $V < \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ then $\mathsf{bad}_3 \xleftarrow{\$} \mathsf{true}$
    if $X_i \in S_2$ then
        $Z \leftarrow \pi_1(X_i \oplus h_1(T_i)) \oplus H$
    else if $X_i \in S_1$
        $Z \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_2)} \setminus (\mathrm{Rng}(\pi_1) \oplus H)$
        $\pi_1^{-1}(Z \oplus H) \leftarrow X_i \oplus h_1(T_i)$
    $\pi_2^{-1}(N_i) \leftarrow Z$
    $M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
return $X_i$
```

Listing 3.4: Since the previous game discarded one of the coupled random variables, we no longer use the coupling distribution here. Instead, we sample directly from the ideal distribution but still set bad_3 with probability $\delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$.

Game G_{10}

```
Oracle LRW2$(T, X, Y)$:
$i \leftarrow i+1$; $X_i \leftarrow X$; $T_i \leftarrow T$
$Y_i \xleftarrow{\$} Y$
$L_i \leftarrow X_i \oplus h_1(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $L_i \in \mathrm{Dom}(\pi_1)$ then
   $M_i \leftarrow \pi_1(L_i) \oplus H$
   if $M_i \in \mathrm{Dom}(\pi_2)$ then
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
   else
      if $Y_i \oplus h_2(T_i) \in \mathrm{Rng}(\pi_2)$ then
         $\mathsf{bad}_2 \leftarrow \mathsf{true}$
      $\pi_2(M_i) \leftarrow Y_i \oplus h_2(T_i)$
else
   $V \xleftarrow{\$} \{w \in \mathbb{R} : 0 \leq w \leq 1\}$
   if $V < \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ then $\mathsf{bad}_3 \xleftarrow{\$} \mathsf{true}$
   if $Y_i \in S_2$ then
      $Z \leftarrow \pi_2^{-1}(Y_i \oplus h_2(T_i)) \oplus H$
   else if $Y_i \in S_1$
      $Z \xleftarrow{\$} \overline{\mathrm{Rng}(\pi_1)} \setminus (\mathrm{Dom}(\pi_2) \oplus H)$
      $\pi_2(Z \oplus H) \leftarrow Y_i \oplus h_2(T_i)$
   $\pi_1(L_i) \leftarrow Z$
   $M_i \leftarrow \pi_1(L_i) \oplus H$
return $Y_i$
```

```
Oracle LRW2$^{-1}(T, Y, X)$:
$i \leftarrow i+1$; $Y_i \leftarrow Y$; $T_i \leftarrow T$
$X_i \xleftarrow{\$} X$
$N_i \leftarrow Y_i \oplus h_2(T_i)$
$H \leftarrow h_1(T_i) \oplus h_2(T_i)$
if $N_i \in \mathrm{Rng}(\pi_2)$ then
   $M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
   if $M_i \in \mathrm{Rng}(\pi_1)$ then
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$
   else
      if $X_i \oplus h_1(T_i) \in \mathrm{Dom}(\pi_1)$ then
         $\mathsf{bad}_2 \leftarrow \mathsf{true}$
      $\pi_1^{-1}(M_i) \leftarrow X_i \oplus h_1(T_i)$
else
   $V \xleftarrow{\$} \{w \in \mathbb{R} : 0 \leq w \leq 1\}$
   if $V < \delta(p_{\mathsf{lazy}}, p_{\mathsf{TBC}})$ then $\mathsf{bad}_3 \xleftarrow{\$} \mathsf{true}$
if $X_i \in S_2$ then
   $Z \leftarrow \pi_1(X_i \oplus h_1(T_i)) \oplus H$
else if $X_i \in S_1$
   $Z \xleftarrow{\$} \overline{\mathrm{Dom}(\pi_2)} \setminus (\mathrm{Rng}(\pi_1) \oplus H)$
   $\pi_1^{-1}(Z \oplus H) \leftarrow X_i \oplus h_1(T_i)$
$\pi_2^{-1}(N_i) \leftarrow Z$
$M_i \leftarrow \pi_2^{-1}(N_i) \oplus H$
return $X_i$
```

Listing 3.5: Game G_{10} gives the adversary control over Y_i values. Such an adversary can set bad flags at least as easily as adversaries for Game G_9 can. Additionally, adversaries for Game G_{10} are, without loss of generality, non-adaptive.

a construction secure against adversaries constrained to chosen-plaintext attacks.

One might also wish to try setting $K_2 = K_1$. While we know of no attacks here, modifying our proof to accommodate this change would be non-trivial. In particular, bounding β_1 required us to trace back through a game's execution history to determine when π_1 became defined at particular points; this task would be messier and more difficult to verify if $\pi_2 = \pi_1$. Still, this may merit future investigation.

4. Wideblock Tweakable Ciphers

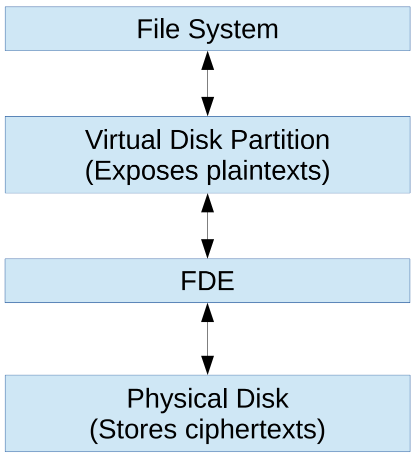

Figure 4.1: FDE works by transparently encrypting data before it is written to the physical disk.

We have examined TBC constructions that inherit the block length of some underlying blockcipher, typically 64 or 128 bits. In some contexts, however, we desire a TBC that operates on a much larger domain — e.g., 512 or 4096 bytes. In still other contexts, we desire tweakable ciphers which, as the reader may recall from Definition 3, support variable input lengths.

Section 4.1 motivates the study of these so-called "wideblock" TBCs by discussing the constraints that preclude traditional encryption in the context of full-disk encryption. Next, Section 4.2 discusses previous wideblock TBC constructions and their limitations. We then introduce the Protected Initialization Vector framework in Section 4.3. We refer to it as a framework because it contains two modular components which can be implemented using a variety of algorithms. Section 4.4 specifies two sets of algorithms to use for these components, yielding the Tweak-Counter-Tweak wideblock TBCs, TCT_1 and TCT_2. These two wideblock TBCs, which we published earlier [53], address some of the limitations of previous constructions. Finally, in Section 4.5 we refine TCT_1 to obtain VCV, and provide benchmarks for our implementation.

4.1 Motivation: full-disk encryption

Most major operating systems include support for full-disk encryption. Windows uses Microsoft's BitLocker, Mac OS uses FileVault (or FileVault 2), while Linux systems, including Android, can use dm-cyrpt. The above systems do not encrypt individual files; rather, they work at a lower level of abstraction and encrypt disk sectors. Working at this layer permits encryption to be file-system agnostic and prevents file-system metadata such as file sizes and directory structure from leaking.

Bitlocker uses *cipher block chaining* (CBC) encryption, a standard blockcipher-based algorithm. CBC, like most other standard encryption algorithms, takes an *initialization vector* (IV) as one of its inputs. (The purpose of an IV is to prevent the same message from being encrypted into the same ciphertext every time, and to prevent similar messages from resulting in similar ciphertexts.) IVs are typically generated at random and then stored or transmitted along with the ciphertext, but Bitlocker instead hashes the sector ID to obtain a fixed IV for that sector. FileVault uses AES-XTS [34], an encryption scheme based on a the XEX TBC that encrypts each 16-byte block using a tweak obtained by concatenating the sector ID with the current block's offset into the sector. Linux's dm-crypt supports both of these algorithms, among others.

These modes seem sufficient in "stolen-laptop" scenarios (provided the laptop isn't stolen while the drive is decrypted), but fail to defend against more sophisticated attacks. For example, an attacker who can see the encrypted disk image at different times can determine which 16-byte blocks have changed when AES-XTS is used, or determine the first modified 16-byte block of each sector when CBC is used. Even more worrisome is an attacker who tampers with the ciphertext to induce changes in the corresponding plaintext. For example, an attacker who knows what sector a CBC-encrypted system binary is saved to can, with some surmountable restrictions, transform the binary into malware by flipping carefully chosen bits of the ciphertext! The situation is less severe with AES-XTS, because an attacker can only corrupt 16-byte blocks of his choosing (or revert them to a previous state). A modified ciphertext block will result in a randomized plaintext block. There are times, however, where even this capability would be problematic; in a white paper [16], Microsoft engineers express concerns over such an attacker being able to toggle sensitive boolean registry settings.

Traditional cryptography would solve the problem of leaking what 16-byte blocks have changed by using a new IV each time a sector is re-encrypted. However, this remedy is unavailable here because it would require a place to store the IVs, forcing the file system to either use smaller logical disk sectors or to touch multiple physical sectors each time a logical sector is accessed. Performance constrains rule out both solutions. Similarly, a traditional solution to the tampering attacks would be to include message authentication codes — the cryptographic equivalent of checksums — in each disk sector. But again, this would require extra information to be stored on the disk, raising the same issues as before.

This is where wideblock tweakable ciphers come in. Given a tweakable cipher $\widetilde{E} : \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^s \to \{0,1\}^s$, we could encrypt an s-bit sector S by setting the ciphertext to be $C = \widetilde{E}_K(\mathsf{SectorID}, S)$. Note that $|S| = |C|$, so this operation can be done without changing the sector size or number of sectors. Furthermore, if $\widetilde{E}$ is a TPRP, then tampering with a ciphertext will essentially randomize the entire s-bit plaintext: adversaries are denied the ability to conduct the precision tampering that CBC affords them, and their ability to corrupt plaintexts is much more coarsegrained than with AES-XTS. Finally, although an adversary who sees the encrypted disk image at different points in time can determine what sectors have changed, he cannot determine where those changes occurred with any finer granularity.

We will discuss further applications of wideblock tweakable ciphers in Chapter 5. For now, we turn our attention to constructions.

4.2 Previous constructions

Researchers have developed three general approaches for constructing wideblock tweakable ciphers from n-bit blockciphers; examples are shown in Figure 4.2. Each approach has yielded a series of increasingly refined algorithms.

We contribute a new, top-down approach that leads us to the first beyond-birthday-bound secure tweakable cipher suitable for encrypting long inputs (i.e., longer than the block length of an underlying blockcipher). Table 4.1 and Figure 4.3 compare existing algorithms with our new Tweak-Counter-Tweak TCT_1 and TCT_2 constructions in terms of computational cost and security, respectively. TCT_1 is the first tweakable cipher to require only a single blockcipher invocation and no extra finite field multiplications for each additional n bits of input, while TCT_2 is the first to provide beyond-birthday-bound security (and still gets away with a fixed number of finite field multiplications).

Note that the finite field operations counted in Table 4.1 take hundreds of cycles in software [32, 2], whereas their cost relative to an AES blockcipher invocation is much lower in hardware [33]. In modern Intel chips, which include some hardware support for both AES and finite field multiplication, the relative cost of these operations depends on the specific architecture. On Ivy Bridge, AES has about twice the throughput of finite field multiplications, whereas on the newer Haswell chips finite field multiplication is slightly faster than AES [18].

	Cost				
Cipher	[BC]	$\mathbb{F}_{2^n}\times$	$[\mathbb{Z}_w+]$	$[\mathbb{Z}_{2w}]$	Ref.
HCTR	ℓ	$2\ell+2$	–	–	[55]
CMC	$2\ell+1$	–	–	–	[21]
EME	$2\ell+1$	–	–	–	[22]
EME*	$2\ell+3$	–	–	–	[19]
PEP	$\ell+5$	$4\ell-6$	–	–	[12]
HCH	$\ell+3$	$2\ell-2$	–	–	[11]
TET	ℓ	2ℓ	–	–	[20]
HEH	$\ell+1$	$\ell+2$	–	–	[50, 51]
TCT_1	$\ell+1$	5	$2\ell\left(\frac{n}{w}\right)^2$	$2\ell\left(\frac{n}{w}\right)^2$	–
TCT_2	$2\ell+8$	32	$4\ell\left(\frac{n}{w}\right)^2$	$4\ell\left(\frac{n}{w}\right)^2$	–

Table 4.1: Tweakable ciphers and their computational costs for ℓn-bit inputs. Costs measured in n-bit blockcipher calls [BC], finite field multiplications $\mathbb{F}_{2^n}\times$, and ring operations $[\mathbb{Z}_w+]$ and $[\mathbb{Z}_{2w}]$, for some word size w. Typically, $\ell = 32$ for FDE, and we anticipate $n = 128$, $w = 64$.

The first approach for constructing tweakable ciphers, "encrypt-mix-encrypt", is used by CMC [21], EME [22], and EME* [19], which employ two rounds of encryption separated by a light-weight "mixing layer". CMC is the first in this line of work, and can be used to encrypt strings whose lengths are integral multiples of n. EME improves on CMC by allowing encryption and decryption to be parallelized, and EME* extends the domain to include strings of arbitrary length.

Naor and Reingold [42] proposed the "hash-ECB-hash" approach, which sandwiches a layer of ECB-mode encryption between two invertible hashes. Informally, the role of the hashing layers is to diffuse the input. The PEP [12] mode of operation employs this approach. TET [20] and HEH [50] provide various improvements, notably in terms of performance. In each case, the two hashing layers require finite field multiplications. A variant of HEH described by Sarkar [51], however, manages to halve the number of multiplications that are required.

The final approach is "hash-CTR-hash". CTR refers to Counter Mode, a stan-

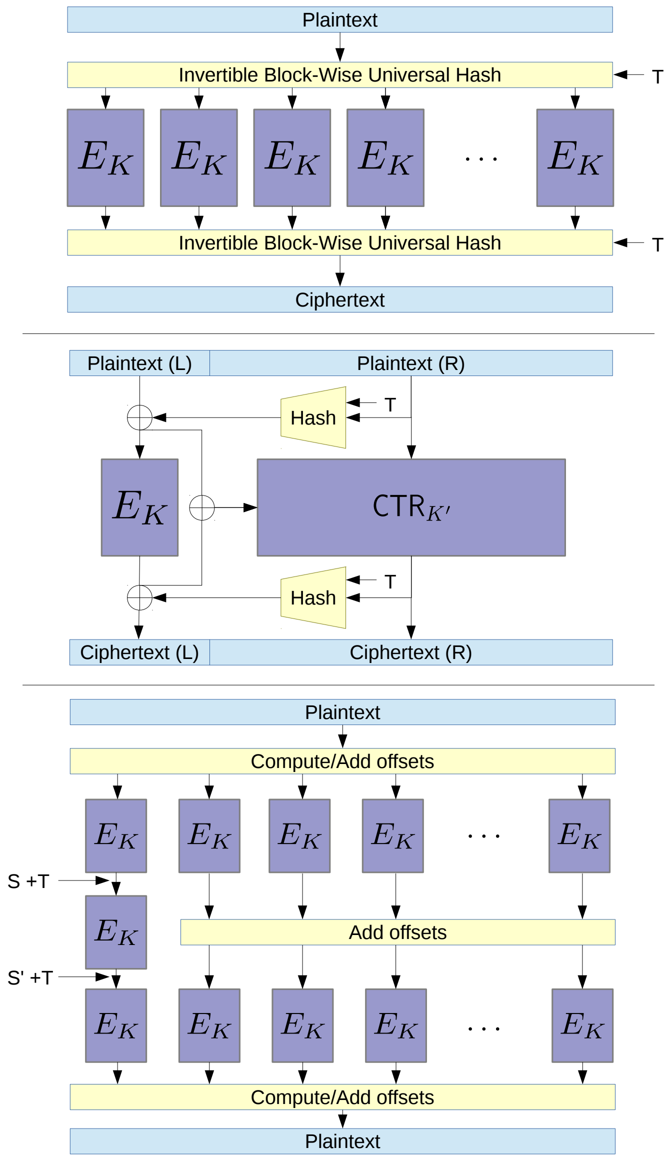

Figure 4.2: Three approaches for constructing wideblock tweakable ciphers. **Top:** Hash-ECB-Hash. **Middle:** Hash-CTR-Hash. **Bottom:** Encrypt-Mask-Encrypt. In these diagrams, E_K is a blockcipher with key K, and T is the tweak. Note that in all three cases, changing a single bit of the plaintext will affect the entire ciphertext (and vice versa).

dard encryption algorithm that in this context is sandwiched between two layers of hashing. The hashing layers are not invertible, but provide the mechanism by which the first output bits become dependent on every input bit. Examples include HCTR [55], which initially offered rather poor security bounds, and HCH, which provides birthday-bound security and requires only a single blockcipher key. Chakraborty and Nandi [10] later gave a birthday-bound-security proof for HCTR.

We mention the LargeBlock constructions due to Minematsu and Iwata [39], since they provide ciphers with beyond-birthday-bound security. These do not support tweaking, but it seems plausible that they could without significant degradation of performance or security. These constructions overcome the birthday bound by using $2n$-bit blockciphers as primitives, which are in turn constructed from an n-bit TBC. To our knowledge, $\mathsf{LRW2}$ is the most efficient n-bit TBC with beyond-birthday-bound security that supports the necessary tweakspace (Minematsu's TBC [38] limits tweak lengths to fewer than $n/2$ bits). Compared to TCT_2, instantiating the LargeBlock constructions with this primitive ultimately requires an extra six finite field multiplications for each n bits of input. Thus, we suspect the LargeBlock designs would be impractical even if adding tweak support proves feasible.

A construction due to Coron, et al. [14], which we refer to as CDMS (after the authors), builds a $2n$-bit TBC from an n-bit TBC, providing beyond-birthday-bound security in n. Like PIV, CDMS uses three rounds of a Feistel-like structure. However, our middle round uses a variable-input-length tweakable cipher, and we require a weaker security property from the round. This allows PIV to efficiently process long inputs. That said, CDMS provides an excellent way to implement a highly-secure $2n$-bit TBC, and we will use it for this purpose inside of TCT_2 to build $\widetilde{F}$. (Nesting CDMS constructions could create $(2^m n)$-bit tweakable blockciphers for any $m > 1$, but again, this would not be practical.) We note that Coron, et al. were primarily concerned with constructions indifferentiable from an ideal cipher, a goal quite different from ours.

The Thorp shuffle [41] and its successor, swap-or-not [24], are highly-secure ciphers targeting very small domains (e.g., $\{0,1\}^n$ for $n \leq 64$). Swap-or-not could almost certainly become a variable-input-length tweakable cipher, without changing the security bounds, by using domain separation for each input length and tweak in the underlying PRF. Essentially, one would make an input-length parameterized family of (tweakable) swap-or-not ciphers, with independent round-keys for each length. While still offering reasonable performance and unmatched security for very small inputs, the result would be wildly impractical for the large domains we are considering: swap-or-not's PRF needs to be invoked at least $6b$ times to securely encipher a b-bit input (below that, the bound becomes vacuous against even $q = 1$ query), and disk sectors are often 4096 bytes. Also, to match TCT_2's security, the PRF itself would need to be secure beyond the birthday bound (with respect to n).

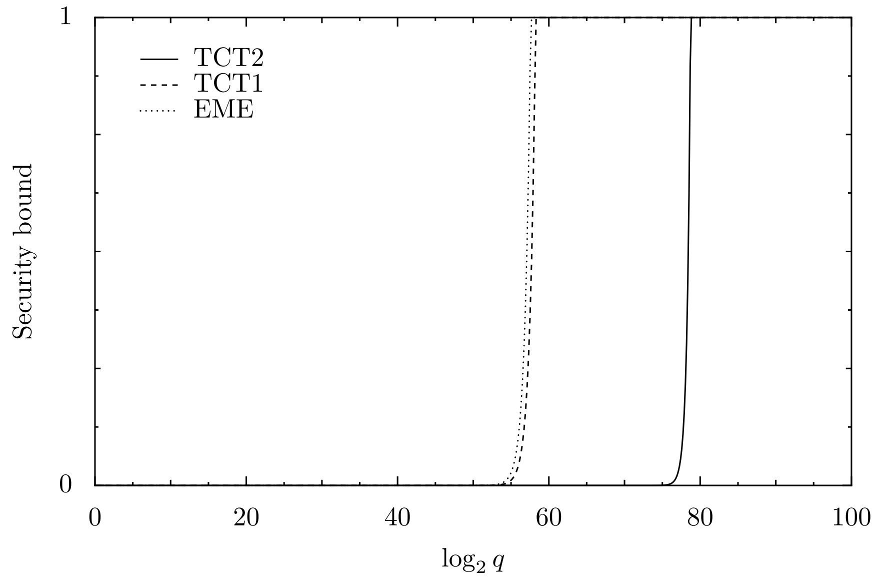

Figure 4.3: Security bounds for TCT_1, EME and TCT_2, all using an underlying 128-bit primitive and 4096-bit inputs, typical for FDE. The EME curve is representative of other prior construction.

4.3 The Protected IV framework

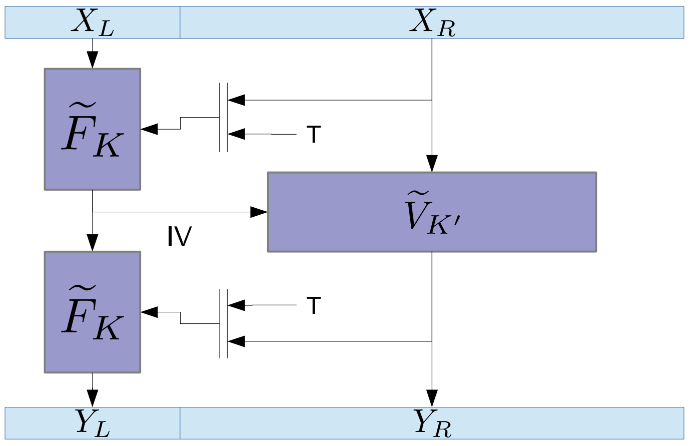

Figure 4.4: The $\mathsf{PIV}[\widetilde{F}, \widetilde{V}]$ tweakable cipher. Input T is the tweak, and $X = X_L X_R$ is a plaintext string of length at least N bits.

We begin by introducing our high-level abstraction, PIV, shown in Figure 4.4. Let $\mathcal{T} = \{0,1\}^\tau$ for some $\tau \geq 0$, and let $\mathcal{Y} \subseteq \{0,1\}^*$ be such that if $Y \in \mathcal{Y}$, then $\{0,1\}^{|Y|} \subseteq \mathcal{Y}$. Define $\mathcal{T}' = \mathcal{T} \times \mathcal{Y}$. Fix an integer $N > 0$. Let $\widetilde{F} \colon \mathcal{K}' \times \mathcal{T}' \times \{0,1\}^N \to \{0,1\}^N$ be a tweakable blockcipher and let $\widetilde{V} \colon \mathcal{K} \times \{0,1\}^N \times \mathcal{Y} \to \mathcal{Y}$ be a tweakable cipher. From these, we produce a new tweakable cipher $\mathsf{PIV}[\widetilde{F}, \widetilde{V}] \colon (\mathcal{K}' \times \mathcal{K}) \times \mathcal{T} \times \mathcal{X} \to \mathcal{X}$, where $\mathcal{X} = \{0,1\}^N \times \mathcal{Y}$. As shown in Figure 4.4, the PIV composition of $\widetilde{F}, \widetilde{V}$ is a three-round Feistel construction, working as follows. On input (T, X), let $X = X_L \,\|\, X_R$ where $|X_L| = N$. First, create an N-bit string $\mathsf{IV} = \widetilde{F}_{K'}(T \,\|\, X_R, X_L)$. Next, use this IV to encipher X_R, creating a string $Y_R = \widetilde{V}_K(\mathsf{IV}, X_R)$. Now create an N-bit string $Y_L = \widetilde{F}_{K'}(T \,\|\, Y_R, \mathsf{IV})$, and return $Y_L \,\|\, Y_R$ as the value of $\mathsf{PIV}[\widetilde{F}, \widetilde{V}]_{K',K}(T, X)$. The inverse $\mathsf{PIV}[\widetilde{F}, \widetilde{V}]^{-1}_{K',K}(T, Y)$ is computed in the obvious manner.

At first glance, it seems that nothing interesting has been accomplished: we took an N-bit TBC and a tweakable cipher, and produced a tweakable cipher with a slightly larger domain. The underlying tweakable cipher, however, only needs to have a very weak type of security property, which we now proceed to define.

The strong-RND (SRND) advantage of an adversary A against a TBC $\widetilde{E}$ is:

$$\mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\widetilde{E}}(A) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K} :\ A^{\widetilde{E}_K(\cdot,\cdot),\widetilde{E}_K^{-1}(\cdot,\cdot)} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\$(\cdot,\cdot),\$(\cdot,\cdot)} \Rightarrow 1 \,\right]$$

where the $\$(\cdot,\cdot)$ oracle always outputs a random string equal in length to its second input: $|\$(T,X)| = |X|$ for all T and X. Adversaries are *nonce-respecting* if the transcript of their oracle queries $(T_1,X_1),\ldots,(T_q,X_q)$ does not include $T_i = T_j$ for any $i \neq j$. Trivially, a TBC cannot be secure against general adversaries; A could, for example, query (T,X) to its first oracle to obtain Y, then query (T,Y) to its second oracle and compare the result to X. Hence SRND security is only meaningful if the TBC is used in some mode of operation that allows a reduction to a nonce-respecting (or similar) adversary. Such is the case with PIV.

Theorem 2. *Let sets $\mathcal{T},\mathcal{Y},\mathcal{T}',\mathcal{X}$ and integer N be as above. Let $\widetilde{F}\colon \mathcal{K}' \times \mathcal{T}' \times \{0,1\}^N \to \{0,1\}^N$ be a tweakable blockcipher, and let $\widetilde{V}\colon \mathcal{K} \times \{0,1\}^N \times \mathcal{Y} \to \mathcal{Y}$ be a tweakable cipher. Let $\mathsf{PIV}[\widetilde{F},\widetilde{V}]$ be as just described. Let A be an adversary making $q < 2^N/4$ queries totaling μ bits and running in time t. Then there exist adversaries B and C, making q and $2q$ queries, respectively, and both running in $O(t)$ time such that $\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathsf{PIV}[\widetilde{F},\widetilde{V}]}(A) \leq \mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\widetilde{V}}(B) + \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\widetilde{F}}(C) + \frac{4q^2}{2^N}$, where B is* nonce-respecting *with queries totalling $\mu - qN$ bits in length.*

The first thing to notice is that the variable-input-length portion of the PIV composition, $\widetilde{V}$, need be SRND-secure against *nonce-respecting* adversaries only. As we will see in the next section, it is easy to build efficient schemes meeting this requirement. Only the fixed-input-length portion, $\widetilde{F}$, needs to be secure against STPRP adversaries that can use arbitrary querying strategies. (Recall from pg. 8 that STPRP adversaries control the tweak, and have access to both encryption and decryption oracles.) Thus the PIV composition promotes nonce-respecting security over a large domain into full STPRP security over a slightly larger domain.

The intuition for why this should work is made clear by the picture. Namely, if $\widetilde{F}$ is a good STPRP, then if any part of T or X is "fresh", then the string IV should be random. Hence it is unlikely that an IV value is repeated, and so nonce-respecting security of the $\widetilde{V}$ component is enough. Likewise when deciphering, if any part of T, Y is "fresh".

The term $4q^2/2^N$ accounts for collisions in IV and the difference between $\widetilde{F}$ and a random function. This is a birthday-bound term in N, the block length of $\widetilde{F}$. Since most TBC designs employ (one or more) underlying blockciphers, we have deliberately chosen the notation N, rather than n, to stress that the block length of $\widetilde{F}$ can be larger than that of some underlying blockcipher upon which it might be built. Indeed, we'll see in the next section that, given an n-bit blockcipher (and a hash function), we can build $\widetilde{F}$ with $N = 2n$. This gives us hope of building beyond birthday-bound secure variable-input-length STPRPs in a modular fashion; we will do so, and with relatively efficient constructions, too.

It will come as no surprise that, if one does away with the lower $\widetilde{F}$ invocation and returns $\mathsf{IV} \parallel Y_R$, the resulting composition does not generically deliver a secure STPRP. On the other hand, it *is* secure as a TPRP (just not a *strong* TPRP). This can be seen through a straight-forward modification of the PIV security proof.

Proof. Fix a message space $\{0,1\}^S$ ($S \subseteq \mathbb{N}$), a tweakspace $\mathcal{T}$, and a non-negative integer $n \leq \min(S)$. Let A be an adversary making at most q queries and running in time t. Halevi and Rogaway [22] show that

$$\mathbf{Adv}^{\widetilde{\text{sprp}}}_{\mathsf{PIV}[\widetilde{F},\widetilde{V}]}(A) \leq \mathbf{Adv}^{\widetilde{\text{srnd}}}_{\mathsf{PIV}[\widetilde{F},\widetilde{V}]}(A) + \frac{q(q-1)}{2^{\min(S)+1}}.$$

This result is essentially a PRF–PRP switching lemma for TBCs, and reduces our problem to that of bounding $\mathbf{Adv}^{\widetilde{\text{srnd}}}_{\mathsf{PIV}[\widetilde{F},\widetilde{V}]}(A)$.

We begin in the information-theoretic setting, and consider $\mathcal{E}[\widetilde{V}] = \mathsf{PIV}[\Pi, \widetilde{V}]$, where $\Pi \xleftarrow{\$} \mathrm{BC}(N)$ is an ideal cipher. The oracles in Game 11 simulate $\mathcal{E}[\widetilde{V}]$ and $\mathcal{E}[\widetilde{V}]^{-1}$ using lazy sampling to define Π, so $\Pr\left[\, A^{\mathcal{E}[\widetilde{V}],\mathcal{E}[\widetilde{V}]^{-1}} \Rightarrow 1 \,\right] = \Pr\left[\, G_{11}(A) \Rightarrow 1 \,\right]$.

In Game 12, we no longer resample "illegal" values when defining Π. The only changes in the code occur after a boolean "bad" flag is set to true; by the Fundamental Lemma of Game-Playing,

$$\mathbf{Adv}^{\widetilde{\text{srnd}}}_{\mathcal{E}[\widetilde{V}]}(A) \le \left(\Pr\left[\, G_{12}(A) \Rightarrow 1 \,\right] - \Pr\left[\, A^{\$(\cdot,\cdot),\$(\cdot,\cdot)} \Rightarrow 1 \,\right]\right) \\ + \Pr\left[\, G_{12}(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \vee \mathsf{bad}_3 \,\right]$$

Note that in Game G_{12}, $\widetilde{V}$ is never queried using the same tweak twice. Hence we may consider a third game (not shown), Game G_{13}, in which $\widetilde{V}$ is replaced by an oracle $\$(\cdot,\cdot)$ that always returns a random string equal in length to its second input. By a reduction standard argument, there exists some nonce-respecting adversary B making q queries and running in $\mathcal{O}(t)$ time such that

$$\Pr\left[\, G_{12}(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_{13}(A) \Rightarrow 1 \,\right] \le \mathbf{Adv}^{\widetilde{\text{srnd}}}_{\widetilde{V}}(B).$$

We now have

$$\mathbf{Adv}^{\widetilde{\text{srnd}}}_{\mathcal{E}[\widetilde{V}]}(A) \le \left(\Pr\left[\, G_{13}(A) \Rightarrow 1 \,\right] - \Pr\left[\, A^{\$(\cdot,\cdot),\$(\cdot,\cdot)} \Rightarrow 1 \,\right]\right) \\ + \Pr\left[\, G_{12}(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \vee \mathsf{bad}_3 \,\right] + \mathbf{Adv}^{\widetilde{\text{srnd}}}_{\widetilde{V}}(B).$$

However, note that now each the first N bits of each oracle output (corresponding to Z_i') are always uniformly random in Game G_{12}, and when we switch from $\widetilde{V}$ to $\$(\cdot,\cdot)$ in the next game, the remaining bits also become uniformly random. Hence $\Pr\left[\, G_{13}(A) \Rightarrow 1 \,\right] = \Pr\left[\, A^{\$(\cdot,\cdot),\$(\cdot,\cdot)} \Rightarrow 1 \,\right]$.

Our final task is to bound the probability that A sets a bad flag in Game G_{12}. The probability that bad_1 is set during query j is less than $j/(2^N - 2j)$. Similarly, the probabilities of bad_2 and bad_3 being set are at most $2j/(2^N - 2j)$ and $2j/2^N$, respectively. Therefore the probability that at least one flag is set during query j is at most $3j/(2^N - 2j) + 2j/2^N$.

Taking the union bound over $j \in 1, 2, \ldots, q$ gives us

$$\Pr\left[\, G_{11}(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \vee \mathsf{bad}_3 \,\right] \le q^2 \left(\frac{1.5}{2^N - 2q} + \frac{1}{2^N} \right).$$

Since $q < 2^N/4$, $1.5/(2^N - 2q) < 3/2^N$. Collecting our previous results and using a standard reduction argument to return to the computational setting completes the proof:

$$\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathsf{PIV}[\widetilde{F},\widetilde{V}]}(A) \le \mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\widetilde{V}}(B) + \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\widetilde{F}}(C) + \frac{4q^2}{2^N},$$

where C makes $2q$ queries, B makes q queries of total length $\mu - qN$ bits without repeating a tweak, and both run in $\mathcal{O}(t)$ time. $\square$

4.4 Concrete Instantiations of PIV

Instantiating a PIV composition requires two objects, a (fixed-input-length) tweakable blockcipher $\widetilde{F}$ with an N-bit block length, and a variable-input-length tweakable cipher $\widetilde{V}$. In this section we explore various ways to instantiate these two objects, under the guidance of Theorem 2 and practical concerns.

Theorem 2 suggests setting N to be as large as possible, so that the final term is

Games $\boxed{G_1 1}$, $G_1 2$

```
Oracle $\mathcal{E}_\ell(T, X)$:
  $j \leftarrow j + 1$
  $T_j \leftarrow T \parallel X[N+1..]$
  $IV_j \xleftarrow{\$} \{0,1\}^N \setminus [(..\Pi]\,[T_j])$
  $\Pi[T_j](X[1..N]) \leftarrow IV_j$
  $IV \leftarrow IV_j$
  if $IV_j \in \{IV_i \,:\, i < j\}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $IV_j \xleftarrow{\$} \{0,1\}^N \setminus \{IV_i \,:\, i < j\}$
    $\boxed{IV_j \leftarrow IV}$ // Rollback to "real" value
  $Z_i \leftarrow \widetilde{V}[IV_j](X[N+1..])$
  $T'_j \leftarrow T \parallel Z_i$
  $Z'_i \xleftarrow{\$} \{0,1\}^N$
  if $IV_j \in \mathsf{dom}(\Pi[T'_j])$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\boxed{Z'_i \leftarrow \Pi[T'_j](IV_j)}$
  else if $Z'_i \in [(..\Pi]\,[T'_j])$
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
    $\boxed{Z'_i \xleftarrow{\$} \{0,1\}^N \setminus [(..\Pi]\,[T'_j])}$
  $\Pi[T'_j](IV_j) \leftarrow Z'_i$
  return $Z'_i \parallel Z_i$
```

```
Oracle $\mathcal{E}_\ell^{-1}(T, Y)$:
  $j \leftarrow j + 1$
  $T_j \leftarrow T \parallel Y[N+1..]$
  $IV_j \xleftarrow{\$} \{0,1\}^N \setminus \mathsf{dom}(\Pi[T_j])$
  $\Pi[T_j]^{-1}(Y[1..N]) \leftarrow IV_j$
  $IV \leftarrow IV_j$
  if $IV_j \in \{IV_i \,:\, i < j\}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $IV_j \xleftarrow{\$} \{0,1\}^N \setminus \{IV_i \,:\, i < j\}$
    $\boxed{IV_j \leftarrow IV}$
  $Z_i \leftarrow \widetilde{V}[IV_j]^{-1}(Y[N+1..])$
  $T'_j \leftarrow T \parallel Z_i$
  $Z'_i \xleftarrow{\$} \{0,1\}^N$
  if $IV_j \in [(..\Pi]\,[T'_j])$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\boxed{Z'_i \leftarrow \Pi[T'_j]^{-1}(IV_j)}$
  else if $Z'_i \in \mathsf{dom}(\Pi[T'_j])$
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
    $\boxed{Z'_i \xleftarrow{\$} \{0,1\}^N \setminus \mathsf{dom}(\Pi[T'_j])}$
  $\Pi[T'_j](Z'_i) \leftarrow IV_j$
  return $Z'_i \parallel Z_i$
```

Listing 4.1: Game G_{11}, which includes the boxed statements, simulates $\mathsf{PIV}[\Pi, \widetilde{V}]$ by defining Π through lazy sampling. Game G_{12}, which does *not* include the boxed statements, never invokes $\widetilde{V}$ with the same tweak twice, and the oracles in this game always return values with a random n-bit prefix. All boolean variables are silently initialized to false.

vanishingly small for any realistic number of queries. But for this to be useful, one must already know how to build a TBC $\widetilde{F}$ with domain $\{0,1\}^N$ for a large N, and for which $\mathbf{Adv}_{\widetilde{F}}^{\widetilde{\mathrm{sprp}}}(C)$ approaches $q^2/2^N$. To our knowledge, there are no efficient constructions that permit $\mathbf{Adv}_{\widetilde{F}}^{\widetilde{\mathrm{sprp}}}(C)$ to be smaller than $\mathcal{O}(q^3/2^{2n})$ when using an n-bit blockcipher as a starting point. (A recent result by Lampe and Seurin [28] shows how to beat this security bound, but at a substantial performance cost.) A construction by Coron, et al., which will be discussed in more detail shortly, does meet this bound[1] while providing $N = 2n$.

So we restrict our attention to building TBC $\widetilde{F}$ with small N. In particular, we follow the common approach of building TBCs out of blockciphers. Letting n be the blockcipher block length, we will consider $N = n$, and $N = 2n$. In the former case, Theorem 2 only promises us security up to roughly $q = 2^{n/2}$, which is the birthday bound with respect to the blockcipher. With this security bound in mind, we can use simple and efficient constructions of both $\widetilde{F}$ and $\widetilde{V}$. On the other hand, when $N = 2n$, Theorem 2 lets us hope for security to roughly $q = 2^n$ queries. To realize this hope we will need a bit more from both $\widetilde{F}$ and $\widetilde{V}$, but we will still find reasonably efficient constructions delivering beyond birthday bound security.

In what follows, we will sometimes refer to objects constructed in other works. These are summarized for convenience in Table 4.2 on pg. 54.

An efficient variable-input-length tweakable cipher. We will start by considering general methods for constructing $\widetilde{V}$. Recall that $\widetilde{V}$ need only be secure against adversaries that never repeat a tweak. In Listing 4.2, we see an analogue of conventional Counter Mode (CTR) encryption, but over an n-bit TBC $\widetilde{E}$ instead of a blockcipher. We call the result tweaked-Counter Mode (TCTR). Within a call (T, X)

[1]However, nesting this construction to provide a variable-input-length tweakable cipher is pro-

Name	Description	Ref
LRW	Birthday-bound TBC. Requires blockcipher E and ϵ-AXU function H. $$\mathsf{LRW}[H,E]_{(K,L)}(T,X) = E_K(X \oplus H_L(T)) \oplus H_L(T)$$	[30]
$\mathsf{PolyHash}^{mn}$	ϵ-AXU function with domain $(\{0,1\}^n)^m$ and $\epsilon = m/2^n$. $$\mathsf{PolyHash}^{mn}_L(T_1T_2\cdots T_m) = \bigoplus_{i=1}^{m} T_i \otimes L^i,$$ all operations in $\mathbb{F}_{2^n}$	[56]
$\mathsf{NH}[\nu w, 2tw]$	ϵ-AU hash function. Fix word size $w > 0$. Requires ν even, inputs are νw bits; here $\epsilon = 1/2^{tw}$. Define: $$H_{K_1 \,\Vert\, \cdots \,\Vert\, K_\nu}(X_1\cdots X_\nu) = \sum_{i=1}^{\nu/2}(K_{2i-1} +_w X_{2i-1}) \cdot (K_{2i} +_w X_{2i}) \bmod 2^{2w}.$$ $$\mathsf{NH}[\nu,t]_{K_1 \,\Vert\, \cdots \,\Vert\, K_{\nu+2(t-1)}}(M) = H_{K_1\cdots K_\nu}(M) \Vert \cdots \Vert H_{K_{2t-1}\cdots K_{\nu+2t-2}}(M)$$	[6]
LRW2	TBC with beyond-birthday-bound security. Requires blockcipher E and ϵ-AXU function H. $$\mathsf{LRW2}[H,E]_{(K_1,K_2)}(T,X) = \mathsf{LRW}[H,E]_{K_1}(T, \mathsf{LRW}[H,E]_{K_2}(T,X))$$	[29]
CDMS	Feistel-like domain extender for TBC $\widetilde{E}$. $$\mathsf{CDMS}[\widetilde{E}]_K(T, L \,\Vert\, R) = \widetilde{E}_K(10 \,\Vert\, T \,\Vert\, R', L') \,\Vert\, R'$$ where $R' = \widetilde{E}_K(01 \,\Vert\, T \,\Vert\, L', R)$ and $L' = \widetilde{E}_K(00 \,\Vert\, T \,\Vert\, R, L)$.	[14]

Table 4.2: Our PIV implementations, TCT_1 and TCT_2, use the above constructions from prior works.

```
procedure TCTR[$\widetilde{E}$]$_K(T,X)$:
$X_1, X_2, \ldots, X_\nu \xleftarrow{n} X$
for $i = 1$ to $\nu$
  $T_i \leftarrow g(T,i)$; $Z_i \leftarrow \langle i \rangle$
  $Y_i \leftarrow \widetilde{E}_K(T_i, Z_i) \oplus X_i$
Return $Y_1, Y_2, \ldots, Y_\nu$

procedure TCTR[$\widetilde{E}$]$_K^{-1}(T,Y)$:
$Y_1, Y_2 \ldots, Y_\nu \xleftarrow{n} Y$
for $i = 1$ to $\nu$
  $T_i \leftarrow g(T,i)$; $Z_i \leftarrow \langle i \rangle$
  $X_i \leftarrow Y_i \oplus \widetilde{E}_K(T_i, Z_i)$
Return $X_1, \ldots, X_\nu$
```

Listing 4.2: The TCTR tweakable cipher.

to TCTR, each n-bit block X_i of the input X is processed using a per-block tweak T_i, this being determined by a function $g\colon \mathcal{T}' \times \mathbb{N} \to \mathcal{T}$ of the input tweak T and the block index i.

Consider the behavior of TCTR when $g(T,i) = T$. The following result is easily obtained using standard techniques.

Theorem 3. *Let* $\widetilde{E}\colon \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ *be a tweakable blockcipher, and let* $\mathrm{TCTR}[\widetilde{E}]_K$ *and* $\mathrm{TCTR}[\widetilde{E}]_K^{-1}$ *be defined as above, with* $g(T,i) = T \in \mathcal{T}$*. Let A be a nonce-respecting adversary that runs in time t, and asks q queries, each of length at most ℓn bits (so, $\mu \le q\ell n$). Then for some adversary B making at most $q\ell$ queries and running in time $\mathcal{O}(t)$,* $\mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\mathrm{TCTR}[\widetilde{E}]}(A) \le \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(B) + 0.5q\ell^2/2^n$.

We note that the bound displays birthday-type behavior when $\ell = o(\sqrt{q})$, and is tightest when ℓ is a small constant. An important application with small, constant ℓ is full-disk encryption. Here plaintexts X would typically be 4096 bytes long, so if the underlying TBC has block length $n = 128$, we get $\ell = 256$ blocks.[2]

Extending tweakspaces. In PIV, the TBC $\widetilde{F}$ will need to handle long tweaks. Fortunately, a result by Coron, et al. [14] shows that one can compress tweaks using

hibitively inefficient.

[2]Actually, slightly less than this when used in the PIV composition, since the first N bits are enciphered by $\widetilde{F}$.

an ϵ-AU hash function at the cost of adding a $q^2\epsilon$ term to the tweakable cipher's TPRP security bound. In particular, we will use (a slight specialization of) the NH hash, defined by Black, et al. [6]; $\mathsf{NH}[r,s]_L$ takes r-bit keys ($|L| = r$), maps r-bit strings to s-bit strings, and is $2^{s/2}$-AU; see Table 4.2 (pg. 54) for the description. Given a TBC $\widetilde{E}$, $\widetilde{E}^{\mathsf{NH}}$ denotes the resulting TBC, whose tweakspace is now the domain of NH, rather than its range.

4.4.1 Targeting efficiency at birthday-type security: TCT_1

Let us begin with the case of $N = n$.

We will use LRW for the fixed-input length portion. We implement LRW's ϵ-AXU hash function with $\mathsf{PolyHash}$, and then extend the tweak space using a fast ϵ-AU hash function[3] as described above.

The TCT_1 Construction. Fix $k, n > 0$, and let $N = n$. Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher, and let $\mathsf{PolyHash}^{mn}$, and NH be as defined in Table 4.2. Then define $\mathsf{TCT}_1 = \mathsf{PIV}[\widetilde{F}, \widetilde{V}]$, where to obtain a τn-bit tweakspace and domain $\{0,1\}^{\{n,n+1,\ldots,\ell n\}}$ we set:

1. n-bit TBC $\widetilde{F} = \mathsf{LRW}[\mathsf{PolyHash}^{2n}, E]^{\mathsf{NH}[(\ell+\tau)n,2n]}$, i.e. LRW with its tweakspace extended using NH. The keyspace for $\widetilde{F}$ is $\{0,1\}^k \times \{0,1\}^{2n} \times \{0,1\}^{(\ell+\tau)n}$, with key K' partitioning into keys for E, $\mathsf{PolyHash}^{2n}$, and $\mathsf{NH}[(\ell+\tau)n, 2n]$. (Since NH supports only fixed length inputs, we implicitly pad NH inputs with a 1 and then as many 0s as are required to reach a total length of $(\ell+\tau)n$ bits.) The tweakspace for $\widetilde{F}$ is $\{0,1\}^{\{0,1,2,\ldots,(\ell+\tau-1)n\}}$.

[3]Indeed, one can show that composing an ϵ-AU hash function with an ϵ'-AXU hash function yields an $(\epsilon+\epsilon')$-AXU hash function; however, we find it convenient to work on a higher level of abstraction.

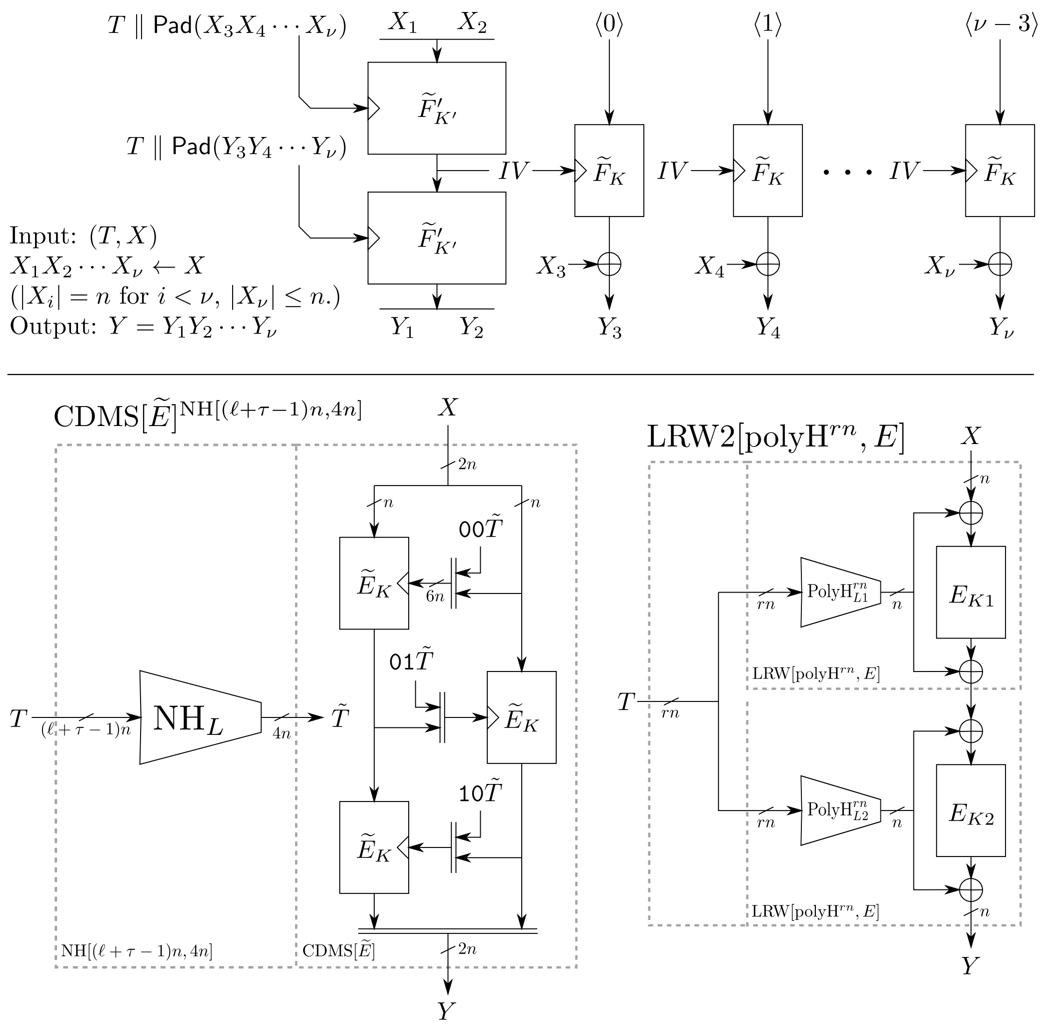
$T \parallel \mathsf{Pad}(X_3X_4\cdots X_\nu)$
X_1 X_2
$\widetilde{F}'_{K'}$
$T \parallel \mathsf{Pad}(Y_3Y_4\cdots Y_\nu)$
$\widetilde{F}'_{K'}$
Input: (T, X)
$X_1X_2\cdots X_\nu \leftarrow X$
$(|X_i| = n$ for $i < \nu$, $|X_\nu| \le n.)$
Output: $Y = Y_1Y_2\cdots Y_\nu$
Y_1 Y_2
$\langle 0\rangle$ $\langle 1\rangle$ $\langle \nu-3\rangle$
IV $\widetilde{F}_K$ IV $\widetilde{F}_K$ $\cdots$ IV $\widetilde{F}_K$
X_3 X_4 X_ν
Y_3 Y_4 Y_ν
$\mathrm{CDMS}[\widetilde{E}]^{\mathrm{NH}[(\ell+\tau-1)n,4n]}$
X
$2n$
n n
$\mathtt{00}\tilde{T}$
$\widetilde{E}_K$
$6n$
$\mathtt{01}\tilde{T}$
T $(\ell+\tau-1)n$ NH_L $4n$ $\tilde{T}$
$\widetilde{E}_K$
$\mathtt{10}\tilde{T}$
$\widetilde{E}_K$
$\mathrm{NH}[(\ell+\tau-1)n, 4n]$
$\mathrm{CDMS}[\widetilde{E}]$
$2n$
Y
$\mathrm{LRW2}[\mathrm{polyH}^{rn}, E]$
X
n
T rn
rn PolyH^{rn}_{L1} n E_{K1}
$\mathrm{LRW}[\mathrm{polyH}^{rn}, E]$
rn PolyH^{rn}_{L2} n E_{K2}
$\mathrm{LRW}[\mathrm{polyH}^{rn}, E]$
n
Y

Figure 4.5: The TCT_2 construction (top). TCT_2 takes τn-bit tweaks, and the input length is between $2n$ and ℓn bits, inclusive. Here, $\widetilde{F}$ is implemented using the $2n$-bit CDMS construction coupled with the NH hash function (bottom left). Both $\widetilde{V}$ and the TBC $\widetilde{E}$ used inside of CDMS are implemented using $\mathsf{LRW2}[\mathsf{PolyHash}^{rn}, E]$ (bottom right), with $r = 6$ and $r = 2$, respectively. The function Pad maps s to $s \,\|\, 10^{(\ell+1)n-1-|s|}$. In the diagram for CDMS, the strings $00\widetilde{T}$, $01\widetilde{T}$, and $10\widetilde{T}$ are padded with 0s to length $5n$ before being used.

2. Variable-input-length tweakable cipher $\widetilde{V} = \mathrm{TCTR}\left[\mathsf{LRW}[\mathsf{PolyHash}^n, E]\right]$ with the TCTR function $g\colon \{0,1\}^n \times \mathbb{N} \to \{0,1\}^n$ as $g(T,i) = T$. The keyspace for $\widetilde{V}$ is $\{0,1\}^k \times \{0,1\}^n$, with key K partitioning into keys for E and $\mathsf{PolyHash}^n$. The tweakspace for $\widetilde{V}$ is $\{0,1\}^n$, and its domain is $\{0,1\}^{\{0,1,\ldots,(\ell-1)n\}}$.

Putting together Theorems 2,3, and results from previous works [6, 30], we have the following security bound.

Theorem 4 (STPRP-security of TCT_1)**.** *Define TCT_1 as above, and let A be an adversary making $q < 2^n/4$ queries and running in time t. Then there exist adversaries B and C, both running in time $\mathcal{O}(t)$ and making $(\ell-1)q$ and $2q$ queries, respectively, such that $\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathsf{TCT}_1[E]}(A) \le \mathbf{Adv}^{\mathrm{prp}}_{E}(B) + \mathbf{Adv}^{\mathrm{sprp}}_{E}(C) + \frac{32q^2}{2^n} + \frac{4q^2(\ell-1)^2}{2^n}$.*

Proof. Using Theorem 2 and security bounds from the respective works cited in Table 4.2,

$$\begin{aligned}
\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathsf{TCT}_1[E]}(A) &\le \frac{4q^2}{2^n} + \mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\widetilde{V}}(t', q) + \mathbf{Adv}^{\mathrm{sprp}}_{\widetilde{F}}(t', 2q) \\
&\le \frac{4q^2}{2^n} + \left[\frac{q(\ell-1)^2}{2^n} + \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\mathsf{LRW}[\mathsf{PolyHash}^n, E]}(t', (\ell-1)q)\right] \\
&\quad + \left[\frac{24q^2}{2^n} + \frac{4q^2}{2^n} + \mathbf{Adv}^{\mathrm{sprp}}_{E}(t', 2q)\right] \\
&\le \frac{4q^2}{2^n} + \left[\frac{q(\ell-1)^2}{2^n} + \frac{3q^2(\ell-1)^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_{E}((\ell-1)q, t')\right] \\
&\quad + \left[\frac{28q^2}{2^n} + \mathbf{Adv}^{\mathrm{sprp}}_{E}(t', 2q)\right] \\
&\le \frac{32q^2}{2^n} + + \frac{q(\ell-1)^2}{2^n} + \frac{3q^2(\ell-1)^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_{E}((\ell-1)q, t') \\
&\quad + \mathbf{Adv}^{\mathrm{sprp}}_{E}(t', 2q).
\end{aligned}$$

□

This algorithm requires $2k + (3 + \tau + \ell)n$ bits of key material, including two keys for $\widetilde{E}$. As we show at the end of this section, we can get away with a single key for E with no significant damage to our security bound, although this improvement is motivated primarily by performance concerns.

Thus TCT_1 retains the security of previous constructions (see Figure 4.3 for a visual comparison), and uses arithmetic in rings with powers-of-two moduli, rather than in a finite field. This may potentially improve performance on some architectures.

4.4.2 Aiming for beyond birthday-bound security: TCT_2

Now let us consider the PIV composition with $N = 2n$. For the fixed-input-length component, we can use Coron et al.'s [14] CDMS construction to get a $2n$-bit TBC from an n-bit TBC, and implement the latter using $\mathsf{LRW2}$. Table 4.2 describes both constructions.[4] We again extend the tweakspace using NH. (To stay above the birthday bound, we set the range of NH to $\{0,1\}^{2n}$). Ultimately, setting $\widetilde{F} = \mathsf{CDMS}[\mathsf{LRW2}]^{\mathsf{NH}}$ is secure against up to around $2^{2n/3}$ queries.

$\mathsf{LRW2}$ also gives us a way to realize a beyond birthday-bound secure variable-input-length component, namely $\widetilde{V} = \text{TCTR}[\mathsf{LRW2}[E, H]$, at least for $\ell = o(q^{1/4})$. (We'll see how to avoid this restriction, if desired, in a moment.)

We are now ready to give our second fully concrete PIV composition, TCT_2, targeted at applications that would benefit from beyond birthday-bound security. This algorithm requires us to nest four layers of other constructions, so we provide an illustration in Figure 4.5 (pg. 57). Again we emphasize that the (admittedly significant) cost of $\widetilde{F}$ can be amortized.

TCT_2 supports τn-bit tweaks and has domain $\{0,1\}^{\{2n, 2n+1, \ldots, \ell n\}}$.

[4]We note that for $\mathsf{CDMS}[\widetilde{E}]$, we enforce domain separation via $\widetilde{E}$'s tweak, whereas the authors of [14] use multiple keys for $\widetilde{E}$. The proof of our construction follows easily from that of the original.

The TCT_2 Construction. Fix $k, \ell, n, \tau > 0$, and let $N = 2n$. Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher, and let $\mathsf{PolyHash}^{\ell n}$, and NH be as defined in Table 4.2. Then define $\mathsf{TCT}_2 = \mathsf{PIV}[\widetilde{F}, \widetilde{V}]$, where:

1. $\widetilde{F} = \mathsf{CDMS}\left[\mathsf{LRW2}[\mathsf{PolyHash}^{6n}, E]\right]^{\mathsf{NH}[(\ell+\tau-1)n, 4n]}$. The keyspace for $\widetilde{F}$ is $\{0,1\}^{2k} \times \{0,1\}^{12n} \times \{0,1\}^{(\ell+\tau-1)n}$, with key K' partitioning into two keys for E, two keys for $\mathsf{PolyHash}^{6n}$, and a key for $\mathsf{NH}[\ell n, 4n]$. The tweakspace for $\widetilde{F}$ is $\{0,1\}^{\tau n}$.

2. $\widetilde{V} = \mathrm{TCTR}\left[\mathsf{LRW2}[\mathsf{PolyHash}^{2n}, E]\right]$, with the TCTR function $g\colon \{0,1\}^n \times \mathbb{N} \to \{0,1\}^n$ as $g(T, i) = T$. The keyspace for $\widetilde{V}$ is $\{0,1\}^{2k} \times \{0,1\}^{4n}$ with key K partitioning into two keys for E and two keys for $\mathsf{PolyHash}^{2n}$. The tweakspace for $\widetilde{V}$ is $\{0,1\}^{2n}$, and its domain is $\{0,1\}^{\{0,1,2,\ldots,(\ell-2)n\}}$.

TCT_2 requires $4k + (\ell + \tau + 15)n$ bits of key material. We have the following security result.

Theorem 5 (STPRP-security of TCT_2)**.** *Define TCT_2 as above, and let A be an adversary making q queries and running in time t, where $6q, \ell q < 2^{2n}/4$. Then there exist adversaries B and C, both running in $\mathcal{O}(t)$ time and making $(\ell-1)q$ and $6q$ queries, respectively, such that* $\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathsf{TCT}_2}(A) \leq 2\mathbf{Adv}^{\mathrm{prp}}_{E}(B) + 2\mathbf{Adv}^{\mathrm{sprp}}_{E}(C) + \frac{12q^2}{2^{2n}} + \frac{q(\ell-1)^2}{2^n} + \frac{6\ell^3 q^3}{2^{2n-2} - \ell^3 q^3} + \frac{6^4 q^3}{2^{2n-2} - 6^3 q^3}$.

Proof. Using Theorem 2 and security bounds for the various components [6, 14, 29],

$$\begin{aligned}
\mathbf{Adv}^{\widetilde{\text{sprp}}}_{\mathsf{TCT}_2}(A) &\leq \frac{4q^2}{2^{2n}} + \mathbf{Adv}^{\widetilde{\text{srnd}}}_{\widetilde{V}}(t', q) + \mathbf{Adv}^{\text{sprp}}_{\widetilde{F}}(t', 2q) \\
&\leq \frac{4q^2}{2^{2n}} + \left[\frac{q(\ell-1)^2}{2^n} + \mathbf{Adv}^{\widetilde{\text{prp}}}_{\mathsf{LRW2}[H^{2n},E]}(t', (\ell-1)q,)\right] \\
&\quad + \left[\frac{4q^2}{2^{2n}} + \frac{4q^2}{2^{2n}} + \mathbf{Adv}^{\widetilde{\text{sprp}}}_{\mathsf{LRW2}[H^{6n},E]}(t', 6q,)\right] \\
&\leq \frac{12q^2}{2^{2n}} + \frac{q(\ell-1)^2}{2^n} + \frac{6\ell^3q^3}{2^{2n-2} - \ell^3q^3} + \frac{6^4q^3}{2^{2n-2} - 6^3q^3} \\
&\quad + 2\mathbf{Adv}^{\text{prp}}_{E}(t', (\ell-1)q) + 2\mathbf{Adv}^{\text{sprp}}_{E}(t', 6q).
\end{aligned}$$

□

Some of the constants in this bound are rather significant. However, as Figure 4.3 shows, TCT_2 nevertheless provides substantially better security bounds than TCT_1 and previous constructions.

4.4.3 Additional practical considerations

Several variations and optimizations on TCT_1 and TCT_2 are possible. We highlight a few of them here. None of these changes significantly impact the above security bounds, unless otherwise noted.

Reducing the number of blockcipher keys. In the case of TCT_1, we can use a single key for both LRW instances provided we enforce domain separation through the tweak. This allows us to use a single key for the underlying blockcipher, which, in some situations, may allow for significant implementation benefits (for example, by allowing a single AES pipeline). One method that accomplishes this is to replace $\mathsf{LRW}[\mathsf{PolyHash}^{2n}, E]^{\mathsf{NH}[(\ell+1)n,2n]}$ with $\mathsf{LRW}[\mathsf{PolyHash}^{3n}, E]^{f(\varepsilon,\cdot)}$ and $\mathsf{LRW}[\mathsf{PolyHash}^{n}, E]$ with $\mathsf{LRW}[\mathsf{PolyHash}^{3n}, E]^{f(\cdot,\varepsilon)}$. Here, f is a 2^{-n}-AU function with keyspace $\{0,1\}^{3n} \times$

$\{0,1\}^{\ell n}$, taking inputs of the form (X, ε) (for some $X \in \{0,1\}^n$) or (ε, Y) (for some $Y \in \{0,1\}^{\{0,1,\ldots,\ell n\}}$), and outputting a $3n$-bit string. Let $f_L(X, \varepsilon) = 0^{2n} \parallel X$ and $f_L(\varepsilon, Y) = 1^n \parallel \mathsf{NH}[(\ell+1)n, 2n]_L(Y)$. The function f described here is a mathematical convenience to unify the signatures of the two LRW instances, thereby bringing tweak-based domain separation into scope; in practice, we imagine the two instances would be implemented independently, save for a shared blockcipher key. We note that TCT_2 can be modified in a similar manner to require only two blockcipher keys.

Performance optimizations. If we need only a tweakable blockcipher (i.e., only need to handle inputs of a predetermined length), we can use $\mathsf{NH}[\ell n, 2n]$ in place of $\mathsf{NH}[(\ell+1)n, 2n]$ by adjusting our padding scheme appropriately. This change reduces the length of the tweak, and hence the number of operations performed during the NH computations. We emphasize that in the TCTR portion, the $\mathsf{PolyHash}$ functions only need to be computed once, since each LRW invocation uses the same tweak. The corresponding optimizations apply to TCT_2, as well.

A naïve implementation of TCT_2 would make a total 72 finite field multiplications during the first and third rounds (a result of evaluating $\mathsf{PolyHash}^{6n}$ twelve times). We can cache an intermediate value of the $\mathsf{PolyHash}^{6n}$ hash used inside of CDMS (four n-bit tweak blocks are constant per invocation), and this saves 32 finite field multiplications. Precomputing the terms of the polynomial hash corresponding to the domain-separation constants eliminates 12 more multiplications, leaving 28 in total. Four more are required during the $\widetilde{V}$ phase, giving the count of 32 reported in Table 4.1.

Handling large message spaces. Both TCT_1 and TCT_2 are designed with FDE applications in mind. In particular, they require ℓ to be fixed ahead of time, and require more than ℓn bits of key material.

These limitations are a consequence of using the NH hash function; however, a simple extension to NH (described by the original authors [6]) accommodates arbitrarily long strings. Fix a positive integer r and define $\mathsf{NH}^*_L(M_1 M_2 \cdots M_\nu) = \mathsf{NH}_L(M_1) \parallel \mathsf{NH}_L(M_2) \parallel \cdots \parallel \mathsf{NH}_L(M_\nu) \parallel \langle |M| \bmod rn \rangle$, where $|M_i| = rn$ for $i < \nu$, $|M_\nu| \le rn$, and NH_L abbreviates $\mathsf{NH}_L[rn, 2N]$. Thus defined, NH^* is 2^{-N}-almost universal, has domain $\{0,1\}^*$, and requires rn bits of key material. This modification shifts some of the weight to the $\mathsf{PolyHash}$ hash; we now require eight extra finite field multiplications for each additional rn bits of input. As long as $r > 4$, however, we require fewer of these multiplications when compared to previous hash-ECB-hash or hash-CTR-hash constructions.

With these modifications, the final two terms in TCT_1's security bound (Theorem 4) would become $8q^2/2^n + 600q^2\ell^2/r^2 2^n + 4q^2(\ell-1)^2/2^n$, where ℓn is now the length of the adversary's longest query, $\ell > 2.5r$, and the remaining terms measure the (S)PRP security of the underlying blockcipher. We also assume $2^n \ge rn$, so that $|M| \bmod rn$ can be encoded within a single n-bit block. Although the constant of 600 is large, we note that setting $r = 16$, for example, reduces it to a more comfortable size — in this case to less than three. The bound for TCT_2 changes in a similar manner. (Note that, if $2^{n-2} \ge rn$, then we can use a single n-bit block for both the tweak domain-separation constants and $\langle |M| \bmod rn \rangle$.)

Beyond birthday-bound security for long messages. When ℓ is not bounded to some small or moderate value, TCT_2 no longer provides beyond-birthday-bound security. The problematic term in the security bound is $q(\ell-1)^2/2^n$. To address this, we return to TCTR (Figure 4.2) and consider a different per-block tweak function.

In particular, $g(T, i) = T \parallel \langle i \rangle$. In the nonce-respecting case, the underlying TBC

$\widetilde{E}$ is then re-tweaked with a never-before-seen value on each message block. Again, think about what happens when $\widetilde{E}$ is replaced by an ideal cipher Π: in the nonce-respecting case, every *block* of plaintext is masked by the output of a fresh random permutation. In other words, every block returned will be uniformly random. Thus we expect a tight bound, in this case. Formalizing this logic yields the following theorem.

Theorem 6. *Let* $\widetilde{E}\colon \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ *be a tweakable blockcipher, and let* $\mathrm{TCTR}[\widetilde{E}]_K$ *and* $\mathrm{TCTR}[\widetilde{E}]_K^{-1}$ *be defined as above, with* $g\colon \mathcal{T}' \times \mathbb{N} \to \mathcal{T}$ *an arbitrary injective mapping. Let A be a nonce-respecting adversary that runs in time t, and asks q queries of total length at most $\mu = \sigma n$ bits. Then there exists some adversary B making at most σ queries and running in time $\mathcal{O}(t)$ such that* $\mathbf{Adv}^{\widetilde{\mathrm{srnd}}}_{\mathrm{TCTR}[\widetilde{E}]}(A) \leq \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(B)$.

Consequently, using this variation of TCTR in Theorems 4 and 5 would remove the $q(\ell-1)^2$ term from the bounds, thereby lifting message length concerns. Note that, if this change is made, then $g(T,i)$ needs to be computed up to ℓ times per invocation, rather than just once. This problem may be mitigated by using the XEX [46] TBC in place of LRW, which makes incrementing the tweak extremely fast without significantly changing our security bound.

When the above changes are made, TCT_1 and TCT_2 offer efficient tweakable ciphers on an unbounded domain, losing security guarantees only after $\mathcal{O}(2^{n/2})$ (resp., $\mathcal{O}(2^{2n/3})$) bits have been enciphered.

4.4.4 Instantiating $\widetilde{V}$ with conventional encryption

To further highlight the flexibility surfaced by our compositional approach, we point out that $\widetilde{V}$ can be realized directly using conventional blockcipher-based encryption.

Consider the implementation of $\widetilde{V}, \widetilde{V}^{-1}$ shown in Listing 4.3. We recognize this immediately as counter-mode encryption, but with the initial value T surfaced as an input to make it a tweakable cipher. (Rogaway [47] formalized this as a "nonce-based encryption scheme".)

```
procedure $\mathsf{CTR}_K(T, X)$:
$X_1, X_2, \ldots, X_b \xleftarrow{n} X$
for $i = 1$ to $b$ do
    $Y_i \leftarrow E_K(T + i) \oplus X_i$
return $Y_1 Y_2 \cdots Y_b$
```

Listing 4.3: Counter mode (CTR) can be used to instantiate $\widetilde{V}$.

Unfortunately, we cannot use the main PIV security statement, Theorem 2, with this $\widetilde{F}$, because it is not SRND-secure against nonce-respecting adversaries. Nonetheless, examination of the proof of Theorem 2 shows that it can be modified to work. To that end, we introduce a new notion of security, SRND\$:

$$\mathbf{Adv}_{\widetilde{E}}^{\widetilde{\mathrm{srnd\$}}}(A) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K} :\, A^{\widetilde{\mathbb{E}}_K(\cdot), \widetilde{\mathbb{E}}_K^{-1}(\cdot)} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\$(\cdot), \$(\cdot)} \Rightarrow 1 \,\right].$$

Given $\widetilde{E}$ with tweakspace $\mathcal{T}$ and message space $\mathcal{X}$, the oracle $\mathbb{E}_K$ takes a query $X \in \mathcal{X}$ and: (1) samples $T \xleftarrow{\$} \mathcal{T}$, (2) returns $E_K(T, X)$. Oracle $\mathbb{E}_K^{-1}$ behaves likewise on input $Y \in \mathcal{X}$, sampling $T \xleftarrow{\$} \mathcal{T}$ and returning $E_K^{-1}(T, Y)$. With this, one can state an alternative composition theorem.

Lemma 1. *Let sets $\mathcal{T}, \mathcal{Y}, \mathcal{T}', \mathcal{X}$, integer N, TBC $\widetilde{F}$ and tweakable cipher $\widetilde{V}$ be as described in Theorem 2. Let A be an adversary making $q < 2^N/4$ queries and running in time t. Then there exist adversaries B and C, making q and $2q$ queries, respectively, and both running in $O(t)$ time such that $\mathbf{Adv}_{\mathsf{PIV}[\widetilde{F}, \widetilde{V}]}^{\widetilde{\mathrm{sprp}}}(A) \le \frac{5q^2}{2^N} + \mathbf{Adv}_{\widetilde{V}}^{\widetilde{\mathrm{srnd\$}}}(B) + \mathbf{Adv}_{\widetilde{F}}^{\widetilde{\mathrm{sprp}}}(C)$.*

Then a standard proof shows that the CTR TBC in Figure 4.3 is SRND\$-secure, up to a birthday bound, if E is secure as a PRP.

Lemma 2. *Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher, and let CTR be as in Figure 4.3. Let A be an adversary running in time t, and ask asking q queries, these totalling at most $\mu = \sigma n$ bits. Then $\mathbf{Adv}^{\widetilde{\mathrm{srnd\$}}}_{\widetilde{V}}(A) \leq \sigma^2/2^n + \mathbf{Adv}^{\mathrm{prp}}_{E}(B)$, where B asks at most σ queries, and runs in time $\mathcal{O}(t)$.*

Thus, one could compose counter-mode encryption with $\widetilde{F}$ based on LRW to build an efficient, birthday-bound-secure STPRP.

Here we point out that there is much similarity between the SIV construction of Rogaway and Shrimpton [49], and PIV using this counter-mode $\widetilde{V}$. Indeed, one can view $\widetilde{F}_{K'}(T \parallel X[N+1..], X[1..N])$ as a special kind of PRF (one with invertibility properties), that takes input (T, X) and returns a "synthetic IV" for use in counter-mode. The second application of $\widetilde{F}_{K'}$ then serves to hide this synthetic IV in way that leaves it recoverable to those in possession of key K'. The SIV construction achieves both privacy and authenticity by using the IV as a plaintext authenticator, too. In the next chapter, we'll look at generic ways to build authenticated encryption with the PIV composition.

4.5 VCV design and implementation

In this section, we introduce a novel wideblock tweakable cipher called "VHASH-CTR-VHASH" (VCV) that is a refinement of TCT_1. VCV completely removes TCT_1's finite field multiplications, simplifying its implementation and offering the potential for performance improvements. We also discuss implementation issues and provide benchmarks.

4.5.1 Removing finite field multiplication

Recall that TCT_1 employs LRW, and that LRW in turn requires an ϵ-AXU hash function. These hash functions are typically instantiated using $\mathsf{PolyHash}$, which evaluates a polynomial (with coefficients determined by the input) in $\mathbb{F}_{2^n}$ at the point encoded by the key $K \in 2^n$. However, finite field multiplications in, e.g., $\mathbb{F}_{2^{128}}$ are expensive (between roughly 63% and 245% the cost of an AES call in modern CPUs [18], in terms of throughput). Although TCT_1 succeeds in limiting the number of multiplications to a small constant (that doesn't grow with the input length), these multiplications still incur a performance penalty and complicate implementations.

However, $\mathsf{LRW}[H, E]$ can work with a group operator $+$ that need not be $\oplus$, provided H is ϵ-AΔU with respect to $+$:

$$\mathsf{LRW}[H, E]_{(K,L)}(T, X) = E_K(X + H_L(T)) + H_L(T).$$

Although this generalization is not explicit in the original paper [30], the authors list UMAC's hash function [6] among the possible instantiations, indicating they had this generalization in mind. In any case, the security proof carries through as-is.

We opted to instantiate VCV by using $\mathsf{LRW}[\mathsf{VHASH}, \mathsf{AES}]$ for the fixed input-length component of PIV, where VHASH is the hash function used by VMAC [15, 27]. VHASH (specifically, VHASH-128) is 2^{-120}-ADU with respect to addition in $\mathbb{Z}_{64}^2$. This corresponds to two standard unsigned 64-bit additions.

4.5.2 Implementation

We used a modified form of Ted Krovetz's public-domain VMAC code[5] for our VHASH implementation. To take advantage of instruction-level parallelism, we in-

[5]`http://fastcrypto.org/vmac/`

terleaved instructions for the final VHASH calculation with the AES-NI instructions for the CTR-mode component of VCV.

In some architectures, this interleaving largely hides the latency of the VHASH calculation. On an i7-4770 with 4096-byte blocks, for example, VHASH runs at 0.31 cpb, while CTR-AES runs at 0.65 cpb; thus, one would expect VCV to run at best at $0.31 + 0.31 + 0.65 = 1.27$ cpb in the absence of pipelining (this conservatively ignores the remaining cost of the two LRW computations). However, our implementation runs at 1.08 cpb on this architecture. The difference is the equivalent of 61% the stand-alone VHASH cost.

When AES-NI instructions are not available, our implementation falls back to using the OpenSSL library. (OpenSSL itself supports AES-NI instructions, but using OpenSSL functions instead of our specialized interleaved code reduces throughput by about 1/3.)

4.5.3 Benchmarks

We compare the throughput of our VCV implementation to both GCM-AES-128[6] and CTR-AES-128 in Table 4.3. The GCM-AES-128 comparisons are interesting for two reasons: (1) VCV can be used for the same purpose as GCM, i.e., authenticated encryption, despite offering a strictly stronger type of security (STPRP vs. AEAD) and (2) although we were unable to obtain any optimized software implementations for HEH or HCTR-style tweakable ciphers, GCM performance should provide an approximate lower bound for these modes because it requires a blockcipher invocation and a finite field multiplication for each n bits of input (cf. Table 4.1). Meanwhile, EME will likely operate at less than half the throughput of CTR-AES-128, since the latter requires two AES invocations per n bits of input, with no possibility of

[6]i.e., Galois Counter Mode (GCM) implemented with the version of AES that uses 128-bit keys.

parallelism between the two passes though the plaintext.

García reports [31] that EME is between 2.21 and 2.34 times as slow as CTR-AES-128 on a 4KB buffer, suggesting that this estimation works in EME's favor (the ratio changes with the architecture). Moreover, EME performed substantially better than HCTR and HEH (2.77 cpb compared to 3.97 and 4.33 cpb, respectively; we emphasize, though, that as these tests were performed on an i5-2400, they cannot be compared directly to our own results.) Unfortunately, we were unable to obtain the source code and so could not run these tests on the same machines as our VCV benchmarks.

CPU	VCV (4KB)	GCM (8KB)	CTR (4KB)
Broadwell i7-4770 (3.4 GHz)	1.08	1.77	0.64 (2x = 1.28)
Ivy Bridge i5-3570 (3.4 GHz)	1.41	2.98	0.71 (2x = 1.42)
Core2 DUO E8400 (3.0 GHz)	9.60	16.01	7.79 (2x = 15.58)

Table 4.3: Performance of VCV, GCM, and CTR in cycles per byte on various architectures. García reports [31] that EME operates at less than half the speed of CTR on an i5-2400.

For VCV and CTR, we performed the benchmarks by timing how long it took to repeatedly encrypt a 4KB buffer 100,000 times. For GCM, we used OpenSSL's native benchmark on an 8KB buffer (this works in GCM's favor when performance is measured in cycles per byte) with the native engine selected (`openssl speed -evp aes-128-gcm`). Both our VCV implementation and OpenSSL 1.0.1f were compiled using `gcc` 4.8.2 with the optimization level set to `-O2`. In neither case did performance significantly improve with `-O3`. We used Intel's optimized AES-NI implementation for CTR [25] when available, which outperforms the OpenSSL implementation.

In all cases, VCV significantly outperforms GCM. On Broadwell and Core2 chips, VCV also outperforms the estimated cycles-per-byte lower-bound for EME, but provides similar performance on Ivy Bridge. Hence, we suspect VCV would do at least as well as an optimized software implementation of any of the modes of Table 4.1.

We note that the recent AEZ tweakable cipher [23] claims performance comparable to CTR. It obtains this by using a scaled-back variant of AES that uses four rounds instead of the standard ten. Hence, unlike VCV, one cannot prove that AEZ is secure under the assumption that AES is secure. There are, however, heuristic arguments suggesting that AEZ is safe, regardless. A comparison to VCV over four-round AES wouldn't be particularly meaningful, since these same heuristic arguments would not apply to VCV.

5. AEAD from Tweakable Ciphers

We now turn our attention to a new use of PIV (and other tweakable ciphers), that of building authenticated encryption with associated data (AEAD) [45] via a generalization of Bellare and Rogaway's "encode-then-encipher" [4]. Bellare and Rogaway show that when messages are augmented with a nonce and redundancy, one can obtain authenticated encryption [3] simply by passing these encoded messages directly through an SPRP-secure blockcipher with a sufficiently large domain. (Similar tricks do not work if the primitive is merely IND-NM or IND-CCA secure [1].) We revisit encode-then-encipher in the tweakable setting. In particular, we precisely identify the salient properties of the mapping from header-message pairs (H, M) to tweakable cipher inputs (T, X), and explore where best to apportion state, randomness, and redundancy in this encoding.

Our results answer natural questions regarding the relationship between tweakable ciphers and nonce-based encryption. But there remains the question of *why* one would adopt this method, given the existence of highly efficient AEAD schemes, such as OCB [48, 46]. In addition to its simplicity, there are two important, practical advantages of our approach:

(1) It admits the use of multiple decryption error messages, while remaining robust against side-channel attacks that seek to exploit them (e.g., padding oracles).

(2) It can eliminate bandwidth overhead by leveraging nonces, randomness and

redundancy already present in plaintext inputs.

The first point holds because, loosely, changing any ciphertext bit results in randomizing every resulting plaintext bit. Thus, descriptive error messages, e.g. `bad_padding` or `bad_seq_number`, cannot be leveraged to forge useful future ciphertexts. We also remark that, under our approach, nonce repetitions only leak equality of plaintexts.

As an example of the second point, messages in protocols that include sequence numbers, human-readable fields, or padding likely contain useful nonces and redundancy. In these cases, the encoding step of encode-then-encipher is implicit, acting as an assumed model for how information is encoded into bit strings before being passed down the stack to encrypt.

Before moving on to formal definitions for our encode-then-encipher scheme, we will provide a brief example of what we have in mind in order to motivate certain departures from the standard AEAD formulation. Given a header H and a message M, we need to encode (H, M) into a tweakable cipher input, (T, X). What if we choose a nonce N of some fixed length, set $T = N \| H$, and $X = N \| M$? We are already sending H down the wire, but our ciphertext cannot simply be $C = \widetilde{E}_K(T, X)$ because we also need N to decrypt. So our ciphertext should be (N, C). In this example, $T = N \| H$ was an encoded header, and $X = N \| M$ was an encoded message. However, we wish to consider encoding schemes separately from tweakable ciphers, so we will discard the symbols T and X in favor of $\overline{H}$ and $\overline{M}$, respectively. The mapping from H to $\overline{H} = N \| H$ is non-deterministic, but can be completely reproduced provided one knows N. We therefore refer to N as our *reconstruction information.* (Note that the message encoding function needs the reconstruction information, and therefore has a different signature than the header encoding function; further, the reconstruction information should not depend on M). The recipient computes $\widetilde{E}_K^{-1}(\overline{H}, C)$ and verifies that N is a prefix.

Authenticated encryption with associated data. An *authenticated encryption scheme with associated data* is a tuple $\Psi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consisting of a key-generation algorithm $\mathcal{K}$, an encryption algorithm $\mathcal{E}$, and a decryption algorithm $\mathcal{D}$. For simplicity, we assume the key-generation algorithm $\mathcal{K}$ samples a key from a non-empty set of the same name. The encryption algorithm, which may be randomized or stateful, is a mapping $\mathcal{E} : \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{R} \times \{0,1\}^*$. Thus, encryption takes a key $K \in \mathcal{K}$, a *header* $H \in \mathcal{H} \subseteq \{0,1\}^*$, and a message $M \in \mathcal{M} \subseteq \{0,1\}^*$, and returns some *reconstruction information* $R \in \mathcal{R}$, along with a ciphertext C. We write $(R, C) \xleftarrow{\$} \mathcal{E}_K(H, M)$ to mean running $\mathcal{E}_K$ on (H, M), this returning (R, C). The deterministic decryption algorithm is a mapping $\mathcal{D} : \mathcal{K} \times \mathcal{H} \times \mathcal{R} \times \{0,1\}^* \to \mathcal{M} \cup \mathsf{Errors}$, where Errors is a set such that $\mathcal{M} \cap \mathsf{Errors} = \emptyset$. (We do not insist that $|\mathsf{Errors}| = 1$.) For proper operation, we require that $\Pr[\, \mathcal{D}_K(H, \mathcal{E}_K(H, M)) = M \,] = 1$ for all $K \in \mathcal{K}$, $H \in \mathcal{H}$ and $M \in \mathcal{M}$. If $\mathcal{E}$ is stateful, this must hold for all states.

Let us discuss this formalization of AEAD. First, in practice, the header H will be sent in the clear along with the ciphertext (e.g., when the header is needed for routing), but our encode-then-encipher AEAD schemes may encode H into some related $\overline{H}$ for internal use. If this encoding is non-deterministic, we use the reconstruction information R to deliver whatever is needed by decryption to properly reconstruct this $\overline{H}$ from H. For example, R may consist of a counter, some random bits, or some redundancy. (It may also be the empty string.)

To avoid constructions that are trivially insecure by virtue of writing message or key bits into R, we require the following. Given any sequence of inputs $\{(H_i, M_i)\}_{i \le q}$ and any two sequences of possible outcomes $\{(R_i, C_i)\}_{i \le q}$ and $\{(R'_i, C'_i)\}_{i \le q}$, we must have that for any $H \in \mathcal{H}$, $M, M' \in \mathcal{M}$, $K, K' \in \mathcal{K}$, and $r \in \mathcal{R}$,

$$\Pr[\, R = r \mid \mathcal{E}_K(H_i, M_i) = (R_i, C_i) \text{ for } i \le q \,] =$$
$$\Pr[\, R' = r \mid \mathcal{E}_{K'}(H'_i, M'_i) = (R'_i, C'_i) \text{ for } i \le q \,]$$

where $(R, C) \xleftarrow{\$} \mathcal{E}_K(H, M)$ and $(R', C') \xleftarrow{\$} \mathcal{E}_K(H, M')$, the states of $\mathcal{E}_K$ and $\mathcal{E}_{K'}$ being conditioned on the two transcripts, respectively. That is, R (hence, R') can depend only on H, q, and coins tossed by $\mathcal{E}_K$ on the query that generates R.

Second, by allowing $|\mathsf{Errors}| > 1$, we let our AEAD schemes return multiple, distinct error messages. This can be useful in practice for, say, diagnostics within a protocol session. Often, allowing decryption to return multiple error messages has been problematic in practice; witness the various "padding oracle" attacks on SSL/TLS [54, 8, 43]. For our encode-then-encipher AEAD schemes, such attacks will not be a concern.

AEAD security notions. Our desired privacy notion is indistinguishability of ciphertexts from random bits. However, we do not require the recovery information to be random (e.g., R might be a counter), so we modify the usual IND\$-CPA notion slightly. To be specific, we measure the privacy of an AEAD scheme Ψ via the following advantage: $\mathbf{Adv}^{\mathrm{priv}}_{\Psi}(A) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}\,;\ A^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1 \,\right] - \Pr\left[\, K \xleftarrow{\$} \mathcal{K}\,;\ A^{\$_K(\cdot,\cdot)} \Rightarrow 1 \,\right]$. Here, $\$_K(\cdot,\cdot)$ is an oracle that on input (H, M), computes $(R, C) \xleftarrow{\$} \mathcal{E}_K(H, C)$, samples $Y \xleftarrow{\$} \{0,1\}^{|C|}$, and returns (R, Y).

The authenticity goal for our AEAD scheme is integrity of ciphertexts, INT-CTXT [3]. Namely, we define $\mathbf{Adv}^{\mathrm{int\text{-}ctxt}}_{\Psi}(A) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}\,;\ A^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \text{ Forges} \,\right]$ where the boolean event Forges is true if and only if A asks a query (H, R, C) to its $\mathcal{D}_K$ oracle such that (1) $\mathcal{D}_K(H, R, C) \notin \mathsf{Errors}$, and (2) no prior query to $\mathcal{E}_K(\cdot,\cdot)$

returned (H, R, C). Without loss of generality, A halts as soon as Forges becomes true.

5.1 Encoding schemes

Informally, an encoding algorithm is responsible for reformatting its input, injecting randomness, state or redundancy, while decoding validates and distills out the original input data.

Fix a message space $\mathcal{M}$, a header space $\mathcal{H}$, an encoded message space $\overline{\mathcal{M}} \subseteq \{0,1\}^*$, and an encoded header space $\overline{\mathcal{H}} \subseteq \{0,1\}^*$. All of these sets must be non-empty (but could equal $\{\varepsilon\}$). Also fix a set Errors such that $\mathsf{Errors} \cap \mathcal{M} = \emptyset$.

As mentioned earlier, we need two types of encoding functions. A *header encoding function* $\mathrm{Encode}_{\mathsf{H}} : \mathcal{H} \to \overline{\mathcal{H}}$ maps headers to encoded headers, possibly in some random or stateful fashion. We require there to be a non-empty set $\mathcal{R}$ and a bijection $\langle \cdot, \cdot \rangle : \mathcal{R} \times \mathcal{H} \to \overline{\mathcal{H}}$ with the property that for all H, $\mathrm{Encode}_{\mathsf{H}}(H) = \langle R, H \rangle$ for some $R \in \mathcal{R}$. In other words, we should always be able to recover H from $\mathrm{Encode}_{\mathsf{H}}(H)$, and any particular output of $\mathrm{Encode}_{\mathsf{H}}(H)$ can be reconstructed from H given the corresponding R. We call $\mathrm{Encode}_{\mathsf{H}}$ an $(\mathcal{H}, \mathcal{R}, \overline{\mathcal{H}})$-encoder (leaving $\langle \cdot, \cdot \rangle$ implicit).

A *message encoding scheme* $\mathsf{EncodeMsg} = (\mathrm{Encode}_{\mathsf{M}}, \mathrm{Decode}_{\mathsf{M}})$ consists of a *message encoding function* $\mathrm{Encode}_{\mathsf{M}} : \overline{\mathcal{H}} \times \mathcal{M} \to \overline{\mathcal{M}}$ and a *message decoding function* $\mathrm{Decode}_{\mathsf{M}} : \overline{\mathcal{H}} \times \overline{\mathcal{M}} \to \mathcal{M} \cup \mathsf{Errors}$. We allow $\mathrm{Encode}_{\mathsf{M}}$ to be randomized or stateful. We require $\mathrm{Decode}_{\mathsf{M}}(\overline{H}, \overline{M}) = M \in \mathcal{M}$ if and only if $\Pr\left[\, \mathrm{Encode}_{\mathsf{M}}(\overline{H}, M) = \overline{M} \,\right] > 0$ for some state of $\mathrm{Encode}_{\mathsf{M}}$; otherwise, $\mathrm{Decode}_{\mathsf{M}}(\overline{H}, \overline{M}) \in \mathsf{Errors}$ (note that we allow, for example, $\mathrm{Decode}_{\mathsf{M}}(\overline{H}, \overline{M}) = (\perp, \overline{H}, \overline{M}) \in \mathsf{Errors}$). The $\mathrm{Decode}_{\mathsf{M}}$ algorithm must be deterministic, and all algorithms should run in linear time. We call $\mathsf{EncodeMsg}$ a $(\mathcal{H}, \mathcal{M}, \overline{\mathcal{H}}, \overline{\mathcal{M}}, \mathsf{Errors})$-encoding scheme.

We only consider encoding functions having an associated *maximal stretch*, defined to be the smallest $s \in \mathbb{N}$ such that, for all $H \in \mathcal{H}$, $\overline{H} \in \overline{\mathcal{H}}$, and $\overline{M} \in \overline{\mathcal{M}}$, $|\mathrm{Encode}_{\mathsf{H}}(H)| \leq |H| + s$ and $\left|\mathrm{Encode}_{\mathsf{M}}(\overline{H}, M)\right| \leq |M| + s$.

Encoding scheme properties. Our encode-then-encipher AEAD security theorems will surface two key properties of the encoding mechanisms.

The first property speaks to the likelihood that an encoding scheme can be made to repeat outputs.

Let A be an adversary that asks q queries (not necessarily distinct) to an oracle f, receiving $\overline{Y}_1, \overline{Y}_2, \ldots, \overline{Y}_q$ in response, and that halts by outputting these values. (Think of f as a message — or header-encoding function.) Without loss of generality, we can assume A is deterministic. Let $\delta\colon \mathbb{N} \to [0,1]$ be a function. Generalizing the definition from [4], we say f is (d,δ)-colliding if

$$\Pr\left[\, A^f \Rightarrow (\overline{Y}_1, \ldots, \overline{Y}_q)\colon \exists i_1 < \ldots < i_d \text{ such that } \overline{Y}_{i_1} = \cdots = \overline{Y}_{i_d} \,\right] \leq \delta(q).$$

This notion is only defined for $d \geq 2$. Given a (d,δ)-colliding oracle, we may assume, without loss of generality that $\delta(0) = \delta(1) = 0$.

The second property captures the probability that a random string (of a given length) is a valid encoding. One can think of this as a measure of the density of encodings. Thus, let $\epsilon \in \mathbb{R}$ be a real number. The we say the $(\mathcal{H}, \mathcal{M}, \overline{\mathcal{H}}, \overline{\mathcal{M}}, \mathsf{Errors})$-encoding scheme $\mathsf{EncodeMsg} = (\mathrm{Encode}_{\mathsf{M}}, \mathrm{Decode}_{\mathsf{M}})$ is *ϵ-sparse* if for all positive integers n and all $H \in \overline{\mathcal{H}}$, $\left|\{C \in \{0,1\}^n \,:\, \mathrm{Decode}(H,C) \notin \mathsf{Errors}\}\right| / 2^n \leq \epsilon$.

```
procedure $\mathcal{E}_K(H, M)$:
$\overline{H} \leftarrow \langle R, H\rangle \xleftarrow{\$} \text{Encode}_{\mathsf{H}}(H)$
$\overline{M} \xleftarrow{\$} \text{Encode}_{\mathsf{M}}(\overline{H}, M)$
return $R, \widetilde{E}_K(\overline{H}, \overline{M})$

procedure $\mathcal{D}_K(H, R, C)$:
$\overline{H} \leftarrow \langle R, H\rangle$
$\overline{M} \leftarrow \widetilde{E}_K^{-1}(\overline{H}, C)$
return $\text{Decode}_M(\overline{H}, \overline{M})$
```

Listing 5.1: The AEAD scheme $\Psi[\text{Encode}_{\mathsf{H}}, \mathsf{EncodeMsg}, \widetilde{E}]$. Reconstruction information R allows decryption to reconstruct $\overline{H}$ from the header. (This is in lieu of sending $\overline{H}$ as part of the ciphertext, or forcing the calling application to replace H with $\overline{H}$.) All authenticity checks are implicitly carried out by Decode_M.

5.2 AEAD via encode-then-encipher

Now, let $\widetilde{E}\colon \mathcal{K} \times \overline{\mathcal{H}} \times \overline{\mathcal{M}} \to \overline{\mathcal{M}}$ be a tweakable cipher. Let $\text{Encode}_{\mathsf{H}}$ be a $(\mathcal{H}, \mathcal{R}, \overline{\mathcal{H}})$-encoder, and let $\mathsf{EncodeMsg} = (\text{Encode}_{\mathsf{M}}, \text{Decode}_{\mathsf{M}})$ be an $(\mathcal{H}, \mathcal{M}, \overline{\mathcal{H}}, \overline{\mathcal{M}}, \mathsf{Errors})$-encoding scheme, for some non-empty sets $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{R}$. From these, we define an encode-then-encipher AEAD scheme $\Psi[\text{Encode}_{\mathsf{H}}, \mathsf{EncodeMsg}, \widetilde{E}]$ in Figure 5.1. As a simple example, let $\text{Encode}_{\mathsf{H}}$ prepend an 64-bit counter R to the header H, and $\text{Encode}_{\mathsf{M}}(\overline{H}, M)$ return $M \,\|\, 0^{80}$. Then $\text{Encode}_{\mathsf{H}}$ is $(2, 0)$-colliding, and $\text{Encode}_{\mathsf{M}}$ is 2^{-80}-sparse (but $(2, 1)$-colliding). We point out that all authenticity checks implicitly take place inside of the $\text{Decode}_{\mathsf{M}}$ function.

Security theorems and discussion. Here we give the privacy and authenticity security statements for our encode-then-encipher AEAD scheme. We'll give the statements first, and then discuss what they imply. Proofs follow the discussion.

Theorem 7 (Privacy). *Let $\Psi = \Psi[\text{Encode}_{\mathsf{H}}, \mathsf{EncodeMsg}, \widetilde{E}]$ be defined as in Figure 5.1. Let s be the maximal stretch of $\mathsf{EncodeMsg}$, s' be the maximal stretch of $\text{Encode}_{\mathsf{H}}$, and let m be the length of the shortest $\overline{M} \in \overline{\mathcal{M}}$ satisfying $\text{Decode}_{\mathsf{M}}(\overline{H}, \overline{M}) \neq \mathsf{Errors}$ for some $\overline{H} \in \overline{\mathcal{H}}$. Let A be an adversary making q queries totaling μ bits, and running in time t. Then if $\text{Encode}_{\mathsf{H}}$ is (d, δ_H)-colliding and $\text{Encode}_{\mathsf{M}}$ is $(2, \delta_M)$-*

colliding for some δ_M that is increasing and convex on $\{0, 1, \ldots, q\}$, there is an adversary B such that

$$\mathbf{Adv}^{\mathrm{priv}}_{\Psi}(A) \leq \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(B) + \left(\delta_M(d-1) + \frac{(d-1)(d-2)}{2^{m+1}}\right)\left\lceil \frac{q}{d-1} \right\rceil + \left(\delta_M(q) + \frac{q(q-1)}{2^{m+1}}\right)\delta_H(q)$$

where B makes q queries of total length at most $\mu + q(s+s')$ bits and runs in time $\mathcal{O}(t)$.

Theorem 8 (Authenticity)**.** *Let $\Psi[\widetilde{E}] = \Psi[\mathrm{Encode}_{\mathsf{H}}, \mathsf{EncodeMsg}, \widetilde{E}]$ be defined as in Figure 5.1. Let s be the stretch of $\mathsf{EncodeMsg}$, s' be the maximal stretch of $\mathrm{Encode}_{\mathsf{H}}$, and define m as the length of the shortest $\overline{M} \in \overline{\mathcal{M}}$ satisfying $\mathrm{Decode}_{\mathsf{M}}(\overline{H}, \overline{M}) \neq \mathsf{Errors}$ for some $\overline{H} \in \overline{\mathcal{H}}$. Let A be an adversary making $q_{\mathcal{E}}$ (resp., $q_{\mathcal{D}}$) queries totaling $\mu_{\mathcal{E}}$ (resp., $\mu_{\mathcal{D}}$) bits to its encryption (resp., decryption) oracle, and running in time t. Then if $\mathrm{Encode}_{\mathsf{M}}$ is ϵ-sparse and $q_{\mathcal{E}} + q_{\mathcal{D}} < 2^{m-1}$, there is an adversary B such that*

$$\mathbf{Adv}^{\text{int-ctxt}}_{\Psi[\widetilde{E}]}(A) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\widetilde{E}}(B) + 2q_{\mathcal{D}}\epsilon.$$

where B makes $q_{\mathcal{E}}$ forward-queries of total length $(\mu_{\mathcal{E}} + q_{\mathcal{E}}s)$ bits, $q_{\mathcal{D}}$ inverse-queries of total length $(\mu_{\mathcal{D}} + q_{\mathcal{D}}s')$ bits, and runs in $\mathcal{O}(t)$ time.

To begin our discussion of these results, consider the case that $\mathrm{Encode}_{\mathsf{H}}$ is $(2,0)$-colliding. Then the privacy bound (Theorem 7) simplifies to $\mathbf{Adv}^{\mathrm{priv}}_{\Psi}(A) \leq \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(B)$. This is intuitive, because, if the tweak $\overline{H}$ never repeats, then the outputs $\widetilde{E}_K(\overline{H}, \overline{M})$ are uniformly random strings for *any* valid encoding of (H, M) into $\overline{M}$; $\mathrm{Encode}_{\mathsf{H}}(H, M) = M$ suffices. Thus, good encodings of the header H can substantially reduce the burden placed upon encoding of (H, M) into $\overline{M}$.

This case generalizes nicely. Say that we can assume that the probability of $\mathrm{Encode}_{\mathsf{H}}$ producing any $\overline{H}$ more than d times, for some small constant $d \ll q$, is negligible. Then the final term in the bound can effectively be ignored. The second term is roughly $q\delta_M(d) + q/2^{m+1}$. Now, notice that δ_M is evaluated at d, not q, and so $q\delta_M(d)$ can be made negligible by encoding a reasonable amount of randomness into $\overline{M}$ (e.g. $\log(q)$ bits). For some natural choices of $\mathsf{EncodeMsg}$ then, $q/2^{m+1}$ will be the dominating term, where m is the shortest length of $\overline{M}$. But to achieve good authenticity bounds, which we will turn to momentarily, m is unlikely to let $q/2^{m+1}$ ruin the bound.

We point out that in the un-tweakable setting considered in [4], privacy must be achieved by encoding randomness or state into $\overline{M}$. The presence of the tweak allows us to shift these "extra" bits into the encoding of the header, which potentially reduces the number of bits that must be cryptographically processed.

In the extreme case that $\overline{H}$ is fixed across all queries (perhaps by design, or perhaps a result of a faulty implementation), the construction reverts to the un-tweakable setting. In this case, $\text{Encode}_{\mathsf{H}}$ is $(2,1)$-colliding, we recover, essentially, the bound of Bellare and Rogaway [4]. (But note that we consider indistinguishability from random bits, which is stronger than the privacy notion considered there.)

Turning now to the authenticity bound (Theorem 8), note that if $\text{Encode}_{\mathsf{M}}$ inserts b redundant bits (so $\epsilon \approx 2^{-b}$) and $q_{\mathcal{E}} + q_{\mathcal{D}} \ll 2^m$, the second term of our authenticity bound is approximately $q_{\mathcal{D}}/2^b$. Thus, if the tweakable cipher $\widetilde{E}$ has STPRP-security up to (say) 2^{80} queries (e.g., an appropriately instantiated PIV with $N = 256$), then encoding the header with a nonce, and the message with 80 bits of redundancy, yields an AEAD scheme with roughly 80-bit privacy and authenticity guarantees, and one that can tolerate nonce-misuse.

We note that the proof of Theorem 8 can be easily modified to show that the stated bound holds even if the adversary controls the coins and state of $\text{Encode}_{\mathsf{M}}$ and $\text{Encode}_{\mathsf{H}}$. Additionally, we assume only that decryption oracle queries are not redundant — adversaries are *not* assumed to respect any nonces encoded into the headers or messages.

Relationship to deterministic authenticated encryption. Motivated by the key-wrapping problem, Rogaway and Shrimpton [49] introduce *deterministic authenticated encryption* (DAE). The encryption and decryption algorithms of a DAE scheme take a header string as an auxiliary input, as in the AEAD case. However, both algorithms are required to be deterministic. The corresponding security notion considers only adversaries that never repeat queries (or that make redundant queries to a decryption oracle).

Our encode-then-encipher AEAD scheme may be viewed as a DAE scheme, provided the $\text{Encode}_{\mathsf{M}}$ and $\text{Encode}_{\mathsf{H}}$ algorithms are deterministic. One can easily show that the DAE security of a scheme is upper bounded by the sum of its privacy and authenticity bounds, as given in Theorems 7 and 8. We note that, under the assumption that adversaries do not repeat queries, the privacy bound from Theorem 7 reduces to $\mathbf{Adv}^{\widetilde{\text{prp}}}_{\widetilde{E}}(B) + q^2/2^{m+1}$ for some adversary B. Obtaining this result requires trivial proof modifications, and generalizes a result from [49], which considers $\text{Encode}_{\mathsf{M}}(\overline{H}, M) = M \,\|\, 0^s$.

Subsequent work. In a recent paper [23], Hoang, Krovetz, and Rogaway introduced a security notion they term *robust authenticated encryption*. In order to meet this security definition, it must be not only be difficult for an adversary to forge a ciphertext, but being able to find one or more forgeries should not increase the adversary's ability to make further forgeries. This property is valuable if bandwidth restrictions permit only small authentication tags. The authors observe that tweakable ciphers provide this stronger form of security.

5.3 Proof of Theorem 7

Proof. Let $\Pi \xleftarrow{\$} \mathrm{BC}(\overline{\mathcal{H}}, \overline{\mathcal{M}})$ be a random cipher. Let $\Psi[\Pi]$ be the AEAD scheme obtained by replacing $\widetilde{E}_K$ and $\widetilde{E}_K^{-1}$ with Π and Π^{-1}, respectively, everywhere in the algorithms of $\Psi[\mathrm{Encode}_{\mathsf{H}}, \mathsf{EncodeMsg}, \widetilde{E}]$. Let $\mathcal{E}_\Pi$ be the corresponding encryption algorithm. We begin by observing that $\Pr\left[\, A^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1 \,\right] - \Pr\left[\, A^{\mathcal{E}_\Pi(\cdot,\cdot)} \Rightarrow 1 \,\right] \le \mathbf{Adv}_{\widetilde{E}}^{\widetilde{\mathrm{prp}}}(B)$, for the standard adversary B that simulates $\mathcal{E}_K$ or $\mathcal{E}_\Pi$, depending on its own oracle. This leaves us with $\mathbf{Adv}_{\Psi}^{\mathrm{priv}}(A) \le \mathbf{Adv}_{\widetilde{E}}^{\widetilde{\mathrm{prp}}}(B) + \Pr\left[\, A^{\mathcal{E}_\Pi(\cdot,\cdot)} \Rightarrow 1 \,\right] - \Pr\left[\, K \xleftarrow{\$} \mathcal{K}\,;\, A^{\$_K(\cdot,\cdot)} \Rightarrow 1 \,\right]$.

Now we proceed by a sequence of games. Let s be the stretch of $\mathrm{Encode}_{\mathsf{M}}$. Game G_1 implements $\mathcal{E}_\Pi$, using lazy-sampling to define Π. In particular, on the i-th query (H_i, M_i), Game G_1 computes the encodings $\overline{H}_i, \overline{M}_i$, and then samples a potential value S_i to assign to $\Pi(\overline{H}_i, \overline{M}_i)$. This value will be valid so long as $\Pi(\overline{H}_i, \overline{M}_i)$ has not already been set, and as long as S_i has not been used with $\overline{H}_i$ before; otherwise, Game G_1 sets bad_1 or bad_2, and then reassigns S_i an appropriate value.

Since Game G_2 is identical to Game G_1 until bad_1 or bad_2 is set, we have

$$\begin{aligned}
\Pr\left[\, A^{\mathcal{E}_\Pi(\cdot,\cdot)} \Rightarrow 1 \,\right] &= \Pr\left[\, G_1(A) \Rightarrow 1 \,\right] \\
&= \Pr\left[\, G_1(A) \Rightarrow 1 \ \wedge\ (\mathsf{bad}_1 \vee \mathsf{bad}_2) \,\right] \\
&\qquad + \Pr\left[\, G_1(A) \Rightarrow 1 \ \wedge\ \neg(\mathsf{bad}_1 \vee \mathsf{bad}_2) \,\right] \\
&\le \Pr\left[\, G_2(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,\right] + \Pr\left[\, G_2(A) \Rightarrow 1 \ \wedge\ \neg(\mathsf{bad}_1 \vee \mathsf{bad}_2) \,\right]
\end{aligned}$$

where in the final line we use an alternative formulation of the fundamental lemma of game-playing [5]. Now, notice that in Game G_2, the value of S_i is always uniformly random in $\{0,1\}^{|M_i|+s}$, and $\mathcal{E}_\Pi$'s outputs are independent of each M_i. Consequently we can postpone assigning values to each M_i until *after* A halts. This in turn allows us to postpone checking to see if bad_1 or bad_2 should be set without altering the probability that they will be. We make both these changes to create Game 3 (so $\Pr[\, G_2(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,] = \Pr[\, G_3(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,]$). Thus,

$$\begin{aligned}\Pr[\, G_2(A) \Rightarrow 1 \ \wedge\ \neg(\mathsf{bad}_1 \vee \mathsf{bad}_2) \,] &= \Pr[\, G_3(A) \Rightarrow 1 \ \wedge\ \neg(\mathsf{bad}_1 \vee \mathsf{bad}_2) \,] \\ &\le \Pr[\, G_3(A) \Rightarrow 1 \,] = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}\,;\, A^{\$_K(\cdot,\cdot)} \Rightarrow 1 \,\right]\end{aligned}$$

where the final equality follows from the fact that in Game G_3, each S_i is sampled independently and uniformly at random from the set of appropriately sized strings. To recap, at this point we have

$$\mathbf{Adv}^{\mathrm{priv}}_{\Psi}(A) \le \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\widetilde{E}}(B) + \Pr[\, G_3(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,]$$

We need an upper bound for $\Pr[\, G_3(A)\,;\, \mathsf{bad}_1 \vee \mathsf{bad}_2 \,]$. Therefore suppose that A has just generated its output after running in G_3. We will first bound the probability that bad_1 gets set. Let $N = \left|\left\{\overline{H}_i \,:\, i \le q\right\}\right|$ be the number of distinct tweak encodings generated during the course of the game. Let $R_1, R_2, \ldots, R_N \subseteq \{1, 2, \ldots, q\}$ be equivalence classes characterized by the property that i and j are in the same class if and only if $\overline{H}_i = \overline{H}_j$. The probability that bad_1 will be set is at most $\sum_k \delta_M(|R_k|)$.

Note that the upper bound is obtained by summing the values of the increasing convex function δ_M at the points $|R_1|, |R_2|, \ldots, |R_N|$ where $|R_1|+|R_2|+\cdots+|R_N| = q$.

Consequently the bound is largest (for fixed q) when $N = 1$ and $R_1 = \{1, 2, \ldots, q\}$. Let Capped denote the event that each $|R_k| < d$; then $\Pr[\, G_3(A)\,;\, \neg\mathsf{Capped} \,] \le \delta_H(q)$. Given that Capped occurs, $\sum_k \delta_M(|R_k|)$ is largest when $N = \lceil q/(d-1) \rceil$ and $|R_k| =$

$d-1$ for $k = 1, 2, \ldots, N-1$. We have

$$\begin{aligned}
\Pr[\, G_3(A) \,;\, \mathsf{bad}_1 \,] \leq \Pr[\, G_3(A) \,;\, \mathsf{bad}_1 \mid \mathsf{Capped} \,] \\
+ \Pr[\, G_3(A) \,;\, \mathsf{bad}_1 \mid \neg\mathsf{Capped} \,] \Pr[\, G_3(A) \,;\, \neg\mathsf{Capped} \,] \\
\leq \delta_M(d-1)\lceil \frac{q}{d-1} \rceil + \delta_M(q)\delta_H(q).
\end{aligned}$$

By a similar argument,

$$\begin{aligned}
\Pr[\, G_3(A) \,;\, \mathsf{bad}_2 \,] \leq \Pr[\, G_3(A) \,;\, \mathsf{bad}_2 \mid \mathsf{Capped} \,] \\
+ \Pr[\, G_3(A) \,;\, \mathsf{bad}_2 \mid \neg\mathsf{Capped} \,] \Pr[\, G_3(A) \,;\, \neg\mathsf{Capped} \,] \\
\leq \frac{(d-1)(d-2)}{2^{m+1}}\lceil \frac{q}{d-1} \rceil + \frac{q(q-1)}{2^{m+1}}\delta_H(q),
\end{aligned}$$

since the standard $i \mapsto i(i-1)/2^{m+1}$ birthday bound (for the probability of a collision among i independent random variables sampled uniformly from $\{0,1\}^m$) meets the criterion of an increasing convex function. This completes the proof. $\square$

5.4 Proof of Theorem 8

Proof. Let B be the STPRP adversary that simulates the INT-CTXT experiment for A and outputs 1 if A would set Forges to true. Then $\mathbf{Adv}^{\text{int-ctxt}}_{\Psi}(A) \leq \mathbf{Adv}^{\widetilde{\text{sprp}}}_{\widetilde{E}}(B) + \mathbf{Adv}^{\text{int-ctxt}}_{\Psi[\Pi]}(A)$, where $\Psi[\Pi]$ is the scheme obtained by replacing $\widetilde{E}_K$ and $\widetilde{E}_K^{-1}$ with Π and Π^{-1}, respectively, everywhere in the algorithms of Ψ, and $\mathbf{Adv}^{\text{int-ctxt}}_{\Psi[\Pi]}(A)$ is defined in the natural way (with probabilities over the random choice of Π, rather than K).

Consider Game G_4. By defining Π through lazy sampling, we have $\mathbf{Adv}^{\text{int-ctxt}}_{\Psi[\Pi]}(A) = \Pr[\, G_4(A) \,;\, \mathsf{Forges} \,]$. Note that in the code for the $\mathcal{D}_\Pi$ oracle, we do not need to check

Games $\boxed{G_1}$, G_2

```
Oracle $\mathcal{E}_\Pi(H, M)$:
  $i \leftarrow i + 1$; $M_i \leftarrow M$
  $\overline{H}_i \leftarrow \langle r, H_i \rangle \xleftarrow{\$} \mathrm{Encode}_T(H_i)$
  $\overline{M}_i \xleftarrow{\$} \mathrm{Encode}_M(\overline{H}_i, M_i)$
  $S_i \xleftarrow{\$} \{0,1\}^{|\overline{M}|}$
  if $S_i \in [(..\Pi]\,(\overline{H}_i, \cdot))$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\boxed{S_i \xleftarrow{\$} \{0,1\}^{|\overline{M}|} \setminus [(..\Pi]\,(\overline{H}_i, \cdot))}$
  if $\overline{M}_i \in \mathsf{dom}(\Pi(\overline{H}_i, \cdot))$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $\boxed{S_i \leftarrow \Pi(\overline{H}_i, \overline{M}_i)}$
  $\Pi(\overline{H}_i, \overline{M}_i) \leftarrow S_i$
  return $r \parallel S_i$
```

Game G_3

```
procedure Main(A):
  $b \xleftarrow{\$} A^{\mathcal{E}(\cdot,\cdot)}$
  for $i \leftarrow 1$ to $q$ do
    $\overline{M}_i \xleftarrow{\$} \mathrm{Encode}_M(\overline{H}_i, M_i)$
      if $\overline{M}_i \in \mathsf{dom}(\Pi(\overline{H}_i, \cdot))$ then $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    if $S_i \in [(..\Pi]\,(\overline{H}_i, \cdot))$ then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $\Pi(\overline{H}_i, \overline{M}_i) \leftarrow S_i$
  return $b$

Oracle $\mathcal{E}_\Pi(H, M)$:
  $i \leftarrow i + 1$; $M_i \leftarrow M$
  $\overline{H}_i \leftarrow \langle r, H_i \rangle \xleftarrow{\$} \mathrm{Encode}_T(H_i)$;
  $S_i \xleftarrow{\$} \{0,1\}^{|M|+s}$
  return $r \parallel S_i$
```

Listing 5.2: Games for the proof of Theorem 7. Boxed commands are omitted in Game G_2, causing the $\mathcal{E}_\Pi$ oracle to always return random strings. We use Game G_3 to bound the probability that this change can be detected by an adversary (as measured by the probability that a bad will be set).

GAME G_4

```
**Oracle $\mathcal{E}_\Pi(H, M)$:**
$\overline{H} \leftarrow \langle r, H \rangle \xleftarrow{\$} \mathrm{Encode}_T(H)$
$\overline{M} \xleftarrow{\$} \mathrm{Encode}_M(\overline{H}, M)$
**if** $\overline{M} \notin \mathsf{dom}(\Pi(\overline{H}, \cdot))$ **then**
    $\Pi(\overline{H}, \overline{M}) \xleftarrow{\$} \{0,1\}^{|\overline{M}|} \setminus [(..\Pi]\,(\overline{H}, \cdot)$
**return** $r \,\|\, \Pi(\overline{H}, \overline{M})$
```

```
**Oracle $\mathcal{D}_\Pi(H, r, C)$:**
$\overline{H} \leftarrow \langle r, H \rangle$
$\overline{M} \xleftarrow{\$} \{0,1\}^{|C|} \setminus \mathsf{dom}(\Pi(\overline{H}, \cdot))$
$\Pi(\overline{H}, \overline{M}) \leftarrow C$
$M \leftarrow \mathrm{Decode}_{\mathsf{M}}(\overline{H}, \overline{M})$
**if** $M \in \mathsf{Errors}$ **then** $\mathsf{Forges} \leftarrow \mathsf{true}$
**return** $M$
```

Listing 5.3: Game G_4 simulates the IND-CTXT experiment for $\mathcal{SE}[\Pi]$.

if $\Pi^{-1}(\overline{H}, Y)$ has already been defined; this possibility is excluded by the fact that A does not repeat queries to $\mathcal{D}_\Pi$, and does not send $\mathcal{D}_\Pi$ a value previously returned by $\mathcal{E}_\Pi$ (while preserving the header).

Fix some query to $\mathcal{D}_\Pi$. The probability that A forges on this query is equal to the probability that the corresponding, randomly chosen value of $\mathcal{M}$ is a valid encoding. There are at most $2^{|Y|}\epsilon$ valid encodings of the correct length, and $\mathcal{M}$ is sampled from a set of size at least $2^{|Y|} - (q_\mathcal{E} + q_\mathcal{D})$. Consequently, A forges on this query with probability at most $2^{|Y|}\epsilon/(2^{|Y|} - (q_\mathcal{E} + q_\mathcal{D})) < 2^m\epsilon/(2^m - 2^{m-1}) = 2\epsilon$. A union bound completes the proof. $\square$

6. Conclusion

We have contributed novel TBC constructions that provide beyond-birthday-bound security ($\mathsf{LRW2}$ and TCT_2), and others that provide standard birthday-bound security more efficiently (TCT_1 and VCV). Moreover, we have shown how wideblock tweakable ciphers can provide authenticated encryption with associated data in a way that is robust against common implementation errors, and that can leverage pre-existing randomness and redundancy in plaintexts to improve security. In the process of proving the security properties of the PIV framework, we believe we have also advanced the central thesis of Liskov, Rivest, and Wagner: namely, that tweakable blockciphers permit short, easily verified proofs, and are therefore the "right" starting point for symmetric-key cryptography.

Our contributions to the design and use of tweakable ciphers were concurrent with those of other researchers. We briefly describe some of the results that have expanded on our own, and then discuss directions for further research.

6.1 Subsequent work

Recall that our first major result was quantifying how much additional security could be obtained by using two rounds of LRW instead of one. We showed that the result, LRW2, is secure as long as an attacker is limited to $q \ll 2^{2n/3}$ queries, where n is the block size. This naturally raises the question of how much security r rounds of LRW could provide. We conjectured in our original paper [29] that r rounds would be sufficient as long as $q \ll 2^{rn/(r+1)}$.

In a follow-up work, Lampe and Seurin [28] proved that this bound holds for $2r$ rounds, but likewise conjectured that r rounds would be sufficient. This remains an open problem.

Others have explored ways of constructing TBCs directly, rather than from an underlying blockcipher. The advantage of this approach is that it offers the possibility of more efficient constructions; the cost is that security proofs are impossible or rely on heuristic models. For example, proofs in the ideal cipher model treat a blockcipher as though it were chosen uniformly at random from the set of all possible blockciphers.

Jean, Nikolić, and Peyrin described the `TWEAKEY` framework [26] for modifying so-called key-alternating blockciphers (of which AES is an example) to transform them into TBCs. Mennink [36] constructed a TBC that is secure for all $q \ll 2^n$, but the security proof is in the ideal cipher model. Cogliati, Lampe, and Seurin have announced [13] that their forthcoming CRYPTO 2015 paper shows LRW2 provides beyond-birthday-bound security even when both instances of the randomly keyed blockcipher are replaced with, public, random permutations (the hash function still has a secret key). More generally, $2r$ rounds of this variant permit $q \ll 2^{rn/(r+1)}$ queries, and again the authors conjecture that r rounds are sufficient for this bound.

Hoang, Krovetz, and Rogaway [23] introduced the AEZ wideblock tweakable cipher. AEZ uses a stripped-down version of AES under the hood, and thus its security does not reduce to the security of (full strength) AES; however, the authors presented some informal arguments that truncating AES is safe in the specific context of AEZ. The benefit of this approach is that AEZ becomes extremely fast in software: the authors reported a throughput of about 0.7 cycles per byte. Moreover, they expanded on our AEAD-from-tweakable-cipher results and prove that using tweakable ciphers provides what they describe as *robust authenticated encryption*.

6.2 Looking forward

While the recent trend towards using heuristic models to analyze TBC constructions may at first appear problematic, we hope that it proves to be a step towards standardizing a dedicated TBC algorithm. Such a TBC would occupy a position similar to the one AES enjoys today: our confidence in its security would rest on sustained, unsuccessful cryptanalysis rather than a standard-model reduction to some underlying primitive. Security proofs in heuristic models would serve to bolster this confidence. TBC-based algorithms, including VCV, would be able to immediately make use of the new primitive, benefiting from its improved efficiency and, one would hope, its security.

However, it will be some time before a TBC algorithm makes it through some standardization process and resists cryptanalysis long enough to garner support from the cryptographic community. Until then, blockcipher-based TBC constructions will continue to offer a versatile and trustworthy foundation for symmetric-key cryptography.

Bibliography

[1] Jee Hea An and Mihir Bellare. Does encryption with redundancy provide authenticity? In Birgit Pfitzmann, editor, *Advances in Crptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2001.

[2] DiegoF. Aranha, Julio Lpez, and Darrel Hankerson. Efficient software implementation of binary field arithmetic using vector instruction sets. In Michel Abdalla and PauloS.L.M. Barreto, editors, *Progress in Crptology – LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2010.

[3] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions andanalysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.

[4] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer Berlin Heidelberg, 2000.

[5] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a frameworkforcode-basedgame-playingproofs. In Serge Vaudenay, editor, *Advances in Crptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[6] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In Michael Wiener, editor, *Advances in Crptology – CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.

[7] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. AES-based authenticated encryption modes in parallel high-performance software. *IACR Cryptology ePrint Archive*, 2014:186, 2014.

[8] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In Dan Boneh, editor, *Advances in Crptology*

– *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599. Springer, 2003.

[9] Mickaël Cazorla, Kevin Marquet, and Marine Minier. Survey and benchmark of lightweight block ciphers for wireless sensor networks. *IDEA*, 64(128):34, 2013.

[10] Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR. In Kaisa Nyberg, editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 2008.

[11] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology – INDOCRYPT 2006*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer Berlin Heidelberg, 2006.

[12] Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew Robshaw, editor, *Fast Software Encryption*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.

[13] Benot Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking Even-Mansour ciphers. Cryptology ePrint Archive, Report 2015/539, 2015.

[14] Jean-Sbastien Coron, Yevgeniy Dodis, Avradip Mandal, and Yannick Seurin. A domain extender for the ideal cipher. In Daniele Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 273–289. Springer Berlin Heidelberg, 2010.

[15] Wei Dai and Ted Krovetz. VHASH security. *IACR Cryptology ePrint Archive*, 2007:338, 2007.

[16] Niels Ferguson. AES-CBC+ Elephant diffuser: A disk encryption algorithm for Windows Vista, 2006.

[17] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function. *Submission to NIST*, 2010, 2008.

[18] Shay Gueron. AES-GCM software performance on the current high end CPUs as a performance baseline for CAESAR competition. *Directions in Authenticated Ciphers (DIAC)*, 2013.

[19] Shai Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology – INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer Berlin Heidelberg, 2005.

[20] Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer Berlin Heidelberg, 2007.

[21] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer Berlin Heidelberg, 2003.

[22] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer Berlin Heidelberg, 2004.

[23] VietTung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption: AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.

[24] VietTung Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2012.

[25] Intel. Intel IPSec v3 library. `https://downloadcenter.intel.com/download/22972/Optimized-IPSec-Cryptographic-Library`, February 2015.

[26] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer Berlin Heidelberg, 2014.

[27] Ted Krovetz. Message authentication on 64-bit architectures. In Eli Biham and AmrM. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 327–341. Springer Berlin Heidelberg, 2007.

[28] Rodolphe Lampe and Yannick Seurin. Tweakable blockciphers with asymptotically optimal security. In Shiho Moriai, editor, *Fast Software Encryption*, volume 8424 of *Lecture Notes in Computer Science*, pages 133–151. Springer Berlin Heidelberg, 2014.

[29] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable blockciphers with beyond birthday-bound security. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Crptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2012.

[30] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin Heidelberg, 2002.

[31] Mancillas López. Efficient hardware implementation of some tweakable enciphering schemes. Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2007.

[32] J. Luo, K.D. Bowers, A. Oprea, and L. Xu. Efficient software implementations of large finite fields GF(2^n) for secure storage applications. *ACM Transactions on Storage (TOS)*, 8(1):2, 2012.

[33] Cuauhtemoc Mancillas-Lopez, Debrup Chakraborty, and Francisco Rodriguez-Henriquez. Reconfigurable hardware impementations of tweakable enciphering schemes. *IEEE Transactions on Computers*, 59:1547–1561, 2010.

[34] L. Martin. XTS: A mode of AES for encrypting hard disks. *Security Privacy, IEEE*, 8(3):68–69, May 2010.

[35] David McGrew and John Viega. The galois/counter mode of operation (gcm). *Submission to NIST. http://csrc. nist. gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec. pdf*, 2004.

[36] Bart Mennink. Optimally secure tweakable blockciphers. In *Fast Software Encryption*, Lecture Notes in Computer Science. Springer, 2015.

[37] Kazuhiko Minematsu. Improved security analysis of XEX and LRW modes. In Eli Biham and AmrM. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2007.

[38] Kazuhiko Minematsu. Beyond-birthday-bound security based on tweakable block cipher. In Orr Dunkelman, editor, *Fast Software Encryption*, volume 5665 of *Lecture Notes in Computer Science*, pages 308–326. Springer Berlin Heidelberg, 2009.

[39] Kazuhiko Minematsu and Tetsu Iwata. Building blockcipher from tweakable blockcipher: Extending FSE 2009 proposal. In Liqun Chen, editor, *Cryptography and Coding*, volume 7089 of *Lecture Notes in Computer Science*, pages 391–412. Springer Berlin Heidelberg, 2011.

[40] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

[41] Ben Morris, Phillip Rogaway, and Till Stegers. How to encipher messages on a small domain. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 286–302. Springer Berlin Heidelberg, 2009.

[42] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-rackoff revisited. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 189–199. ACM, 1997.

[43] KennethG. Paterson and Arnold Yau. Padding oracle attacks on the ISO CBC mode encryption standard. In Tatsuaki Okamoto, editor, *Topics in Crptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 305–323. Springer, 2004.

[44] Gordon Procter. A note on the CLRW2 tweakable block cipher construction. *IACR Cryptology ePrint Archive*, 2014:111, 2014.

[45] Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[46] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In PilJoong Lee, editor, *Advances in Crptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[47] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–358. Springer, 2004.

[48] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[49] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Crptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[50] Palash Sarkar. Improving upon the TET mode of operation. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Crptology – ICISC 2007*, volume 4817 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2007.

[51] Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Trans. Inf. Theor.*, 55(10):4749–4760, October 2009.

[52] Rich Schroeppel and H Orman. The hasty pudding cipher. *AES candidate submitted to NIST*, page M1, 1998.

[53] Thomas Shrimpton and R. Seth Terashima. A modular framework for building variable-input-length tweakable ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Crptology – ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 405–423. Springer, 2013.

[54] Serge Vaudenay. Security flaws induced by cbc padding applications to SSL, IPSEC, WTLS.... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–545. Springer Berlin Heidelberg, 2002.

[55] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[56] M.N. Wegman and J.L. Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.